Community Experience Distilled

WordPress 3.0 jQuery

Enhance your WordPress website with the captivating effects of jQuery

Tessa Blakeley Silver

[PACKT] PUBLISHING open source* community experience distilled

Wordpress 3.0 jQuery

Enhance your WordPress website with the captivating effects of jQuery

Tessa Blakeley Silver

[PACKT] open source
community experience distilled
PUBLISHING

BIRMINGHAM - MUMBAI

Wordpress 3.0 jQuery

First published: September 2010

Production Reference: 1170910

Published by Packt Publishing Ltd.
32 Lincoln Road
Olton
Birmingham, B27 6PA, UK.

ISBN 978-1-849511-74-2

www.packtpub.com

Cover Image by Jose Argudo (jose@joseargudo.com)

Credits

Author
Tessa Blakeley Silver

Reviewer
Chris Gossmann

Acquisition Editor
Chaitanya Apte

Development Editor
Ved Prakash Jha

Technical Editors
Aaron Rosario
Mohd. Sahil

Copy Editors
Janki Mathuria
Laxmi Subramanian

Indexer
Rekha Nair

Editorial Team Leader
Akshara Aware

Project Team Leader
Priya Mukherji

Project Coordinator
Vincila Colaco

Proofreader
Aaron Nash

Graphics
Nilesh R. Mohite
Geetanjali Sawant

Production Coordinator
Aparna Bhagat

Cover Work
Aparna Bhagat

About the Author

Tessa Blakeley Silver has prior experience in print design and traditional illustration. She evolved over the years into web and multi-media development, where she focuses on usability and interface design.

Prior to starting her consulting and development company *hyper3media* (pronounced hyper-cube media) `http://hyper3media.com`, Tessa was the VP of Interactive Technologies at *eHigherEducation*, an online learning and technology company developing compelling multimedia simulations, interactions, and games which met online educational requirements like 508, AICC and SCORM. She has also worked as a consultant and freelancer for *J. Walter Thompson* and the *Diamond Trading Company* (formerly known as *DeBeers*) and was a Design Specialist and Senior Associate for *PricewaterhouseCoopers*' East Region Marketing department.

Tessa has authored a few books for Packt Publishing, including **WordPress 2.8 Theme Design** and **Joomla 1.5 Template Design**.

I send a huge "thank you" to the Packt team who have made this title possible and whose help in getting it out into the world has been invaluable. Special thanks to Chaitanya and Thorsten for their editing work. Additional big-time "thank you" goes out to Vincila for the backbreaking work and diligence it takes to keep to a schedule.

I'd also like to thank the exemplary WordPress and jQuery community (Matt and John, you guys Rock) and all who participate and power the Open Source world and strive to improve the accessibility of the Web for all.

Additional thanks goes out to my very patient partner and our little daughter (who's not so patient) who per usual, spent quite a few evenings without me while I worked on this title. I love you both and appreciate your flexibility with me while I work on interesting books and projects (yes, I'm working on getting better at estimating how much time it really, really takes to write a chapter).

About the Reviewer

Chris Gossmann is Lead Developer and co-owner of the well-known Thematic, an open-source WordPress Theme Framework. Chris has over four years experience with WordPress. Today he is constantly enhancing Thematic. As the co-worker of Milo, one of the best web designers, Chris Gossmann creates highly customized WordPress and BuddyPress solutions for international customers.

Table of Contents

Preface

This easy-to-use guide will walk you through the ins and outs of creating sophisticated professional enhancements and features, specially tailored to take advantage of the WordPress personal publishing platform. It will walk you through clear, step-by-step instructions to build several custom jQuery solutions for various types of hypothetical clients and also show you how to create a jQuery and Wordpress plugin.

What this book covers

Chapter 1, *Getting Started: WordPress and jQuery*...This chapter introduces the reader to the core fundamentals that they need to be familiar with in order to get the most out of the book. HTML, CSS, PHP, and JavaScript syntax, and how to recognize the various parts of those syntaxes are covered, as well as a list of "tools of the trade" which covers what features their code editor, browser, and even image editor should have. The chapter also illustrates exactly how CSS, JavaScript, and jQuery work in the browser with the HTML served up from the WordPress site.

Chapter 2, *Working with jQuery in WordPress*...This chapter goes into the details of how to start working with jQuery specifically within WordPress. It covers how to properly include jQuery using the Script API and focuses on jQuery's selectors (very important for working in WordPress) as well as jQuery's top functions.

Chapter 3, *Digging Deeper: Understanding jQuery and WordPress Together*...This chapter takes the reader to a deeper level and introduces them to all the ways that jQuery can be applied to a WordPress site: Through a custom script in the WordPress theme, as a jQuery plugin called in through the theme, and lastly, as a custom jQuery script or plugin applied to a WordPress plugin! The ways to affect a WordPress site with jQuery are numerous, and the pros and cons of each method is considered so that the reader can assess their own projects accurately. The chapter also introduces the reader to their first "hypothetical client" and covers how to create their own jQuery plugin and then wrap that jQuery plugin into a WordPress plugin so that a site administrator could easily implement the enhancement without having to know how to edit the theme.

Chapter 4, *Doing a Lot More with Less: Making Use of Plugins for Both jQuery and WordPress*...You thought you learned quite a bit in *Chapter 3*? Hang on to your mouse. You're about to embark on a nice little project that requires you getting familiar with the popular jQuery plugin Colorbox, as well as the popular WordPress plugin Cforms II and mashing the two with your own custom jQuery magic to whip up some slick event registration that will knock a client's socks off.

Chapter 5, *jQuery Animation within WordPress*...If you're going to use jQuery, you might as well really use it to its fullest, which means animation. This chapter covers using jQuery's animation functions and shortcuts to create some sharp, well timed visual enhancements that grab the site user's attention as well as create a super slick navigation enhancement and an awesome rotating slideshow of sticky posts.

Chapter 6, *WordPress and jQuery's UI*...Now that we have some animation chops under our belt, we can make that work even easier by using jQuery's UI plugin which includes the Easing and Color plugins we learned about in *Chapter 5*. In this chapter, we're going to also take advantage of the UI plugin's widgets and events features to create some super useful interfaces in our WordPress site.

Chapter7, *AJAX with jQuery and WordPress*...This chapter introduces you to what AJAX is and isn't along with the top ways to get started using AJAX techniques in your WordPress site; you'll load in HTML from other pages on your site, get your tweets and favorite flickr pictures pulled in through JSON, and last but not least, custom AJAXing the built in WordPress comment form.

Chapter 8, *Tips and Tricks for Working with jQuery and WordPress*...This chapter covers the top tips and tricks for getting the most out of jQuery specifically within WordPress. Most of these best practices are covered throughout the title but in this chapter we take a look at exactly why they're so important, espeically within the context of WordPress and how to implement them.

Appendix A, *jQuery and WordPress Reference Guide*...Dog-ear this appendix and consider it your "cheat sheet". Once you work your way through the book, why waste time hunting and pecking your way back through it to recall some function's bit of syntax and what its parameters are? This book extracts the most important information about jQuery and WordPress and breaks it down into an easy-to-skim reference guide so that you can easily find the syntax for most jQuery selectors, remind yourself of the top jQuery functions that you'll need for most WordPress development and their parameters, as well as helpful WordPress template tags and API functions and other useful WordPress know-how such as structuring the Loop and the Theme Template Hierarchy.

What you need for this book

- WordPress (2.9.2 or 3.0)
- The jQuery library (1.4.2)
- A web server (local WAMP or MAMP installation or hosted by a provider)
- A web browser (Firefox or better)
- A good code or HTML editor

Who this book is for

This book is for anyone who is interested in using jQuery with a WordPress site. It's assumed that most readers will be WordPress developers with a pretty good understanding of PHP or JavaScript programming and at the very least, experience with HTML/CSS development who want to learn how to quickly apply jQuery to their WordPress projects.

Conventions

In this book, you will 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: "We can include other contexts through the use of the `include` directive."

A block of code is set as follows:

```
<script type="text/javascript">
     jQuery("document").ready(function(){
       jQuery("p").css("background-color", "#ff6600");
 });
</script>
```

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

```
<script type="text/javascript">
     jQuery("document").ready(function(){
       jQuery("p").css("background-color", "#ff6600");
 });
</script>
```

For for clarity and conciseness, many code examples in this title are extracted. An extracted block of code is set as follows:

```
...
jQuery("p").css("background-color", "#ff6600");
}
...
```

Code and markup preceded and ended with ellipses "..." are extracted from the full context of code and/or a larger body of code and markup. Please refer to the downloadable code bundle to see the entire work.

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: "clicking the **Next** button moves you to the next screen".

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 for this book

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
Getting Started: WordPress and jQuery

Welcome to WordPress and jQuery. The WordPress web-based publishing platform and jQuery's JavaScript library are two of the most powerful tools used on the Web today. Combining these tools together doubles the power and flexibility of your websites. Both technologies, being easy and fun to learn, create a recipe for web magic. I hope you're ready for some fun and interesting insights by learning how jQuery can improve your WordPress development experience.

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

- This book's approach and the core JavaScript language and WordPress skills that you should understand in order to gain maximum advantage from it
- The essential software tools that you'll need to get your project up and running
- The basic overviews of jQuery and WordPress

If anything in any of the following sections puzzles you, you may need a little more background information and understanding before moving forward with this title. No worries though, I'll point you towards some excellent sources for more information.

This book's approach

This title introduces you to the essentials and best practices of using jQuery with WordPress. It is not an introduction to programming with JavaScript and PHP, nor is it a primer on using CSS with HTML mark-up. I'm assuming that you're a WordPress site developer and/or a WordPress theme designer. Maybe you're just someone who spends enough time managing and tinkering with a WordPress site that you probably qualify as one of the above or both. Regardless of how you label yourself, you use WordPress. WordPress helps you, or your clients, get content out there quickly and simply, and you're always looking for ways to do more, faster, and easily.

jQuery is a library that speeds the time and reduces the complications of writing custom JavaScripts. I'm sure you must know that JavaScripts can be useful to a website in a number of ways. They can also enable it with really cool-looking features. While I'll be covering, in depth, as much on jQuery as possible, we won't be looking at jQuery as the "big deal", JavaScript library entity that most other books stress. Instead, we'll be considering jQuery a great tool that can help us get more done, more easily (and yes, "with less" code) using WordPress.

To recap: So, you're a WordPress user, developer, or designer? Great. Let's look at this "tool" called jQuery. It's going to make your WordPress development a lot easier and probably look a lot better. Ready to get started?

Core fundamentals you need to know

As I mentioned, this book is geared toward WordPress users, visual theme designers, and developers who are looking to learn to do more with WordPress by using jQuery. I've tried to write this title so that client-side and server-side scripting or programming experience isn't explicitly required. However, you'll see at the very least that a general familiarity with the given subjects will help.

Regardless of your web development skill-set or level, you'll be walked through with clear, step-by-step instructions. Let's go over the web development skills and WordPress know-how that you'll need to be familiar with to gain the maximum benefit from this book. Again, I'll also point you to good resources if you feel you need a little more background.

WordPress

To start with, you should already be familiar with the most current, stable version of WordPress. You should understand the basics of getting WordPress installed and running on a web server or locally on your machine (especially as you'll need an installation to tinker with the examples in this book). Not to worry, I'll point you in the right direction for getting a basic local installation of WordPress on your Mac or PC. Plus, many hosting providers offer easy one-click installs. You'll have to check with your hosting provider to see if they offer WordPress. I'll also point you toward a few other good resources for WordPress installations. Getting WordPress up and running is often the easiest part of using WordPress.

Going a tad more in-depth, you'll do well to know your way around the WordPress administration panel. You'll need to be familiar with adding content to the WordPress publishing system and how posts, categories, static pages, and sub-pages work. You'll also want to understand using the **Media** upload tools to add images to posts and pages, as well as creating galleries. Lastly, understanding the basics of installing and using different themes and plugins will also be helpful, though we will cover this to some extent in this title.

Even if you'll be working with a more technical WordPress administrator, you should have an overview of what the WordPress site that you're developing for entails, and what (if any) themes or additional plugins or widgets will be needed for the project. If your site does require a specific theme or additional plugins and widgets, you'll want to have those installs handy and/or already installed in your WordPress development installation (or **sandbox**—a place to test and play without messing up a live site).

What version of WordPress does this book use?

This book focuses on the new features introduced in versions 2.8, 2.9, and 3.0 RC (Release Candidate—as of the writing of this book). Everything covered in this book has been tested and checked in WordPress 2.9.2 and 3.0 RC. While this title's case studies are developed using version 2.9.2 and 3.0 RC, any newer version of WordPress should have the same core capabilities, enabling you to enhance themes and plugins with jQuery for it using these techniques. Bug fixes and new features for each new version of WordPress are documented at `http://WordPress.org`.

If you are completely new to WordPress, then I recommend you read **WordPress 2.7 Complete** by *April Hodge Silver* and *Hasin Hayder*.

Basic programming

Having an understanding of programming in any client-side or server-side language will help you out here, no matter what language—JavaScript, VBScript, .NET, ASP, PHP, Python, Java, Ruby, you name it. If you're familiar working with, or at the very least looking at, any of those languages, you'll do fine. Of course, the following specific languages will really help.

JavaScript and AJAX techniques

OK, you definitely don't need to have any experience with AJAX whatsoever. But if you know a bit about JavaScript (that's the "J" in "AJAX") you're off to a great start. In particular, you should be able to understand how to recognize the overall syntax and structure of JavaScript statements. For example: what variables look like in JavaScript and how **blocks** for **functions** or **conditions** are set up using "{ }" (curly brackets). You'll also want to know how to properly end a line of JavaScript code with a ";" (semicolon). Again, you don't need direct experience, but you should be comfortable looking at a block of JavaScript code and understanding how it's set up.

For example, let's take a quick look at the following code example, which includes explanatory comments:

```
<script type="text/javascript"> /*this is an XHTML script tag with the
type attribute set to define javascript*/
/*
This is a multi-line Comment.
You can use multi-line comments like this to add instructions or notes
about your code.
*/

//This is a single line comment for quick notes
function writeHelloWorld(){ /*this line sets up a function and starts
block of code*/
   var text1 = "Hello"; //this is a variable called text1

   document.write(text1); /*This writes "Hello" to the HTML body via
the variable "text1"*/

   document.write(" World!"); /*Writes the string " World!" to the
HTML body. Note the  ";" semi-colons ending each statement above, very
important!*/

}// this bracket ends the function block
writeHelloWorld(); /*evokes the function as a statement again, ending
with a ";" semi-colon.*/
//this closes the HTML script tag
</script>
```

If you can follow what's happening in the given code snippet, and you're confident that you could alter, say, the variable without breaking the script, or change the name of the function and where it's evoked, you're doing well enough for this title.

Of course, the more you know about working with different **types** of information such as **strings**, **integers**, and **arrays** as well as **loops** and **if/else** statements, the better. But again, just understanding the general syntax for now, will certainly get you started with jQuery and this title.

AJAX is not really a language. As we'll learn in *Chapter 7*, *AJAX with jQuery and WordPress*, it's simply a set of *techniques* for working with Asynchronous JavaScript and XML, using JavaScript and HTTP requests together to develop highly dynamic pages. Developers like this approach as it allows them to create pages that respond more like desktop programs, than standard web pages. If you're interested in using AJAX with WordPress, in *Chapter 7*, *AJAX with jQuery and WordPress*, we'll get into how jQuery can help you with various AJAX techniques. But it's by no means essential for taking advantage of jQuery with WordPress.

If you're new to JavaScript and want a quick, fun primer, I highly recommend the W3Schools' site. This site is a great resource for priming yourself with all W3C compliant web technology. `http://w3schools.com/js/`. You can find out about AJAX too: `http://w3schools.com/ajax/`.

PHP

You definitely don't have to be a PHP programmer to get through this book, but PHP is what WordPress is built with and its themes use liberal doses of PHP to work their magic! WordPress plugins are almost pure PHP. Any hope of adding jQuery functionality to a WordPress theme or plugin will require braving a little PHP syntax.

As with JavaScript, if you at least understand how basic PHP syntax is structured, you'll be much less likely to make mistakes while retyping or copying and pasting code snippets of PHP and WordPress template tags, in your theme's template files.

The good news is PHP syntax is structured similarly to JavaScript syntax. PHP also uses curly brackets in the same way to denote blocks of code for functions, loops, and other conditions. You also end every statement in PHP with a semicolon just as you would in JavaScript. The main difference is that PHP is evoked by wrapping code snippets inside `<?php ?>` tags, which are not part of the XHTML tag set and JavaScript is evoked by placing code snippets inside the XHTML `<script>` tags. Also, variables in PHP are denoted with a "`$`" (dollar) sign, permanently prepended to the variable name you create, rather than established once with the `var` statement.

The biggest difference is that PHP is a server-side scripting language and JavaScript is client-side. That means that JavaScript downloads and runs inside the user's browser on their machine, while PHP code is pre-interpreted on the web server and only the final, resulting XHTML (and sometimes CSS and JavaScript—you can do a lot with PHP!) is served up into the user's web browser.

Let's take a quick look at some basic PHP syntax:

```
<?php /*All PHP is evoked using greater-than brackets and a "?"
question mark, followed by the letters "php"*/

//This is a single-line comment

/*
This is multi-line
comment block
*/

function newHelloWorld(){/*this sets up a function and code block*/

     $text1 = "Hello"; //creates a variable called: $text1

     echo $text1." World!";  /*tells the HTML page to print , aka:
"echo" the variable $text1 with the string " World!" concatenated onto
it.*/

}//this ends the code block

newHelloWorld(); //calls the function as a statement ending with a
semi-colon.

//the question mark and closing less-than tag end the PHP code.
?>
```

I'm sure you recognize some differences between PHP and JavaScript right away, but there are also quite a few similarities. Again, if you're confident that you could swap out a variable value without breaking the function, you'll do fine with WordPress and this title. As always, the more you know about PHP the better.

Do I have to add "php" to my `<?` starter block?

You'll notice I've set up my PHP starter block as: "`<?php`". Those of you with some PHP knowledge or having some WordPress experience, may be familiar with PHP blocks that just start with `<?` and end with `?>`. On servers with **shorthand support** enabled, you can start a scripting block with just "`<?`" (as well as use a few other cool PHP shorthand tricks).

However, while shorthand support is usually enabled, not everyone's PHP installation will have it enabled. When I have clients or friends who can't seem to get a new plugin or theme to work with their WordPress installation, this often comes up as the culprit. The theme or plugin was written using shorthand and the client's PHP installation doesn't have it enabled and for some reason, their IT guy or hosting provider doesn't want to enable it. To stay as compatible as possible, we'll be using the standard form in this book (`<?php`) rather than the shorthand form.

If you'd like to understand WordPress a little better by knowing more about PHP, again, that W3School site is a great place to start! (`http://w3schools.com/php/`).

After reading this book, if you find PHP really interests you as well as JavaScript, AJAX, and jQuery, you might want to move onto reading **AJAX and PHP: Building Modern Web Applications 2nd Edition** by *Audra Hendrix*, *Bogdan Brinzarea*, and *Cristian Darie*.

More of a visual "see it to do it" learner? `lynda.com` has a remarkable course selection from the top CSS, XHTML/XML, PHP, JavaScript (and yes, even jQuery) people in the world. You can subscribe and take the courses online or purchase DVD-ROMs for offline viewing.

The courses or the monthly subscription might seem pricey at first, but if you're a visual learner, it's worth spending money and time on them. You can refer to the official site at `http://lynda.com`.

Essential tools

Skills are one thing, but the better your tools are, and the more command you have over those tools, the better your skills can be put to use (you can just ask any carpenter, golfer, or app programmer about the sheer importance of the "tools of the trade").

Code/HTML editor

First up, we'll need to meddle with markup and code—lots of markup, CSS, PHP, and jQuery. So, you'll need a good code or HTML editor. Dreamweaver is a great option (`http://www.adobe.com/products/dreamweaver/`), although I prefer to use Coda for Mac (`http://www.panic.com/coda/`). Before I discovered working with Coda, I was very happy with the free editor TextWrangler (`http://www.barebones.com/products/textwrangler/`). When I was working on a PC, I loved the free text/code editor HTML-kit (`http://www.htmlkit.com/`).

There are thousands of editors out there, some free, some expensive, and with varying degrees of features. Just about every developer and designer I've talked to, uses something different and has a ten-minute "schpiel" about why their editor is the best. Ultimately, any HTML or text editor that lets you enable the following features will work just great. I recommend you enable/use all of the following:

- **View line numbers**: This comes in very handy during the validation and debugging process. It can help you find specific lines in a jQuery script, theme, or plugin file, for which a validation tool has returned a fix. This is also helpful for other theme or plugin instructions given by their author, which refer to a specific line of code that might need customizing or editing under different conditions.
- **View syntax colors**: Any worthwhile code and HTML editor has this feature usually set as a default. The good editors let you choose your own colors. This displays code and other markup in a variety of colors, making it easier to distinguish various types of syntax. Many editors also help you identify broken XHTML markup, CSS rules, or PHP code.
- **View non-printing characters**: You might not want this feature turned on all the time. It makes it possible to see hard returns, spaces, tabs, and other special characters that you may or may not want in your markup and code.
- **Text wrapping**: This of course lets you wrap text within the window, so you won't have to scroll horizontally to edit a long line of code. It's best to learn what the key-command shortcut is for this feature in your editor, and/or set up a key-command shortcut for it. You'll find it easier to scroll through unwrapped, nicely-indented, markup and PHP code to quickly get a general overview or find your last stopping point; however, you will still want to turn wrapping on quickly so you can easily see and focus your attention on one long line of code.

- **Load files with FTP or local directories**: An editor that allows you to connect through FTP or see your local working directory in a side panel, is extremely helpful. It saves you from having to manually find files locally in your OS explorer or finder, or from having to upload through an additional FTP client. Being able to connect to your files in a single application just speeds up your workflow.

Free open source HTML editors:

I've also used Nvu (`http://www.net2.com/nvu/`) and KompoZer (`http://kompozer.net/`). They're both free, open source, and available for Mac, PC, and Linux platforms. KompoZer was made from the same source as Nvu and, apparently, fixes some issues that Nvu has. (I haven't run into any major issue with Nvu myself). Both editors are too limited for my regular use, but I do like being able to format HTML text quickly and drag-and-drop form objects onto a page. Both editors have a **Source** view, but you must be careful while switching between the **Normal** and the **Source** view tabs. Nvu and KompoZer are a little *too helpful*, and will try to rewrite your handcoded markup if you haven't set your preferences properly!

Linux users of Ubuntu and Debian (and Mac users with Fink) might also be interested in checking out the Bluefish editor (`http://bluefish.openoffice.nl`). I use Bluefish when working on Ubuntu Linux. I prefer it when on Linux, though it's robust enough to probably be considered more of an IDE (Integrated Development Environment), similar to Eclipse (`http://www.eclipse.org`), rather than just a basic code or HTML editor. Many of you may find that a tool like Bluefish or Eclipse is overkill for your general WordPress development and maintenance needs. On the other hand, if you're serious about WordPress development, they may have features you find invaluable and they are worth downloading and checking out.

Firefox

Finally, you'll need a web browser. I strongly suggest that you use the latest stable version of the Firefox browser, available at `http://mozilla.com/firefox/`.

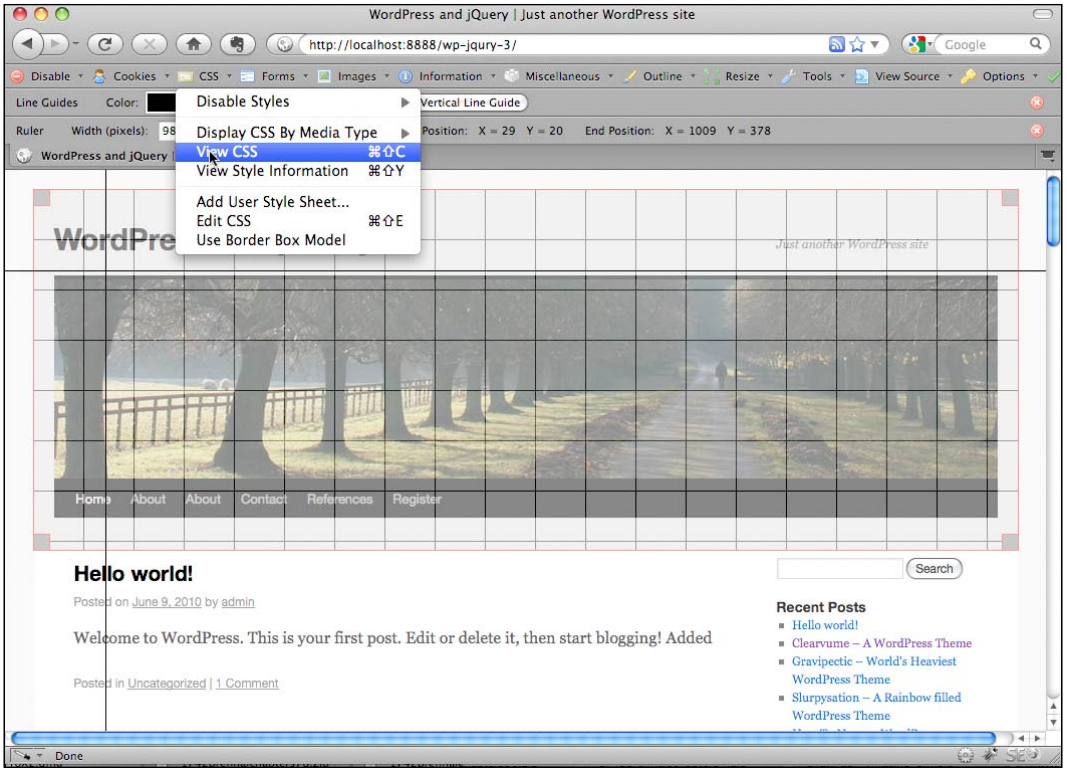
Now one may ask, why use Firefox? While this browser has its setbacks (like any other), on the whole, I view it as an excellent web development tool. For me, it's as essential as my HTML editor, FTP programs, and graphics tools. Firefox has great features that we'll be taking advantage of to help us streamline our WordPress and jQuery enhancements and site production. In addition to built-in features such as the DOM Source Selection Viewer and adhering to CSS2 and some CSS3 standards as specified by the W3C, Firefox also has a host of extremely useful **extensions** such as the **Web Developer Toolbar** and **Firebug**, which I recommend to further enhance your workflow.

If you have some experience with jQuery, you've probably noticed that the great documentation on jQuery's site as well as most jQuery books, tend to focus on the intricacies of jQuery, using very simple and basic HTML markup examples with minimal CSS attributes added. Within WordPress, you'll find yourself working with a theme or plugins that were most likely created by someone else. You'll need an easy way to explore the **Document Object Model** (**DOM**) and CSS that the theme, plugins, and WordPress are generating in order to get jQuery to do what you want with the resulting markup that is generated. The Firefox browser and its extensions allow you to do this more easily than any other browser.

Web Developer toolbar

This is a great extension that adds a toolbar to your Firefox browser. The extension is also available for the Seamonkey suite and the new Flock browser, both of which, are powered by the open source code of Mozilla, just like Firefox. Get it from `http://chrispederick.com/work/web-developer/`.

The toolbar lets you link directly to the browser's DOM and Error Consoles, as well as W3C's XHTML, and CSS validation tools. It also lets you toggle and view your CSS output in various ways, and lets you view and manipulate a myriad of information your site outputs on-the-fly. The uses of this toolbar are endless. Every time I develop a design or create jQuery enhancements, it seems I discover some feature that I have never previously used and yet find quite useful.

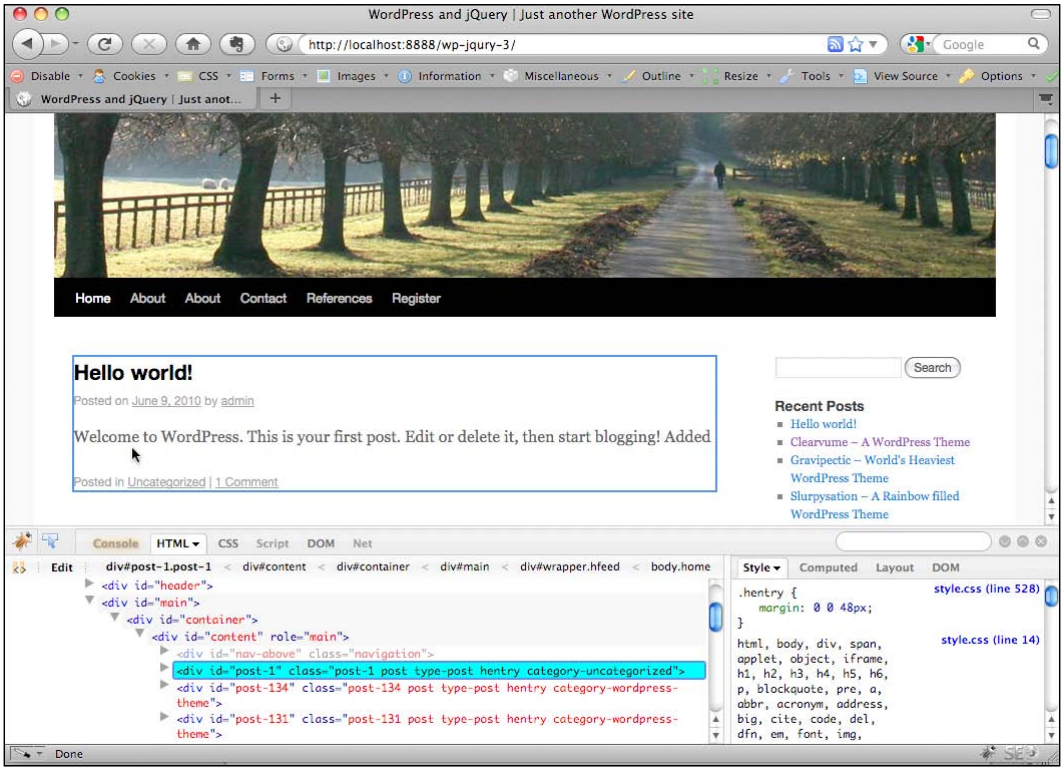


Firebug

A more robust tool is Joe Hewitt's Firebug extension for Firefox found at `http://www.getfirebug.com/`. You'll note, there's also a "Firebug Lite" version available for Internet Explorer, Safari, and Opera. But the full Firebug for Firefox is your best bet.

This extension is a powerhouse when combined with the features of the Web Developer toolbar. On its own, Firebug will find anything you need to manipulate or debug: HTML, CSS, JavaScript, you name it. It can even help you find little "weirdo" tidbit things happening to your DOM on-the-fly. There's a variety of fun inspectors and just about all of them are invaluable.

My favorite Firebug features are the options for reviewing HTML, CSS, and the DOM. Firebug will show you your box models and let you see the measurements of each ledge. Plus, the latest version of Firebug lets you make edits on-the-fly to easily experiment with different fixes before committing them to your actual source files. (There are features that let you edit on-the-fly in the Web Developer Toolbar as well, but I find the Firebug interface more in-depth and easier to use.)



Not essential, but helpful: Image editor

The last tool that I'd like to mention is an image editor. While you can certainly do plenty of cool enhancements with pure CSS, chances are you'll want to expand on your WordPress design and jQuery enhancements a little more by being able to add some slick visual elements such as cool icons or custom backgrounds. These are best achieved by using a graphic editor such as GIMP, Photoshop, or Fireworks.

Adobe owns both Photoshop and Fireworks. It also offers a light and less-expensive version of Photoshop, called Photoshop Elements that will allow you to do basic image edits (`http://www.adobe.com/products/`).

Any graphic editor you prefer is fine. One that allows you to work with layers is best.

Free open source image editors

If you're on a budget and in need of a good image editor, I'd recommend GIMP. It's available for PC, Mac, and Linux. You can get it from `http://gimp.org/`.

On the other hand, if you prefer vector art as I do, then try Inkscape, which is also available for PC, Mac, and Linux. Bitmap graphic editors are great in that they also let you enhance and edit photographs and do some drawing. But if you just want to create neat buttons and icons or other interface elements and vector-based illustrations, Inkscape gives you detailed drawing control and is worth trying out (`http://inkscape.org`). You'll find that many graphic examples created for this book were done primarily with Inkscape.

I personally use both, a bitmap image editor like GIMP or Photoshop in conjunction with a solid vector drawing program like Inkscape. I find it is often necessary to use both types of image editors together to create most of my site designs and effects.

jQuery background and essentials

jQuery, created by **John Resig** is a free, open source JavaScript library. It simplifies the task of creating highly responsive web pages and works well across all modern browsers. John took specific care when developing jQuery so that it abstracts away all the differences between browsers. So you can focus on your project's function and design without getting caught up in elaborate JavaScript coding to handle all the different browsers out there, and the different ways in which individual browsers like to handle the DOM and their own browser event models.

What jQuery does (really well)

jQuery at its core, excels at manipulating the DOM by finding and selecting (hence the word "query" in the name) DOM elements into a **jQuery object**, often called a **wrapper**. This allows you to easily get and set page elements and content, and work with all the modern browser event models allowing you to add sophisticated features to your site. Last but not least, jQuery has a really cool set of effects and a UI library. Animation and interface widgets are now at your complete command.

Wait! DOM?!

Don't panic. I know, we're barely into the first chapter and I've mentioned this mysterious acronym **DOM** several times. I'll be mentioning it a lot more. Learning about the **Document Object Model** can really enhance your understanding of your HTML for WordPress theme design and jQuery enhancements.

It will also help you better understand how to effectively structure your CSS rules and write cleaner and accurate jQuery scripts. For more information, you can of course refer to the W3Schools website: (`http://w3schools.com/htmldom/`.

Beyond all that cool DOM manipulation stuff, jQuery has a nice easy learning curve. You CSS gurus will especially enjoy picking up jQuery. Again, in finding the best way to select elements easily, John developed jQuery so that it leveraged web developers' existing knowledge of CSS. You'll find jQuery selectors a snap, especially as you can grab and select sets of elements almost as easily as you can style them with CSS!

How we got here: From JavaScript to jQuery

JavaScript, originally named LiveScript, was invented by Netscape's developers in the early 90s. By 1996, Netscape had renamed LiveScript to JavaScript in order to boost its popularity by linking it to Java (developed separately by Sun Microsystems). Java, which had been around a few years itself already, was becoming even more popular because people were starting to run it in websites by using a separate plugin called an "applet". There are some ways in which Netscape's developers took care to make JavaScript syntax and functions very similar to Java, but there are differences of course. The biggest difference is that JavaScript is a client-side scripting language that is interpreted, which means it runs live in the browser and is not pre-compiled the way Java is in order to execute and run.

It's a bit complicated and beyond the scope of this book to explain it all, but of course, Microsoft's browser, Internet Explorer, in competition with Netscape, took a completely different route and released IE with the ability to run Microsoft's own VBScript. VBScript was made to look and work similar to VisualBasic, but again as an interpreted language, instead of a compiled one like VB. When JavaScript seemed to be gaining more popularity with budding web developers than VBScript, Microsoft introduced JScript. JScript was crafted to be very similar to JavaScript, in order to appeal to JavaScript developers without any licensing hassles for Microsoft, but there were still quite a few differences. You could however, if you were very careful and didn't have high expectations, write a script that executed as JavaScript in Netscape and JScript in IE 3.0.

Yes. What a pain! To this day, IE still only executes VBScript and JScript! The difference is, both Microsoft and Mozilla (Netscape's creation foundation) submitted JavaScript and JScript to **ECMA International**, an organization which focuses on creating and maintaining standards for information communication systems. In addition to JavaScript, you can thank ECMA Int. for standards running the gamut from CD-ROM and DVD formatting specs to the newer Open XML standards used in Office suites like MSOffice and OpenOffice.

It has taken well over ten years from JavaScript's initial submission in 1997. But as of 2010, both JavaScript and JScript standards are very similar, and both are now technically named ECMAScript (but who wants to try and say that all the time?).

Many developers who came of age in the later 90s and early 2000 use the terms JScript and JavaScript interchangeably without realizing there's a difference! And yet, there are still differences. IE handles ECMAScript in some ways differently compared to Firefox and other browsers. For clarity and sanity, this title will continue to call ECMAScript JavaScript.

Once upon a time, there was JavaScript

Back in the "dark ages", that is before jQuery came along in early 2006, in order to create a more dynamic page that responded to events or manipulated the DOM using JavaScript, you had to spend a lot of time writing long and often clumsy JavaScript using `while` and `foreach` loops, with perhaps a few or many `if/else` statements squashed inside those loops.

If you wanted to evoke your JavaScript immediately, it had to be placed in the header tags or in the body with an `onload` event handler. The problem is that this method waits for the *entire* page and all its content to load, including things such as CSS files and images. If you created a loop to select and manipulate a set of elements, and wanted to perform an additional change to that set of elements, you had to select them again in an additional loop or have a long loop with `if/else` statements that could become complicated to track and maintain.

Lastly, many events you might want the page to respond to, often had to be called separately. I recall sometimes having to create an event script for Firefox (or way, way back in time, on Netscape) and a separate event script for IE. Occasionally, I'd even devise little creative ways to detect different browsers or "trick" them into responding to different events that on the whole were just to make the page appear to look and respond somewhat similarly between the two browsers.

As much as I was enjoying programming and adding engaging interactivity to my sites, I was often a little less than enthused to embark on an in-depth JavaScript endeavor.

Why jQuery is simpler than JavaScript

All that ended with jQuery. jQuery does not stand alone, meaning it's not a new language that browsers support. It essentially boils down to just creating better JavaScript that works. As mentioned, it's a JavaScript library that gives you simpler, easier-to-construct syntax to work with. That jQuery syntax gets interpreted by the browser's JavaScript engine as plain JavaScript. jQuery simply hides a lot of the "ugly" and complicated things that you used to have to do yourself with JavaScript and does them for you.

One of the first things that I grew to love about jQuery (other than its excellent, clear documentation) is that it is essentially a fantastic "loop engine". Now, I call it "looping", but those of you with a more formal programming background or some previous experience with jQuery have probably heard the term used as: **implicit iteration**. Essentially, jQuery iterates, that is, repeats (aka: loops) through the selected elements of its container object without the introduction of an *explicit* iterator object, hence, using the term *implicit*. OK, complicated definitions aside, it simply means you can do just about anything you need to a set of elements, without ever having to write a `foreach` or `while` loop! Most people I chat with about jQuery, have no idea this is what jQuery is really doing under the hood.

What's even cooler than being able to easily loop through selected elements is the ability to select them in the first place using standard CSS notation. Then, as if those two features weren't wonderful enough, once you've grabbed a set of elements, if you have more than one operation that you want to apply to the selected set of elements, no problem! Rather than evoking individual functions and scripts on the selection over and over, you can perform *multiple* operations all at once, in a single line of code. This is called **statement chaining**. Statement chaining is awesome and we'll learn all about it and take advantage of it often throughout this title.

Lastly, jQuery is extremely flexible and most importantly, extensible. In the four years it's been around, there have been thousands of third-party plugins written for it. It's also very easy to write your own jQuery plugins as we'll discover in this book. However, you'll probably find that for most of your more practical day-to-day WordPress development and maintenance needs, you won't have to! Just as WordPress saves you loads of time and work, you'll find with jQuery that a lot of the work has already been done as well.

Whatever you wish to create, you can probably find a way to do it fairly easily with a jQuery plugin and a tweak or two to your WordPress theme. Perhaps you might just need to write a quick and simple jQuery script to enhance one of your favorite WordPress plugins. We'll go over the basics of jQuery and the most common uses of applying it to WordPress in this book and you'll quickly see that the possibilities are endless.

Getting to know jQuery

This book is here to help you create solutions for scenarios and problems that tend to confront WordPress users. I'm hoping to help you save a little time having to poke through WordPress' wonderful yet extensive codex and jQuery's API documentation. But by no means will this book replace those resources or the great resources maintained by jQuery and WordPress' community members.

For jQuery, I highly recommend you check out jQuery's documentation and the Learning jQuery site:

`http://docs.jquery.com`

`http://www.learningjquery.com`

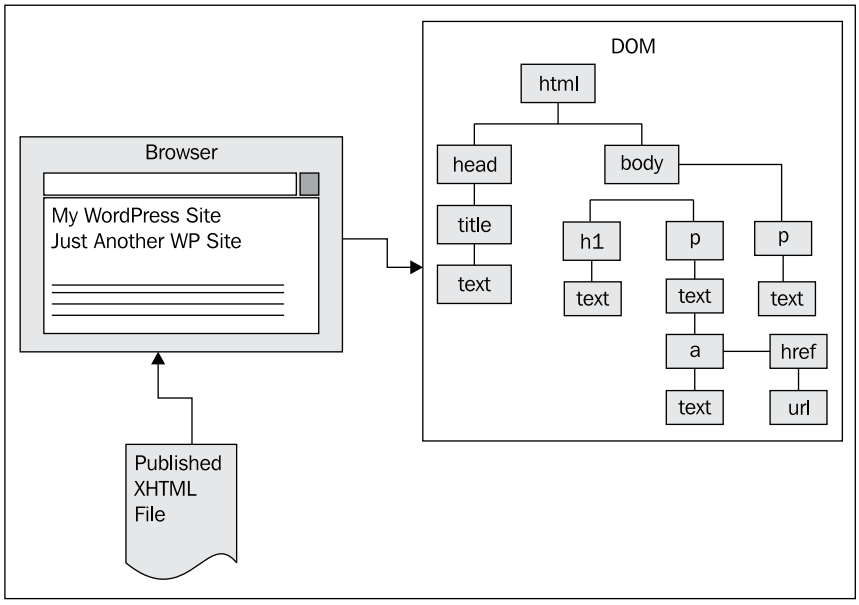
Understanding the jQuery wrapper

As we move through this title, you'll hear and learn a lot more about the jQuery object, also called the "wrapper" or "wrapper set", which probably makes the most sense, as it's a "set" of elements you've selected to work with. But as it's essential to how jQuery works, we'll do a quick introduction now.

To fully understand the wrapper, let's back up a bit outside of jQuery. Ultimately, it all starts with your browser. Your browser has a JavaScript engine and a CSS engine. The browser can load, read, and interpret properly formatted HTML, CSS, and JavaScript (and yes, a host of plugins for Java, Flash, and many different media players that we won't worry about for the purposes of this explanation).

Now this is a very crude, high-level overview. But I think it will help you understand how jQuery works. The browser takes the HTML document that loads into it and creates a map of the document called the DOM (Document Object Model). The DOM is essentially a tree of the HTML document's objects.

You'll recognize most objects as the the markup tags in an HTML document, like `<body>`, `<h1>`, `<div>`, `<p>`, `<a>`, and so on. The DOM tree is laid out, displaying the parent-child relationships of those objects to each other as well as mapping relationships to each object's attributes and content. For example, take a look at the following sample DOM tree illustration:

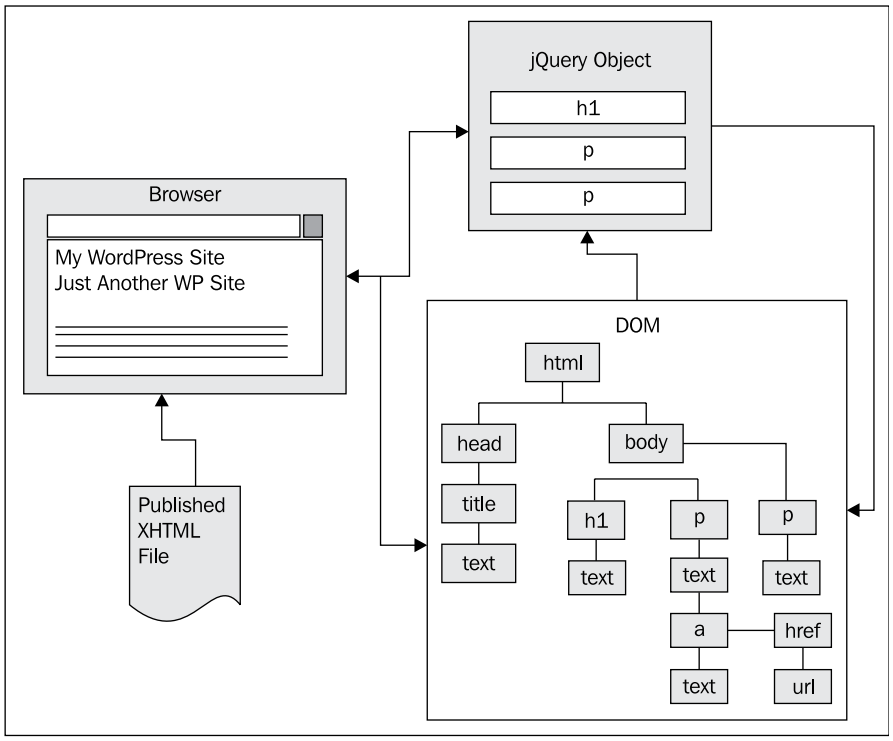


Now for the fun stuff. If a CSS stylesheet is attached or embedded into the document, the browser's CSS engine traverses the DOM tree and styles each of the elements as specified by the style rules. And of course, if there is any JavaScript attached or embedded into the document, the browser's JavaScript engine is also able to traverse the DOM tree and perform the instructions the script contains.

The jQuery library is attached to your XHTML document as a JavaScript file. The library is then able to prepare the JavaScript engine to create an object that will have all of jQuery's functionality inside it, ready to be used upon being evoked (also known as the jQuery object). When you create jQuery code, you automatically evoke that jQuery object and you're ready to start working with it.

Most commonly, you will instruct the jQuery object to traverse the DOM through CSS selectors and place specific elements inside of it. The selected elements are now "wrapped" in the jQuery object and you can now start performing additional jQuery functionality on the selected set of elements. jQuery can then loop through each element that it is wrapped around, performing additional functions. The jQuery object stops looping when it comes to the last object in the set and has performed all the instructions passed to it through statement chaining.

The following illustration shows some of the DOM's objects passed to the jQuery object.



Getting started with jQuery

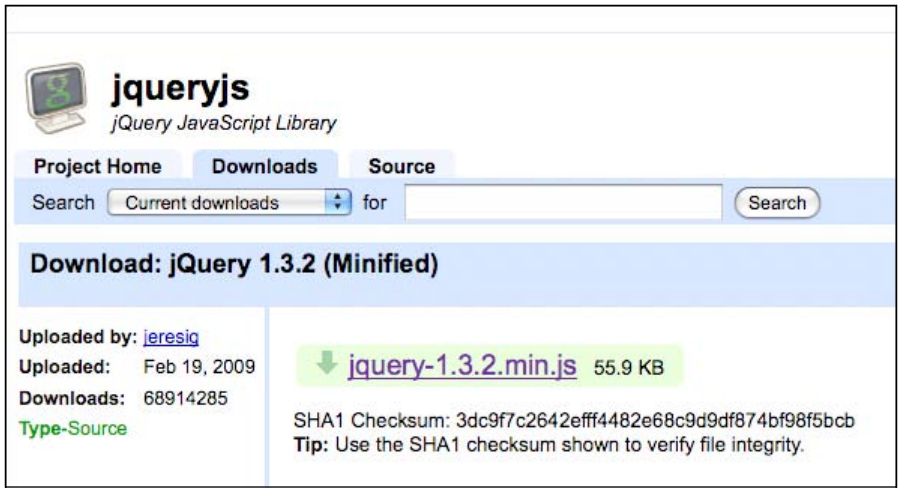
It's very easy to get started with jQuery. We'll cover the most direct basic method here and in the next chapter, we'll explore a few other ways to work with jQuery in WordPress.

Downloading from the jQuery site

If you head over to the jQuery site at `http://jquery.com`, you'll find that the home page offers you two download options: production and development libraries of version 1.4.2, the most current stable version available at the time of this writing.

The production version has been compressed and "minified" into a smaller file size that will load much more quickly. It weighs in at 24KB. The development version, which hasn't been compressed, comes in at 155KB. That's quite a bit larger, but it's much easier to open up and read if you ever run into a debugging problem and should need to.

The ideal scenario is, that you're supposed to use the development version of jQuery while creating your site, and when you release it live, switch over to the production version, which will load much more quickly. Many of you will probably never want to look inside the jQuery library, but it's a good idea to download both anyway. In the event your debugging process keeps showing you a line of code in the jQuery library that is giving you problems, you can switch over to the development version to see more clearly what the line of code is trying to do. I can tell you, the odds that something in the jQuery library has a bug in it is slim! It will almost always be your jQuery script or plugin that has the problem, but being able to look at the full jQuery library may give you an insight as to what's wrong with your script's code and why the library can't work with it. There's no difference between the production and development libraries, just file size and human readability.

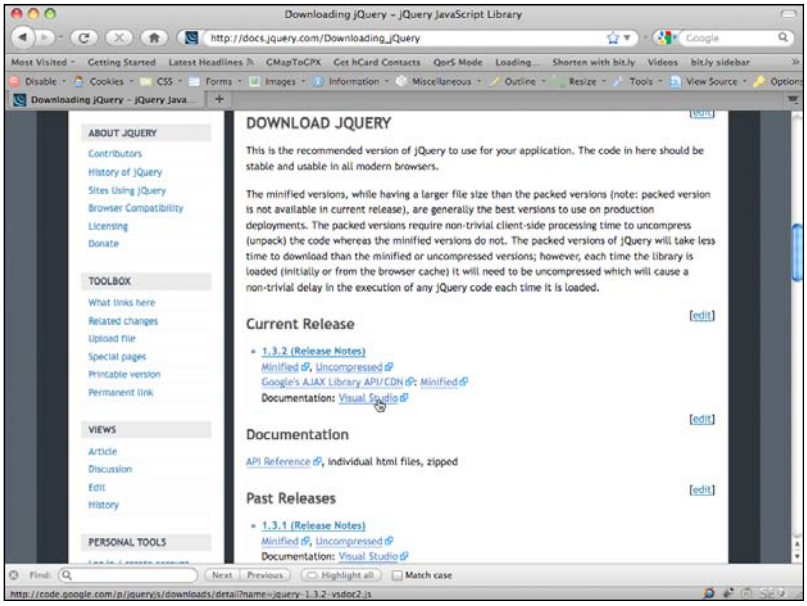


On jQuery's home page, when you click on **Download**, you'll be taken over to the Google code site. You can then go back and select the other version for download. Note that the library is not zipped or packaged in any way. It downloads the actual `.js` JavaScript file ready to be placed into your development environment and used. If you click on the **Download** button and see the jQuery code appear in your browser, just hit the back button and *right-click* or *control-click*, and then click on **Save Target As** to download it.

Using Visual Studio?

If your code/HTML editor happens to be Visual Studio, you can download an additional documentation file that will work in Visual Studio and give you access to comments embedded into the library. This allows the Visual Studio editor to have statement completion, sometimes called IntelliSense, when writing your jQuery scripts.

To download the definitions file, click on the blue **Download** tab at the top of the home page. On the **Download jQuery** page, you'll find the link to the Visual Studio documentation file in the most current release.



You'll place this file in the same location as the jQuery library you downloaded (production or development) and it should now work with your Visual Studio editor.

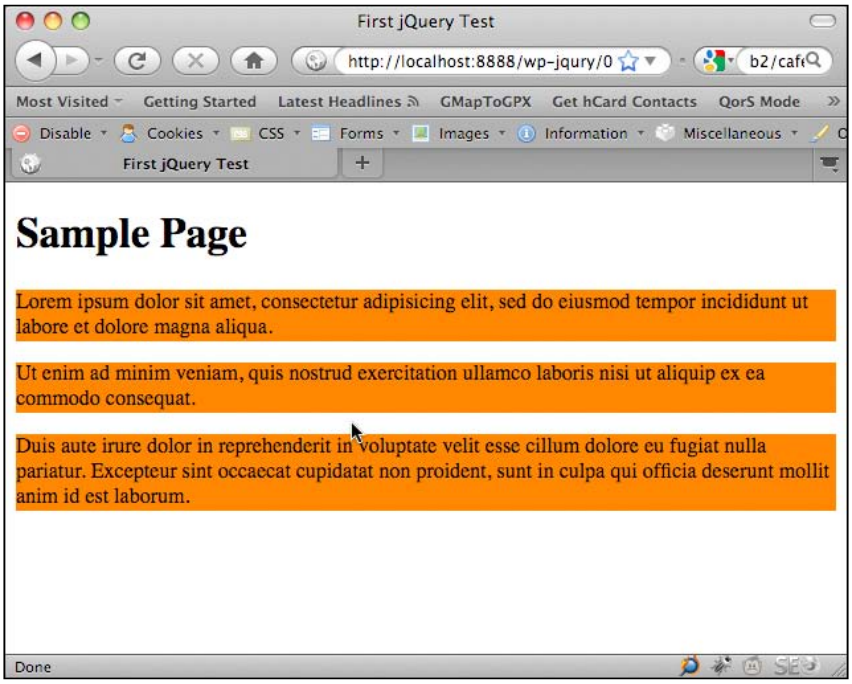
Including the jQuery library

Let's get right down to it and set up a basic HTML document that includes the jQuery library file we just downloaded. I went ahead and downloaded the smaller production version.

In the following markup, we'll attach the library and write our first jQuery script. Don't worry so much about the jQuery code itself at this point. It's just there so you can see it working. We'll go over really understanding jQuery functionality in the next chapter.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
  <head>
    <title>First jQuery Test</title>

      <script type="text/javascript"
        src="jquery-1.3.2.min.js"></script>

      <script type="text/javascript">
           jQuery("document").ready(function(){
             jQuery("p").css("background-color", "#ff6600");
       });
      </script>

  </head>
  <body>
    <h1>Sample Page</h1>

    <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed
do eiusmod tempor incididunt ut labore et dolore magna aliqua. </p>

    <p>Ut enim ad minim veniam, quis nostrud exercitation ullamco
laboris nisi ut aliquip ex ea commodo consequat.</p>

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

  </body>
</html>
```

That's it! Without any CSS included or embedded into the page or the markup, we've used jQuery to change the CSS `background` property of the paragraph tags. Now, ultimately we wouldn't want jQuery to replace our regular use of CSS by any means! But from this quick example, you can see how jQuery can be used to alter the look and layout of your site's pages on-the-fly, and in response to events, making your site's pages very responsive to users; it is a powerful feature. You should now be able to load up this file into Firefox to see your first jQuery script in action.



If you've worked at all with WordPress, based on the previous sample, you can probably easily see how to include the jQuery library in your WordPress theme and start working with it. You'd do just fine including jQuery into your theme in this way. However, in the next chapter, we will discuss the more optimal way to include the jQuery library into your WordPress installation.

WordPress background and essentials

Now that you have a little background with jQuery and understand how to get it up and running in an HTML document, let's take a look at WordPress. Again, most of you are already WordPress users and developers. At the very least, you've probably worked with it in some way. You might even have a WordPress site that you own or maintain.

For those of you with minimal experience with WordPress, we'll quickly go over some background and essentials to getting started with it. Even you more experienced users may want to read on, as I'll cover setting up a "sandbox" or development installation of WordPress. This way, you can experiment, learn, and play with WordPress and jQuery without having to have any of it appear on your actual site until you're ready to deploy it.

Overview of WordPress

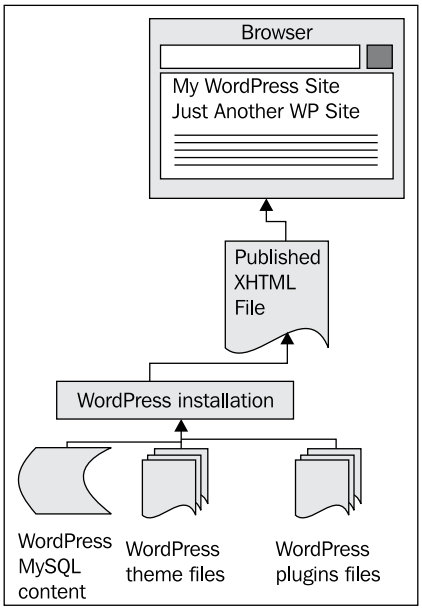
WordPress, developed as a fork off the original b2/cafelog software, was co-developed by *Matt Mullenweg* and *Mike Little*. It first appeared in 2003. Originally a blog platform, it has grown over the years into a robust publishing platform that millions of people and organizations use in a myriad of ways for maintaining their site's content.

Like jQuery, WordPress is flexible and extensible. Matt and his fellow WordPress developers at Automattic have taken care to make sure WordPress conforms to current W3C web standards. A WordPress site's design and additional, custom functionality can be easily controlled and updated using the platform's APIs, which streamline theme and plugin development.

You should keep in mind, as someone who is looking to enhance your site with jQuery, just how dynamic a WordPress site is. WordPress uses a MySQL database and set of theme template pages as well as plugin pages, not-to-mention hundreds of core functionality pages to generate your site. This means a final displayed XHTML page's markup comes from many places; from the theme's template files, from post and page content stored in the MySQL database, and some of it may be defined in the code of a plugin or widget that the installation is using.

The more you know and understand about your WordPress installation and how its files come together, the more easily you'll be able to enhance the site with jQuery.

The next diagram illustrates how WordPress serves up a complete HTML page to the browser:



Completely new to WordPress?

Again, I highly recommend the book **WordPress 2.7 Complete** by *April Hodge Silver* and *Hasin Hayder*. This book is an excellent resource. It covers everything you need to know about WordPress and will also get you started on working with WordPress themes and plugins.

Interested in going deeper with WordPress?

If you're comfortable with using WordPress but would like to understand more about theme and plugin development, then you should definitely check out **WordPress Plugin Development** by *Vladimir Prelovac*, and, if you'll excuse the shameless plug for my own book, **WordPress 2.8 Theme Design**.

Essentials for getting WordPress running

If you have a version of WordPress running that you can play with, great. If you don't, I highly recommend having a locally running installation. Installing and running a small web server on your local machine or laptop has become very easy with the release of WAMP (Windows, Apache, MySQL, and PHP) and MAMP (Mac, Apache, MySQL, and PHP). A local server offers you several conveniences compared to working with WordPress installed on a hosting provider.

I often find that when I travel, despite more and more Internet WiFi bubbles popping up, I am often somewhere that doesn't have one, or I'm in a Starbucks and I don't feel like shelling out cash to T-Mobile for the "privilege" of being connected. With a local installation of WordPress, I have no worries. I can develop and tinker to my heart's content regardless of Internet connectivity and most importantly, without worry that I'll break something on the live site that I'm developing or designing for.

If you're interested in a local sandbox installation of WordPress, I recommend you download WAMP for Windows or MAMP for Mac.

Using WAMP

WAMP stands for Windows, Apache, MySQL, and PHP and it makes it very easy to have a local web server running on your computer in just a few clicks. If you're using a Windows operating system such as XP, Vista, or Windows 7, you can head over to `http://www.wampserver.com` and download WAMP 2.

Be sure to follow the directions in WAMP's installation wizard! If you already have a web server running as localhost and/or a previous version of WAMP installed, carefully read the wizard instructions for disabling or uninstalling that server, backing up your data, and installing the latest version of WAMP.

You can also agree to let WAMP install a start page for you. From this start page as well as from the WAMP icon in the taskbar, you'll be able to easily launch **phpMyAdmin**. phpMyAdmin will allow you to easily create a database and the database user account required for installing WordPress.

Using MAMP

Similar to WAMP, MAMP stands for (you guessed it!) Mac, Apache, MySQL, and PHP. Mac users will head on over to `http://mamp.info` and download the free version of the server.

Once you download and unpack the ZIP and launch the `.dmg` file, it's a pretty straightforward process for copying the MAMP folder to your `Applications` folder and launching the app.

Again, like WAMP, MAMP from the start page offers you an easy way to launch **phpMyAdmin**. phpMyAdmin will allow you to easily create a database and database user account, which is required for installing WordPress.

Using Ubuntu?

If you're using Ubuntu and need a local server, you're in luck. Linux afterall is the OS most web servers use (I think you know what LAMP stands for at this point).

I'd recommend you do a little research through Google to find the best way to install your own local web server. I found the following resource to to be the most useful for me and what I used to install LAMP on my Ubuntu 10.04 installation: `http://www.unixmen.com/linux-tutorials/570-install-lamp-with-1-command-in-ubuntu-910`.

Choosing a hosting provider

If you are using a school's or library's computer and can't (or otherwise just don't want to) locally install software, you'll need an account with a web hosting provider. The hosting provider you choose must be running Apache, MySQL, and PHP, in order to accommodate WordPress. It will greatly benefit you to choose a hosting provider that offers an easy-to-understand account panel, which allows you to easily access phpMyAdmin.

Easy, one click installs—Easy, yes. Just be careful!

Many web hosting providers offer super easy "one-click" installs of many of today's top CMS publishing platforms and other useful web applications including WordPress. Be sure to check out your hosting provider's services and options as this will let you fill out one easy form and will save you the hassle of dealing directly with phpMyAdmin or the WordPress install wizard.

Be careful with one-click installs! While many providers simply install a single installation of WordPress on to your account for you, which is perfect, some providers may have **WordPressMU** running. These providers will create an MU account, which will map to your domain name, but not give you access to any installation files. If that's the case, you will *not* have complete control over your WordPress site!

You'll **must** be able to FTP into your hosting account and see your WordPress installation's files, particularly the `wp-content` directory, which will contain your theme and plugin directories and files that you'll need to be able to edit in order to enhance your site with jQuery. Be sure to double-check with your hosting provider before choosing a one-click install.

WordPressMU is multi-user WordPress. It is what powers **WordPress.com** accounts. While it's super easy to set up a site on `WordPress.com` and have them host it, you cannot upload or customize your own themes and plugins. This is why this title doesn't even attempt to cover `WordPress.com` accounts as you need access to the `wp-content` folder in order to enhance your site with jQuery.

Rolling out WordPress

WordPress itself is very easy to install. Once you have a MySQL database set up with a username and password for that database, you'll unzip the latest WordPress version and place it into your local `httpdoc` or `www` root folder and then run the installation by navigating to `http://localhost-or-domainname-url/my-wp-files/wp-admin/install.php`.

WordPress in 5 minutes (or less!)

For a complete overview of installing WordPress, be sure to check out **WordPress' 5-Minute Installation Guide** from the Codex: `http://codex.wordpress.org/Installing_WordPressAgain`. The book **WordPress 2.7 Complete** will walk you through a WordPress installation, step-by-step.

jQuery and WordPress: Putting it all together

You probably come from one of two camps: you might know and have experience with jQuery and you're looking at WordPress to help maintain your site. Or, more likely, you have experience with WordPress and you're looking to see what jQuery can do for you.

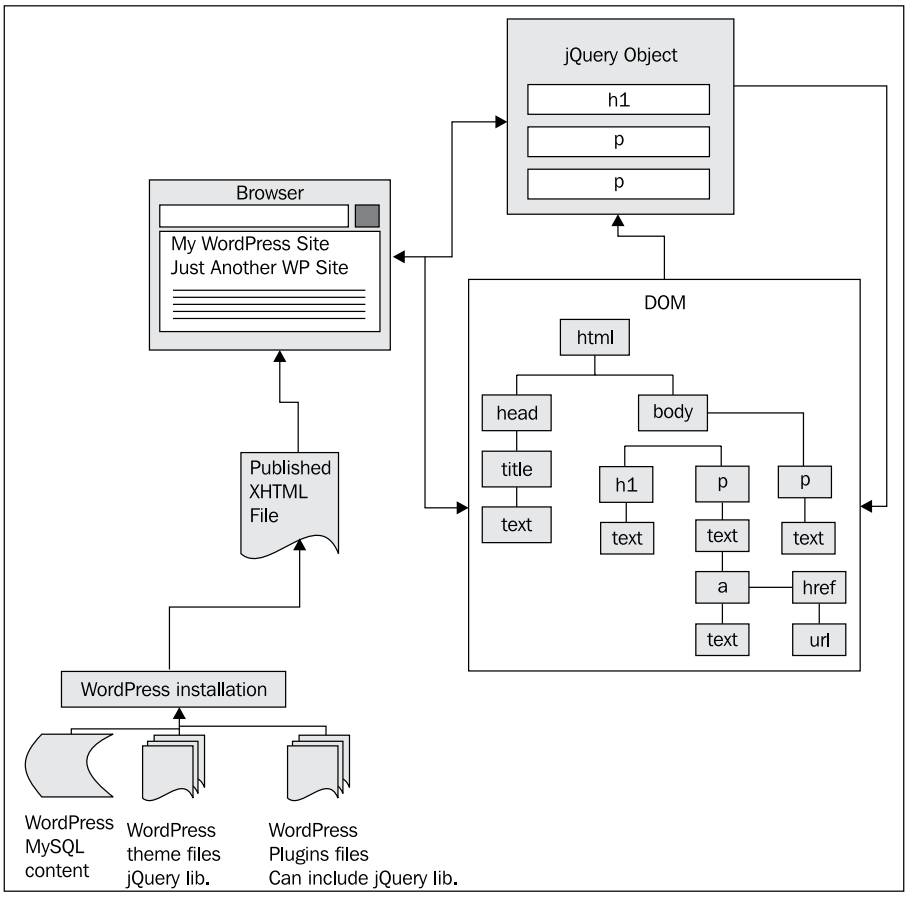
If you have some experience with jQuery but are fairly new to WordPress you're probably familiar with all sorts of jQuery examples that show clean and clear, hand-coded HTML and CSS, which you then craft your jQuery scripts to fit. It can become easy to just open up an HTML file and be able to quickly see and even directly manipulate all the HTML markup and CSS `id` and `class` references right there in order to make your jQuery script as straightforward as possible.

With WordPress, as we've discussed in some detail here, all that HTML is generated dynamically. There's no single file that you can open into your editor to get an overview of what jQuery has to work with. You'll have to get to know the WordPress publishing system and most importantly, the WordPress theme and any plugins that you're using, to be able to get your jQuery scripts to target and affect the elements that you want to affect. As I've already mentioned, this is where you'll discover the Web Developer toolbar and Firebug extensions for Firefox to be your best friends.

On the other hand, you WordPress experts who are becoming familiar with jQuery ultimately have the same problem, but you're coming at it from a slightly different angle. You might be used to just having WordPress generate everything for you and not give much thought to it. In order to get jQuery to affect your WordPress content, you're going to have to become a lot more familiar with what's going on under the hood in WordPress and your theme.

Your advantage to implementing jQuery will be in your familiarity with how your theme is set up in your WordPress system and any WordPress plugins you're using. You're going to want to really focus and get a handle on understanding jQuery selectors to be able to navigate all the possible DOM elements being generated by WordPress and create the enhancements you desire.

The following illustration shows how WordPress serves up a complete HTML page to the browser that then interprets the DOM so that CSS styles can be applied, and jQuery and other JavaScript can enhance it:



Summary

We've taken a look at the essential background knowledge you'll need and tools required for working effectively with jQuery and WordPress.

We also took a look at the following topics:

- Software tools that you need to get your project up and running
- Background and basics of jQuery and WordPress

Now that you're up to snuff on these topics, in the next chapter, we'll enable jQuery in our WordPress installation and take a deeper look at jQuery's immense possibilities. Get ready to have some serious fun with our WordPress site. Let's get started!

2
Working with jQuery in WordPress

Now that we understand the basics of jQuery and WordPress and have a little background on how they'll interact with each other, we're now ready to take a look at using jQuery to dynamically enhance a WordPress installation. We'll start with getting jQuery included in WordPress and end up with our first cool project: Expanding and collapsing content. This is only the beginning of the jQuery possibilities in store for your WordPress site! Again, we'll be using WordPress 3.0 in this title and the new default Twenty Ten theme with jQuery 1.4.2, but rest assured that if your site or project is still using WordPress 2.9, these jQuery techniques will work just fine.

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

- Registering jQuery in WordPress
- Using Google's CDN to include jQuery
- Reviewing all of jQuery's "secret weapons"
- Our first jQuery and WordPress enhancement

Getting jQuery into WordPress

jQuery can be included into WordPress in three different ways as follows:

- You can download it from `jQuery.com`, and include it directly with a `script` tag into your XHTML header tags, inside your theme's `header.php` file (this method works, but is not really recommended for a variety of reasons)
- You can register WordPress' bundled jQuery in themes and plugins
- You can also take advantage of Google's CDN (Code Distribution Network) to register and include jQuery into your theme and plugins

We covered the basics of the first method in *Chapter 1*, *Getting Started: WordPress and jQuery*. WordPress is so flexible that any user with the right admin level can come along and update, enhance the theme, or install additional plugins which may also use a version of jQuery or other JavaScript libraries. Therefore, including jQuery or any JavaScripts directly into the theme with hardcoded script tags is not recommended as it could cause conflicts with other scripts and libraries included into the WordPress site through theme customizations or plugins added to the WordPress installation. In this chapter, let's take a look at using the two remaining methods, registering jQuery through WordPress' **Script** API and using Google's CDN.

jQuery now comes bundled with WordPress

As of WordPress 2.7, jQuery and several other JavaScript libraries and plugins have been bundled and are available through WordPress' Script API through a handy function called `wp_enqueue_script`. Actually, WordPress has had jQuery and quite a few other JavaScript libraries (including `Script.aculo.us` with Prototype and many more) bundled into the `wp-includes` directory for some time, but until version 2.7, these includes were not so easily accessible.

Registering jQuery in a WP theme

You can activate WordPress' bundled jQuery in two different ways:

First, you can place the following code in your `header.php` file before the closing `</head>` tag:

```
<?php wp_enqueue_script("jquery"); ?>
<?php wp_head(); ?>
<script type="text/javascript">
  //add jQuery code here
  jQuery(document).ready(function() {
  jQuery("p").click(function() {
     alert("Hello world!");
  });
  });
</script>
```

Alternatively, you can register the `wp_enqueue_script` (and any custom jQuery code you write) in your theme's `functions.php` file. If your theme doesn't have a `functions.php` file, simply create a new file, name it `functions.php`, and place it in your theme's root directory with your other template files (`functions.php` is a standard template file that's included with the default theme we're using). Place the following code into your `functions.php` file:

```
<?php wp_enqueue_script('jquery');/*this registers jquery*/
    function jq_test(){ /*This is your custom jQuery script*/
?>
<script type="text/javascript">
 jQuery(document).ready(function() {
    jQuery("p").click(function() {
      alert("Hello world!");
    });
  });
</script>

<?php
}
add_filter('wp_head', 'jq_test');/*this adds your script to the wp_
head() hook in the header.php file and ensures your custom jQuery
script is run*/
?>
```

Avoiding problems registering jQuery

The first time that I ever attempted to load up jQuery using the `wp_enqueue_script` (both in the `functions.php` file and through the `header.php` file), I just could not get it to work. After some hair pulling and a few hours on the WordPress Codex, I finally realized the following facts:

- If you're loading directly into your `header.php` template file, make sure that the `wp_enqueue_script` function is above your `wp_head` function. Your custom jQuery code must go below the `wp_head` function.
- If you're registering the `wp_enqueue_script` in the `functions.php` file, make sure that it comes before any custom functions that load through the `add_filter` function into the `wp_head`.

Read up on the `wp_enqueue_script` function!

This function is part of WordPress' Script API and it actually does a lot more than just load up jQuery! As I mentioned, there are many, in fact well over fifty, JavaScript toolkits, frameworks, user interface libraries, plugins, and helpers that you can load up safely using the `wp_enqueue_script` function. Check it out here: `http://codex.wordpress.org/Function_Reference/wp_enqueue_script`.

Using Google's CDN

Personally, I am a little torn about registering and referencing the copy that comes with WordPress. I've discovered that loading the library from **Google Code's Code Distribution Network** (**CDN**) is sometimes a better way to go. The CDN saves on bandwidth, allowing your site to do some parallel processing while downloading other scripts and collateral. Plus, it's easy to always get the most current version of jQuery. jQuery's library loads very quickly from Google's CDN and, as a bonus, the library will already be cached if your site's user has previously visited another site that delivers jQuery from Google Code's CDN.

Registering and including jQuery through Google's CDN into a theme

To include jQuery from Google Code's CDN, we'll be sure to deregister jQuery then register through Google's CDN. This is the beauty of registering and using the `wp_enqueue_script` function: if any other plugin or script requires jQuery, and doesn't have any conflicts with the version loading up from Google, that script will use the already loaded Google CDN library. If a script depends on a specific version of jQuery, say 1.3.2 or 1.2.6, and the CDN is loading up version 1.4.2, then that script will go ahead and load the version of jQuery it requires. Because (as we'll learn) every script loaded through the Script API stays in `noConflict` mode, it's OK to have the two library versions loaded as long as they're registered and required.

```
...
wp_deregister_script( 'jquery' );
wp_register_script(   'jquery', 'http://ajax.googleapis.com/ajax/libs/
jquery/1.4/jquery.min.js');
...
```

Google offers a great versioning system that allows you to be as precise as you want, or just pull the latest stable version. Consider the previous code example (note the highlighted number, 1.4, in the previous code example).

Understanding Google's versioning system

That previous registration script references version 1.4.2 of jQuery (the most recent version as of writing this title). When jQuery's developers release a new version, say, 1.4.3, that version will automatically be called by that same URL because I did not pinpoint the version's specifics. In the same vein, I could choose to call `...jquery/1.3/jquery...` that would give me `1.3.2` the highest version in the `1.3` release. And you guessed it, targeting a simple `...jquery/1/...` would pull the most recent version of jQuery, up to version 1.9.x, until jQuery turns over to version 2.0!

Generally, it's good practice to always have the most recent library load, but you never know, you may use a jQuery plugin or write some of your own code that doesn't work well with a newer version. You'd then want to target the last specific version of the library that works with your plugins or custom scripts, until you can fix and update them.

Using WordPress' bundled jQuery versus including your own jQuery download or using Google's CDN

As I mentioned earlier, the `wp_enqueue_script` function allows for a safe load of jQuery (and other includes) into `noConflict` mode. As long as you deregister and register for jQuery from the Google CDN, the library will load into WordPress with the same `noConflict` mode protection. I really like to take advantage of Google's CDN, for the variety of performance reasons I mentioned, but for large projects with lots of editors and administrators making different decisions on how to manage the WordPress site and what WordPress plugins to use, I play it safe and register the bundled version into the theme. Also, for development, I find it nice to have jQuery already running locally on my MAMP or LAMP server, if I'm developing a theme and yet have disconnected from the Web due to traveling (or the need for enhanced productivity). Once a site is live, I'll consider switching it over to the Google CDN version of jQuery.

Keeping conflicts out!

Because WordPress and jQuery are anticipating other libraries to be loaded which may use the short variable, `$`. The `wp_enqueue_script` ensures jQuery is loaded up in `noConflict` mode. Therefore, you'll also need to make sure to write your custom jQuery code in `noConflict` mode's **syntax**. The easiest way to do this is to replace the `$` variable (common in many jQuery scripts) with the full `jQuery` variable, as I've discussed in *Chapter 1*, *Getting Started: WordPress and jQuery*, and done in my two previous samples.

Setting your own jQuery variable

If you find the `jQuery` variable tedious to write out, yet want to remain in `noConflict` mode, you can replace the standard `$` variable to any variable you want as follows:

```
<script type="text/javascript">
var $jq = jQuery.noConflict();
  $jq(document).ready(function() {
    $jq("p").click(function() {
      alert("Hello world!");
    });
  });
</script>
```

But I really want to use the $ variable!

You **should not** use the `$` variable for jQuery within WordPress. OK, I know, you've got a good reason. Say for instance, you're copying a jQuery script over from another non-WordPress project and it's proving cumbersome to covert all the `$` variables to `jQuery` or some other custom shortcut variable. Fine. (Never heard of "Find and Replace"?) At any rate, here is an example of how to shortcut jQuery to safely use the `$` variable:

```
jQuery(function ($) {
        /* jQuery only code using $ can safely go here */
});
```

The only drawback to the above solution is, I've found it's easy to start working with the `$` variable and then forget to encapsulate other scripts in the above jQuery function. If all my jQuery scripts use the `jQuery` variable or a custom variable (such as `$jq`), I'm much better at staying in `noConflict` mode.

Including jQuery in a WordPress plugin

You can include jQuery in a WordPress plugin using any of the earlier mentioned methods. However, you'll need some familiarity working with WordPress plugins. We'll cover this topic in detail by learning more about WordPress plugins later in *Chapter 3, Digging Deeper: Understanding jQuery and WordPress Together*.

Launching a jQuery script

Most of the time you'll want your script to launch and/or be available as soon as the DOM is loaded and ready. For this, you can use the standard "on document ready" technique as follows:

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

   // Your jQuery script go here
});
```

You can reduce the previous code, just a bit, by using the following code:

```
jQuery(function(){
   // Your jQuery script go here
});
```

If the `jQuery` variable is evoked and a function immediately passed, jQuery assumes the `.ready` event is implied and will run the next selection and function as soon as the DOM is loaded.

Our first WordPress and jQuery setup

I hear you. Enough talking already. Let's get jQuery rolling. The majority of this book's code and samples use WordPress 3.0 RC and the brand new default theme is "Twenty Ten". It's a great, clean, HTML5 valid theme. Even if you want to enhance an older version of WordPress, say 2.8 or 2.9, you'll be glad to know that every one of this title's scripts (or approximate versions of it) was originally written and tested in version 2.8.6 and 2.9.2 before being ported over to 3.0.

Where applicable, I'll show you alternative jQuery solutions for WordPress' 2.9.2 default theme as well as point out differences between jQuery's 1.3.2 library, which comes bundled with version 2.9.2, and jQuery's 1.4.2 library, which is bundled with WordPress version 3.0.

The point of every example is to show you not just how to enhance WordPress' default theme, but any theme, and I hope you get creative with the examples and find ways to apply them in unique ways to all sorts of WordPress versions, themes, and plugins!

Registering jQuery in our setup

Because the bundled version of jQuery that comes with WordPress 3.0 also happens to be the most current version of jQuery available, 1.4.2, I'll simply navigate to `wp-content/themes/twentyten` and open up the `header.php` file and use the basic `wp_enqueue_script` function to evoke jQuery as shown:

```
//placed right above the wp_head function
wp_enqueue_script( 'jquery' );
wp_head();
```

Registering your own custom script file

Next, we'll need to include a separate script file into our theme that will have our custom jQuery scripts. I would like to create a directory in the theme named `js` where I will keep all of my JavaScripts. Inside that directory, I'll create a file and name it as `custom-jquery.js`.

Here's the neat bit: you can use `wp_enqueue_script` to include any script that you write. You'll do this so that you can announce that the script is dependent on jQuery and WordPress will therefore, load jQuery as well, if for some reason, jQuery isn't loaded already! You'll want to place your custom scripts below the jQuery call, yet before the `wp_head()` call.

```
...
wp_enqueue_script( 'jquery' );
wp_enqueue_script('custom-jquery', get_bloginfo('stylesheet_
directory') . '/js/custom-jquery.js', array('jquery'), '20100510' );
wp_head();
```

In the above function, `wp_enqueue_script`, I first registered a name for my script as `custom-jquery`. Then in the next parameter, I told WordPress where to find my script, using the `get_bloginfo` template tag to direct WordPress to the `twentyten` theme's folder "`. . ./js/custom-jquery.js`". For the third parameter of the function, I set the script as dependent on `jquery`, and in the final parameter I simply set a version number. I usually set this number as the day's date. If I update the script, I try and update this date in the function, and as a result, when the theme "renders" my script loads in looking like this:

```
<script type='text/javascript' src='http://localhost/wp-content/
themes/twentyten/js/custom-jquery.js?ver=20100510'></script>
```

This helps a browser load the script "fresh" instead of loading it from the cache if I ever update it.

The previous custom script include method works for the jQuery library itself too!

Say in the near future jQuery updates to version 1.4.3 (or 1.5 and so on) but it's going to be a while before WordPress updates and includes that version. You could of course use the Google CDN to register the latest script version but if, for some reason, you didn't want to use the Google CDN, you could simply download the latest version of jQuery from the jQuery.com site and place it inside your theme's root folder and register it using the custom registration method we just used to include our `custom-jquery.js` file.

Don't forget to `deregister` the bundled jQuery first!

Also: Calling a script in through `wp_enqueue_script` "registers" it at the same time so there's no need to call the `register` function separately if using `wp_enqueue_script`.

Setting up the custom-jquery file

Finally, let's open up the `custom-jquery.js` file, and using the technique we learned earlier, set up the shortcut for jQuery's document ready function as the following:

```
jQuery(function(){ /*<- shortcut for document ready*/
/*any code we write will go here*/
});//end docReady
```

That's it! Let's get started discovering jQuery's "secret weapons" and putting them to use. You can now place any code described in the following sections in your `custom-jquery.js` file and experiment with it!

jQuery secret weapon #1: Using selectors and filters

It is time to start having some fun with jQuery! I feel jQuery can be broken down into three core strengths, what I deem as its "secret weapons":

- Understanding selectors and filters
- Manipulating CSS and content
- Working with events and effects

If you get a handle on these top three strengths, you're well on your way to being a jQuery rockstar!

This first item, understanding selectors and filters, is **essential**. You need to have a strong understanding of selectors and filters if you're going to be able do anything else with jQuery. The better you are at using selectors and filters, the better you'll be with jQuery period.

Selectors and filters give you the ability to (you guessed it!) select objects on your page into the jQuery wrapper object and then work with and manipulate them in just about any way you'd see fit. The selectors will allow you to easily grab an array of elements using easy CSS syntax. Filters will then further narrow down and refine the results of that array.

Keep in mind, the objects selected into the jQuery wrapper using selectors and filters are not really DOM elements anymore. They are an array of objects in the jQuery object wrapper that have a whole set of functions and capabilities available. If you ever need to, you can weed down through all the jQuery added items and functionality in each array element to the actual DOM element, but why? The whole point of jQuery is to get you around that but it's good to know it's there.

Selecting anything you want from the document

In the following examples, we'll be looking at selectors and filters; but to illustrate jQuery's selection, I'll be using a function called `css()`. I'll cover that function and a lot more in later sections. Right now, just focus on the selector and filter at the beginning of the samples.

The essence of jQuery selectors is that they are CSS syntax based. This means that most of you are going to find you can work with jQuery very easily, as far as how you use CSS to target and style specific elements on your page.

Selections are declared in the beginning of the main jQuery function as:

```
jQuery(function(){
  jQuery("selector:filter").jqFunctionName();
});
```

You can also select the following elements into the jQuery wrapper based on CSS syntax:

- HTML **tag names** such as `body`, `p`, `h1`, `h2`, `div`, and so on
- The **id attribute** that is used to select instances and is denoted by a `#` (hash) in CSS, as in `#header` or `#sidebar`
- And the **class attribute**, which is denoted by a `.`(dot) in CSS as in `.body` or `.post`

Of course, any of the combinations that you're allowed to use in CSS to target an element, you can perform with jQuery. For example:

- `Tag` (space, or no space) `#id` or `.className`, such as `div#sidebar li`—this will grab *all* `li` instances in a `div` with the ID name of `sidebar`
- `Tag,` (comma) `.class` such as `p, .post`—the comma ensures this will grab everything that is *either* a paragraph *or* marked with the `.post` class

To clarify, just like in CSS, you can also use **syntax** to *structure* the selector:

- A **comma** means select this element, (and) this element. For example: `div, p` (selects all `div` tags *and* all `p` tags).
- A **space** means select this element (which has) this element within it. For example: `div p .className` (selects all `div` tags that have paragraph `p` tags inside them *with* any other elements assigned to `.className` class *inside* the `p` tag).
- Last, **no space** would indicate a class applied directly to an element not just held within it: `p.className` (selects all paragraph `p` tags with the `.className` assigned to it. This *would not* select a `div` tag that had the same `.className` class assigned to it).

In addition to standard CSS comma space and attached id and class names, within jQuery you can also use these additional symbols to clarify your selections:

- The greater than sign `>` will only find child elements of a parent that meets the selection.

  For example, `.post > p` will find paragraph `p` tags that are **directly** inside the `.post` class. `p` tags inside a different class, within the `.post` class *will not* be selected.

Let's compare "`.post (space) p`" to "`.post > p`" and take a look at the results.

In our first example, we will examine the code as follows:

```
jQuery(function(){
  jQuery(".post p").css("background", "#f60");
});
```

Note that this code produces an output similar to the next screenshot, which shows how all paragraphs are highlighted, even though they are nested another level deep with a class named `.entry-content`:

However, let's look at this code example:

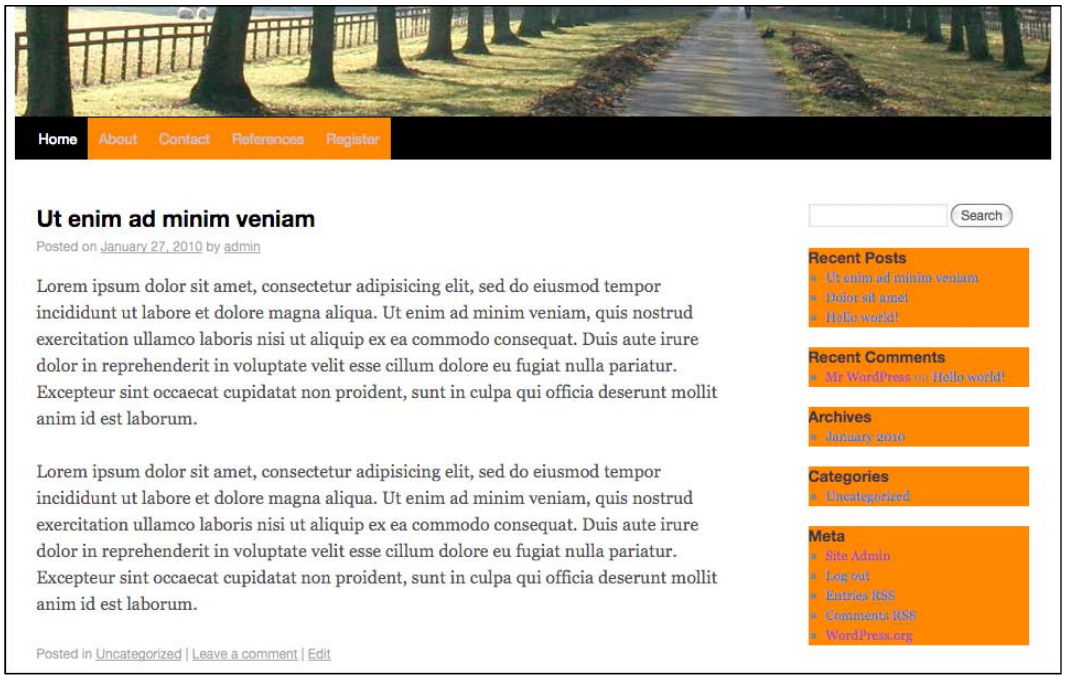
```
jQuery(function(){
  jQuery(".post > p").css("background", "#f60");
});
```

And let's also look at the following screenshot. We find that no paragraphs are highlighted, because they are inside another `div` tag with a class named `.entry-content` and thus, *not* a child of the `.post`.

The `+` selector will find all *next* elements to the matching selector. For example: `li + li` will select every list `li` item within a list, *except* for the first item. Just the items *next* to that first item as shown:

```
...
  jQuery("li + li").css("background", "#f60");
...
```

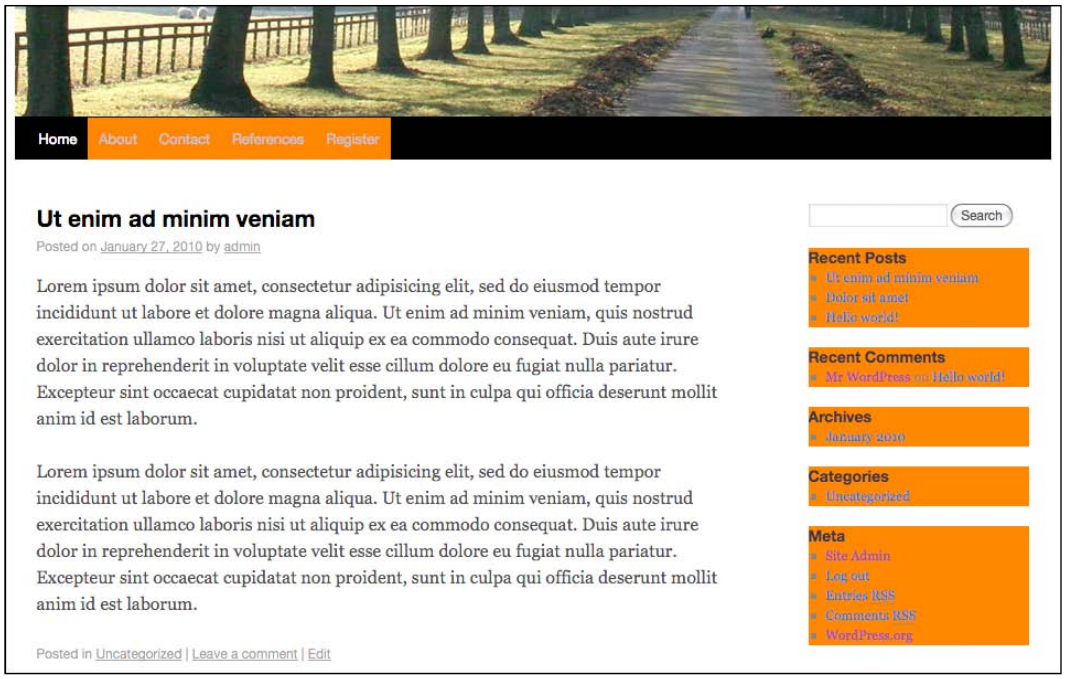
The following screenshot illustrates this:



The ~ selector will find all the siblings of the selector. For example: `li ~ li` will select every list item within a list again, except for the first item, just the sibling items of that first item. The code example is as follows:

```
...
  jQuery("li ~ li").css("background", "#f60");
...
```

As siblings are often next to a selected item, the + and ~ selectors can often receive similar results. Note how the following screenshot looks similar to the previous one:



Filtering those selections

Many of you can probably do most of what you need with just the basic CSS style selectors. But wait, there's more! Filters are the part of selections that I find incredibly useful, especially considering that we're working with WordPress. Again, with a WordPress theme, a lot of your HTML elements, IDs, and class names are probably being generated by a theme that you're not the author of or, for various reasons, you don't want to edit or perhaps you're just not allowed to edit the theme. (What's that? Designers get a little "testy" when developers start mucking about with their markup? I had no idea.) But that's OK. With filters, you simply don't have to.

The thing is, starting out with jQuery, it's tempting to want to go in and change the HTML markup to something that is easier to select with jQuery. But with WordPress, this is not easy. Changing the markup means you run the risk of breaking the theme or worse, having to remind content editors to manually add specific markup to posts and pages (which in some ways, defeats the purpose of using WordPress in the first place). Understanding filters will allow you to have precise control over your selections in every case and scenario, every time.

It's very easy to refine a filter, you're just going to include these items that will take your selected elements and match them to specific conditions, like their position or index relative to other elements. Again, in keeping with spirit of CSS selection syntax, some of these filters look similar to **CSS pseudo classes**, such as `:hover` and `:first-child`. These are not all actually CSS pseudo classes; they won't work in a CSS stylesheet, but they'll work in jQuery.

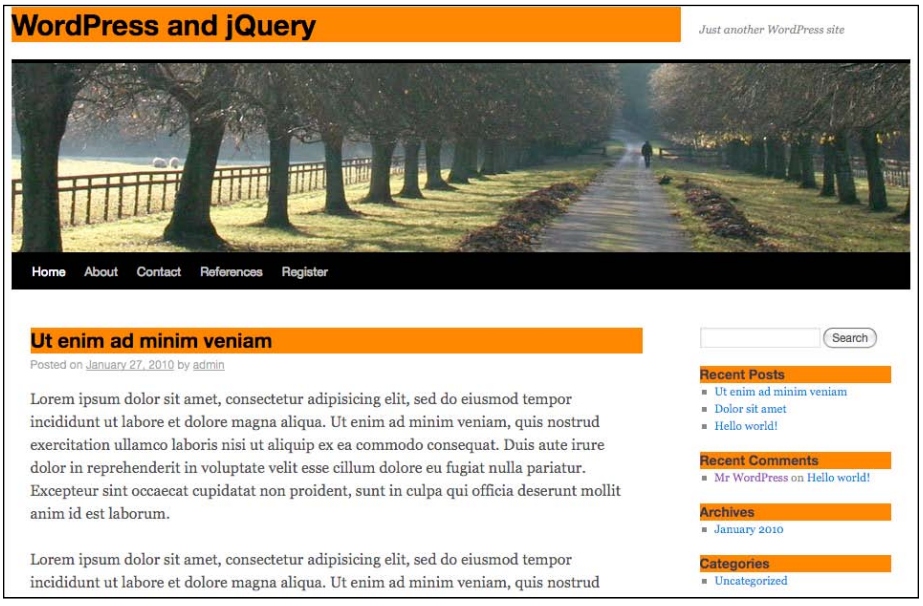
These filters are broken down in the jQuery API in the following categories (listed as I find them most useful to WordPress development): Basic filters, Content filters, Child filters, Form filters, Attribute filters, and Visibility filters.

Basic filters

As you work with WordPress, I believe you'll find the `:not()` filter and the `:header` filters incredibly useful. The `:header` filter allows you to simply select *all* the headers in a selection, no matter what level header they are. Rather than having to select `h1` and `h2` and so on, adding the `:header` filter to your selection will grab up all the headers, `h1` through `h6` into the wrapper. Try it out, in your `custom-jquery.js` file, and add the following code (don't worry about the `.css(...);` part of the code; we'll get to that later. I'm just using it to help us to visualize what jQuery can do):

```
jQuery(function(){
  jQuery(":header").css("background", "#f60");
});
```

You'll see in the next screenshot that all headers are selected, `h1`, `h2`, and so on:

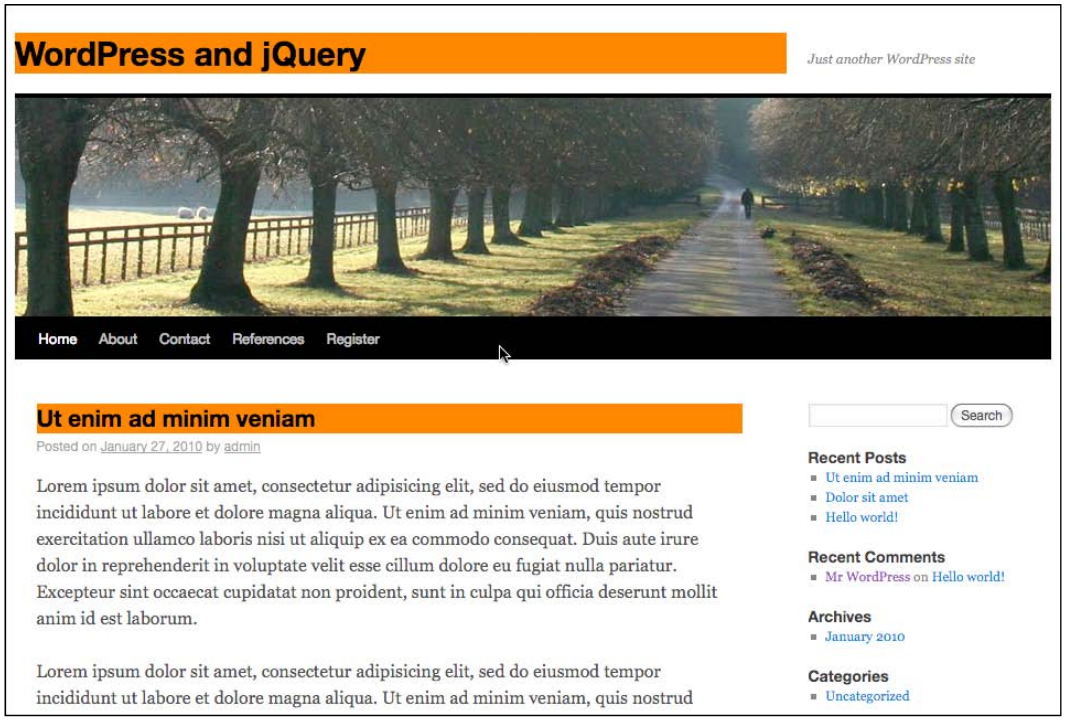


My favorite filter is the `:not` filter. Ever noticed on an airplane, you're often reminded that the "nearest exit may be located behind you"? The same principle holds true when you're trying to scoop up the right elements into your jQuery wrapper. Sometimes it's easier to tell jQuery what you *don't* want in the wrapper! I once worked with a theme that had some very pretty e-mail and PDF icon elements tucked inside the `.post` class. The theme did not have an `.entry` class. This was irritating as I wanted to apply a general transformation to images that were loaded into the WordPress posts, but these icons were affected! The theme author had them wrapped in a class named `.postIcons`. Using the `:not()` filter, I was able to transform all `img` tags that were in the `.post` class but *not* in the `.postIcons` class. Sweet.

Take a look at what happens when you add the `:not` filter with our previous `:header` selection:

```
...
jQuery(":header:not(li :header)").css("background", "#f60");
...
```

The following filters now show us all headers selected, except for headers in list items:



You've probably noticed just from the previous example that you can get pretty clever with filters, using them multiple times within a selection.

What's that, you say? Yes, you're correct: `(":headers:not(li h2)")` achieves the *exact* same results as the previous example, and yes, it's always better to take the most direct route to your selections. I'm just trying to illustrate how these two filters can be used. Eventually, you will run into more complex situations where they'll come in very handy. For everything else, use plain selectors first, before resorting to filters.

Let's take a look at each Basic filter, what it's syntax looks like, and what it does in detail. Because most WordPress theme authors use the `.post` class, and most of the time you'll be targeting post elements to make the syntax have the most sense. I'll use `.post` class name often in my examples, but remember, your main selector can be any `tag`, `id` name, or `class` name used in CSS selector syntax!

Example	Syntax	Description
:not(selector)	`jQuery(".post img:not(.pIcon)").jqFn();`	Filters out all elements matching the given selector.
:header	`jQuery(".post:header").jqFn();`	Filters down to all elements that are headers, such as h1, h2, h3, and so on.
:first	`jQuery(".post:first").jqFn();`	Filters down to the first selected element only.
:last	`jQuery(".post:last").jqFn();`	Filters down to the last selected element only.
:even	`jQuery(".post:even").jqFn();`	Filters down to even elements only. Note: Arrays are zero-indexed! Zero is considered an even number so your first item will be selected!
:odd	`jQuery(".post:odd").jqFn();`	Filters down to odd elements only. Note: Arrays are zero-indexed! Zero is considered an even number so your second item will be selected!
:eq(number)	`jQuery(".post:eq(0)").jqFn();`	Filters down to a single element by its index, which again is zero-indexed.
:gt(number)	`jQuery(".post:gt(0)").jqFn();`	Filters down to all elements with an index **above** the given one, again this is zero-indexed.
:lt(number)	`jQuery(".post:lt(2)").jqFn();`	Filters all elements with an index below the given one.
:animated	`jQuery(".post:animated").jqFn();`	Filters down to all elements that are currently being animated (we'll get to animation later in this chapter).

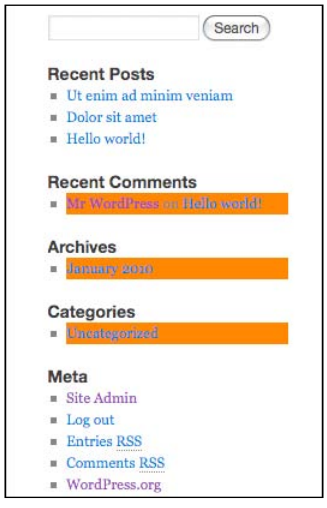
Child filters

Anything in the jQuery wrapper is an array, and these child filters will come in handy, but you'll probably find these filters come in most handy when working with `li` tags or definition list elements in WordPress. By default, WordPress splits a fair amount of its link content into `li` tag elements and galleries that are are created by wrapping the images and descriptions in definition lists (`dt dd` elements).

Example	Syntax	Description
:nth-child(number/even/odd)	`jQuery(".linkcat li:nth-child(1)").css("background", "#f60");`	Filters down to the elements that are the "nth" child of its selector. Note that this is **not** zero-indexed! `1` and odd selects the **first** element.
:first-child	`jQuery(".linkcat li:first-child").css("background", "#f60");`	Filters down to the elements that are the first child of their parent.
:last-child	`jQuery(".linkcat li:last-child").css("background", "#f60");`	Filters down to the elements that are the last child of their parent.
:only-child	`jQuery(".pagenav li:only-child").css("background", "#f60");`	Filters down to the elements that are only-children of their parent. If a parent has more than one child, no elements are selected.

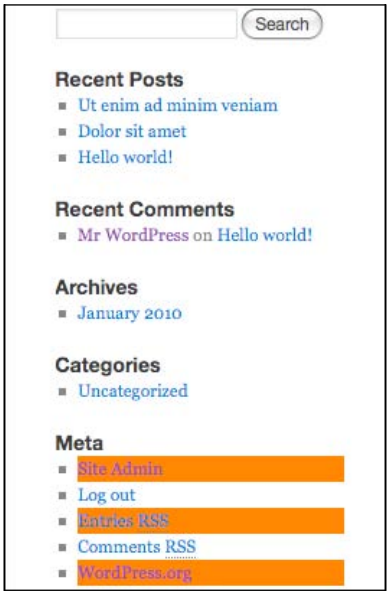
Here you can see the `:only-child` filter in action:

```
...
jQuery("li:only-child").css("background", "#f60");
...
```



Here's the `:nth-child` filter at work in the Meta list:

```
...
jQuery(".widget_meta li:nth-child(odd)").css("background", "#f60");
...
```



Content filters

After the basic and child filters, the next most useful filters you'll run into are content filters. Content filters allow you to make selections based on **matching** various types of elements and content. The most useful content filter—I often use it in WordPress—is the `:has()` filter. I often need to select elements that have something *inside* them, like anchor `a` tags that have `img` image tags inside them, or paragraph `p` tags that have list `li` tags, or other elements with a particular class name inside them. It's easy to target a specific object, but if you find you need to target a larger, parent object, based on what kind of elements are inside it, the `:has()` filter will become your best friend.

The next most useful item is the `:contains()` element which, at first blush, might seem very similar to `:has()`! But this filter is very different (and really cool), in that it allows you to target specific *text* inside an element.

Be careful with these two filters and make as many "preselections" as possible. You want to make sure jQuery is pointed in the right direction for the elements and text you're trying to select. Just specifying `...(p:contains('my text'))...` may be too general for a large page of content; you'll cause jQuery to lag, or worse, hang and timeout because it has to search every single little `p`, `div`, or `a` element on the page for your text or elements. A jQuery that specifies `...(#divIdName .className a:contains('my text'))...` is much better because jQuery only has to search through the text of every `a` element within one specific ID container's specified classes, as opposed to the *entire* page of content.

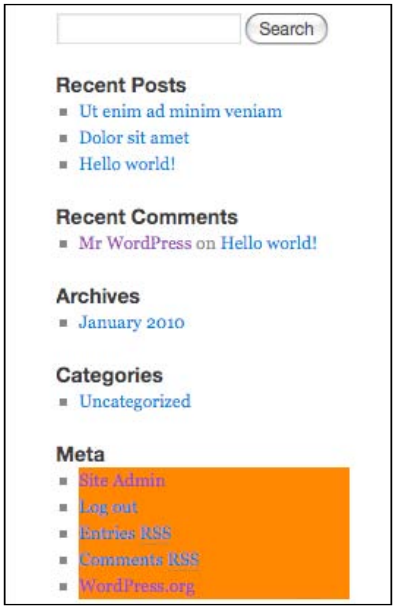
Let's take a look at the following content filters in more detail:

Example	Syntax	Description
:has(selector)	`jQuery(".post:has(.entry)") .css("background", "#f60");`	Filters down to elements that have at least one of the matching elements inside it.
:contains(text)	`jQuery(".post:contains('Hello world')").css("background", "#f60");`	Filters down to elements that contain the specific text. Note: This is **case sensitive**!
:empty	`jQuery(":empty')") .css("background", "#f60");`	Filters down to elements that have no children. This includes text nodes.
:parent	`jQuery(":parent')") .css("background", "#f60");`	Filters down to elements that are the parent of another element. This includes text nodes.

For an in-depth example, let's look at the sidebar of the default theme. The sidebar has some items that are not denoted with a special `id` name or `class`. If I want to target the `ul` list that is only under the Meta header, I can target it using `:has()` and `:contains()`. Notice how I "direct" jQuery, by preselecting, or pointing to the `.widget-area li` tags first, so that jQuery ignores the rest of the page *before* I tell you to look for children elements and containing text.

You can see the result of the following code in the next screenshot:

```
...
jQuery(".widget-area li:has(h3:contains('Meta')) ul")
  .css("background", "#f60");
...
```



Form filters

As if all the previous selectors and filters weren't cool enough, you can also explicitly filter to several types of form elements as well as types of events for those elements. Using these filters, you'll be able to take control of your WordPress generated comment forms as well as custom and WordPress plugin forms and make them even more intuitive and easier to use. Later on in this book, we'll see how jQuery can make form use and validation dead simple.

Example	Syntax	Description
:input	`jQuery("form:input").css("background", "#f60");`	Filters to all input, text area, select, and button elements
:text	`jQuery("form:text").css("background", "#f60");`	Filters to all input elements that are of type text
:password	`jQuery("form:password"). css("background", "#f60");`	Filters to all input elements that are of type passwords
:radio	`jQuery("form:radio").css("background", "#f60");`	Filters to all input elements that are of type radio
:checkbox	`jQuery("form:checkbox"). css("background", "#f60");`	Filters to all input elements that are of type checkbox
:submit	`jQuery("form:submit"). css("background", "#f60");`	Filters to all input elements that are of type submit
:image	`jQuery("form:image").css("background", "#f60");`	Filters to all image elements (classified as a form filter, but useful for regular images)
:reset	`jQuery("form:reset").css("background", "#f60");`	Filters to all input elements that are of type reset
:button	`jQuery("form:button") .css("background", "#f60");`	Filters to all input elements that are of type button
:file	`jQuery("form:file").css("background", "#f60");`	Filters to all input elements that are of type file

Using the following code, I've highlighted only the `text` input and `submit` buttons, as shown in the next screenshot:

```
...
jQuery(":text, :submit").css("background", "#f60");
...
```

Attribute filters

Attributes are those additional properties found inside HTML tags that allow the tag to refine itself. You're probably most familiar with the `id` and `class` attributes as well as the `src` attributes for `img` and `script` tags and of course the `href` attribute for `a` tags.

Attributes are powerful properties for defining and refining HTML elements, so you can imagine how powerful being able to filter using them can be. Powerful yes, but do keep in mind the simplest and the most direct approach to selecting items into the jQuery wrapper is often the best. My examples will show different class selections because they create nice visual examples, but in reality, you're better off using regular selectors to target class items and saving attribute filters for your more refined, tricky work.

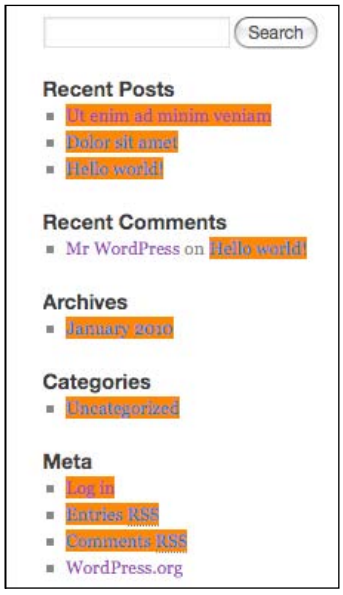
You'll note that these filters differ from the other filters. Instead of `:` (colon marks), these filters use `[]` (square brackets). This means you can easily see in your selector syntax if you're filtering for an attribute. You'll also note that for every attribute out there in HTML's DOM, you can filter for it. There's no standard set of "attribute filter names"; you simply use the square brackets to indicate whatever attribute you want to filter for. You can even structure your attribute filter in a few ways:

Example	Syntax	Description
[attribute]	`jQuery("div [href]") .css("background", "#f60");`	Filters for an attribute, regardless of its value
[attribute=value]	`jQuery("div [class='entry']") .css("background", "#f60");`	Filters for an attribute and an *exact* specified value
[attribute!=value]	`jQuery("div [class!='entry']") .css("background", "#f60");`	Filters for attributes that do not have a specified value
[attribute^=value]	`jQuery("div [href^='http://']") .css("background", "#f60");`	Filters for attributes that have a value that *begins* with a specific string
[attribute$=value]	`jQuery("div [href$='/']") .css("background", "#f60");`	Filters for attributes that have a value that *ends* with a specific string
[attribute*=value]	`jQuery("div [href*='page_ id']").css("background", "#f60");`	Filters for attributes that contain a string

Here, we can take a look at targeting only the local links in our sidebar with the following jQuery code:

```
...
jQuery(".widget-area [href^='http://localhost']").css("background",
"#f60");
...
```

The following screenshot shows the result, and only `localhost` links referencing the WordPress installation are highlighted:



Visibility

I've saved these two filters for last, mostly because I don't use them very much in most of my WordPress projects, but they are part of the selector/filter API so I'll go ahead and cover them here.

Most of the time, everything you'll need to target with jQuery is by default, visible. But occasionally, you may have an item that you've previously hidden with a jQuery transformation or a form field that is hidden and you'll want to run a transformation on it. For that, you can use the `:hidden` filter. This is a little tricky, as you've selected the item into your wrapper, but you won't necessarily see any transformation (unless the transformation is to make it visible). If you find yourself with quite a few hidden elements, you can always filter for what's visible, if that's easier.

Example	Syntax	Description
:hidden	`jQuery("form:input:hidden") .css("background", "#f60");`	Filters for elements that have a display value of none or type value of hidden or have an explicit width and height of `0`
:visible	`jQuery("div .post:visible") .css("background", "#f60");`	Filters for elements that are visible

I've covered the main selectors and filters that I get the most use of being a WordPress developer. Be sure to look through the jQuery documentation for all the selectors and filters available listed in alphabetical order: `http://api.jquery.com/category/selectors/`.

jQuery secret weapon #2: Manipulating CSS and elements in the DOM

Now that we can reliably select any *object* our WordPress site displays on a page, let's start manipulating and enhancing our selections! We can manipulate our CSS styles which display our objects and as if that isn't cool enough, we can also manipulate the HTML objects themselves in the DOM. Let's get started with manipulating CSS.

Manipulating CSS

So far, everything that we've looked at regarding selectors and filters is essential for targeting the elements you want to affect. Now that you can select anything you want into the wrapper, let's start making stuff happen! Thanks to all of my previous examples, you're already familiar with the `css()` function. Mostly, you'll use this function to assign standard CSS property values, such as: `background`, `border`, `padding`, `margins`, and so on. If you can assign the property in a CSS stylesheet, you can assign it using the `css()` function. You can also retrieve and get CSS properties with this function.

Within the Attributes API of jQuery, you'll find more CSS manipulation features such as the `.addClass`, `.removeClass`, and `.toggleClass`. These three functions alone will give you a lot of power in making your WordPress site dynamic. Don't be confused by my continued talk of attributes! We're not dealing with selectors and filters anymore. We're dealing with functions that allow you to manipulate those selections. Let's take a look at some of jQuery's CSS and class attribute manipulation functions in detail:

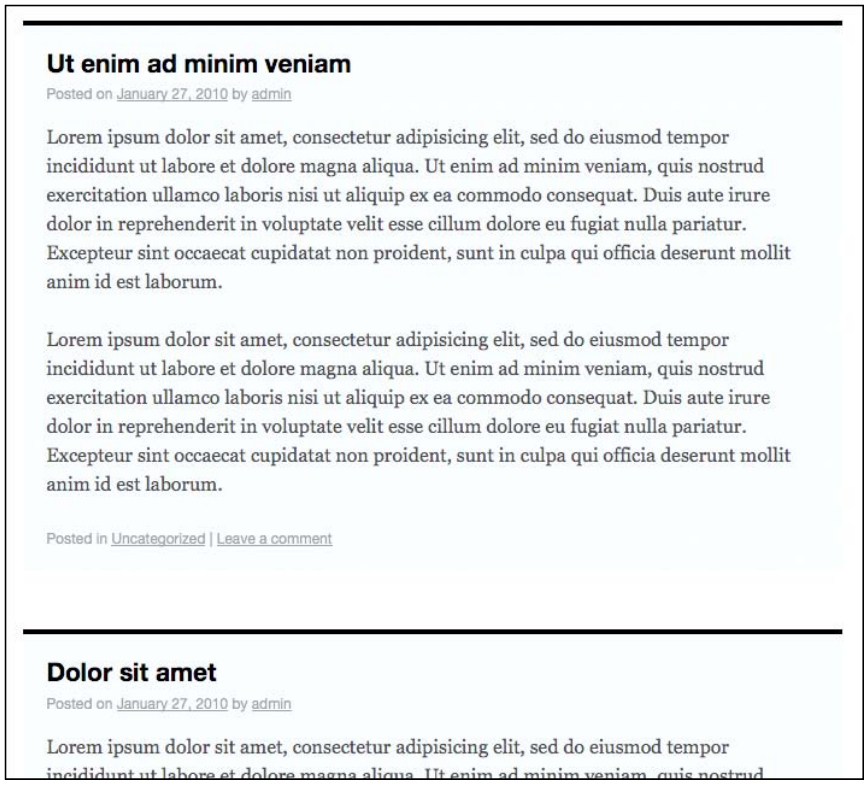
Example	Syntax	Description
.css('property', 'value')	`jQuery(".post").css("background", "#f60");`	Adds or changes the CSS properties of the selected elements.
.addClass('className')	`jQuery(".post").addClass("sticky");`	Adds listed class(es) to each of the selected elements.
.removeClass('className')	`jQuery(".post").removeClass("sticky");`	Removes listed class(es) from each of the selected elements.
.toggleClass('className', switch-optional)	`jQuery(".post").toggleClass("sticky");`	Toggles listed class(es) from each of the selected elements based on their current state. If the class is there, it's removed, and if it's not, it's added.
.hasClass('className')	`jQuery(".post").hasClass("sticky");`	Returns true or false if listed class(es) from each of the selected elements exist.

Let's check out that `addClass()` function by adding the default's theme `sticky` class to all posts.

When making selections, you'll need to denote `class` names from `id` names from `tag` names, but in these jQuery class attribute functions, you only need to put in the name of the class. You don't need to denote it with a "." period. The function is only expecting a class name so it's not necessary. As you might expect, you obviously can't add an `id` name to a selection using the `addClass` function (and nope, sorry, there's no `addId` function!)

```
...
jQuery(".post").addClass("sticky");
...
```

You can now see in the next screenshot that the `.sticky` class has been added to all the `.post` classes through jQuery, not WordPress!



Manipulating attributes

You can also affect the attributes of specific objects (this comes in handy for switching our image paths, and provides another way to work with `class` names and even `object` ID names)

Example	Syntax	Description
.attr	`jQuery(".post").attr();`	Retrieves the attribute's value for the first element of the selected elements
.removeAttr	`jQuery(".post a").removeAttr("href");`	Removes an attribute from each of the selected elements

More power over CSS:

If you ever need to work with HTML objects in a nice, cross-browser friendly way, it's easy to retrieve and set a host of property and height and width variables on any selector you target. Occasionally, these will come in handy, but you'll find the brunt of your work done with the functions as listed in the previous table. None-the-less, you'll want to take a look at the positioning and height and width functions under jQuery's CSS API: `http://docs.jquery.com/CSS`.

Manipulating elements and content

The Manipulation section of jQuery's API is again extensive, but I find some of the functions useful for helping along my WordPress and jQuery enhancements. For example, if you make something expandable or retractable, you'll need an element for the user to handle that event, rather than having to go into every post and add control buttons (or remind your client or site editors to add control links or buttons to each post—yeah, they'll do that). You can add and remove content and HTML elements on the fly, using jQuery.

The most useful functions are the `prepend()` and `append()` functions allowing you to include text before or after your selection. These functions allow you to focus on content, or specific selectors within your selection, whichever is easiest for you to target.

The next most useful functions are the `before()` and `after()` and `instertBefore()` and `instertAfter()` functions. If you find you need to wrap elements inside a class name or HTML element to add extra styling, that's no problem with the `wrap()` function. You can even remove and clone elements! Let's take a look at these manipulation functions in more detail.

Example	Syntax	Description
.append(html & text)	`jQuery(".post").append("<b>post ends here</b>");`	Inserts content in the parameter, to the end of each selected element.
.appendTo(selector)	`jQuery("<b>post ends here</b>").appendTo(".post");`	Does the same thing as append, just reverses the element selection and content parameter.
.prepend(html & text)	`jQuery(".post").prepend("<b>post starts here</b>");`	Inserts content in the parameter, to the beginning of each selected element.

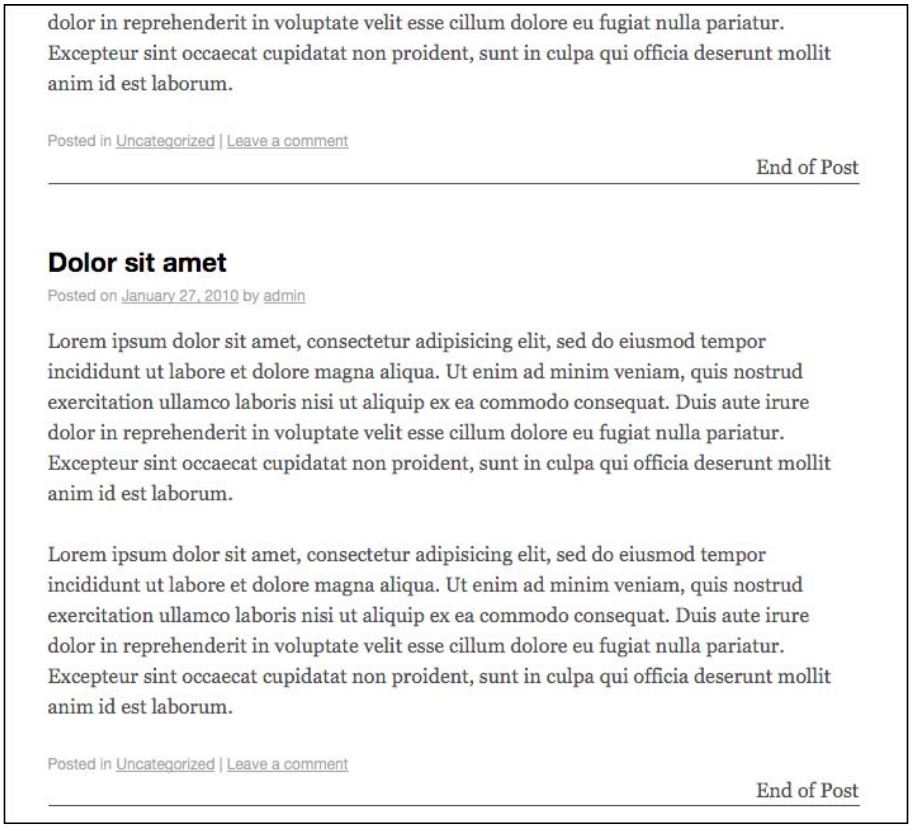
Example	Syntax	Description
.prependTo(selector)	`jQuery("<b>post starts here</b>").prependTo(".post");`	Does the same thing as prepend, just reverses the element selection and content parameter.
.after(string)	`jQuery(".post").after("<b>This goes after</b>");`	Inserts content in the parameter, after and outside of each selected element.
.insertAfter(selector)	`jQuery("<b>This goes after</b>").insertAfter(".post");`	Does the same thing as after, just reverses the element selection and content parameter.
.before(HTML & text)	`jQuery(".post").before("<b>This goes before</b>");`	Inserts content in the parameter, before and outside of each selected element.
.insertBefore(selector)	`jQuery("<b>This goes before</b>").insertBefore("class");`	Does the same thing as before, just reverses the element selection and content parameter.
.wrap(html or functionName)	`jQuery(".post").wrap("<div class=".fun" />");`	Wraps an HTML structure around each selected element. You can also construct a function that will wrap each element in HTML.
.wrapAll(HTML)	`jQuery(".post").wrapAll("<div class=".fun" />");`	Similar to wrap, but places the HTML structure around all of the elements together, not each individual element.
.wrapInner(selector)	`jQuery(".post").wrapInner("<div class=".fun" />");`	Similar to wrap, but it places the HTML structure *inside* each of the selected elements around any text or child elements of each selected element.
.html(HTML & text)	`jQuery(".post").html("<h2>Replacement Text</h2>");`	Replaces any content and child elements of selected items with the content in the parameter.
.text(text only–HTML chars will be escaped)	`jQuery(".post").text("Replacement Text");`	Similar to HTML, but text only. Any HTML characters will be escaped as ASCII codes.

Example	Syntax	Description
.empty(selector)	`jQuery(".post").empty(".entry");`	Deletes any content and child elements of a selected element. Leaves the element.
.remove(selector)	`jQuery(".post").remove();`	Similar to empty but deletes the entire element.
.clone(selector)	`jQuery(".post").clone();`	Duplicates the selected elements.

Here we can see how easy it is to use these types of functions:

```
...
jQuery(".post").append("<div style='text-align:right;
  border-bottom: 1px solid #333'>End of Post</div>");
...
```

The above jQuery script adds **End of Post** to the end of every post as seen in the following screenshot:



Working with the DOM

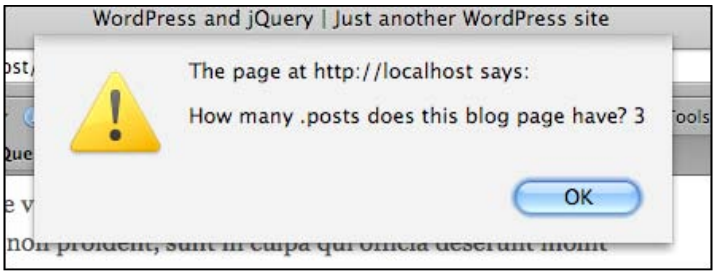
With jQuery, you can actually traverse and handle the DOM itself instead of just dealing with the elements that are in the jQuery wrapper set (remember, these are no longer pure DOM elements in the array). In order to work directly with the DOM, you can use a few jQuery functions and properties. jQuery's documentation site itself has a pretty exhaustive list of 20 or 30 functions that you can use to help you traverse the DOM, though again working with WordPress, you most likely will not need to work directly with it. The ones I use most are actually part of the jQuery core and not found in the Traversing API, but I use them similarly to help me refine and navigate DOM objects.

Example	Syntax	Description
.length or size()	`jQuery(".post").length;`	Returns the number of elements in the selected set.
.get(number-optional)	`jQuery(".post").get(3);`	This will return the array of native DOM elements. Comes in handy if you don't want to deal with DOM directly and not the jQuery wrapped elements.
.find(selector)	`jQuery(".post").find(".entry b");`	Returns an array of jQuery elements inside the first selector that match the find function's selector.
.each(functionName)	`jQuery(".post").each(function(){// code});`	This will run a function on every element that matches the jQuery selector.

As these functions return numbers and arrays, you'll find them most useful for troubleshooting. To easily reference one of these functions, I simply set up `alert()` functions with my jQuery statements as follows:

```
...
alert("How many posts does this blog have? "+jQuery(".post").length);

jQuery(".post").each(function(){
    alert("one alert for each .post")
});
...
```

You can see the resulting alert here in the following screenshot:



Be sure to take a look at the full traversing functions.

Again, the point of jQuery is to get you away from the details of the DOM, but as you get more sophisticated with your use of jQuery, you don't want to forget these functions are available to you at `http://docs.jquery.com/Traversing`.

You can also take a closer look at the jQuery core at `http://docs.jquery.com/Core`.

jQuery secret weapon #3: Events and effects (aka: the icing on the cake)

All right, you are a *selection* master; you can grab anything you want from anyone's CSS and WordPress theme and you can *manipulate* those selections' CSS properties and attributes until the cows come home. Just from these first examples, you've probably managed to come up with your very own impressive jQuery enhancements. But wait, there's more! Let's bring it all together with events and effects.

Working with events

There are lots of events that you can handle with jQuery. You can manually **bind** and **unbind** events to elements, you can reference the **unified event object**, and you can use event helpers. We're going to save looking at the jQuery's unified event object until a little later in this book and for now, take a look at the most direct ways to get started with events.

Helpers are so helpful!

The helper functions, also often referred to as "shortcuts", let you easily set up events on a click or hover. You can also easily toggle events. We saw how useful the `toggleClass()` function was in the CSS Manipulation section; imagine being able to toggle *more* functions.

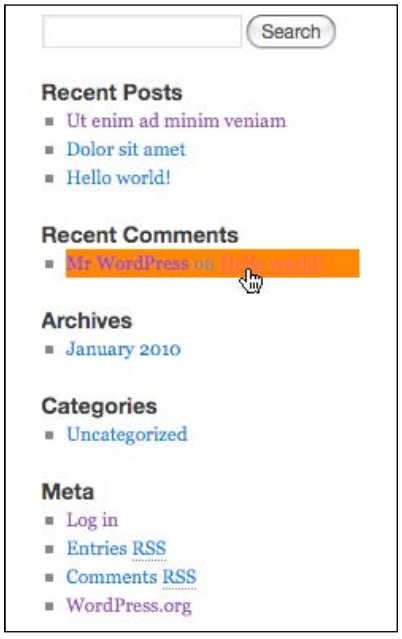
Most of the time, `hover()` will accomplish your needs, but if you want your event to be a click, then the `toggle()` function will probably work best. The `toggle()` function allows a bit more flexibility than `hover` because you can add in additional functions and not be constrained to just one or two functions.

Example	Syntax	Description
.click(functionName)	`jQuery(".post") .click(function(){// code});`	Binds a function to the `click` event type, executed on a single click.
.dbclick(functionName)	`jQuery(".post") .dbclick(function(){// code});`	Binds a function to the `click` event type, executed on a double click.
.hover(functionName1, functionName2)	`jQuery(".post") .hover(function(){// code});`	Works with the `mouseenter`/ `mouseleave` event types and binds just two functions to the selected elements, to be executed on `mouseenter` and `mouseleave`.
.toggle(functionName1, functionName2, functionName3, etc)	`jQuery(".post") .toggle(function(){// code});`	Works with the `click` event type and binds two or more functions to the selected elements, to be executed on alternate clicks.
.mouseenter(functionName)	`jQuery(".post") .mouseenter(function(){// code});`	Binds a function to be executed when the mouse enters the selected elements.
.mouseleave(functionName)	`jQuery(".post") .mouseleave(function(){// code});`	Binds a function to be executed when the mouse leaves the selected elements.

Example	Syntax	Description
.keydown(functionName)	`jQuery(".post").keydown(function(){// code});`	Binds a function to the `keydown` event type, executed only when the selected element has a *focus* and keys are pressed down.
.keyup(functionName)	`jQuery(".post").keyup(function(){// code});`	Binds a function to the `keyup` event type, executed only when the selected element has a *focus* and keys are pressed then released.

With events comes a more live and dynamic page. Let's set up a very simple hover on our sidebar navigation items:

```
...
jQuery(".widget-area li ul li").hover(function(){
    jQuery(this).css("background", "#f60");
  },
  function(){
    jQuery(this).css("background", "none");
});
...
```



Working with bind, unbind, and the event object

We'll just do a quick overview of these functions; using this method is a bit of overkill, but it might come in handy for specific uses and if nothing else, make you appreciate all the helper shortcut functions jQuery provides you with.

Occasionally, you may need to really refine the control of your events, and using the `bind()` and `unbind()` functions, you may well handle this for yourself. You can pass parameters of event types, such as `click` or `mouseenter`; you can pass some data as well as an event handler (or you can call another function). The data is an optional parameter and it's a tad beyond the scope of this chapter to get into, but for those of you who have become really interested in developing with jQuery, it's good to know you can pass data around if need be (and we'll do our bit even in this chapter)!

Let's take a closer look and break down the parts of these functions:

Example	Syntax	Description
.bind(event type, data, functionName)	`jQuery(".post").bind("mouseenter", function(){//code});`	Attaches a function to be triggered on a type of event to the selected elements.
.unbind(event type, functionName)	`jQuery(".post").bind("mouseenter", function(){//code});`	Removes the event type from the selected elements.

We can recreate what we achieved with the hover class by using `bind` and `unbind`. It's a bit more cumbersome, and ultimately not the most elegant way to go for a simple hover effect. The advantage of bind is that you can pass data around. The following example demonstrates passing data, that is, the color of our background, to the event's function:

```
...
  jQuery(".widget-area li ul li").bind("mouseenter", {color: "#f60"},
function(event){
      jQuery(this).css("background", event.data.color);
         jQuery(this).unbind("mouseleave");
  });
  jQuery(".widget-area li ul li").bind("mouseleave", function(){
      jQuery(this).css("background", "none");
      jQuery(this).unbind("mouseenter");
  });
...
```

In the previous code sample, we worked with jQuery's event object to pass the data. Working with the data, the unified event object returns can help you create refined jQuery transformations, and I often use the object's information to help pass event information to functions for cleaner code and to also help me with troubleshooting.

Example	Description
`event.type`	Returns the type of event, such as a click or `mouseenter` or `keyup`.
`event.target`	Returns the selected element the event was triggered from.
`event.data`	Returns and contains the optional data passed through the bind function.
`event.pageX, .pageY`	Determines the mouse position relative to the left edge (pageX), or top (pageY) of the document.
`event.result`	Returns the last value returned by an event handler that was triggered by this event. Very useful for troubleshooting.
`event.timeStamp`	Returns the Unix timestamp of when the event was triggered.

The following code will track event object attributes on click:

```
...
jQuery(".post").click(function(event){
    jQuery(this).html("event type: "+event.type+"<br/>event timestamp:
"+event.timeStamp+"<br/>event x: "+event.pageX+"<br/>event y: "+event.
pageY);
});
...
```

Here's one event object function which you may find useful—the `preventDefault()function`. It can stop an element's default action. The most common example would be making a `link` tag not executing its `href`. If you need to know if an element's default event has had this called on it, you can use the `isPreventDefault()` function to test for it.

Example	Syntax	Description
.preventDefault()	`jQuery(.post a).preventDefault();`	Will prevent the selected elements from their browser-set default actions.
.isPreventDefault()	`jQuery(.post a).isPreventDefault();`	Returns true or false if `ispreventDefault` was called on a set of selected elements.

Adding effects

So now we're ready for the fun section of this chapter—adding slick effects. The jQuery library provides some very basic animation and effects functions for us to work with. These are all visual effects such as showing and hiding, fading in and out, sliding up and down, or using the `animate` function to move around elements on the screen, more precisely. Most of you will be very happy with the standard shortcut animation functions, but we'll take a look at the `animate` function as well.

The majority of these functions also allow for a callback function which makes it easy to trigger additional animations or functionality that you want to have completed when the element's animation is complete. Let's get started with effects and animation.

Showing and hiding

The first thing you'll want to note about showing and hiding is that the size and the fade of the targeted elements are affected. If you want to just fade or affect the size, then you'll want to look at the other animation functions. You can also very easily use the `toggle` event we discussed before to aid in your effects.

Example	Syntax	Description
.show(speed-optional, functionName)	`jQuery(".post").css("background", "#f60").show("slow");`	Displays the matched elements; if a speed is set, the object grows in from left to right and alpha fade 0 to 1. A function can be called upon completion. Speed can be "slow" or "fast" or milliseconds.
.hide(speed-optional, functionName)	`jQuery(".post").css("background", "#f60").show(200);`	Similar to show but hides. If a speed is set, the element shrinks from right to left and alpha fade 1 to 0. A function can be called upon completion. Speed can be "slow" or "fast" or milliseconds.

Sliding in and out

You'll notice that showing and hiding "grew" the object from the right to left. Sliding is an elegant way to handle opening and closing elements with a more direct up and down motion.

Example	Syntax	Description

.slideUp(speed, functionName)	`jQuery(".post") .slideUp('slow', function() {`
	`// code`
	`});`
		Slides the selected element up from bottom to top until it is hidden. Speed can be "fast" or "slow" or milliseconds. A function can be called when the animation is finished.

.slideDown(speed, functionName)	`jQuery(".post") .slideDown('slow', function() {`
	`// code`
	`});`
		Slides a hidden selected element down from top to bottom until its size is defined. Speed can be "fast" or "slow" or milliseconds. A function can be called when the animation is finished.

.slideToggle()	`jQuery(".post") .slideToggle('slow', function() {`
	`// code`
	`});`
		Toggles the visibility of the selected element using the slide animation. Speed can be "fast" or "slow" or milliseconds. A function can be called when the animation is finished.

Fading in and out

A good fade in and out is nice as well. I do want to point out that `fadeIn()` and `fadeOut()` only work when starting from an alpha of `0` or `1`. For example: `fadeOut` only works if the element's alpha is set to `1`, and `fadeIn` only works if the element's alpha is at `0`.

I'd also like to point out that if you've previously used the `fadeTo()` function to fade to a specific alpha number, and then try to `fadeOut()` all the way or `fadeIn()` all the way, it doesn't work. Just continue to use the `fadeTo()` function to smooth your transitions up and down. Also, when using `fadeOut()`, once the element's alpha is at `0`, it disappears completely. Any space it was taking up collapses in a somewhat jarring effect. Take this into consideration when deciding to use `fadeOut()`.

Example	Syntax	Description
.fadeOut(speed, functionName)	`jQuery(".post") .fadeOut("slow", function(){//code});`	Fades a selected element that's visible or alpha is `1` to `0`
.fadeIn(speed, functionName)	`jQuery(".post") .fadeIn("slow", function(){//code});`	Fades a selected element who's visibility is hidden or alpha is set to `0` to `1`
.fadeTo(speed, alpha, functionName)	`jQuery(".post") .fadeTo("slow", .3, function(){//code});`	Fades a selected element to a specific alpha from `0` to `1`

Working with the animate function

The three animation functions in the previous table will do most of what you need. You may, however, find yourself in a situation that requires a tad more control. In that rare instance, you can use the `animate` function.

Example	Syntax	Description
.animate(css properties, duration, easing, functionName)	`jQuery(".post") .animate({width: 200, opacity: .25}, 1000, function(){//code});`	Creates a custom transition of CSS properties on the selected elements
.stop()	`jQuery(".post").stop();`	Stops an animation on a selected element

Here's an example of custom animating an `img` in a post with the `animate()` function:

```
...
  jQuery(".post img").animate({
    opacity: 0.25,
    left: '+=50',
    height: 'toggle'}, 1000, function() {
    //alert("animate function finished");
  });
...
```

It's tough to capture animation in a book so I haven't tried with the other examples, but here you get the idea of the post's image half way animated (the image height is closing and the alpha is on it's way to 0):

Making it all easy with statement chaining

As I've mentioned, one of jQuery's many powerful features is statement chaining, that is, stringing multiple functions together that will be performed in the order they're added to the chain (left to right) on the selected set all in one nice string of code. For example, we can change a CSS property, hide the selected elements, and fade them smoothly with one line of code:

```
...
jQuery(".post").css("background", "#f60").hide().fadeIn("slow");
...
```

For a more in-depth example of statement chaining, let's get to our first jQuery project in WordPress.

Our First Project: Expanding/collapsing WordPress posts

OK, this is a quick project, but it requires that we use a little bit of everything we just covered. I've always liked that WordPress had the `<!--more-->` feature to make posts "condensable" for the main post view page, but that doesn't always suit my purposes for some types of posts. Let's assume that my blog will have relatively short posts, yet I really want a reader to be able to see as many headlines as possible, above the fold, without having to scroll or scan any content (we'll suspend reality and pretend that my post headers are just unbelievably interesting and compelling).

I'd like the user to have the option to expand the post that interests him, while keeping him in the context of all the other post headlines. You've probably seen similar enhancements to this on many sites. This is a very popular jQuery enhancement for FAQ and press release posts.

Let's take a look at how we'd do that. Set up a clean `custom-jquery.js` file in your theme and let's get started.

First, we'll have to hide our post content:

```
jQuery(".post .entry-content").hide();
```

Next, we'll need some sort of control for people to click on which also gives them some intuitive instructions. Of course, it would be very inefficient to have an editor add a control element to each post, so we won't do that (but sadly, I've seen this done on a few projects). We could add it to the theme's `post.php` page, but then, the control would appear even if the user had JavaScript disabled. We want this to degrade gracefully, it's an *enhancement* after all.

If someone comes across this content in a mobile browser without JavaScript support or a text-only or text-to-speech browser, we'll want them to just view the content as normal without any non-functional elements distracting them. We'll use jQuery to add our control element. If JavaScript is disabled, it simply won't appear.

```
jQuery(".post").after("<div class='openIt' style='border-top: 1px
solid #666; color: #036; text-align:right; cursor:pointer;'>Expand</
div>");
```

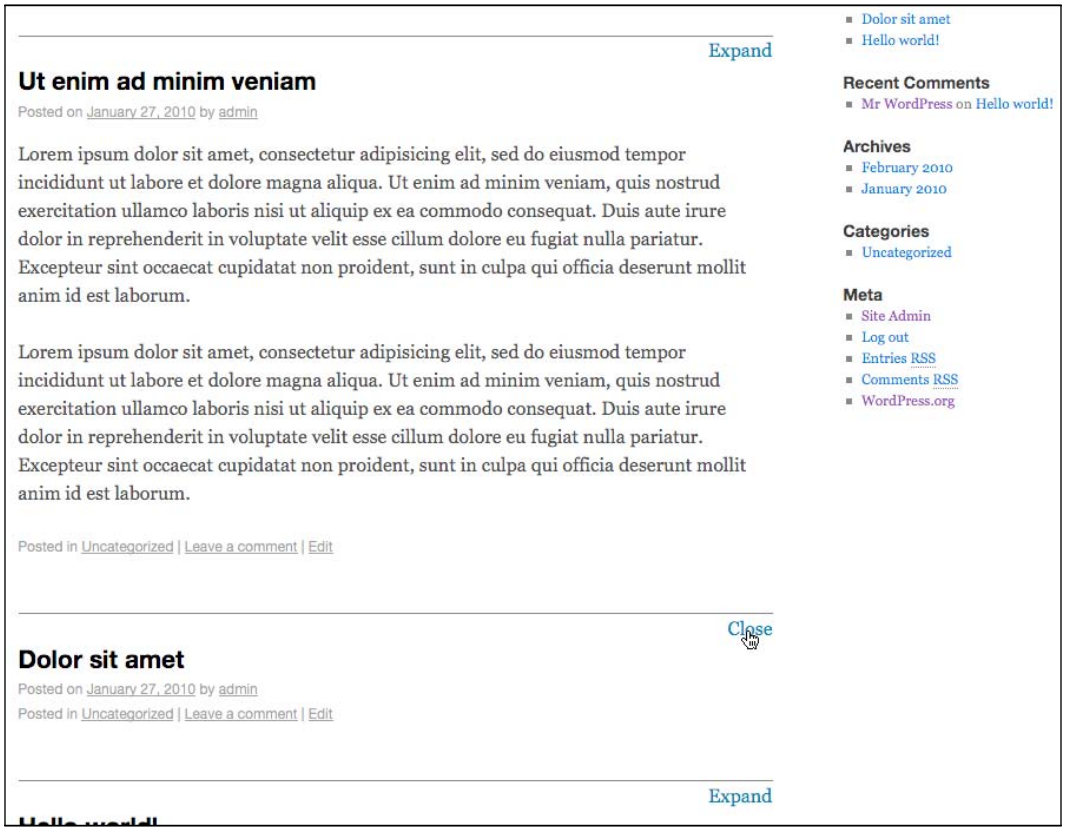
We now just need a nice way to show and hide the post's content:

```
jQuery(".openIt").click(function() {
  jQuery(this).prev(".post").find(".entry").slideToggle("slow");
});
```

Last, let's make sure the instructions in the `.openIt` div update:

```
        jQuery(".openIt").toggle(function(){
             jQuery(this).html("Close")},
           function(){
             jQuery(this).html("Expand")
         });
...
```

That's it! Your very first, *useful* jQuery enhancement for WordPress. Here's a screenshot of what it looks like:



Keeping jQuery readable

In the real world this enhancement could be cleaned up and refined quite a bit. For example, it would be better to have an existing CSS style for `.openIt` instead of applying styles to the `div`.

Also, I highly recommend writing separate, named functions. For example, it's much easier to read:

```
...
jQuery(".openIt").toggle(closePost, expandPost);
```

And then, below that see:

```
function expandPost(evt){
  //jQuery(evt.target)...
}
function closePost(evt){
  //jQuery(evt.target)...
}
...
```

If you find yourself working on a project with other developers, consider breaking your functions down like this rather than packing them directly into jQuery functions as my first example did. It makes for more maintainable code and you can reuse your functions with other jQuery functions and scripts.

Summary

To recap, we took a look at getting jQuery included into WordPress by registering WordPress' bundled version and by using Google's CDN. We also took a look at jQuery's top three "secret weapons":

- Selectors and filters
- Manipulating and changing content
- Events and effects

After exploring the basics of jQuery within WordPress and getting a feel for how they work, you may feel like you're good to go! In many ways you are, but we're going to continue exploring WordPress and jQuery in more detail about the parts of WordPress that generate content we can enhance with jQuery: We'll look deeper into WordPress themes and plugins as well as take a look at another type of plugin, the jQuery plugin. Themes and plugins can make our WordPress development work very powerfully and flexibly across multiple sites and projects.

3
Digging Deeper: Understanding jQuery and WordPress Together

Now that we've gotten a look at the basics of jQuery within WordPress, we're ready to dig a little deeper by understanding the following:

- What WordPress themes, WordPress plugins, and jQuery plugins are and do
- The basics of creating your own WordPress themes, plugins, and jQuery plugins
- Best practices for how and when to apply jQuery directly to a theme or to WordPress plugin, as a script or as a jQuery plugin

By taking a closer look at these two main components of WordPress, the theme and the plugin as well as how to encapsulate our jQuery code for easier use across projects inside a jQuery plugin, we're well on our way to mastering dynamic WordPress development.

Two ways to "plugin" jQuery into a WordPress site

You're aware that WordPress is an impressive publishing platform. Its core strength lies in its near perfect separation of content, display, and functionality. Likewise, jQuery is an impressive JavaScript library with a lot of effort spent on making it work across platforms, be very flexible and extensible, and yet, elegantly degradable (if a user doesn't have JavaScript enabled for some reason).

You're aware that WordPress themes control the look and feel of your site and that WordPress plugins can help your site do more, but we're going to take a look at exactly how those two components work within the WordPress system and how to use jQuery from either a theme or a WordPress plugin. In doing so, you'll be better able to take advantage of them when developing your jQuery enhancements.

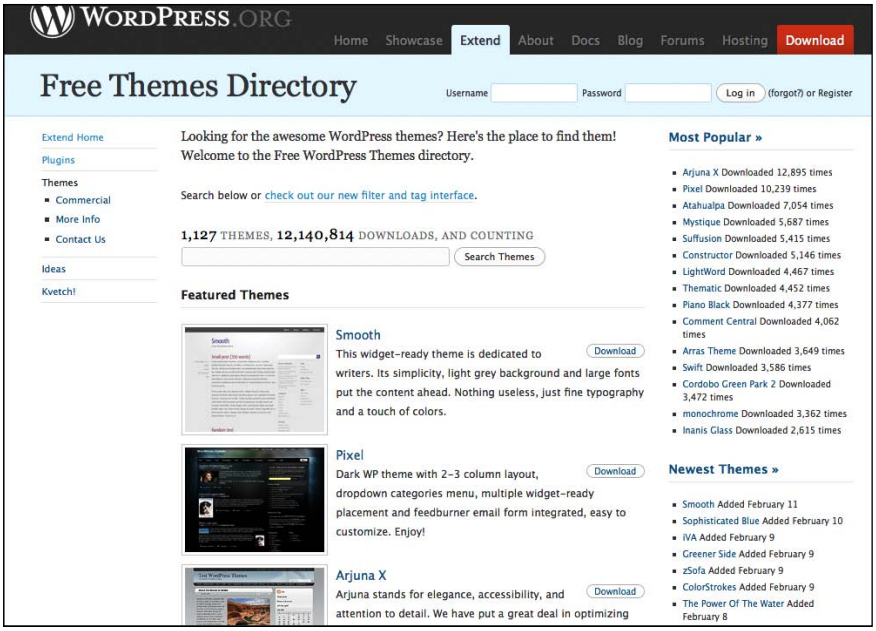
Speaking of jQuery enhancements, jQuery scripts can be turned into their own type of plugins, not to be confused with WordPress plugins. This makes the work you do in jQuery easily portable to different projects and uses.

Between these three components, themes, WordPress plugins, and jQuery plugins, you'll find that just about anything you can dream of creating is at your fingertips. Even better, you'll realize that most of the work is already done. All three of these component types have extensive libraries of already developed third-party creations. Most are free! If they aren't free, you'll be prepared to determine if they're worth their price.

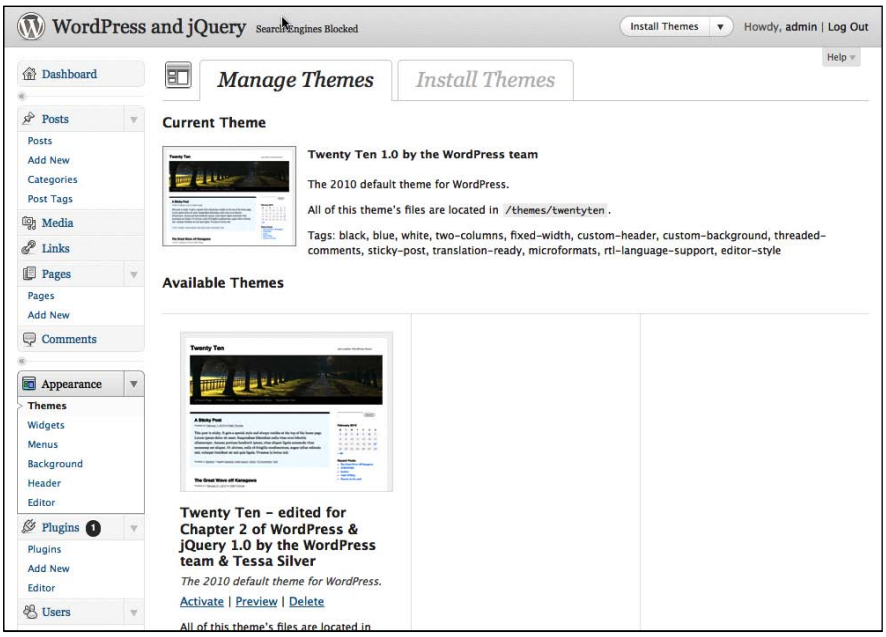
By understanding the basics of editing themes and creating your own WordPress and jQuery plugins, you'll be ready to traverse the world of third-party creations and find the best solutions for your projects. You'll also be able to determine if it's better or faster to work with another developer's themes, plugins, or jQuery plugins, versus creating your own from scratch.

WordPress themes overview

A WordPress theme is, according to the WordPress codex, *a collection of files that work together to produce a graphical interface with an underlying unifying design for a weblog*. Themes comprise a collection of template files and web collateral such as images, CSS stylesheets, and JavaScript. Themes are what allow you to modify the way your WordPress site looks, without having to know much about how WordPress works, much less change how it works. There are plenty of sites that host free themes and or sell premium WordPress themes. A quick Google search for "wordpress themes" will give you an idea of the enormity of options available. However, when first looking for or researching themes, a good place to start is always WordPress' free theme gallery where you can easily review and demo different themes and styles: `http://wordpress.org/extend/themes/`. The next screenshot shows the main page of the WordPress theme's directory:



Once you've selected a theme to use or work with, you'll activate the theme by navigating to **Administration | Appearance | Themes** in the left-hand side panel of your WordPress installation's administration panel. The next screenshot displays the **Manage Themes** panel:

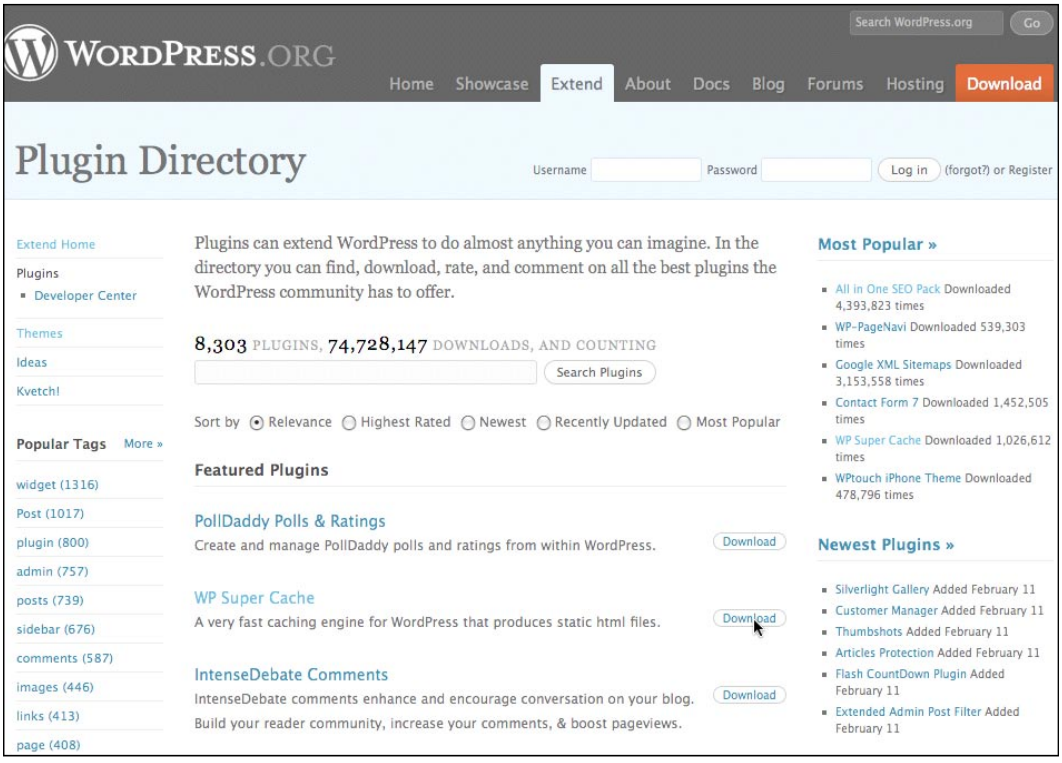


That's the minimum you need to know about themes as a WordPress user. Before we get into more detail, let's get an overview of WordPress plugins and jQuery plugins first.

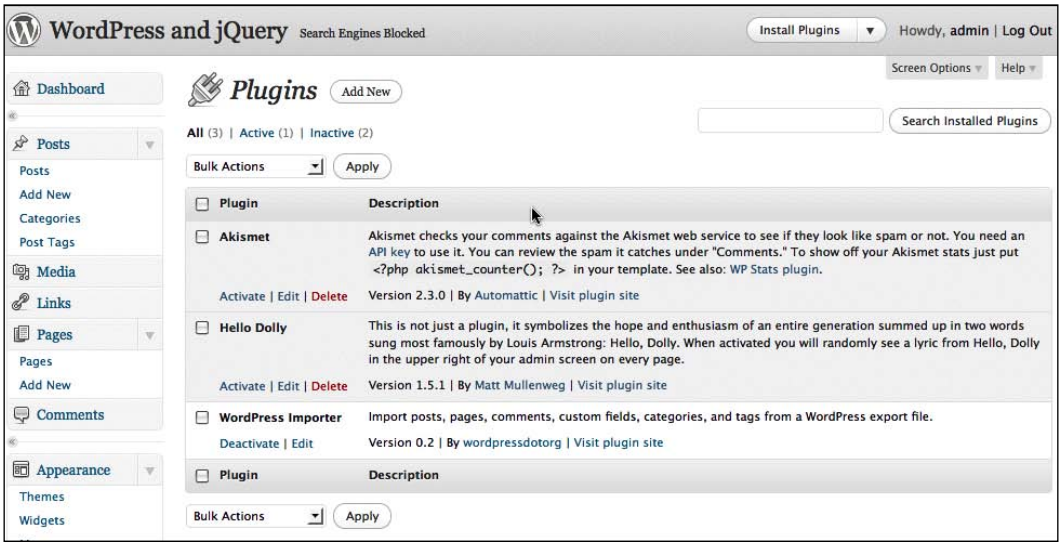
WordPress plugins overview

So themes change the look of WordPress without affecting its functionality. But what if you want to change or add functionality? WordPress plugins allow easy modification, customization, and enhancement to a WordPress site. Instead of having to dig in to the main files and change the core programming of WordPress, you can add functionality by installing and activating WordPress plugins.

The WordPress development team took great care to make it easy to create plugins using access points and methods provided by the WordPress' **Plugin API** (Application Program Interface). The best place to search for plugins is: `http://wordpress.org/extend/plugins/`. The following is a screenshot of the WordPress plugin directory's main page:



Once you have a plugin, it's a simple matter of decompressing the file (usually just unzipping it) and reading the included `readme.txt` file for installation and activation instructions. For most WordPress plugins, this is simply a matter of uploading the file or directory to your WordPress installation's `wp-content/plugins` directory and then navigating to the **Administration | Plugins | Installed** panel to activate it. The next screenshot shows the **Plugins** admin panel with the activation screen for the default **Askimet**, **Hello Dolly**, and new **WordPress Importer** plugins:

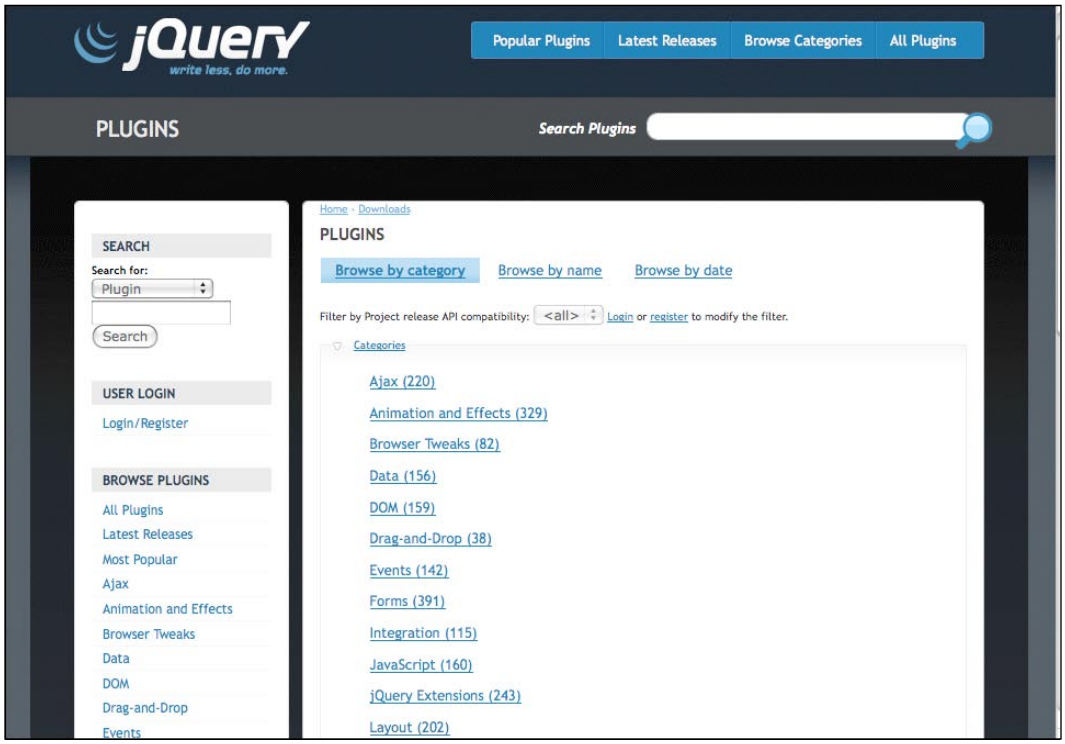


So how does a WordPress plugin differ from a jQuery plugin? In theory and intent, not that much, but in practice, there are quite a few differences. Let's take a look at jQuery plugins.

jQuery plugins overview

jQuery has the ability to allow you to take the scripts that you've created and encapsulate them into the jQuery function object. This allows your jQuery code to do a couple of key things. First, it becomes more easily ported over to different situations and uses. Second, your plugin works as a function that can be integrated into larger scripts as part of the jQuery statement chain.

The best place to browse for jQuery plugins is the jQuery plugins page (`http://plugins.jquery.com/`), as seen in the next screenshot:



In addition to having jQuery already bundled, WordPress has quite a few jQuery plugins already bundled with it as well. WordPress comes bundled with **Color**, **Thickbox** as well as **Form** and most of the **jQuery UI** plugins. Each of these plugins can be enabled with the `wp_enqueue_script` either in the theme's `header.php` file or `function.php` file, as we learned in *Chapter 2*, *Working with jQuery in WordPress*. In this chapter, we'll shortly learn how to enable a jQuery plugin directly in a WordPress plugin.

Of course, you can also download jQuery plugins and include them manually into your WordPress theme or plugins. You'd do this for plugins that aren't bundled with WordPress, or if you need to amend the plugins in anyway.

Yes, you've noticed there's no easy jQuery plugin activation panel in WordPress. This is where understanding your chosen theme and WordPress plugins will come in handy! You'll soon find you have quite a few options to choose from when leveraging jQuery. Now that we have an overview of what WordPress themes, plugins, and jQuery plugins are, let's learn how to take better advantage of them.

The basics of a WordPress theme

By now you've gotten the point that the WordPress theme essentially contains the HTML and CSS that wrap and style your WordPress content. Thus, it's usually the first place you'll start when incorporating jQuery into a site. Most of the time, this is a good approach. Understanding a bit more about how themes work can only make your jQuery development go a little smoother. Let's take a look at how themes are structured and best practices for editing them.

Want to know more about WordPress theme design?

This title focuses on what you most need to know to work with jQuery in WordPress. If you're interested in WordPress theme development I highly recommend *April Hodge Silver* and *Hasin Hayer's* **WordPress 2.7 Complete**. Along with covering the complete core competencies for managing a WordPress site, *Chapter 6*, *WordPress and jQuery's UI* has an overview on editing and creating standard themes for WordPress.

If you want to really dig deep into theme design, my title **WordPress 2.8 Theme Design** will walk you through creating a working HTML and CSS design mockup and coding it up from scratch.

Understanding the template's hierarchy

We've discussed that a WordPress theme comprises many file types including template pages. Template pages have a structure or hierarchy to them. That means, if one template type is not present, then the WordPress system will call up the next level template type. This allows developers to create themes that are fantastically detailed, which take full advantage of all of the hierarchy's available template page types, to make the setup unbelievably simple. It's possible to have a fully functioning WordPress theme that consists of no more than an `index.php` file!

To really leverage a theme for jQuery enhancement (not to mention help you with general WordPress troubleshooting), it's good to start with an understanding of the theme's hierarchy.

In addition to these template files, themes of course also include image files, stylesheets, and even custom template pages, and PHP code files. Essentially, you can have 14 different default page templates in your WordPress theme, not including your `style.css` sheet or includes such as `header.php`, `sidebar.php`, and `searchform.php`. You can have more template pages than that if you take advantage of WordPress' capability for individual custom page, category, and tag templates.

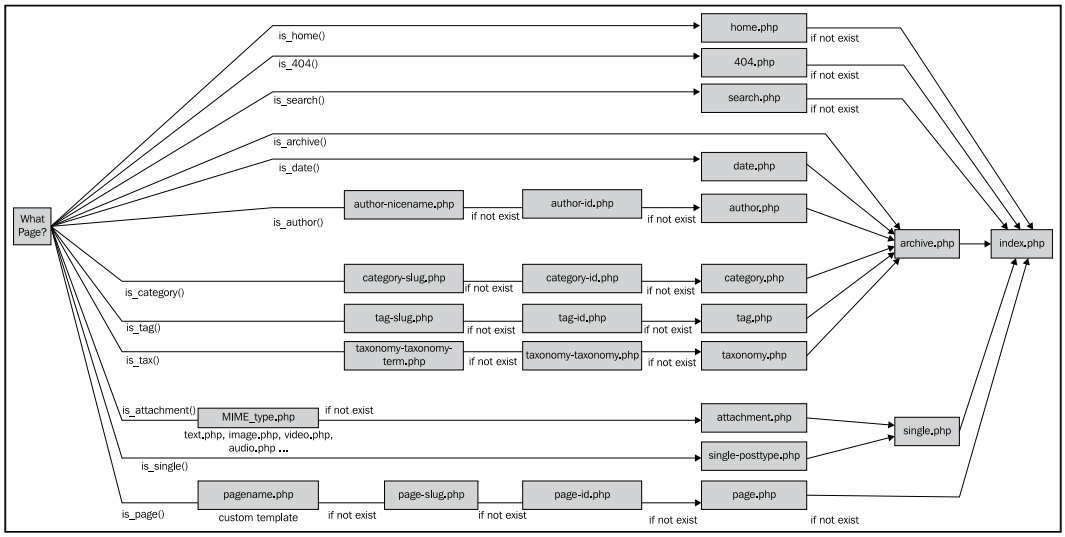
If you open up the default theme's directory that we've been working with, you'll see most of these template files as well as an image directory, `style.css` and the `js` directory with the `custom-jquery.js` file we started in *Chapter 2*, *Working with jQuery in WordPress*. The following screenshot shows you the main files in WordPress 3.0's new default theme, **Twenty Ten**:

The next list contains the general template hierarchy rules. The absolute simplest theme you can have must contain an `index.php` page. If no other specific template pages exist, then `index.php` is the default.

You can then begin expanding your theme by adding the following pages:

- `archive.php` trumps `index.php` when a category, tag, date, or author page is viewed.
- `home.php` trumps `index.php` when the home page is viewed.
- `single.php` trumps `index.php` when an individual post is viewed.
- `search.php` trumps `index.php` when the results from a search are viewed.
- `404.php` trumps `index.php` when the URI address finds no existing content.
- `page.php` trumps `index.php` when looking at a static page.
    - A custom **template** page, such as: `page_about.php`, when selected through the page's **Administration** panel, trumps `page.php`, which trumps `index.php` when that particular page is viewed.
- `category.php` trumps `archive.php`, which then trumps `index.php` when a category is viewed.
    - A custom **category-ID** page, such as: `category-12.php` trumps `category.php`. This then trumps `archive.php`, which trumps `index.php`.
- `tag.php` trumps `archive.php`. This in turn trumps `index.php` when a tag page is viewed.
    - A custom **tag-tagname** page, such as: `tag-reviews.php` trumps `tag.php`. This trumps `archive.php`, which trumps `index.php`.
- `author.php` trumps `archive.php`. This in turn trumps `index.php`, when an author page is viewed.

- `date.php` trumps `archive.php`. This trumps `index.php` when a date page is viewed.

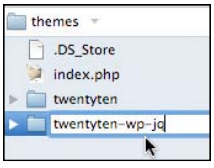


You can learn more about the WordPress theme template hierarchy here: `http://codex.wordpress.org/Template_Hierarchy`.

A whole new theme

If you wanted to create a new theme, or as in the case of this book, if you'll be modifying a theme considerably, you'll want to create a directory with a file structure similar to the hierarchy explained previously. Again, because it's hierarchal, you don't have to create every single page suggested, higher up pages will assume the role unless you decide otherwise. As I've mentioned, it is possible to have a working theme with nothing but an `index.php` file.

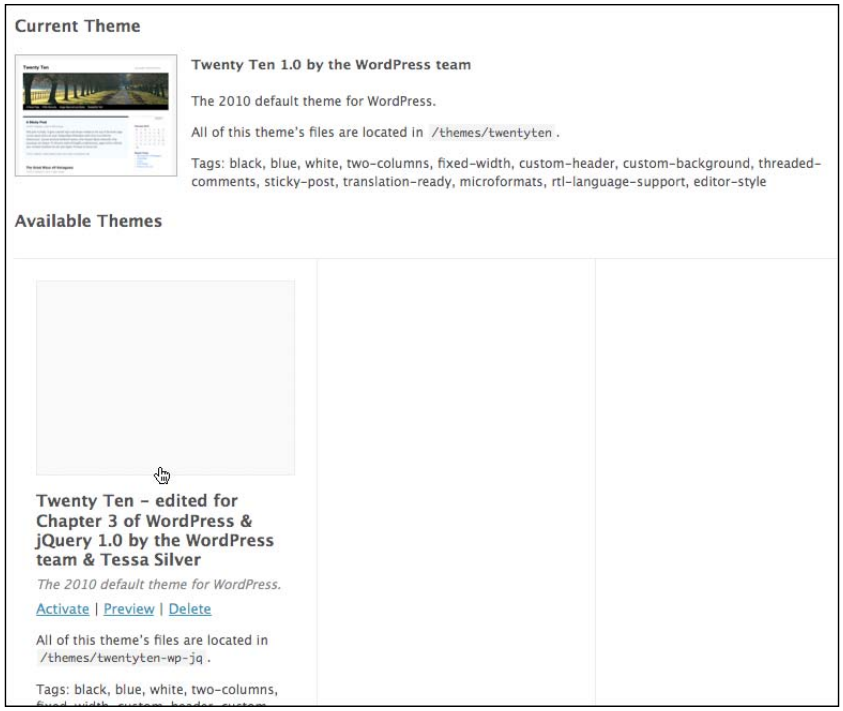
I'll be modifying the default theme, yet would like the original default theme available for reference. I'll make a copy of the default theme's directory and rename it to: `twentyten-wp-jq`. WordPress depends on the theme directories namespace. Meaning, each theme requires a uniquely named folder! Otherwise, you'll copy over another theme. The next screenshot shows this directory's creation:



I'll then open up the `style.css` file and modify the information at the beginning of the CSS file:

```
/*
Theme Name: Twenty Ten - edited for Chapter 3 of WordPress & jQuery
Theme URI: http://wordpress.org/
Description: The 2010 default theme for WordPress.
Author: the WordPress team & Tessa Silver
Version: 1.0
Tags: black, blue, white, two-columns, fixed-width, custom-header,
custom-background, threaded-comments, sticky-post, translation-ready,
microformats, rtl-language-support, editor-style

*/
...
```

My "new" theme will then show up in the administration panel's **Manage Themes** page. You can take a new screenshot to update your new or modified theme. If there is no screenshot, the frame will display a grey box. As the look of the theme is going to change a little, I've removed the `screenshot.png` file from the directory for now, as you can see in the next screenshot:



The Loop

In *Chapter 1, Getting Started: WordPress and jQuery* and *Chapter 2, Working with jQuery in WordPress* we learned how useful it was that jQuery "looped" through the selected elements in the wrapper for you. WordPress does a little looping of its own; in fact, it's important enough to be named "The Loop". **The Loop** is an essential part of your WordPress theme. It displays your posts in chronological order and lets you define custom display properties with various WordPress template tags wrapped in HTML markup.

The Loop in WordPress is a **while loop** and therefore starts with the PHP code: `while (have_posts()):` followed by the template tag `the_post()`. All the markup and additional template tags are then applied to each post that gets looped through for display. The Loop is then ended with the PHP `endwhile` statement.

Every template page view can have its own loop so that you can modify and change the look and layout of each type of post sort. Every template page is essentially, just *sorting* your posts in different ways. For example, different category or tag template pages sort and refine your posts down to meet specific criteria. Those sorted posts can appear different from posts on your main page, or in your archive lists, and so on. The next example is a very simple loop taken from WordPress 2.9.2's default Kubrick theme:

```
...
<?php while (have_posts()) : the_post(); ?>

    <div <?php post_class() ?> id="post-<?php the_ID(); ?>">
      <h2>
        <a href="<?php the_permalink() ?>"
          rel="bookmark" title="Permanent Link to
          <?php the_title_attribute(); ?>">
          <?php the_title(); ?>
        </a>
      </h2>
      <small><?php the_time('F jS, Y') ?>
        <!-- by <?php the_author() ?> -->
      </small>

      <div class="entry">
        <?php the_content('Read the rest of this entry &raquo;'); ?>
      </div>

      <p class="postmetadata">
      <?php the_tags('Tags: ', ', ', '<br />'); ?>
        Posted in <?php the_category(', ') ?> |
```

```
            <?php edit_post_link('Edit', '', ' | '); ?>
            <?php comments_popup_link('No Comments &#187;',
              '1 Comment &#187;', '% Comments &#187;'); ?>
          </p>
        </div>

    <?php endwhile; ?>
...
```

The loop is tucked into a large `if/else` statement that most importantly checks if there are posts to sort. If there are no matching posts to display, a "Sorry" message is displayed, and the `searchform.php` file is included with the `get_search_form()` include tag.

The new WordPress 3.0 Twenty Ten theme has its loop separated out into its own template page called `loop.php`, and it has quite a few more `if/else` statements within it so that the same loop code can handle many different situations, instead of writing individual loops for different template pages. On the whole, the same basic template tags as well as conditional and include tags are used in the new theme as they were before in the previous default theme. There are now just a few new template and include tags that help you streamline your theme.

Let's take a closer look at some of these template tags, include and conditional tags, and the API hooks available to us in a WordPress theme.

Tags and hooks

Within The Loop, you probably noticed some interesting bits of code wrapped in PHP tags. The code isn't pure PHP, most of them are WordPress-specific tags and functions such as **template tags**, which only work within a WordPress system. The most obviously important template tags in The Loop are `the_title()`, and `the_content()`. You'll notice that most tags and functions can have various parameters passed through them. You'll notice that in the previous code snippet, `the_content` tag has a parameter `'Read the rest of this entry &raquo;'` passed to it. That string of text with a right angle quote, will appear if the `<!--more-->` tag is placed into a post.

Not all WordPress tags and functions go inside the loop. If you poked around the `header.php` file at all in *Chapter 1*, *Getting Started: WordPress and jQuery* and *Chapter 2*, *Working with jQuery in WordPress*, you probably noticed things such as `blog_info()` and `body_class()`, and of course the `wp_enqueue_script()` that we used to register jQuery in our installation.

When having to work with theme template files for development and enhancement, I've found that the following template tags and functions are useful to recognize and know:

- `bloginfo()` — this tag can be passed parameters to retrieve all sorts of information about your blog. In the `header.php` file, you'll note it's most commonly used to find your stylesheet directory `bloginfo('stylesheet_directory')` and stylesheet URL `bloginfo('stylesheet_url')`. It can also display your RSS URL, what version of WordPress your site is running, and quite a few other details. For more details, have a look at: `http://codex.wordpress.org/Template_Tags/bloginfo`.
- `wp_title()` — this tag can be outside the loop and it displays the title of a page or single post (not a sorted list of several posts). You can pass it a few options such as what text separator to use in the title, and if the separator text should show up on the left or the right. You can also pass this tag a Boolean true or false to display the title. `wp_title("--",true,"right")`. For more details, have a look at `http://codex.wordpress.org/Template_Tags/wp_title`.
- `the_title()` — this tag goes inside the loop. It displays the title of the current post and you can pass it any text characters you'd like the title to be wrapped in: `the_title("<h2>", "</h2>")`. For more details, have a look at `http://codex.wordpress.org/Template_Tags/the_title`.
- `the_content()` — this tag goes inside the loop and it displays the post's content. If you don't pass it any `params`, it will display a generic **Read More** link if the `<!--more-->` tag is used in the post. Otherwise, you can pass it what you'd like the 'read more' instructions to say (I've even found passing an existing tag works here. `the_content("Continue Reading".the_title())`. For more details, have a look at `http://codex.wordpress.org/Template_Tags/the_content`.
- `the_category()` — this tag also has to go into the loop and it displays a link or links to the categories assigned to the post. You can pass it the text separators of your choice if there's more than one category. `the_category(", ")`. For more details, have a look at `http://codex.wordpress.org/Template_Tags/the_category`.
- `the_author_meta()` — this tag also has to go into the loop. It has a wealth of parameters that can be passed to it. You'll be most familiar with `the_author_meta("nickname")`, or `the_author_meta("first_name")`, or `the_author_meta("last_name")`. You can also get the author's bio, `the_author_meta("description")`, as well as e-mail and website URLs. Your best bet is to review the codex for all that you can do with this tag: `http://codex.wordpress.org/Template_Tags/the_author_meta`.

The WordPress template tag library is extensive and the creative ways you can use the tags in your themes can just stretch to infinity. I've included the tags that make a template useful and great, but by all means, do check out the codex:

`http://codex.wordpress.org/Template_Tags`.

Conditional tags

The conditional tags can be used in your template files to change what content is displayed and how that content is displayed on a particular page depending on what conditions that page matches. For example, you might want to display a snippet of text above the series of posts, but only on the main page of your blog. With the `is_home()` conditional tag, that task is made easy.

There are conditional tags for just about everything; out of all of them, these are the few that I find I need most in my theme development:

- `is_page()`
- `is_home()` or `is_front_page()`
- `is_single()`
- `is_sticky()`

All of those functions can take the following parameters: the post ID or page ID number, the post or page title, or the post or page slug. As great as themes are, I'm sure you've run into the conundrum that you or your client doesn't want the exact same sidebar on every single page or post.

I use these conditional tags to ensure specific pages can have particular styles or divs of content turned on and off and display or not display specific content. These tags really help give project sites a true, custom website feel.

The conditional tag fun doesn't end there. There are many more that you may find invaluable in aiding your theme's customization:

`http://codex.wordpress.org/Conditional_Tags`.

Template include tags

In the `index.php` template page and other template pages like `single.php` or `page.php` and so on, you probably noticed these include tags. They let you include standard page includes into the other template pages:

- `get_header()`
- `get_footer()`
- `get_sidebar()`
- `comments_template()`
- custom include: `include(TEMPLATEPATH."/file-name.php")`

Creating custom header, footer, sidebar includes

A while back, WordPress 2.7 introduced the ability to create *custom* headers, footers, and sidebar templates for a theme. You simply create your custom header, footer, or sidebar and call it using the standard include template tag. Be sure to add a file *prefix* of `header-`, `footer-`, or `sidebar-`, and your own file name. You can then call your custom template file as follows:

- `get_header('customHeader')` will include `header-customHeader.php`
- `get_footer('customFooter')` will include `footer-customFooter.php`
- `get_sidebar('customSidebar')` will include `sidebar-customSidebar.php`

Plugin hooks

In general, unless you're a plugin developer, you probably don't have much need to pour over the plugin API. There are, however, a few hooks that should be placed into themes in order for plugins to work effectively with your theme. If you're editing a theme, be sure to not remove these hook tags, or if you're creating a custom theme, be sure to include them:

- `wp_head`: Place within the `<head>` tags of a `header.php` template:

  ```
  <?php wp_head(); ?>
  ```

- `wp_footer`: Place within the `footer.php` template:

  ```
  <?php wp_footer(); ?>
  ```

- `wp_meta`: You'll most likely place this hook within the `sidebar.php` template. However, it's best to add this hook wherever you intend plugins and widgets to appear:

  ```
  <?php wp_meta(); ?>
  ```

- `comment_form`: Goes in `comments.php` and `comments-popup.php`, before the `</form>` closing tag:

  ```
  <?php do_action('comment_form'); ?>
  ```

Project: Editing the main loop and sidebar in the default theme

Alright! That may seem like a lot to know about themes! As someone just looking to enhance a WordPress site with jQuery, you may be asking: "Is it really necessary to know all that?" Even if you have no interest in creating custom themes, from time to time, when working with jQuery, you'll find it very useful to be able to understand how WordPress themes work, what HTML markup the theme is outputting, and what most of the different tags and functions do.

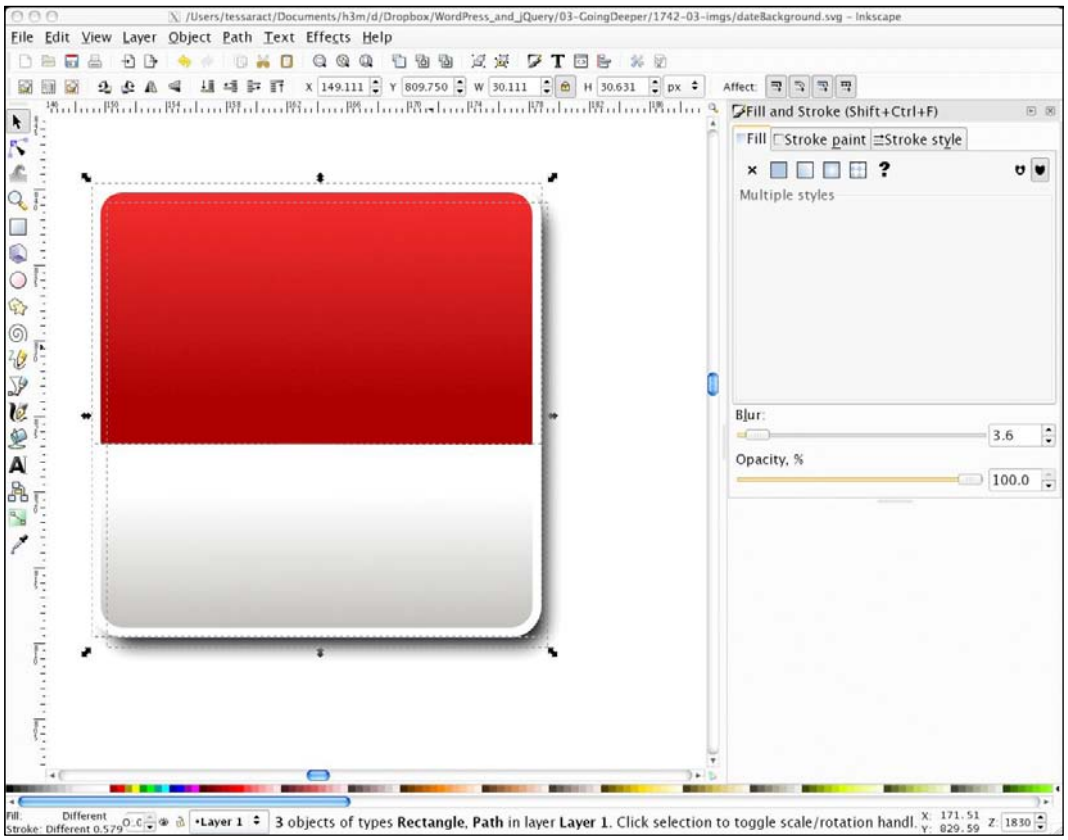
Granted, in *Chapter 2*, *Working with jQuery in WordPress*, I strongly advocated that you learn how to handle jQuery's selectors, inside and out, specifically so that you would be able to enhance any WordPress site without having to edit its theme. While you should know your jQuery selectors and filters like the back of your hand, it's not always the quickest or easiest approach. Sometimes, while you can select and edit anything that you want on the page, the jQuery selection process and statement chain is bloated; it could be cleaned up and trimmed down if only some element just had a specific HTML tag, `class` or `id` attribute. There will be lots of situations where being able to edit the theme directly will allow you to create your jQuery enhancements faster and with less code. Not to mention, many themes are great, but can usually be made a little better and more customized to your site with just the simplest theme tweaks. Let's do that now and take what we've just learned about themes and put it to use.

Now, the new Twenty Ten default theme we're using is great, but it would be better if the date was a bit more prominent in the posts and if the Sidebar was cleaned up to look more like "official" links, instead of just lists of bullets.

Changing the loop

Since we're touching up the theme, I want to change what the loop displays. We're going to assume this is a site for a client, and I know the client will *eventually* want to focus on the post's authors (there are many authors on this "hypothetical" site) and while the date is important, it shouldn't be on the same line as the author's name. I'm sure you've seen some blogs that have a little calendar or iCal-ish icons next to the post. I think that's a visually appealing way to display information like that, and not have the date take up a lot of room.

Using the free open source vector editor Inkscape (`http://inkscape.org`), I made a calendar background icon that can have the day's date up top in red and the three letter month below it. The icon is about 32 pixels square. You can use whichever graphic program you prefer, GIMP, Photoshop, Illustrator, and so on, to create a similar icon, or you can probably find a royalty-free image on the Web.



To get our calendar background behind our date and formatted properly, let's dig into the loop. The default theme's loop is located inside the template file called `loop.php`. This is a much longer loop than you may be used to if this is your first time working with the Twenty Ten default theme. Ultimately, we're interested in the "normal" or "everything else" view that is displayed on the site's "home" or default blog page. You'll find this code around line **127** starting with `<div class="entry-meta">`.

To start, comment out the custom PHP function `twentyten_posted_on` (it references a custom function in the theme's `function.php` file, getting into which is a bit beyond the scope of this title), and then add the following HTML markup and PHP template tags in bold:

```
...

<div class="entry-meta">
      <?php //twentyten_posted_on();//comment this out ?>
      <small class="date">
        <?php the_time('d') ?><br/>
        <span><?php the_time('M') ?></span>
      </small>
</div><!-- .entry-meta -->
...
```

What we're focusing on is the date display. The date is displayed with a template tag called `the_time` which has parameters set to display the full month, the day "as said", and the year; for example; February 4, 2010.

I just want to display the date's number and the three-letter abbreviation of the month underneath that. `the_time` tag's parameters don't really let me add HTML break tags, so I'll separate my date into two separate `the_time` tag calls so that I can control the HTML a little better. I'll also want to ensure my style only applies to this loop and not the `<small>` date and content that's wrapped in other template page's loops, so I'll be sure to add a custom `date` class to the `<small>` tag. I'll also wrap the year date display inside some `<span>` tags so that I can have some additional style control over it. My date display and classes end up looking like this:

```
...

        <small class="date">
          <?php the_time('d') ?><br/>
          <span><?php the_time('M') ?></span>
          <!-- by <?php the_author() ?>-->
        </small>
...
```

We'll then open up the CSS `style.css` stylesheet and add the rules for the special class name that we added to the date display, and modify the header display. I simply add my modifications to the very bottom of the `style.css` stylesheet. If on the odd chance, any of my style names are the same as anything already defined in the stylesheet, my rules will inherit from the previous rule and amend it (Either that, or make it blatantly clear that I need a more unique style name.)

First, I'll move the `h2` headers on the home page itself that are inside the `.post` class over 40 pixels, in order to make room for my date. Next, I'll move my date inside the `.post` class up about 25 pixels to have it sit next to the header. Within this rule, I also assign the `dateBackground.png` that I created in Inkscape and tweak the date number's size, color, and a few other properties a bit. Lastly, I set my month display size and color inside the span tag as follows:

```
...
/*----------twentyten chapter 3 customizations------------*/

.home .post .entry-title{
  padding-left: 40px;
}

.post small.date{
  display:block;
  background: url(images/dateBackground.png) no-repeat;
  margin-top: -25px;
  padding-top: 4px;
  width: 32px;
  height: 32px;
  font-size: 20px;
  line-height: 12px;
  text-align: center;
  color: #eee;
}

.post small.date span{
  font-size: 10px;
  color: #666;
}
...
```

And with that, the next screenshot shows what our post's headers and dates appear like now:

Not bad! Now, let's tackle the sidebar.

Changing the sidebar

The sidebar will be easy. The whole thing in the Twenty Ten default theme is widgetized, so any reordering that we want to do can be done through the administration panel. However, we do want to adjust the CSS of the sidebar's bulleted lists a bit. When amending a theme you didn't create yourself from scratch, it's always best to add new classes to the markup and stylesheet, rather than change or edit any of the original styles that the author put in. This just makes it easier to revert for various reasons. As you must have noticed earlier, I always add my new custom styles to the bottom of the `style.css` stylesheet.

Let's start off by opening up `sidebar.php` in our editor and just adding in a new class name that we can use to style any widgets that get loaded up into any of the widget areas. Wherever I find a `<ul class="xoxo">` tag, I'll just add an additional class called `.currentsidebar` after the .xoxo class. This appears twice in the `sidebar.php` file approximately around line **12**, and again, approximately around line **51**.

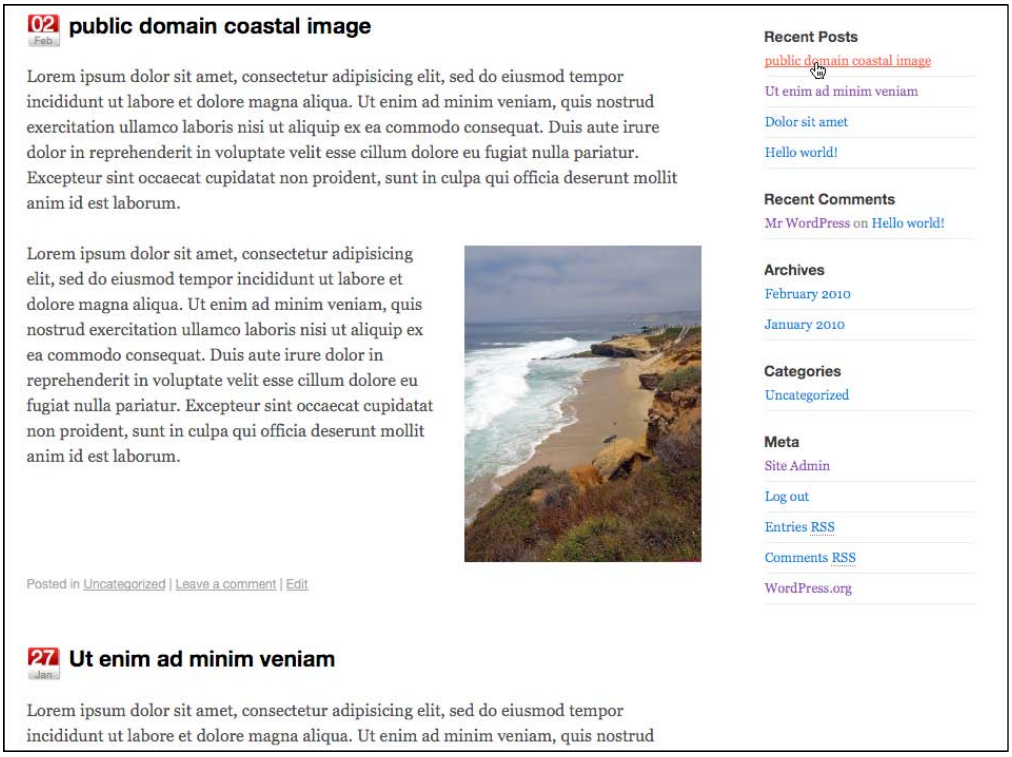
```
...
<ul class="xoxo currentsidebar">
...
<ul class="xoxo currentsidebar">
...
```

Next, we'll now simply open up our `style.css` stylesheet, and again at its bottom, let's write up our new `.currentsidebar` CSS rules to affect the list items:

```
...
.currentsidebar li{
    padding: 0;
    margin: 15px 0 20px 0;
}
```

```
.currentsidebar li ul li{
    list-style: none;
  padding: 5px 0; margin: 0 0 0 -15px; border-bottom: 1px solid #ddd;
  font-size: 105%;
}
...
```

Tada! As you can see in the next screenshot, our page and sidebar navigation now look like this:



As you can see, touching up a WordPress theme is easy. Not only can you customize your theme to look and work the way you want, you can imagine how easy it is to tweak the theme's HTML markup so that your jQuery enhancements are easier to add in. Next, let's move on to WordPress plugins.

The basics of a WordPress plugin

Now honestly, the details of writing WordPress plugins are far beyond the scope of this title; my goal is to show you the structure of a simple WordPress plugin and the basics of how to construct one. Understanding this, you can begin to write your own basic plugins and feel more confident looking through other people's plugins when assessing what kind of features they provide to your WordPress site and if you need to tweak anything for your jQuery enhancements. Even as simply and basically as we're going to work, you'll see how truly powerful WordPress plugins can be.

Want to become a WordPress plugin rockstar?

You can start off with, yet again, **WordPress 2.7 Complete** by *April Hodge Silver* and *Hasin Hayder*. There's a chapter on plugins that walks you through creating very useful simple plugins, as well as a more complex plugin that writes to the WordPress database. Beyond that, you'll want to check out **WordPress Plugin Development: Beginner's Guide** by *Vladimir Prelovac*. Don't let the title fool you, Vladimir will have you generating feature rich and dynamic WordPress plugins using WordPress' coding standards all explained with clear, step-by-step code.

Working with plugins does require some experience with PHP. I'll keep this explanation fairly straightforward for non-PHP developers, and those of you with PHP experience should be able to see how to expand on this example to your advantage in WordPress. On the whole, if you've been following the jQuery and WordPress PHP examples in this book so far, you should be fine.

Just as with themes, WordPress plugins require a little structure to get started with them. There's no defined hierarchy for plugin files, but you do need, at the very least, a PHP file with a special comment up top so that WordPress can display it within the Plugin Management page. While there are some single-file plugins out there, such as the Hello Dolly plugin, which comes with your WordPress installation, you never know when you first start developing, the ways in which a plugin may grow. To be safe, I like to organize my plugins into a uniquely named folder. Again, like with themes, WordPress relies on the plugin directory's namespace, so uniqueness is of key importance!

In the `wp-content/plugins` directory you can place a unique folder and inside that, create a `.php` file, and at the beginning of the file, inside the `<?php ?>` tags, include the following header information. Only the bold information is absolutely required. The rest of the information is optional and populates the **Manage Plugins** page in the Administration panel.

```
<?php
/*
Plugin Name: your WordPress Plugin Name goes here
Plugin URI: http://yoururl.com/plugin-info
Description: Explanation of what it does
Author: Your Name
Version: 1.0
Author URI: http://yoururl.com
*/
//plugin code will go here
?>
```

Make sure that you don't have any **spaces** *before* your `<?php` tag or after your `?>` tag. If you do, WordPress will display some errors because the system will get some errors about page headers already being sent.

Once you have your `.php` file set up in its own directory, inside your plugin directory, you can add a basic PHP function. You can then decide how you want to evoke that function, using an **action hook** or a **filter hook**. For example:

```
<?php
/*
Plugin Name: your WordPress Plugin Name goes here
Plugin URI: http://yoururl.com/plugin-info
Description: Explanation of what it does
Author: Your Name
Version: 1.0
Author URI: http://yoururl.com
*/

function myPluginFunction(){
  //function code will go here
}
```

```
add_filter('the_title', 'myPluginFunction');

//or you could:
/*add_action('wp_head', 'myPluginFunction');*/

?>
```

Remember that in the theme section earlier, I covered plugin hooks and how it's important to have them in your theme? This is why. If you didn't have `wp_head` or `wp_footer` in your theme, many plugins can't function, and you limit yourself to the plugins you can write. In my plugins, I mostly use `wp_header` and the `init` action hooks.

Luckily, most filter hooks will work in your plugins as WordPress will run through them in The Loop. For the most part, you'll get the most work done in your plugin using `the_title` and `the_content` filter hooks. Each of these filter's hooks will execute your function when WordPress loops through those template tags in the loop.

Want to know what filter and action hooks are available?

The list is exhaustive. In fact, it's so immense that the WordPress codex doesn't seem to have them all documented! For the most complete listing available of all action and filter hooks, including newer hooks available in version 2.9.x, you'll want to check out Adam Brown's **WordPress Hooks Database**: `http://adambrown.info/p/wp_hooks`.

Overwhelmed by the database? Of course, checking out Vladimir's **WordPress Plugin Development: Beginner's Guide** will get you started with an arsenal of action and filter hooks as well.

You now understand the basics of a WordPress plugin! Let's do something with it.

Project: Writing a WordPress plugin to display author bios

As we've discussed, plugins can help expand WordPress and give it new functionality. However, we've seen that adding jQuery scripts directly to the theme and editing its template pages here and there will do the trick in most cases. But let's imagine a more complicated scenario using our modified default theme and the hypothetical client mentioned in the previous project in this chapter.

While we tweaked the default theme, I figured that this client might want to have her site's focus be more journalism oriented, and thus, she'd want some attention drawn to the author of each post upfront. I was right, she does. However, there's a catch. She doesn't just want their WordPress nickname displayed; she'd prefer their full first and last name be displayed as well, as it's more professional. She'd also like their quick biography displayed with a link to their own URL and yet, not have that information "get in the way" of the article itself, or lost down at the bottom of the post. And here's the really fun part; she wants this change affected not just on this site, but across her network of genre-specific news sites, over 20 of them at last count (dang, I forgot she had so many sites! Good thing she's hypothetical).

For this specific WordPress site, it's easy enough to go in and comment out the custom function we dealt with earlier: add in `the_author` tag and display it twice, passing each tag some parameters to display the first and last name. I can also add a tag to display the author's biography snippet from the user panel and URL (if they've filled out that information). Also, it's certainly easy enough to add a little jQuery script to make that bio `div` show up on a rollover of the author's name. However, having to take all that work and then re-copy it into 20 different sites, many of which are not using the default theme, and most of which have not had jQuery included into their theme, does sound like an unnecessary amount of work (to boot, the client has mentioned that she's deciding on some new themes for some of the sites, but she doesn't know which sites will get what new themes yet).

It is an unnecessary amount of work. Instead of amending this theme and then poking through, pasting, testing, and tweaking in 20 other themes, we'll spend that time creating a WordPress plugin. It will then be easy to deploy it across all the client's sites, and it shouldn't matter what theme each site is using. Let's get started!

Coding the plugin

First up, looking through the client's network of sites, not many display the author's nickname or name. Only a handful do and of those, the name is listed unobtrusively. It will be much easier to have a plugin display the author's name and then comment out or remove `the_author` tag from a few themes.

Here's a quick detail to note: template tags don't do so well in plugins. This is because the template tag, which is a function, is set to display text, which, within another function, we don't really want. What we want to do is get the information and pass it to our hook, which will display it when the plugin function runs. Most template tags have comparable WordPress functions, which will only get the information and not write or display it immediately. For writing plugins, instead of looking through the WordPress Codex's **Template Tag** function list, I like to look through the **Function Reference**. Just about anything that starts with `get_` will work great in a plugin. For more details, have a look at `http://codex.wordpress.org/Function_Reference`.

The Codex Function Reference has `get_the_author()` which would suit some of my needs for this project, but I prefer to use a newer function that came about in WordPress version 2.8, called `get_the_author_meta()`. Unlike `get_the_author`, you can pass this function over 25 parameters to find out just about anything you care to on a WordPress user.

Given next is my plugin's base `addAuthor` function, followed by my `add_filter` hook which will run my function on every post's content. You can read the comments in bold for more detail:

```
...
//add author function
function addAuthor($text) {
  /*the $text var picks up content from hook filter*/
  //check if author has a url, a first name and last name.
  //if not, no "Find out more" link will be displayed
  //and just the required nickname will be used.
  if (get_the_author_meta('user_url')){
    $bioUrl = "<a href='".get_the_author_meta('user_url')."'>
                Find Out More</a>";
  }
  if (get_the_author_meta('first_name')
    && get_the_author_meta('last_name')){
      $bioName = get_the_author_meta('first_name').
        " ".get_the_author_meta('last_name');
  }else{
    $bioName = get_the_author_meta('nickname');
    }

  //check if author has a description, if not
  //then, no author bio is displayed.
  if (get_the_author_meta('description')){
    $bio = "<div class='authorName'>by <strong>".$bioName."</strong>
              <div class='authorBio'>"
                .get_the_author_meta('description')." ".$bioUrl."
              </div>
            </div>";
  }else{
    $bio = "<div class='authorName'>
              by <strong>".$bioName."</strong>
           </div>";

  }
```

```
    //returns the post content
    //and prepends the bio to the top of the content
    return $bio.$text;
}//addAuthor

//calls the post content and runs the function on it.
add_filter('the_content', 'addAuthor');
...
```

You'll note that in the previous code snippet I took some extra care to check if the WordPress user has a URL filled out in their profile, and if they've added in their first and last name as well as a bio description. If they don't, my plugin will merely display the user's nickname (the nickname is a required field) which is usually the same as the user's login name.

If any author doesn't have their first and last name, or a biography filled out, I'll leave it up to our client to force them to update their profile. In the meantime, the plugin won't display anything blank, empty, or broken, so no harm done.

Right now I'm just focused on getting the author's name and bio into WordPress, and now that the name and bio should be getting generated, I just want to make sure that the biography is styled nicely so that it stands apart from the post content but doesn't draw too much attention to itself.

To accomplish this, I'll add a stylesheet called `authover.css` to my plugin directory and add the following style to it:

```
.authorBio {
    border-top: 2px solid #666;
    border-bottom: 2px solid #999;
    background-color: #ccc;
    padding: 10px;
    font-size: 10px;
}
```

Now, the reason why I placed the CSS inside its own stylesheet instead of scripted as a string into the plugin as another function was mostly to illustrate the best practice of using the `wp_register_style` and `wp_enqueue_style` functions from the Script API. Just as using the `wp_enqueue_scripts` function helps us avoid conflict with other JavaScript and jQuery libraries, these functions register the new stylesheet and load it up, ensuring that there won't be any conflicts with other same-named stylesheets.

For a stylesheet I'm pretty sure it will be unique to my plugin, and even more, just for a single rule, this may be overkill, but you should be aware of this method, especially since you'll probably run into it looking through more robust popular plugins. Also, this makes the plugin more easily extendable in the future. You won't need to futz through your PHP string to edit or amend the CSS. In fact, if you were to write a plugin that had a lengthy enough stylesheet, you could hand the stylesheet over to a CSS designer while you focused on the PHP functionality. Not to mention, this makes your plugin useful to other users. A WordPress user with no PHP experience could download and install this plugin and easily edit its CSS stylesheet so that it looks good with their site's design.

Here's my `addCSS` function. Also, afterward instead of activating the stylesheet off a filter hook, I want the stylesheet to register and load as soon as WordPress loads up, even before the `wp_head` hook! Hence, you'll see that I've used the `init` action hook.

You'll note in addition to my comments in bold, the use of the `WP_PLUGIN_URL` variable. This is similar to the `TEMPLATEPATH` variable I showed you in the theme section to create a custom include, except of course, this works inside plugins to help WordPress dynamically find your plugin files without you hard coding them in.

Please read the bold comments in the next code block to understand what each code statement does:

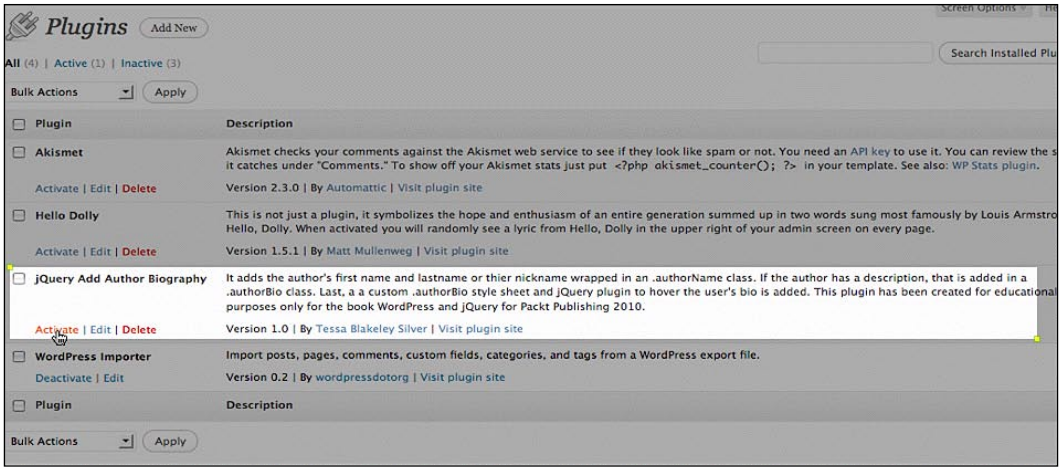
```
...
// Some CSS to position for the paragraph
function authorCSS() {
  //These variables set the url and directory paths:
  $authorStyleUrl =
    WP_PLUGIN_URL . '/add_author_bio-tbs/authover.css';
  $authorStyleFile =
    WP_PLUGIN_DIR . '/add_author_bio-tbs/authover.css';
  //if statement checks that file does exist
  if ( file_exists($authorStyleFile) ) {
    //registers and evokes the stylesheet
      wp_register_style('authorStyleSheet', $authorStyleUrl);
      wp_enqueue_style( 'authorStyleSheet');
    }

}

//evoke the authorCSS function on WordPress initialization
add_action('init', 'authorCSS');
```

OK! That should do it. We now need to activate our plugin and check it out in WordPress.

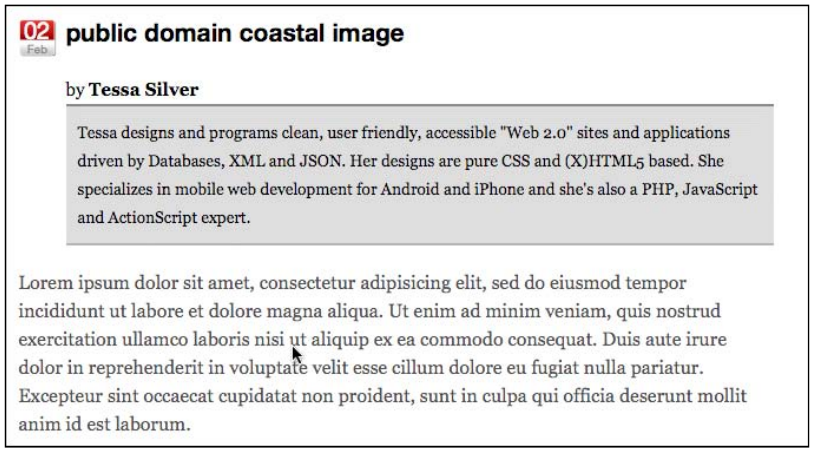
Activating our plugin in WordPress

Our plugin is already in the WordPress `wp-content/plugins` directory. That means all we have to do is navigate over to our **Manage Plugins** page and activate it.

The plugin called **jQuery Add Author Biography** in the `Plugin Name:` space in the code's comment header appears in the plugins table as shown in the next screenshot:



Once the plugin is activated, we can navigate to the site to see it in action:



It's working! The theme, which does not have `the_author_meta` tags in it, is now displaying the author's full name and bio description underneath it. The biography description is styled using the CSS rule in our plugin's class.

You've now edited a theme by hand and further extended the site by creating a WordPress plugin from scratch. Great job! But what's that you say? You were expecting to do a little more jQuery? You're right. Let's enhance this site a little further by creating a jQuery plugin.

The basics of a jQuery plugin

We'll discover that compared to WordPress themes and plugins, jQuery plugins are actually not that complex.

To set up a jQuery plugin, you need to follow jQuery's **plugin construct**. The basic construct consists of setting up a jQuery function for your plugin as follows. Note the bold `.fn` added to the jQuery object. This is what makes your function a jQuery function.

```
jQuery.fn.yourFunctionName = function() {
  //code
};
```

Within that, it's best practice to then add a return `this.each(function(){...});` so that your function will run through each item in the jQuery selector.

```
jQuery.fn.yourFunctionName = function() {
  return this.each(function(){
    //code
  });
};
```

Unlike WordPress, which requires specifically formatted comments in theme CSS stylesheets and in plugin headers, jQuery does not require a commented-out header, but it's nice to add one up top.

```
/*
You can name the plugin
Give some information about it
Share some details about yourself
Maybe offer contact info for support questions
*/
jQuery.fn.yourFunctionName = function() {
  return this.each(function(){
    //code
  });
};
```

Note that each function and method you wrap your plugin in and use inside your plugin *must* end in a ";" semicolon. Your code may otherwise break, and if you ever compress it, it will definitely break.

That's it, all that's required of a jQuery plugin. Now, let's dive in to enhancing the output of our WordPress plugin with a jQuery plugin.

Project: jQuery fade in a child div plugin

Taking the required jQuery function discussed previously, I'm going to write up a basic function, which can be passed not only to the main jQuery wrapper selection, but an additional selector parameter so that it's easy to target the child `div` of a selection, or the specific parameter of the jQuery selection that's passed the parameter.

Again, note the bold comments in my `authorHover` function to follow along:

```
...
//sets up the new plugin function: authorHover
jQuery.fn.authorHover = function(applyTo) {
  //makes sure each item in the wrapper is run
  return this.each(function(){

  //if/else to determine if parameter has been passed
  //no param, just looks for the child div
  if(applyTo){
    obj = applyTo
  }else{
    obj = "div";
  }

  //hides the child div or passed selector
  jQuery(this).find(obj).hide();

  //sets the main wrapper selection with a hover
  jQuery(this).css("cursor", "pointer").hover(function(){

      //restyles the child div or passed selector
      // and fades it in
  jQuery(this).find(obj).css("position","absolute")
    .css("margin-top","-10px").css("margin-left","-10px")
    .css("width","400px")
    .css("border", "1px solid #666").fadeIn("slow");
      }, function(){
```

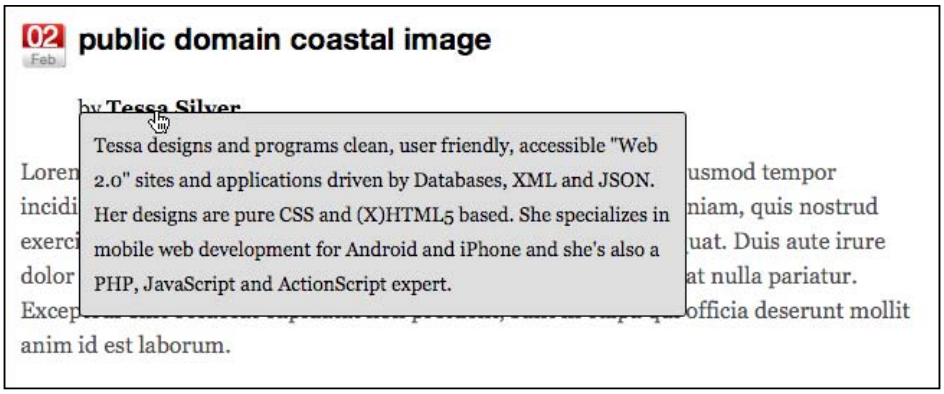
```
            //fades out the child selector
            jQuery(this).find(obj).fadeOut("slow");

        });
    });

};
```

That's all it takes. Now that we've created a jQuery plugin script, let's quickly test it out in our theme first. All we have to do is embed our new jQuery plugin named `jquery.authover.js` to our theme, *under* the `wp_enque_script` call, *below* the `wp_head` hook and evoke it with a simple script:

```
...
<script type="text/javascript">
jQuery(function(){
  jQuery(".authorName").authorHover();
});
</script>
...
```

We can take a look at the results in our site. I've grabbed two screenshots so that you can see the fade-in effect. In the following screenshot you can see the new `div` start to fade in:

In this next screenshot you can see the completed fade animation:



Extra credit: Adding your new jQuery plugin to your WordPress plugin

Now you're free to go and install your WordPress plugin and include jQuery plugin on as many sites as needed! However, in case you're wondering, yes, we can refine the installation process a bit more and just incorporate this jQuery plugin inside our WordPress plugin.

The first step is to simply drop our `jquery.authover.js` script inside our plugin directory and then use the `wp_enqueue_script` to evoke it. You'll want to pay particular attention to this use of the `wp_enqueue_script` function, as it will also include jQuery 1.4.2 IF its NOT already registered in the theme or plugin! This means that client's sites, which don't already have jQuery included, don't need to worry! Just installing this plugin will automatically include it!

```
...
function addjQuery() {

    wp_enqueue_script('authover',
     WP_PLUGIN_URL . '/add_author_bio-tbs/jquery.authover.js',
     array('jquery'), '1.4.2' );
}
...
```

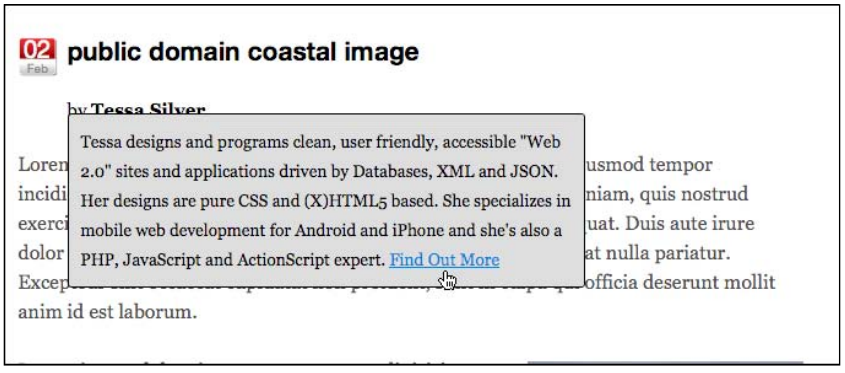
We'll then add a function to our WordPress plugin which writes in the jQuery script that uses the `authorHover` function of the plugin. Normally, it would be better, and it is recommended to load up all scripts through the `wp_enque_script` function, but for very small scripts that are so customized, you're sure will not ever conflict, and you know jQuery is already loading in properly (as we are with the plugin), if you want, you can just hardcode script tags like so:

```
...
function addAuthorHover(){
  echo '<script type="text/javascript">
jQuery(function(){
  jQuery(".authorName").authorHover();
});
</script>';
}
...
```

Lastly, we add the action filters which will evoke those functions:

```
...
add_action('init', 'addjQuery');

add_action('wp_head', 'addAuthorHover');
?>
```

Now, if you remove your jQuery plugin from your theme and make sure that your plugin is activated, you should see the exact same results as before! In the next screenshot, you'll notice that I've added a URL to my profile, and now the **Find Out More** feature set to degrade nicely if no URL was present, just automatically works. Wonderful.



Putting it all together: Edit the theme or create a custom plugin?

We've learned in this chapter how easy it is to edit a theme, create a WordPress plugin, and a jQuery plugin. For the majority of your WordPress development work, adding jQuery enhancements right to the theme will do the trick. If you feel your jQuery scripts are a bit cumbersome and you're allowed to edit the theme (assuming of course, you don't break the layout or dramatically alter the look) you'll probably find that being able to wrap WordPress content in custom HTML tags with special `class` or `id` attributes is a huge help and time saver.

This chapter's project example's "hypothetical client request" also showed that if there's any chance that your work can or will be reused or deployed across multiple individual WordPress installations, you should consider encapsulating the work in either a jQuery plugin, a WordPress plugin, or as we discovered, both.

In addition to considering if your work will need to be reused or deployed, you may also want to consider the lifespan of the jQuery enhancement and that of the WordPress theme. It's easy to think that the jQuery enhancement is really more a part of the theme as it visually affects it, but is it really? I've found that more often than not, a large part of my WordPress and jQuery development seems to center around encapsulating jQuery development into a WordPress plugin, or making WordPress plugins more effective with jQuery.

As there are only two ways to include jQuery into a WordPress site, through the theme, or a plugin, if you're at all comfortable with editing and creating plugins, you'll probably start to find that its the better way to go (sure, there are always exceptions). Enhancing WordPress plugins with jQuery and even encapsulating jQuery plugins in WordPress plugins will allow you to easily scale your theme design and any jQuery functionality/enhancements independently of each other.

This approach comes in very handy if you do like to redesign or update your theme a lot, or perhaps you have a client who's a little "theme swap happy". If you want to keep the cool jQuery enhanced forms, image and gallery lightboxing, and various other functionality, or even just "neat eye candy" that you've created for a site, without having to manually update a new theme constantly with all of that over and over again, creating a plugin is the way to go, be it for jQuery, WordPress, or both.

Ultimately, it's up to you and your comfort level, and what's best for the project, but I've found, with a few exceptions, which we will cover examples of in later chapters, that trying to keep most jQuery enhancements from being embedded in the WordPress theme has served me well.

Summary

You should now understand the following:

- What WordPress themes, WordPress plugins, and jQuery plugins are.
- How to edit a theme and create your own basic WordPress and jQuery plugins.
- Best practices for knowing when to edit and customize a theme, or make a WordPress plugin, a jQuery plugin, or all three!

Armed with this information, we're going to move on to the next chapter where we'll take a look at using a jQuery plugin with a plug-n-play WordPress plugin. We will also discuss enhancing and expanding the capabilities of WordPress plugins with jQuery. Get ready to dazzle with lightbox modal windows and wow users with easy-to-use forms.

4

Doing a Lot More with Less: Making Use of Plugins for Both jQuery and WordPress

At this point, you understand enough about jQuery and WordPress basics—as well as the different ways to integrate them together—that you can start to get truly creative in your solutions. In this chapter and the following three chapters, we're going to roll up our sleeves and work out solutions for some often requested projects and start getting jQuery to do some useful and just plain cool work within our WordPress sites.

We're going to bring all available components of WordPress and jQuery together. In this chapter, we'll be:

- Working with the very robust and popular jQuery plugin, ColorBox, by *Jack Moore* of Color Powered.
- We'll also work with the robust and popular WordPress plugin, cforms II, by *Oliver Seidel* of Deliciousdays.
- We'll then customize our default theme so that it works seamlessly with cforms II and ColorBox, giving our site a seamless event registration experience.
- We're not done! We'll then enhance cform II's already great validation with jQuery for a smooth user experience.

Get ready to put your WordPress site to work!

The project overview: Seamless event registration

While we will continue to work with the default theme, we're going to imagine a different hypothetical client and scenario for this chapter's jQuery enhancement.

In this scenario, the "client" is a not-for-profit/awareness group. They've created an **Events** category in their WordPress site and whenever a new event is planned, it is up to each event's coordinator to post information about their upcoming event to the Events category.

Most of their events are free but very disorganized as it's up to each coordinator to decide how they want to accept registration for an event, through e-mails or phone calls. People get confused and e-mail the wrong people on the site, and then there's no reliability of who's coming to what events so that the organization's leaders can gather stats from busy event coordinators in order to keep track of how effective the events are for their cause.

The good news is, we can still help them fix all this.

What the "client" wants

After sitting down and discussing all the options, ultimately, they want one, simple registration form that can have the event name passed to it, and then e-mailed on to the event administrator, who will dole the RSVPs out among the various event organizers.

They've also received feedback by registrants who have complained that the event's publish date confuses them: They don't register for events because, unless the coordinator makes the date bold or places it inside the title, it looks like the event is happening on that day, or has happened in the past. Because of this, the client would like their event posts template restyled and cleaned up a bit so that it's easier to recognize them as events and not the same as other posts on the site.

Last, and most importantly, they've been really impressed and influenced by the feedback and other forms they've seen on several sites lately, and would really like it if their registration form opened up in a modal box so that people can register for an event while staying on the **Events** page. When they're done registering for an event, they can continue browsing the **Events** category and easily register for more.

Part 1: Getting everything set up

Luckily for us, with a little WordPress and jQuery knowledge under our belt, this task is not as complicated as it sounds. In the last chapter, I extolled the virtues of keeping design and functionality separate and wrapping your jQuery enhancements in WordPress plugins. I also mentioned the fact that there are always exceptions. Well, here's a scenario where we'll be inclined to apply our enhancements directly to the theme for a couple of reasons:

- We'll already be tweaking the theme to create a custom category page for events
- And, we'll also need to create a custom page template for the form that can load into a modal box without reloading the rest of the site's headers and footers

Because these requests require that the client understand they'll need to take care if they ever want to update or swap out their theme, we might as well leverage the full power the WordPress theme API can provide us for this enhancement.

What we'll need

Let's start with the main parts of the enhancement: We'll need a form with e-mail capability and a modal box to load it in. The rest we'll do with a few custom jQuery scripts and customizations to the WordPress theme.

ColorBox

For the modal box, there are several good jQuery plugins. You've probably heard of ThickBox which is very good, but I myself prefer ColorBox for several usage and aesthetic reasons.

You can download the jQuery ColorBox plugin from here:
`http://www.colorpowered.com/colorbox/`.

Why ColorBox and not ThickBox?

The ThickBox plugin comes bundled with Wordpress and I was a big ThickBox fan, yet, I also preferred the simplicity of jQuery LightBox (jQuery LightBox only works with images). I was quite impressed when I came across ColorBox, there are a few reasons for this:

- Both ThickBox and ColorBox plugins offer modal windows for more than just images.
- You can call up inline elements, external files, and iFrames as well as basic AJAX calls. No problems at all.

However, ColorBox, compared to ThickBox has a few advantages. For ColorBox, *Jack Moore* really took some time and effort to come up with five, very slick styles for the modal window, and a nice set of callback and extension functions for the more advanced developer to take advantage of. Second, all image loading for the modal window components (the transparent background, close, **Next**, and **Back** buttons) is handled entirely in the stylesheets, so it's very easy for a designer to custom-style the modal window. On several occasions, I've had to hand edit the `thickbox js` file in order to get the images to load properly if they were not relative to the `plugins js` file the way ThickBox intended or if I needed to add a new image.

Last, ThickBox relies on you hand adding the `.thickbox` class attribute to elements you want to launch the modal window from. While this approach works great for web developers who have the luxury of handcoding everything up, it's a pain for implementing inside a publishing platform such as WordPress for less technical users. Having to coach (and coax) clients through setting their Administration panel editor to **HTML** and custom adding `class` attributes to their markup is just painful. None of that is necessary for ColorBox. It is easily controlled with jQuery selectors, so the theme or plugin developer can take care of selections based on WordPress's HTML output allowing the content editors to simply focus on their content without any HTML understanding.

Cforms II

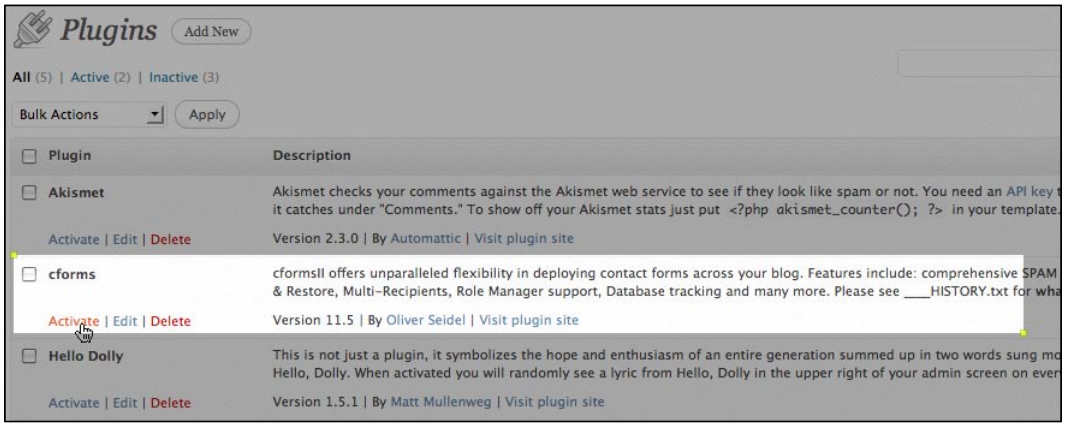
To create the registration form we have numerous WordPress plugins to choose from, but I find the best to be **cforms II**. Cforms II states upfront: "Admittedly, cforms is not the easiest form mailer plugin, but it may be the most flexible." And they're right. And after working with it just once, you'll find that it's much easier than you'd think.

You can download the WordPress cformsII plugin from here:
`http://www.deliciousdays.com/cforms-plugin/`.

Installing the WordPress plugin

Once you've downloaded the cforms II plugin, you should follow the instructions provided by the developer for installing and activating them.

Essentially, this entails unzipping the package, placing `cforms` directory in your WordPress installation's `wp-content/plugins` directory, and then navigating to the Administrator's **Manage Plugins** page. You'll then select **Activate** for the plugin.

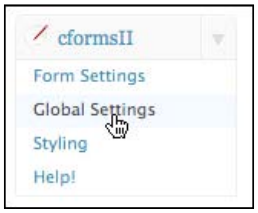


Once you have the plugin installed and activated, let's get to work with it.

Setting up the registration form with cforms II

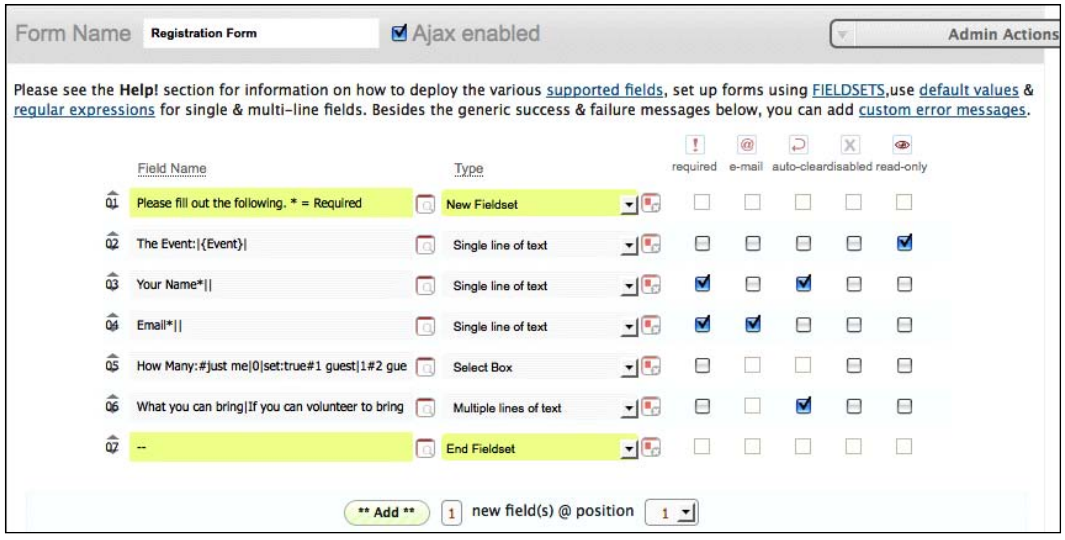
The cforms II plugin offers many powerful form-building features for the not-so-technical WordPress administrator. As a more technical WordPress developer, it flat out saves you tons of time. The cforms administration interface does take a while to get used to, but it is by far the most powerful and flexible form plugin I've used with WordPress.

CformsII is a complex plugin that requires a large amount of administration real estate, and several screen pages. For this reason, once you activate the plugin, you'll find a whole new panel available on your left-hand side Administration area.



Out of the box, cformsII allows for the use of AJAX, in that it will submit and update the form without reloading the page. It also allows for the very easy creation of all basic types of form elements: `input`, `select`, `check`, and `radio` boxes as well as `textarea` boxes. You can wrap form elements in custom `fieldset` tags with `legend` text tags for easy grouping of related form elements.

Powerful server-side validation is built right in. It's very easy to assign fields to be required and check for valid e-mail addresses. Beyond that, you can also easily assign your own, custom regular expressions for custom validation. The cforms **Help!** panel even gives you helpful examples of regular expressions that you can use.



Setting up cforms II securely

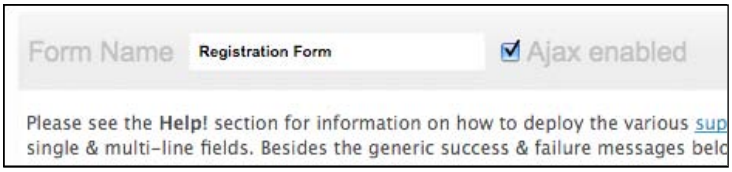
You'll want to read up on the cforms documentation, and if possible, have a chat with the site's Admin or Hosting provider's tech support. You'll want to take care to go through the **Global Settings** panel and make sure your installation of cforms II is as secure as possible for your needs of the forms.

If at all possible, try to use Captcha fields to reduce spam and turn off file upload capabilities if you don't need them.

Striking a balance: Forms should be short and easy and yet retrieve useful information

The goal of a form is to strike a balance between gaining as much information from the user as possible without making them feel as though you're asking for too much personal information or, if nothing else, boring them with the tediousness of filling out too many fields.

I've kept this registration form very short and to the point. First, I filled out the form name and set it to **Ajax enabled**. This will help us out as the page will be loading in a modal box, so when it's refreshed it won't pop out into a new page, outside the modal window.



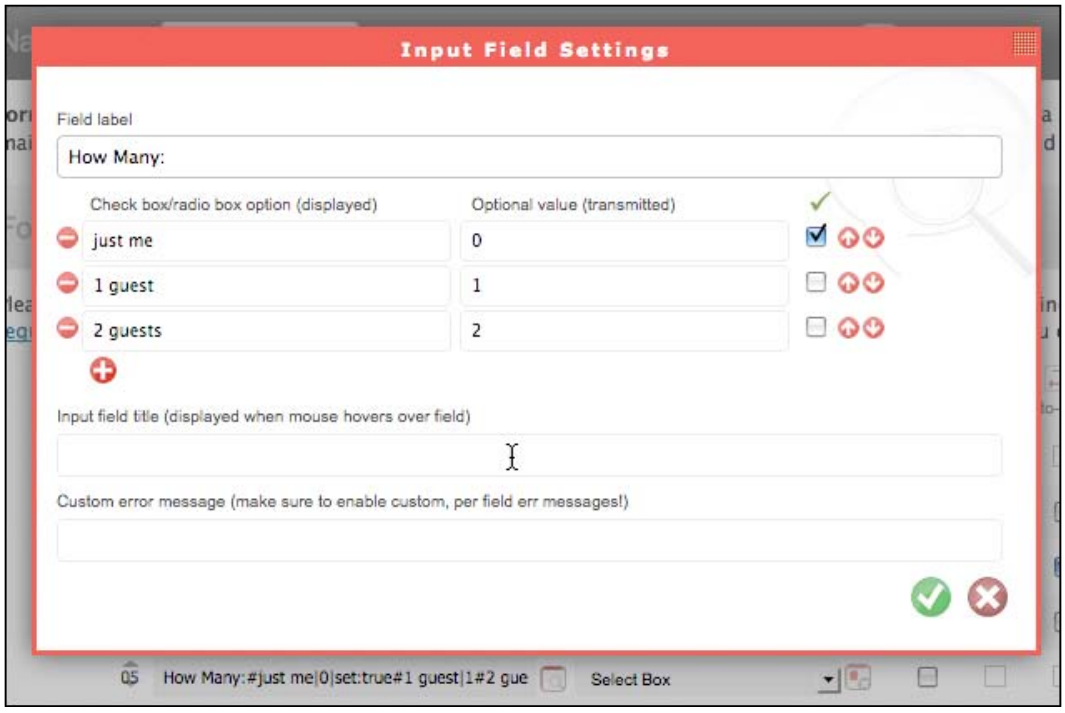
Next, using the default form set that's already provided, let's set up a `fieldset` with `legend` text and the five form inputs that the event coordinators need to know for planning the events.

First up the event, that is, the name of the event will be passed by the post and not filled out by the user, but I want to show it and it needs to be in a form element to be e-mailed to the administrator.

After the event field, we'll need to ask for the user's name. As no money is being exchanged and this form is more to have a "head count", I've left this to a single field. It's a required field, but I'll allow the user to be as casual or formal as they please.

Next, we'll ask for an e-mail address. This is required and I've opted for server-side validation using the checkboxes to the right. If a user suggests they can bring something to the event, the event coordinator may want to reply to them and get back in touch with them. Also, updates about the events may need to be broadcast to registrants, so a valid e-mail is essential.

Now, let's set up a select box to ask how many guests a registrant may bring to the event.



Last, the message area is a text area that has some text suggesting the registrant offer to bring something and if they can, to state what they can bring in the message area.

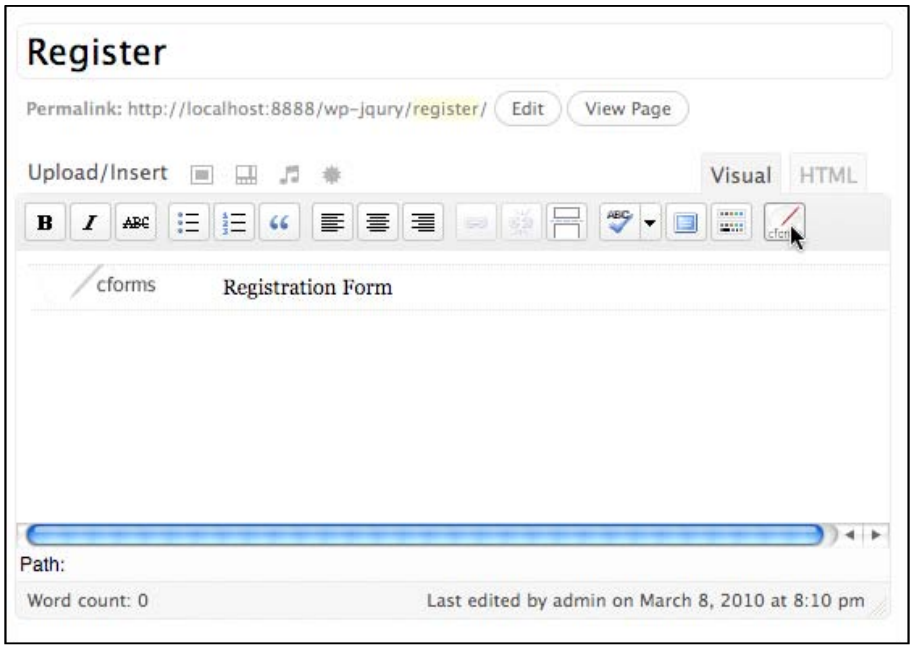
OK. So there's our form. In order to view it, it now needs to be placed in a WordPress page or post. We'll be placing it in it's own page, which means we should create that page in WordPress.

Creating the register page using WordPress 3.0's custom menu option

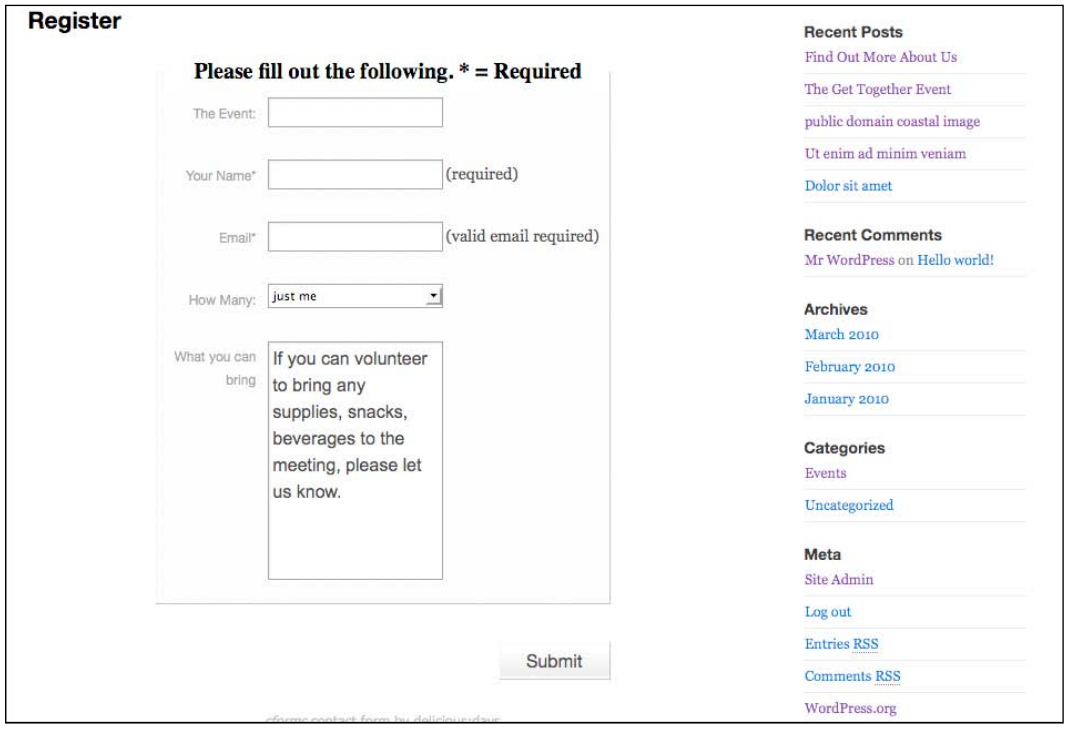
If you look at the Pages column on the left-hand side of your Administration panel, you'll now see that the pageMash plugin is part of the options.

We'll simply select **Add New** from the option in the left-hand menu under **Pages** and create a page named **Register**. We'll leave the content area blank, but you'll notice now, in **Edit** view, there's a **cforms** button in the editing panel for the page/post.

Clicking on to that button will allow you to select the form you'd like placed on the page (you can create multiple forms in cforms II and even place multiple forms in a single post or page, but that's overkill for our purposes). Once you select your form, you should see a placeholder for it.



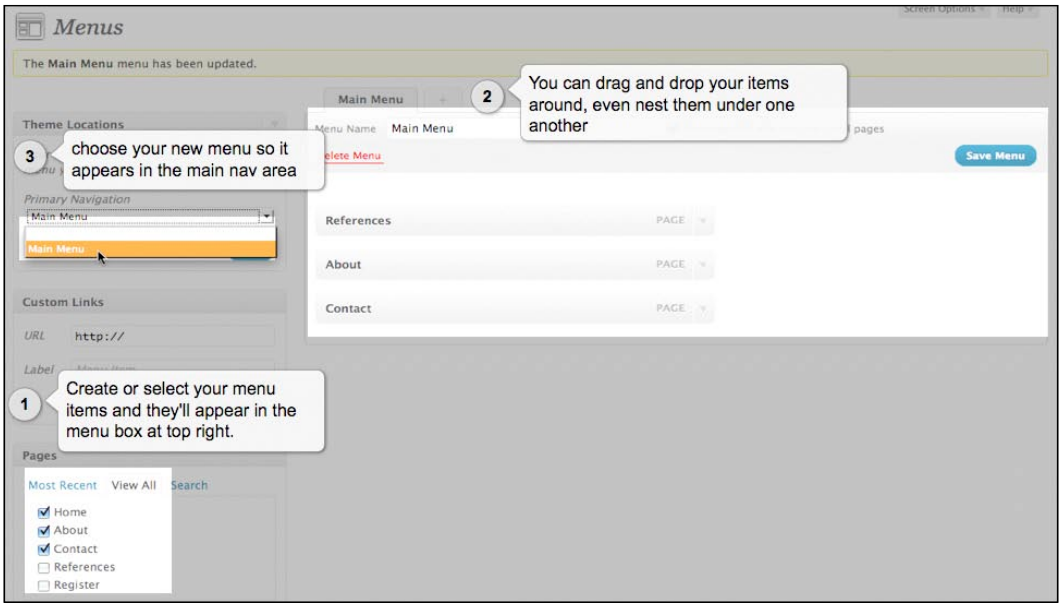
You should now see your form in the **Register** page on your site, as shown in the following screenshot:



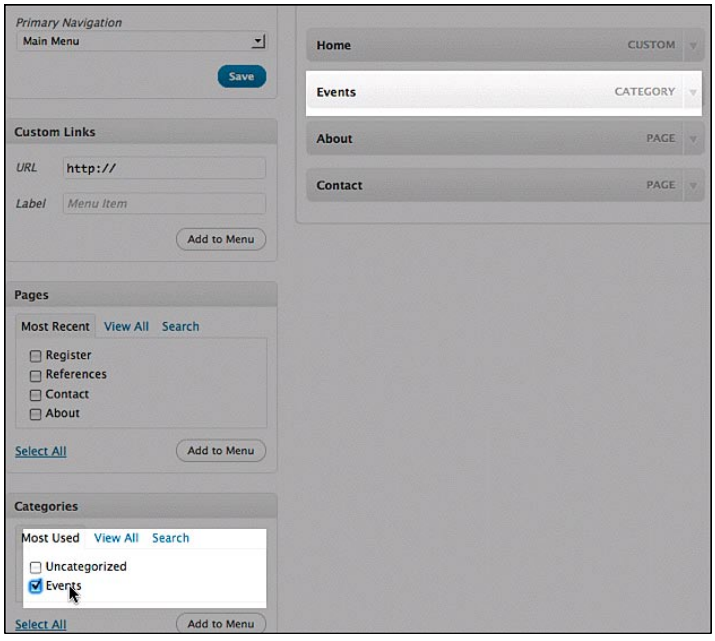
Working with WordPress 3.0's custom menu option

However, we don't want the **Register** page to show up in our Page navigation and we need it to be in its own template page so that it will load more nicely into our modal box, without the theme's header and footer graphics and styling. We'll need to modify our template, but first, let's create a custom menu in WordPress 3.0 that will override the **Page** menu and then we can easily specify what pages should show up so the registration page won't appear in our site's navigation.

First up, you'll want to navigate to **Appearance | Menus** in the Administration panel. Once there, you can click on the + (plus) tab to create a new menu and then select options from the right to add to it. You can then, in the upper-right panel set the menu to be your "primary navigation", which will overwrite the standard Page navigation in the Twenty Ten default theme's header. The following screenshot illustrates the three main steps to set up a new main navigation and assign it as the site's primary navigation:



You can also include the **Event** category into the menu as shown in the following screenshot (we'll be needing access to this page later):



OK! We now have a "hidden" page holding our registration form. Let's get started with the theme customizations.

Customizing the theme

Again, we'll need to customize the theme in two ways: First, we want a custom page template to hold our registration form that will load into the modal box and second, we'll need to create a custom category template and modify it so that it will display only posts assigned to the **Events** category and launch the modal box with the registration form in it.

Creating the custom page template

First up, we'll need to create a new page template that we can assign our registration page to. We'll start by creating a copy of our **page.php** template and renaming it **registration-page.php**.

The whole point of this form is to load in the ColorBox modal window, so having our theme's header and footer styling will be a bit distracting. We'll simply remove the `get_header()` and `get_footer()` WordPress template tag commands from this template page.

Next, while we don't want the header and footer styles, we do need the page to be a properly formatted HTML page that loads in jQuery. We'll manually add a doctype and borrow some of the WordPress header code from the `header.php` file, just from a `body` tag to the beginning of this template's loop, as shown here:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" <?php language_
attributes(); ?>>

<head profile="http://gmpg.org/xfn/11">

<meta http-equiv="Content-Type" content="<?php bloginfo('html_type');
?>; charset=<?php bloginfo('charset'); ?>" />

<title><?php wp_title('&laquo;', true, 'right'); ?> <?php
bloginfo('name'); ?></title>

<link rel="stylesheet" href="<?php bloginfo('stylesheet_url'); ?>"
type="text/css" media="screen" />
<link rel="pingback" href="<?php bloginfo('pingback_url'); ?>" />
```

```
<?php wp_enqueue_script("jquery"); ?>
<?php wp_head(); ?>

<style type="text/css">
<!--

.cform fieldset{
  border:1px solid #036;
}

.success{
  font-size: 140%;
  font-weight: bold;
}

-->
</style>
</head>
<body>
...
```

You'll notice that we've simplified it a lot compared to the `header.php` file's `head` tag code. We don't need to worry about comments or a sidebar on this page, so those bits of PHP WordPress code have been removed. We do need jQuery to load in and I've also gone ahead and added a few manual styles the cforms use to spruce up our form a bit.

We'll then add this new footer markup; just the closing body and HTML tags just below the template page's loop:

```
...
<?php wp_footer(); ?>
</body>
</html>
```

Don't forget about your plugin hooks when customizing template pages

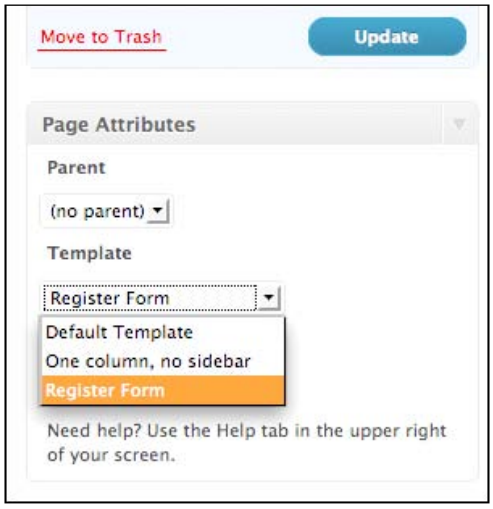
Note, I made sure that the `wp_head()` and `wp_footer()` were manually placed in our registration template page. Many plugins require those hooks to be in a theme to work. If you're creating custom pages, make sure that they be included in the header or footer or that you place them in manually if the theme's `header.php` and `footer.php` files won't be included, as this template page does.

Last, for this new page to be recognized as a special template for WordPress, we have to add a **template header** to the very top of the document in commented out PHP as shown here:

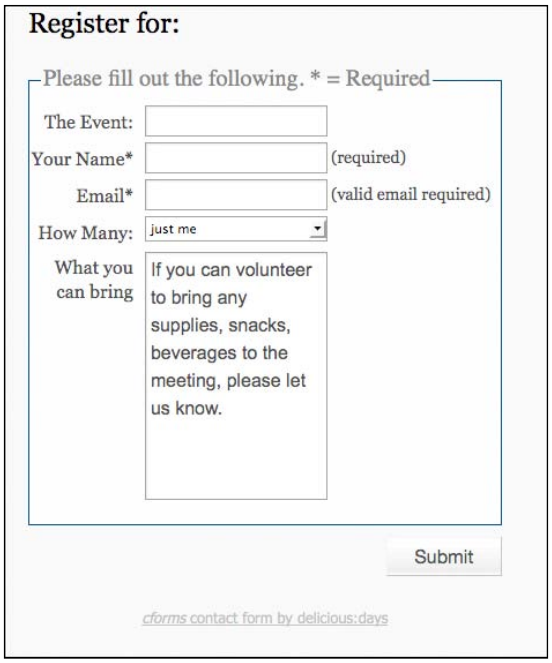
```
<?php
/*
Template Name: Register Form
*/
?>
...
```

As with our plugin in *Chapter 3, Digging Deeper: Understanding WordPress and jQuery Together*, make sure there are no spaces or hard returns before the `<?php` tag. Otherwise, you may get an error about headers already having been sent.

Now, if we return to our **Registration** page in the Administration panel, on the right-hand side, we'll see that our new page template is available to be assigned to the page.



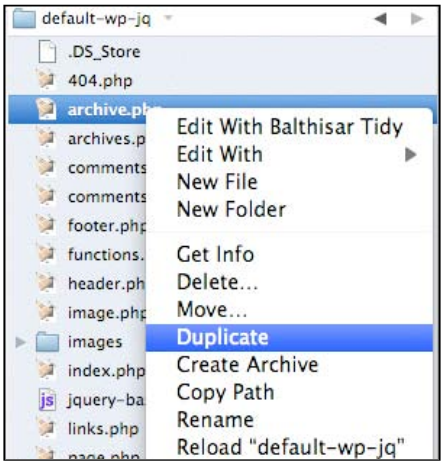
We can now see that if we use the browser's address bar to navigate to the URL of our **Register** page, it loads up without any other WordPress styling and is ready to be loaded into our ColorBox modal window.



That's the first half of our solution. Let's now finish it.

Creating the custom category template

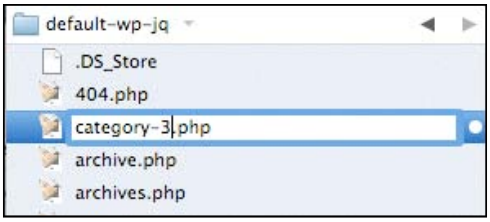
We now need to create a special category template for the **Events** category. Again, we'll want each event to have a registration link. That link will pass the event's title to the registration form.



To get started with this, if you'll recall the Template Hierarchy from *Chapter 3, Digging Deeper: Understanding WordPress and jQuery Together,* the `category.php` template page trumps the `archive.php` template page. Turns out, the default template that we're using doesn't have a `category.php` page. We'll simply create one by duplicating the `archive.php` page and naming it `category.php`.

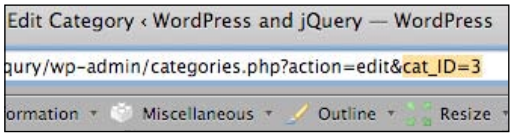
But wait; we don't just want this page to show *any* category. We want to only show the Events category. You'll also recall from *Chapter 3, Digging Deeper: Understanding WordPress and jQuery Together*, that you can further trump the `category.php` template with a specific `category-ID.php` template page such as **category-3.php**.

It just so happens that in my local setup of WordPress, the ID for the Event category is 3, so that's what we'll name the file.



Finding your category ID

Those of you working in your own WordPress installation, the category ID numbers are assigned as you create categories. First, you'll need to determine what your **Events** category's ID number is. You can do this by navigating to the **Posts | Categories** page in the Administration panel and selecting **Edit** under the **Events** category. You can then check the URL in that category's **Edit** page. At the very end of the URL you'll see the category's ID number.



We're now ready to customize the `category-3.php` template for the **Events** page.

First up, as we saw in the previous chapter, this page calls the `get_template_part( 'loop', 'category' )` function from the `loop.php` template page. We actually want a custom, very simple setup for the **Events** category *only*. While we certainly could update the `if...else` statement of `loop.php` with an additional custom loop (which is a tad beyond the scope of this book, stay tuned for the upcoming **WordPress 3.0 Theme Design** book from Packt!), for this custom client project we'll just comment out the call to that loop and add in our own very simple loop that, thanks to the Template Hierarchy, will only work on our **Events** category page.

```
<?php
//start the loop:
while (have_posts()) : the_post(); ?>
    <div <?php post_class() ?>>
        <h2 id="post-<?php the_ID(); ?>" class="entry-title">
          <a href="<?php the_permalink() ?>" rel="bookmark"
              title="Permanent Link to
              <?php the_title_attribute(); ?>">
              <?php the_title(); //adds the title ?></a></h2>

        <div class="entry">
          <?php
            //add the content
            the_content() ?>
        </div>
        <?php //add the registration button ?>
        <p><a class="register"
          href="/wp-jqury/register/?evnt=<?php the_title(); ?>">
          Register</a>
        </p>

        <div class="register-separate"></div>

      </div>

    <?php endwhile; ?>
```

If you haven't done so, you might want to deactivate your **Add Author Bio** plugin that we built in the previous chapter. It's not necessary for this project, though it doesn't hurt to have it activated (it will just be sitting there in the Event's posts).

Notice that in the loop towards the bottom, we made a `link` tag that references the register form. I've ammended a **variable string** named `evnt` to that link and added the title of the event using the `get_title()` template tag as shown:

```
...
<p><a class="register"
  href="/wp-jqury/register/?evnt=<?php the_title(); ?>">
  Register</a>
</p>

<div class="register-separate"></div>
...
```

We'll now go back up to the top of the template page and make sure that the header title makes sense. There's a bunch of `if...else` statements at the top checking to see if the content returned is from an archive, category, or tag (remember, this page was duplicated from the default template's `archive.php` page). As this template page will only load up Event posts now, we don't really need all that PHP, but it doesn't hurt. Mostly, we'll want to add in a main header with the following markup and code:

```
...
<h1 class="pagetitle"> Upcoming Events <?php single_cat_title(); ?></
h1>
...
```

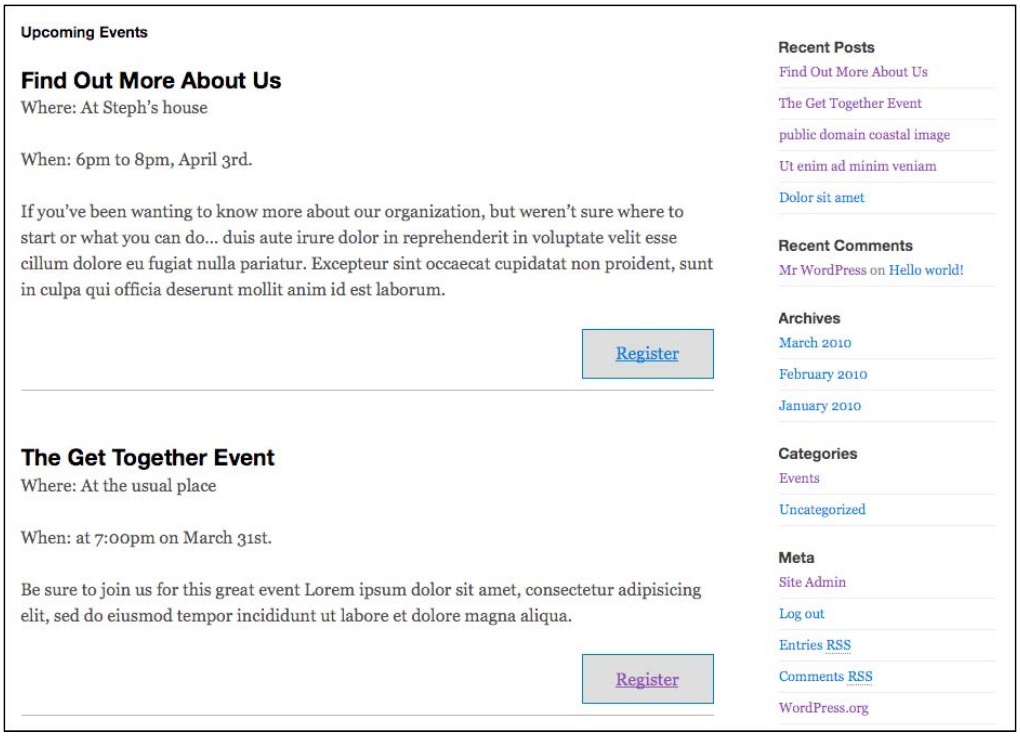
That will give us a title named **Upcoming Events** on the page (or whatever you actually named your own **Events** category, that is, Shows, Parties, and so on—you get to name it. That `single_cat_title()` template tag will pull it in for you).

At the very bottom of the theme's `style.css` sheet, we'll add the following rules to style our registration link and float it to the right:

```
...
.register {
  display:block;
  background-color: #ccc;
  border: 1px solid #069;
  width: 100px;
  padding: 10px;
  text-align: center;
}

p a.register{
  float: right;
}
```

```
.register-separate{
  clear:both;
  padding-top: 10px;
  border-bottom:1px solid #999;
}
```

When we look at an event post now, we'll see our Event post with a dynamic link at the bottom to **Register**:



Getting jQuery in on the game plan

Alright! I don't know about you, but I feel that was quite a bit of prep-work. It's all going to come together now as we load up the ColorBox plugin and cook up a few final custom jQuery scripts.

Including the ColorBox plugin

In our theme, let's create a `js` directory, and within that directory, let's create an additional directory named **colorbox**. This will allow us to place in the CSS sheet and image collateral for the ColorBox plugin and keep everything tidy and working the way it prefers to work in the **colorbox.css** sheet.

We'll unzip our ColorBox plugin and move over the minified version into our `js/colorbox` directory. We'll then take the stylesheets and collateral from the `example1` folder (I like it the best, the striped, transparent background and rounded corners are great), and drop them into the **colorbox** directory. We'll then go to our theme's `header.php` file and include both the **colorbox.css** stylesheet, underneath the theme's main stylesheet as shown:

```
...
<link rel="stylesheet" type="text/css" media="all"
  href="<?php bloginfo( 'stylesheet_url' ); ?>" />

<link rel="stylesheet" href="<?php bloginfo('stylesheet_directory');
?>/js/colorbox/colorbox.css" type="text/css" media="screen" />

...
```

Then, above the `wp_head` function, we'll add in our main jQuery include as well as the ColorBox plugin using the methods that we learned in the previous chapters, taking advantage of the script API as shown:

```
...
wp_enqueue_script( 'jquery' );
  wp_enqueue_script('colorbox', get_bloginfo('stylesheet_directory') .
'/js/colorbox/jquery.colorbox-min.js', array('jquery'), '20100516' );
...
```

Writing a custom jQuery script

Now, in the root of our `js` directory, let's create a new `custom-jquery.js` file and also be sure to include it in our `header.php` file, *under* our ColorBox includes as follows:

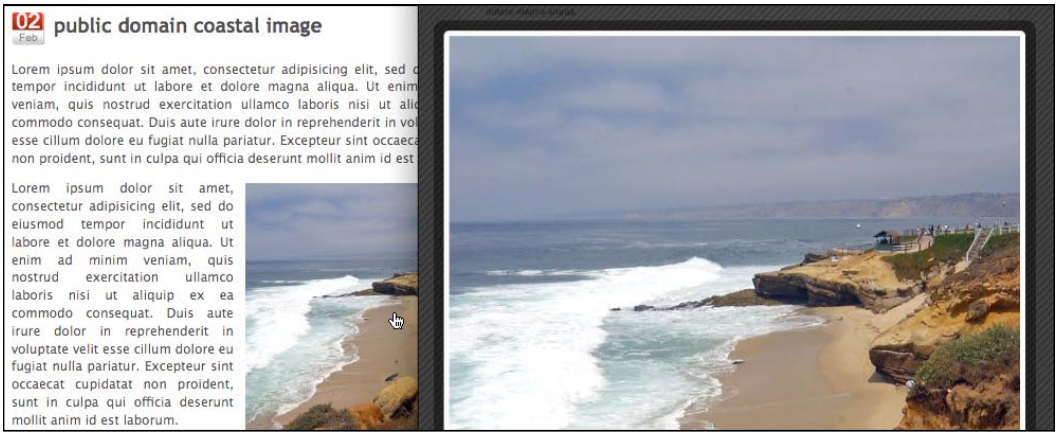
```
...
wp_enqueue_script('custom-jquery', get_bloginfo('stylesheet_
directory') . '/js/custom-jquery.js', array('jquery'), '20100510' );
...
```

Get set for some jQuery fun now. Since we've gone through the trouble of hand including the ColorBox plugin into our WordPress theme, we might as well make sure it can load up images in addition to our registration form.

To ensure that ColorBox only loads up images, and not every link on the page, we'll think of some examples back to *Chapter 2*, *Working with jQuery in WordPress*, and do a little creative selecting. We'll add this rule to our `custom-jquery.js` file:

```
jQuery(function(){
  jQuery(".entry-content a:has(img)").colorbox({height:"98%"});
});//end docReady
```

This selection only works on tag links that are in posts, inside the `.entry` class, that *have* thumbnail `img` tags. No other links will activate ColorBox.



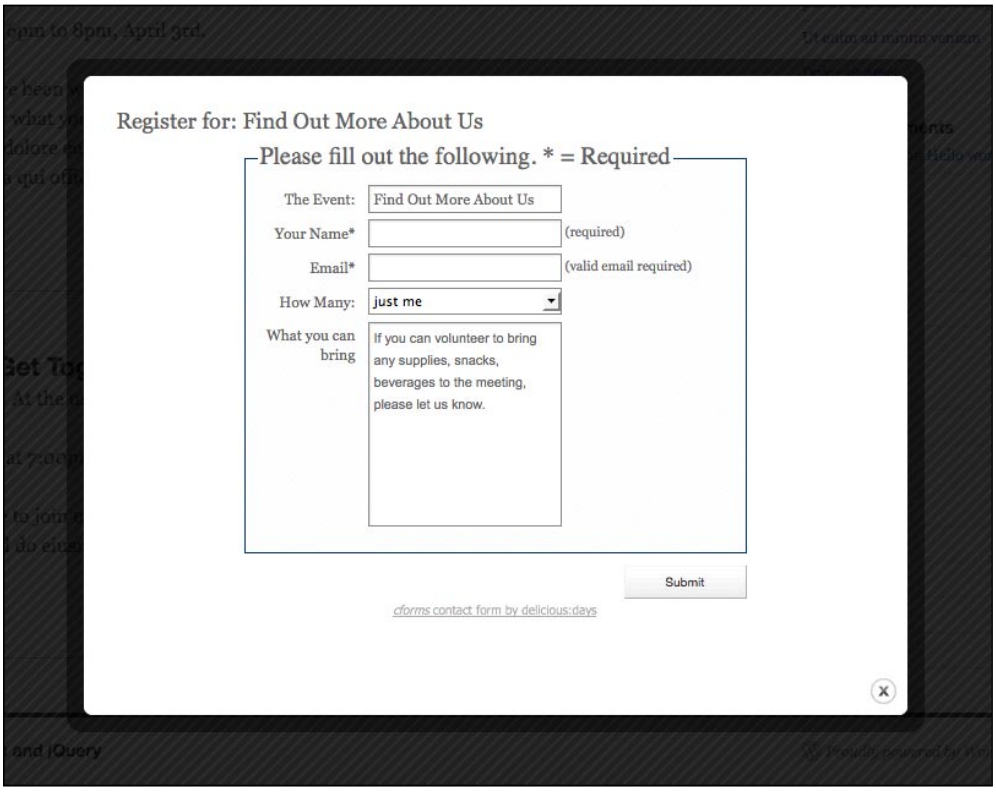
Now, to get the registration page to launch ColorBox, we'll be a bit more clever. While I've added a `.registration` class, I don't want to rely on it to launch ColorBox. The only link that should trigger the modal window is a link to the registration form, so I'll be sure to select for that. In my `cb-registration.js` file, inside my document ready function and after my image select `colorbox` function, I'll add this script:

```
...
jQuery("a[href*='register']")
  .colorbox({iframe:true, width:"500px", height: "600px"});
...
```

That particular jQuery selector will make sure that again, only links that contain (that's what the asterisk `*` is for) the word `register` in the `href` attribute will trigger the ColorBox modal window, in ColorBox's iframe mode.

You'll notice that I also used ColorBox's available parameters to set a constrained height and width for the registration form's modal box.

Now, only our Registration links and image links with thumbnails launch ColorBox:



Pulling it all together: One tiny cforms II hack required

You'll recall that we set up our registration link to pass the name of the event in the URL via a **GET call**, sometimes called a **variable string**, to the Registration form.

Right now, there is no way that the cforms administration panel will pick up that variable, but there's a quick, and fairly commonly used "hack" for cforms to make sure you can pass your own custom variables to the form.

cforms II creates little-bracket-enclosed template names for the variables it can pick up, like adding the variable template:`{Title}` will display post or page title that the form is on. We want to pass it a post title from another page (rather than having to manually add this form to every Event post), so we'll add our own variable template to the cformsII plugin.

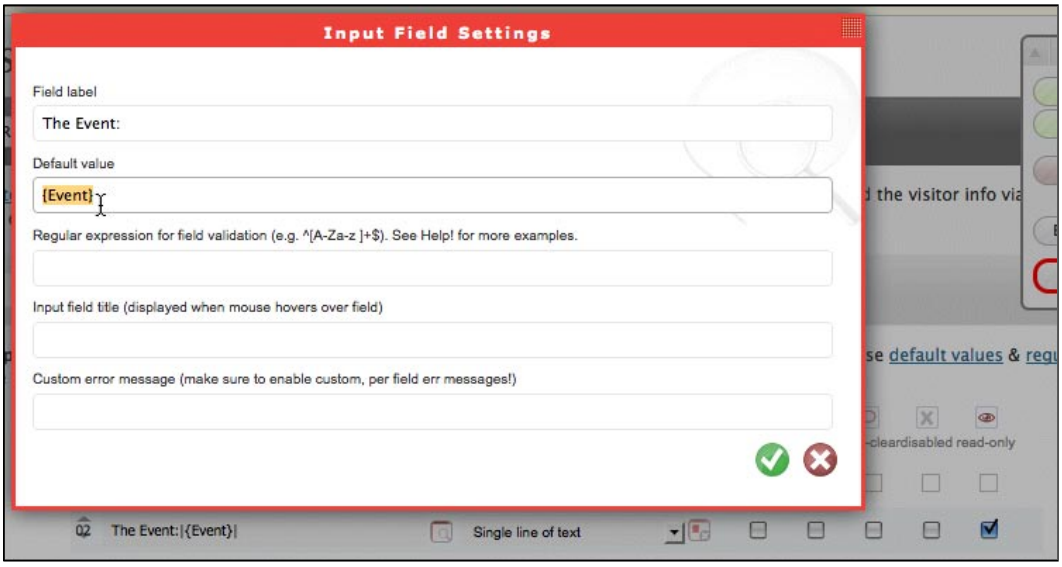
Customizing a plugin? Keep copious notes!

WordPress and jQuery developers usually keep updating and refining their plugins. Your best bet is to try to find a solution that doesn't require you to edit the actual plugin source files. However, as in this case, if you find you do edit a plugins source file, add your own `customization-readMe.txt` file to the directory and keep detailed notes of what you amended or edited in the plugin. When the developer releases and upgrades their plugin, especially WordPress plugins that sometimes need updating to keep up with the current WordPress core version, you'll lose your amendments and hacks when you update the plugin. Your notes will make it easy to reintegrate them.

In the cforms II plugin directory, locate the `lib_aux.php` file. Around line 421, just after code that looks like `...$m = str_replace( '{BLOGNAME}',...` add this line of code:

```
...
$m = str_replace( '{Event}', esc_attr($_GET['evnt']), $m );
...
```

Then, in the cforms administration panel for my Registration form, we can now add the **{Event}** variable to the `Event` field that I added to the `lib_aux.php` page in the plugin. Let's also make sure the field is set to "read only".



Just for clarity, I'd like the event name to show up in the header of the form as well. The header is not part of cforms, but part of the page template. In my theme directory, I'll open up `registration-page.php` and next to the header's `the_title()` template tag on line 41, I'll add the following code:

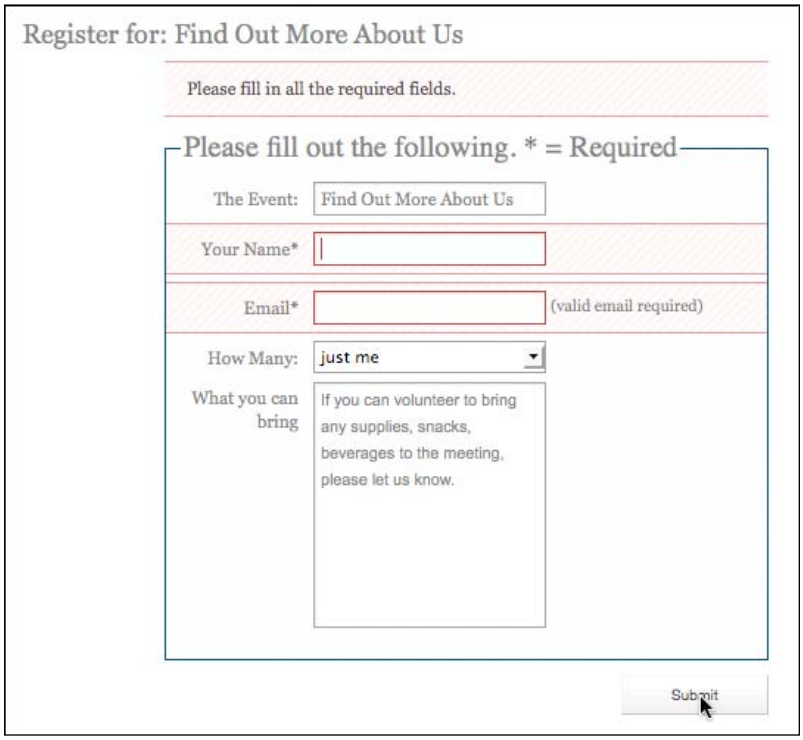
```
...
<h2><?php the_title(); ?> for: <?php $evnt = esc_attr($_GET['evnt']);
echo $evnt;?></h2>
...
```

When the form launches, you'll now see the name of the event in the header and in the Event field, which is set to read only and not editable by the user. Now when the form is submitted and e-mailed to the administrator, it's clear what event the registration is for.

We now have an Event page that shows the user's upcoming events and lets them seamlessly register for those in a form that loads in a modal window as planned. Great job! Let's see about making this experience even better.

Part 2: Form validation—make sure that what's submitted is right

The great news is, cformsII provides nifty, awesomely CSS styled, server-side validation already built-in and ready to go. You can see if I click on **Submit** on my form without filling out the required details, or an improperly formatted e-mail address, the form reloads showing me the fields that are incorrect.



But why wait until the user clicks on the **Submit** button? While **server-side validation** is essential and the only way to properly validate data, by adding in a little **client-side** validation, with jQuery, we can easily enhance and speed up the user's process by alerting them as they work through the form that they're missing details or have data improperly formatted.

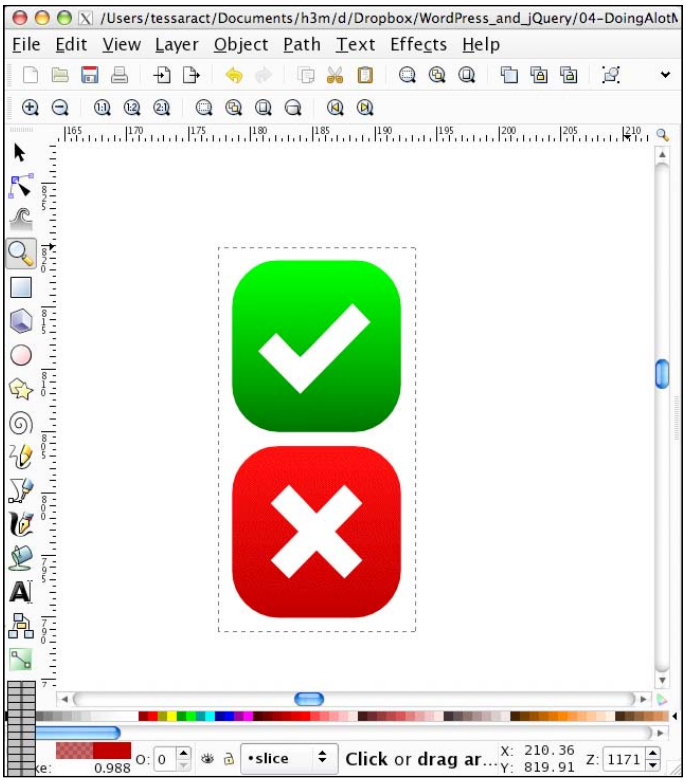
Why is server-side validation important?

Client-side validation with JavaScript and jQuery should never be relied on for data validation or to prevent improperly formatted information from being submitted and sent to the server. Users can always disable JavaScript in their browser and then submit any values they want (sometimes using improperly formatted values to hack into your server through the form). Client-side validation, for this reason, should only be used to *enhance* the user's experience and not actually protect the server or data integrity.

The trick to client-side validation: Don't just tell them when it's wrong!

Everyone responds to positive feedback. Instead of waiting for your users to mess up or forget a field, with the use of jQuery and a few styles you can let them know that they filled the field out correctly and are ready to move on.

Using Inkscape, I've made a simple little "check" and "x" set of icons that can be applied as a background image to a span added by jQuery. Using the CSS sprite image technique of adjusting the background position to display the "check" or the "x" icons, the user will visually see quickly if the form field is correctly filled out and that it's OK to move on.



Blank input validation

In order to set up this basic validation, we'll write up a jQuery script that selects for the input items and on `blur`, sets off a function. Let's place the script in the `registration-page.php` just below the loop code, above the `wp-footer()` hook, as shown (note the bold comments in the code to follow along with what each jQuery statement does):

```
...
jQuery(".cform :input").blur(function(){

  /*this "if" makes sure we don't target the submit button or email
field*/
  if (jQuery(this).val() != "Submit") {

    /*this "if" targets only empty fields*/
    if (jQuery(this).val().length == 0) {
       jQuery(this).after('<span class="wrong"> ! </span>');
    }else{
```

```
        /*"else" otherwise field is fine*/
            jQuery(this).after('<span class="correct"> thanks. </span>');
        }//end if no length

      }//end ifelse !submit

      });//end blur function
    ...
```

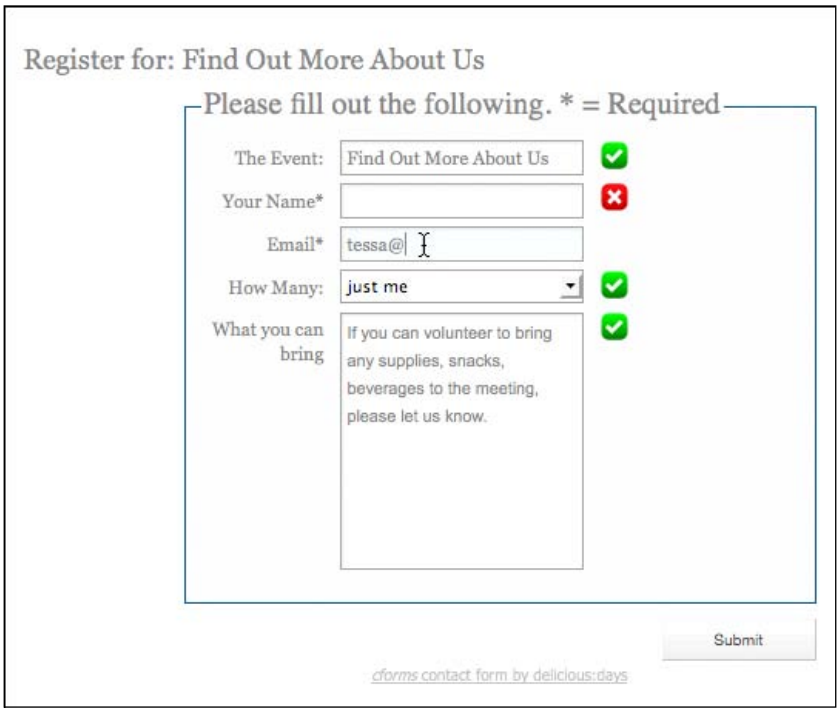
The previous code appends an exclamation point (`!`) for an invalid, empty field, or a quick `thanks.` for a valid, filled-out one. However, as the user focuses and blurs the input fields, the spans keep getting appended with the `after` function. To compensate for that, we'll add a custom script that works on `focus`, just underneath our `blur` script. It will remove the `after` appended spans as shown:

```
...
jQuery(".cform :input").focus(function(){
     jQuery(this).next("span").remove();
});//end focus function
...
```

This gives us some very nice, basic validation that checks for blank inputs. You'll note that our `span` tags have classes amended to them. I've added the "check" and "x" images to my theme's image directory, and now, at the very bottom of my theme's `style.css` stylesheet, I'll add the following class rules:

```
...
/*for registration form*/
.wrong{
  display:block;
  float:right;
  margin-right: 120px;
  height: 20px;
  width: 20px;
  background: url(images/form-icons.png) no-repeat 0 -20px;
  text-indent: -3000px;

}

.correct{
  display:block;
  float:right;
  margin-right: 120px;
  height: 20px;
  width: 20px;
  background: url(images/form-icons.png) no-repeat 0 0;
  text-indent: -3000px;

}
```

The end result is a pretty nice, obvious visual display of what happens when you mouse or tab through the fields, but leave the two required fields blank, before ever clicking on the **Submit** button.



Properly formatted e-mail validation

Let's just take this one small step further. It's one thing to leave the e-mail address blank, but we might as well point out if it's not well formed. Steve Reynolds, has an excellent how-to post on his site about using jQuery to validate an e-mail address. You can read up on it here: `http://www.reynoldsftw.com/2009/03/live-email-validation-with-jquery/`.

Steve's code demonstration is particularly interesting and worth a look at, as he uses jQuery's `keyup` function to check validation against the e-mail expression in real time.

For our purposes, we'll be borrowing Steve's regular expression function and fitting it into the validation check we've already started, which works on the `blur` function.

First up, below our existing script, we'll add in Steve's `isValidEmailAddress` function as follows:

```
...
function isValidEmailAddress(emailAddress) {
           var pattern = new RegExp(/^(("[\w-\s]+")|([\w-]+(?:\.[\
w-]+)*)|("[\w-\s]+")([\w-]+(?:\.[\w-]+)*))(@((?:[\w-]+\.)*\w[\w-
]{0,66})\.([a-z]{2,6}(?:\.[a-z]{2})?)$)|(@\[?((25[0-5]\.|2[0-4][0-
9]\.|1[0-9]{2}\.|[0-9]{1,2}\.))((25[0-5]|2[0-4][0-9]|1[0-9]{2}|[0-
9]{1,2})\.){2}(25[0-5]|2[0-4][0-9]|1[0-9]{2}|[0-9]{1,2})\]?$)/i);
           return pattern.test(emailAddress);
  }//is valid e-mail
...
```

Next, we'll take a close look at our existing script. What we want to do, is after checking for a value of nothing (`val().length == 0`), we'll double-check that the input field is not the email field.

Using Firefox and Firebug, I explored the DOM and discovered that the email field form has a unique class named as `.fldemail`.

We'll place our new statement as an extension of our current if statement with an `else if` statement before our general `else` statement.

Our updated `blur` script now looks like this (note the new email validation, `if...else` statement in bold):

```
...
jQuery(".cform :input").blur(function(){

  /*this if makes sure we don't target the submit button or email
field*/
  if (jQuery(this).val() != "Submit") {

    /*this "if" targets empty fields*/
    if (jQuery(this).val().length == 0) {
       jQuery(this).after('<span class="wrong"> ! </span>');

    /*This "else if" targets if the field is the email field*/
    }else if(jQuery(this).hasClass("fldemail") == true){

        var email = jQuery(this).val();

        /*Run's Steve's function and return true or false*/
        if(isValidEmailAddress(email)){
```

```
            //This shows the user the form is valid
            jQuery(this).after(
              '<span class="correct"> thanks. </span>');

          }else{
            //This shows the user the form is invalid
            jQuery(this).after('<span class="wrong"> ! </span>');
          }//if...else

        //end email check
        }else{
          /*otherwise field is fine*/
            jQuery(this).after('<span class="correct"> thanks. </span>');
        }//end if no length

      }//end if...else !submit

      });//end blur function
    ...
```

We can now not only check for empty fields, but also check for a valid e-mail address on `blur` of a field input:

Validation tip: don't go overboard!

The cforms II plugin server-side validation is more than adequate. Again, we're just trying to speed things along with a little client-side validation, and not frustrate our users because we've created a bunch of strict formatting rules for data. Some people may have phone numbers, or zip codes that are a little differently formatted than you would expect, and for most intents and purposes, this is OK. Your best bet is to use your jQuery validation to prompt hints and inline help and guide the user, rather than force them to comply with exact data formatting.

Final thoughts and project wrap up: It's all about graceful degrading

As with everything you do with jQuery, you need to keep in mind that you're creating useful enhancements that are great to have, but if for some reason a user didn't have JavaScript enabled or available, the process or site won't break.

Our client is very happy with our seamless registration solution. Going through the registration process with JavaScript disabled, the registration process does work just fine using the standard browser back keys. The only drawback I find is that the registration form loads up *outside* of the WordPress theme, which is what we had to do so it would load in nicely into the ColorBox modal window.

On the whole, I don't think this is that big of a problem. Going through my various website stats, I'm hard-pressed to find a visitor who didn't have JavaScript enabled. The two or three who didn't were probably in text-only browsers, so a lack of WordPress theming would probably not be noticed at all (in fact, it's probably nice for disabled users using text-to-speech browsers, not having to wade through the header info to get to the form).

Because we're always thinking of hypotheticals in this title, if by some chance, the client ever decided they'd like the form to work outside of ColorBox within the WordPress template in the event JavaScript was disabled, I've come up with the following solution:

First, you'd need to load the form into *two* WordPress pages. One named `register`, as we've done with the special template and another one named `register-b` (that's just the permalink slug, the header could still say **Register** on both pages). For the `register-b` page, you would not assign the special template; you'd leave the **Page Template** as **Default Template**. You can place a cform on to as many pages and posts as you like, so having this form in two places definitely won't be a problem.

Next, you'll go into the `category-3.php` Events template page and change the link to call the alternative, default themed page as follows (note the bold `-b` is the only difference from our original link):

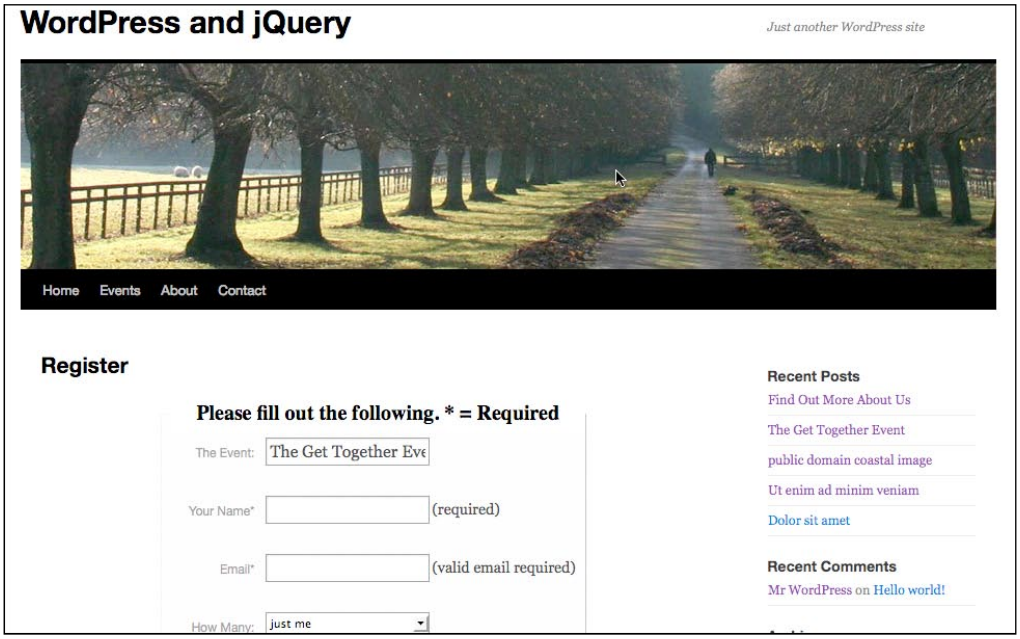
```
...
<p><a class="register" href="/wp-jqury/register-b/?evnt=<?php the_
title(); ?>">Register</a></p>
...
```

Last, in your `custom-jquery.js` file, you'll simply create a jQuery script that will rewrite that `href` link to the modal page form by removing the `-b`. Be sure to place this script *before* your colorBox function scripts, just to make sure the `href` transforms before setting up the colorBox functions.

```
...
jQuery("a[href*='register']").each(function(){
  this.src = this.src.replace(/register\-b/, "/register/");
});
...
```

If JavaScript is enabled, jQuery will change all the register `href` instances and the whole process will flow as planned using the ColorBox plugin. If not, the user will register using the standard WordPress theme form and not be any-the-wiser.

As you can see in the following screenshot, the form would just load in as part of the site if JavaScript were disabled:



Summary

We've now learned how to:

- Really leverage a theme to aid in jQuery enhancements
- Enhance the very robust cforms II WordPress plugin with the jQuery ColorBox plugin and a few custom scripts

And this was just one of many ways to achieve this particular solution! As the aim of this book is using jQuery within WordPress, I went down a route that focused more on jQuery and accessible WordPress features. But sure, we could have plugin-ized the ColorBox plugin; we could have plugin-ized the whole thing! Or made a plugin that just extended the cforms plugin. The list of solution strategies is almost endless.

Again, you'll need to look at each project and assess it accordingly. Coming up in the next chapter, get ready to bust out the "eye candy" with some slick HTML and CSS-based chart animation as well as image gallery slideshows and rotators, and a few other clever ways to catch your user's attention.

5
jQuery Animation within WordPress

We're going to continue to build on our knowledge of jQuery and WordPress while delving deeper into animation using jQuery. Animation is one of jQuery's strong suites and while you may eschew animation as frivolous or a cheap trick, just for "eye candy", it can be very useful when properly implemented.

jQuery animation of CSS properties, colors, and interface elements can ensure that users clearly see alert, error, and conformation messages. Animation also enables interface objects to fade and slide in and out of view for a better user experience. Last but not least, a little "eye candy" certainly never hurt a site's interest and popularity level with users.

In this chapter we will be using animation to:

- Grab your user's attention and direct it to alerts
- Save space and animate through a series of rotating sticky posts
- Create some slick, animated mouse-over effects and easy animated graph charts

Let's get started applying useful, high-end animations to our WordPress site.

jQuery animation basics

To start off, we already have a little experience with jQuery animation. Let's refresh: In *Chapter 2*, *Working with jQuery in WordPress*, in the *Events and effects* section, we learned about the following functions: `show()`, `hide()`, `fadeIn()`, `fadeOut()`, `fadeTo()`, `slideUp()`, `slideDown()`, and `slideToggle()`. I had also mentioned the `animate()` and `stop()` functions.

We've already worked with several of these functions in our previous projects in *Chapter 2*, *Working with jQuery in WordPress*; *Chapter 3*, *Digging Deeper: Understanding jQuery and WordPress Together*; and *Chapter 4*, *Doing a Lot More with Less: Making Use of Plugins for Both jQuery and WordPress*, particularly, `show()` and `hide()`, as well as `fadeTo()` and `slideToggle()`. As we've seen, a very large portion of your animation needs are easily met with these shortcut functions, though at the same time, limited by them. Let's now take a closer look at the `animate()` function and pick up some fine grain control over our jQuery animations.

CSS properties made magical

The `.animate()` function allows you to animate any *numerical* CSS property. Pixels `px` are the understood norm for most numerical property values, but you can specify `em` and `%` (percentage) units. Pretty much anything you can place in the handy `.css()` function, can be used in the `.animate()` function.

Additionally, rather than numeric values, you can add the shortcut strings `"show"`, `"hide"`, and `"toggle"` to any property. They will essentially take the value from 0 to 100, or vice versa, or toggle from 0 or 100 to the opposite number for you.

Let's take a look at a quick example of this clever function. Remember, you'll want to place any jQuery scripts you write inside the document ready function: `jQuery(function(){//code here});` also inside `<script>` tags, so that your jQuery will launch when the DOM has finished loading:

```
...
jQuery('.post p').animate({ fontSize: '140%',
  border: '1px solid #ff6600',}, 3000);
...
```

This snippet will animate all `.post p` paragraph tags on the page, increasing the font size and adding a border.

You'll notice that I added a `border` property that does not have a single numeric value. You'll also notice that when you test this code on your site, the border does not animate in; instead, it just appears at the very end as the animation completes. Adding CSS properties that are not basic numeric values (like borders or background color, hex values) will not animate, but you can add all CSS properties using the `.animate()` function, which will act like the `.css()` function once it's *completed* running. This is probably not the best way to add regular CSS properties, but if you're animating something anyway, just know you can add other non-numeric CSS properties, they just won't animate.

Your property doesn't work?

You probably noticed this with the `.css()` function as early as *Chapter 2, Working with jQuery in WordPress* already, but just in case you didn't: property names must be **camel cased** in order to be used by the `.animate()` and `.css()` function. It's a bit confusing as you're probably just thinking of them as properties that you'd use in an actual CSS stylesheet but you'll need to specify `paddingBottom` instead of `padding-bottom` and `marginRight` not `margin-right`.

Making it colorful

You probably agree that as cool as the `.animate()` function is, it's not that impressive without color (and a little jarring with color that just changes abruptly at the end of the animation). You long to cross fade in brilliant color. Who doesn't? Unfortunately, the core animate function isn't robust enough to calculate all the variances of numbers in a single hex web color, much less between two hex colors (let's just say, some serious math is involved). It's much more complicated than moving a value anywhere from 0 to 100, or down again.

The good news is, the animate function can be *extended* with the Color plugin. The even better news? Yes, this plugin comes bundled with WordPress!

Let's add this plugin to our theme with the `wp_enqueue_script` like so:

```
...
<?php wp_enqueue_script("jquery-color"); ?>
<?php wp_head(); ?>
...
```

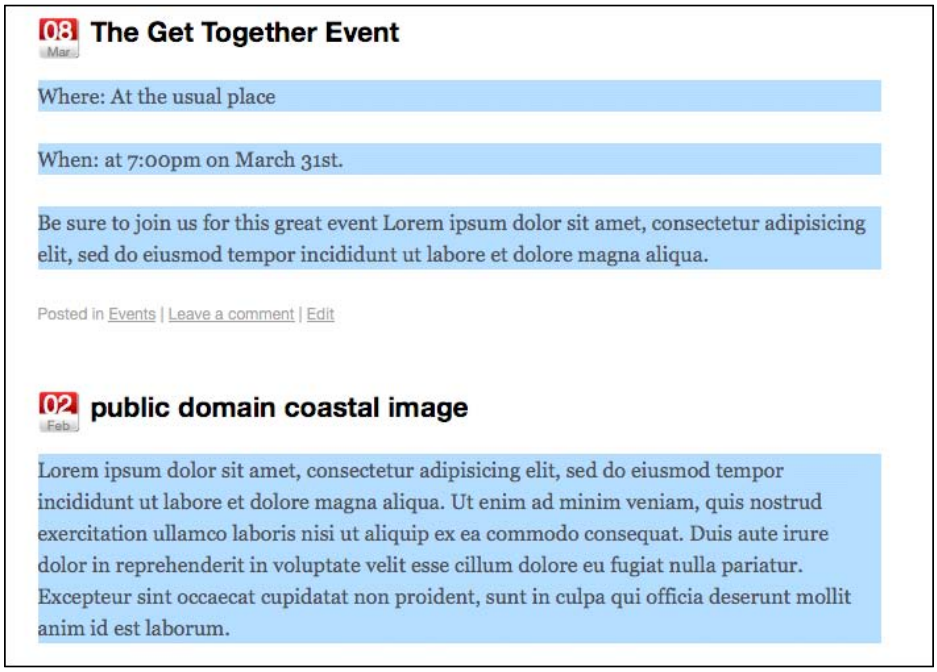
Registering and including a script that only needs to load on a particular page?

You'll recall in *Chapter 2, Working with jQuery in WordPress*, that you can wrap your `wp_enqueue_script()` functions in `if` statements that use WordPress' conditional tags that check for what page the site is on: `is_home()`, or `is_front_page()`, or `is_admin()`, and so on. Be sure to use these to your advantage to help keep your site running as optimized as possible and not unnecessarily slowed down by loading scripts that aren't needed. To find out more about conditional tags, check out their use with the Script API in *Chapter 2, Working with jQuery in WordPress*, and the conditional tag quick reference in *Chapter 9, jQuery and WordPress Reference*. You can also check out WordPress' Codex at `http://codex.wordpress.org/Conditional_Tags`.

Again, this plugin *extends* the existing `.animate()` function, so there are no new properties to learn! Once you've included the Color plugin into your project you can animate in background colors to your heart's content.

```
...
jQuery('.post p').animate({'backgroundColor':'#99ccff'}, 2000);
...
```

You should now see the `.post` paragraphs fade elegantly to a nice, light-blue color, as seen in the next screenshot:



Taking it easy, with easing control

If you're familiar with animation using various video editing tools or Adobe Flash, you've probably heard of easing. **Easing** is the control of acceleration and deceleration in an animation. It's most common use is to give animations a more natural feel, mimicking various properties of physics found in the real world, instead of calculated and rigid movement.

Almost as complicated as animating hex color values, easing applies virtual physics properties to the object being animated using various algorithms to control the speed of an animation as it starts off and ends. Serious math indeed. jQuery comes with a type of built-in easing, so we're saved from having to really think about any of it.

jQuery's default easing option is called "swing". Technically, there are two options—"linear" and "swing". **Linear easing** simply animates the object along its values from point A to point B, like a good programming script should. No acceleration or deceleration, so yeah, it is a tad "stiff".

Swing easing starts off more slowly, gains full speed, and then slows down again as the animation completes. jQuery chose swing as the default easing option as it looks best in most situations. That's probably because this is how most objects react in our real physical world; heading off a little slower while gaining speed, then decelerating and slowing down as they come to rest (provided the object didn't crash into anything while at the maximum speed).

As swing easing is the *default*, let's take a look at our previous script that animates in our post's paragraph blue background color and see if we can detect the difference:

```
...
jQuery('.post p').animate({'backgroundColor':'#99ccff'
  }, 2000, 'linear');
...
```

It's subtle, but a definite difference is there. Linear easing is much more rigid.

Advanced easing: There's a plugin for that!

As you've probably guessed, plenty of "mathy" people have figured out all sorts of variations in the easing algorithm to mimic all sorts of different physics environments and yes, there's a jQuery plugin for that. While this plugin doesn't come bundled with WordPress, that shouldn't stop you from downloading and experimenting with it. You can download and test out all the available easing options here: `http://gsgd.co.uk/sandbox/jquery/easing/`.

The plugin, like the Color plugin, *extends* the `.animate()` function and provides you with over 25 easing options, which include some pretty cool options such as jswing bounce and elastic, as well as a host of vector easing paths such as circular and sine wave.

The majority of these options are a bit of overkill for most projects that I've been on but I do love the elastic and bounce easing options. By the way, if you're one of those "mathy" people I referred to a second ago, you'll enjoy checking out the magic behind the easing algorithms here: `http://www.robertpenner.com/easing/`.

Timing is everything: Ordering, delaying, and controlling the animation que

Again, if you're familiar with animation, be it traditional animation, video, or multimedia work with Flash, you've probably learned—*timing is everything*. The more control you have over the timing and playback of your animations the better. Easing, for example, depends on how much time to give the object to animate and move. No matter how "smooth" you'd like an object to move, it's going to look fairly jerky if you only give it a second or less to get across the screen. Let's take a look at the three main ways to get a handle on your timing and playback.

Getting your ducks in row: Chain 'em up

We've discussed chaining functions in previous chapters, and you're most likely aware that any events you've chained together in a jQuery statement kick off in the order that they were *appended* to the chain. As far as I can tell, and based on what the experts say, you can chain to your heart's content as many functions as you'd like, infinitely (or until the browser crashes).

On the whole, I find laying out jQuery functions in separate lines, with their own selector sets, can take up some space, but keeps your jQuery scripts much more organized and manageable. Remember, you always start a jQuery statement with an initial selector for a wrapper set, but based on additional chained functions that can move you around the DOM and take their own selectors, you'll find that you can move around and affect the DOM a whole lot just from one statement! Along the way, possibly generating some quite magnificent "spaghetti code" that's hard to follow and will make any developer who has to work with you hate your guts.

However, for functions that need to be run on a single initial selector set, especially animation functions, I really like jQuery chains as they help keep my animation sequences in the order that I want them to kick off, and it's clear what wrapper set is going to be affected by the chain.

Here's an example:

```
...
jQuery('.post:first').hide().slideDown(5000, 'linear').fadeTo('slow',
.5);
...
```

Now, even initially concise animation chains can get a little complicated. That's OK; unlike some scripting languages, JavaScript and jQuery rely on the semi colon ";" as a clear ending statement, not the actual end of the line. So you can organize your chains into separate lines so that it's a little easier to follow and edit like so:

```
...
jQuery('.post:first')
.hide()
.slideDown(5000, 'linear')
.fadeTo('slow', .5);
...
```

Delay that order!

Because timing is everything, I often discover I want a function's animation to complete, and yet, depending on the easing option, especially those that are *elastic* or *bounce*, I don't necessarily want the very next function to kick off quite so fast! As of jQuery 1.4, it's easy to **pause** the chain with the `.delay()` function. Let's place a three second pause in our chain like so:

```
...
jQuery('.post:first')
.hide()
.slideDown(5000, 'linear')
.delay(3000)
.fadeTo('slow', .5);
...
```

Check your jQuery version! `delay()` **requires 1.4+**

As soon as this function became available, I've put it to use in all sorts of invaluable ways with my jQuery animations. However, if you find the delay function is not working for you, you're probably working with version 1.3.2 or older of jQuery. The delay function is only available with version 1.4+. You may want to go back to *Chapter 2*, *Working with jQuery in WordPress* and see about registering jQuery from the Google CDN or including it directly in your theme.

Jumping the queue

Queues—those irritating lines that ensure everyone or everything in them is processed fairly and in the order they arrived. jQuery's animation queue works similarly, only processing each object's animation request, in the order it was assigned to the object. Sometimes special needs and requirements arrive that shouldn't be forced to waste time in the queue.

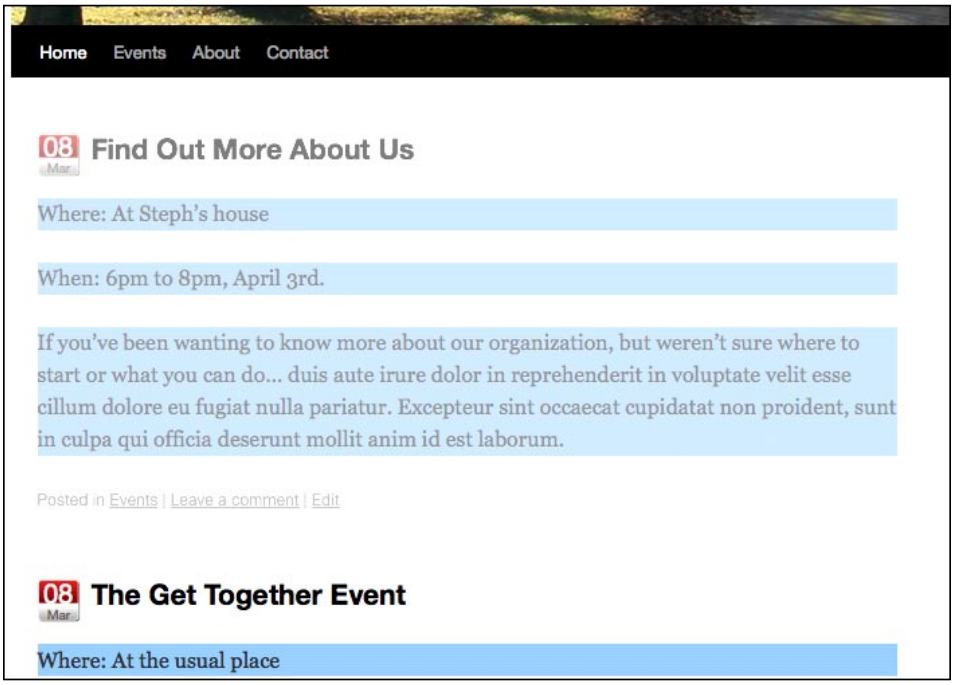
So far, we've seen how the `.animate()` function, in addition to CSS properties, can be passed various optional parameters specifying the *duration*, (slow, fast, or numerical milliseconds) and the type of *easing* (swing, linear, or plugin extended easing).

The `que` parameter is a **true or false** Boolean that can be set if you don't want the animate function to have to wait its turn. For the instances that you'd like an object to have several animations to run in parallel with each other, like sliding *and fading* at the same time, disabling the `queue` in your animate function will do the trick.

In order to set the `queue` option in your code, instead of using the previous syntax we've been working with, you will have to *wrap* all the other options into a more advanced syntax which clearly labels each optional parameter like so:

```
...
jQuery('.post:first')
    .hide()
    .fadeTo(0, .1)
    .css("height","5px")
    .animate({
        height: '+=500px',
     },
     {
        duration: 4000,
        easing: 'swing',
        queue: false
     }
    )
    .fadeTo(4000, 1);
...
```

The following screenshot shows the post is fading out *while* changing in height at the same time:



You can see by the previous screenshot that the code we just wrote fades the first `.post` div in *while* it's sliding down. If you change `false` to `true`, and reload the page, you'll discover that the first `.post` div slides all the way down to `500` pixels high *and then* fades in.

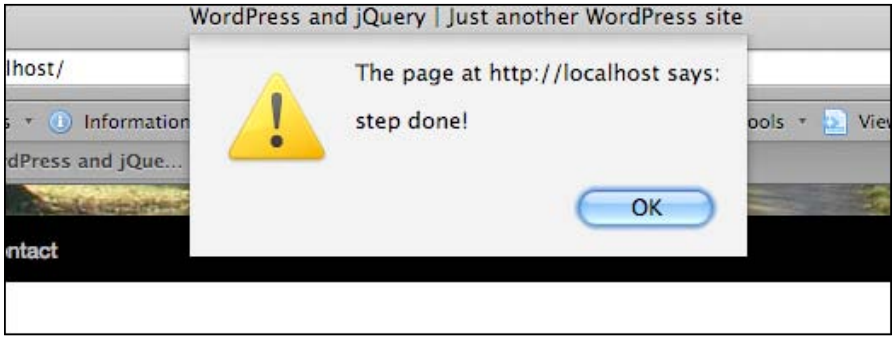
Stepping to completion

The final options that can be passed into the `.animate()` function are `step` and `complete`. The `step` parameter allows you to set up an additional function that can be called after each step of the animation is complete (sometimes useful if you have multiple CSS properties you're animating). The `complete` parameter allows you to specify a callback function when the entire animation function has been completed. Keep in mind, you can chain multiple animation functions together, and the steps with complete parameters are unique to each instance of the animation functions that they belong to.

If you have an animation that absolutely should not kick-off until the current animation function has completed, the `.delay()` function might not be the best way to go. You can use the `step` and `complete` parameters to kick off other functions and animations in the exact order you wish.

```
...
jQuery('.post:first')
    .hide()
    .fadeTo(0, .1)
    .css("height","5px")
    .animate({
        height: '+=500px',
     },
     {
        duration: 4000,
        easing: 'swing',
        queue: false,
       step: function() {alert('step done!');},
         complete: function() {alert('completely done!');}
     }
    )
    .fadeTo(4000, 1);
...
```

The previous code snippet will generate JavaScript alerts at the `.animate()` function's completed steps once it's completely finished.



I've personally never needed to hook into the `step` parameter for a WordPress project, but I can see how it could be very useful for hooking into and creating a chain of cascading type effects. I have found the complete `parameter` very useful for many of my animations.

Grabbing the user's attention

OK, sample code snippets aside, it's time to get to work! Back in "hypothetical land", our previous clients have enthusiastically touted our jQuery solutions to a few associates and we now have quite a few requests for help with WordPress sites. Let's walk through a few new hypothetical client situations and see if we can solve their problems.

First up: a common way many sites employ "sticky" posts and how to enhance them with a little jQuery animation.

Project: Animating an alert sticky post

Here's a quick and simple one. You've got a client who has a good friend, who runs a non-profit educational organization's site, and they need a favor (meaning; do this "for free" please).

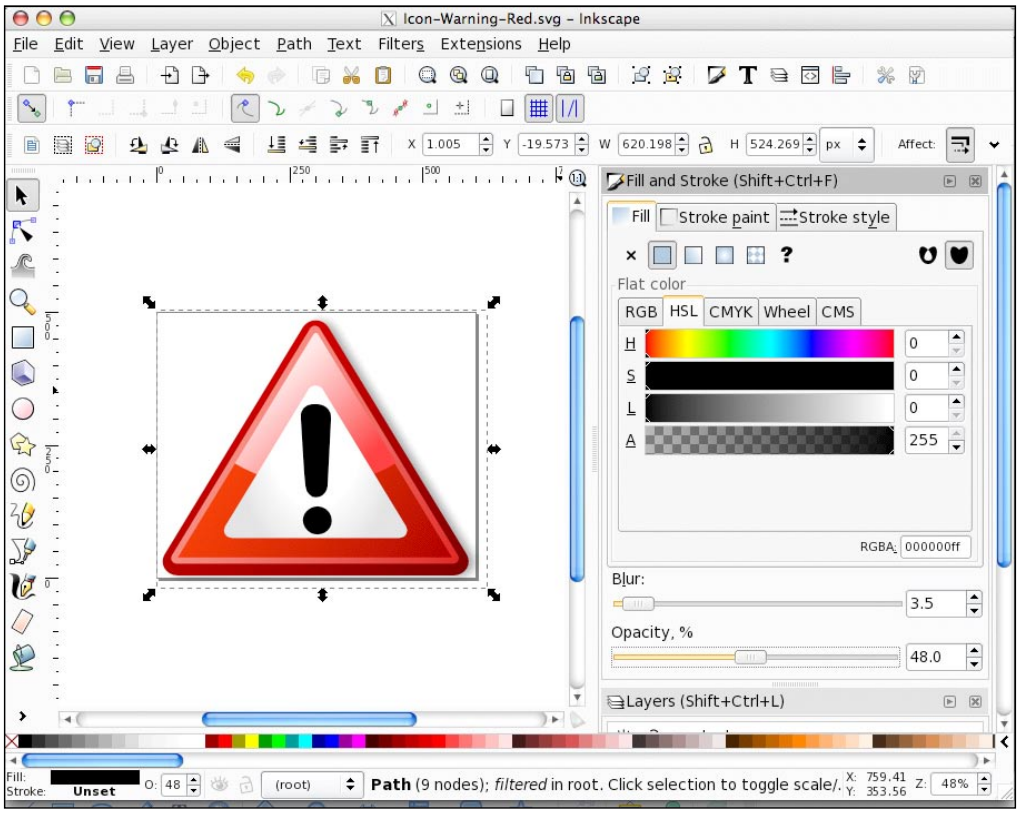
The organization's after-school care runs on the public school's schedule (as many kids are bussed over from different schools). If the public school system takes a snow day or some other emergency day, the after-school program also closes down. The organization does their best to notify people through their WordPress site.

Despite making it clear to parents that it's their responsibility to check the site, or call to find out the center's schedule, there's been a few misunderstandings with people who claim that they checked the site but "*didn't see the closing alert*". Apparently, even though they've been making the posts "sticky" so they stay at the top, the posts look awfully similar to the rest of the site's content.

You're happy to help (especially as they were referred to you by a client with well-paying gigs). It helps that this is a really easy fix. First off, you can simply add a few `.sticky` styles to their theme's `style.css` file, which makes the sticky posts stand out a lot more on the site.

They've made it clear they only use the "sticky" feature for daycare and other center alerts that affect the organization's center building being open to the public so you decide to do a quick Google search for "`creative commons, public domain, alert icon svg`" and download a very nice SVG file from `http://commons.wikimedia.org/wiki/File:Nuvola_apps_important.svg`.

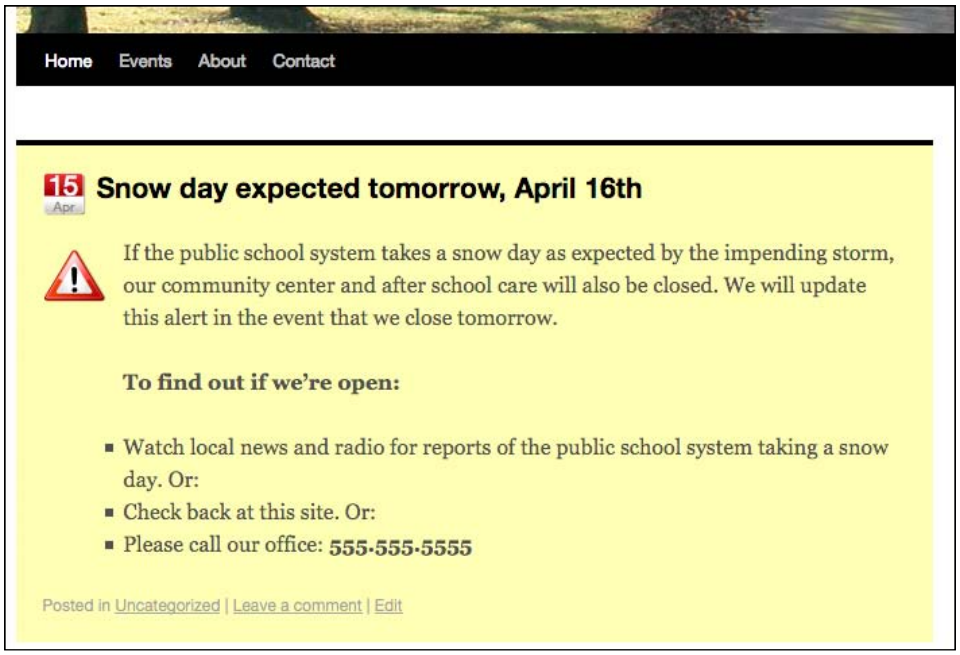
Let's open that SVG file into Inkscape and size it down to 48 pixels wide to save a transparent `.png` of it (I took the liberty of adding a little blur to the shadow, but you may not want to). Name the PNG `sticky-alert.png`.



You then add the new `sticky-alert.png` image to their theme's image directory and update the stylesheet at the very bottom *or below* the existing `.sticky` class, if one exists, with a few class rules for the `.sticky` calls like so:

```
...
/*change the .sticky background */
.home .sticky { background-color: #ffff9c;}
/*add the icon to the entry-content div inside the sticky post*/
.home .sticky .entry-content{
  background: url(images/sticky-alert.png) no-repeat 0 20px; }

/*nudge the paragraph and lists out of the way of the icon*/
.home .sticky .entry-content p,
  .sticky .entry-content ul{margin-left: 60px;}
...
```

The following screenshot shows the newly re-styled sticky posts:



This is more than good enough. Now anyone going to the site regardless of JavaScript being available will certainly notice that. But hey, since you're poking around in the theme anyway, and you've decide to register jQuery, the jQuery Color plugin from the WordPress bundle, and include a `custom-jquery.js` page to their `header.php` file, you might as well add in this nice and simple few lines of code.

```
jQuery(function(){
     jQuery('.home .sticky')
      .animate({'backgroundColor':'#ff6600'}, 'slow')
      .animate({'backgroundColor':'#ffff99'}, 'slow')
      .animate({'backgroundColor':'#ff6600'}, 'slow')
      .animate({'backgroundColor':'#ffff99'}, 'slow');
});
```

The previous code will fade our sticky posts from light yellow to darker orange, and then *repeat* it again for emphasis. The following image shows the post faded to darker orange:

Again, a bit hard to see the animation in a book, but we just made sure that the alert `.sticky` post, upon loading, will fade up to orange (`#ff9900`) and back down to the yellow (`#ffffcc`), and then repeat one more time for quite the "orange alert" effect.

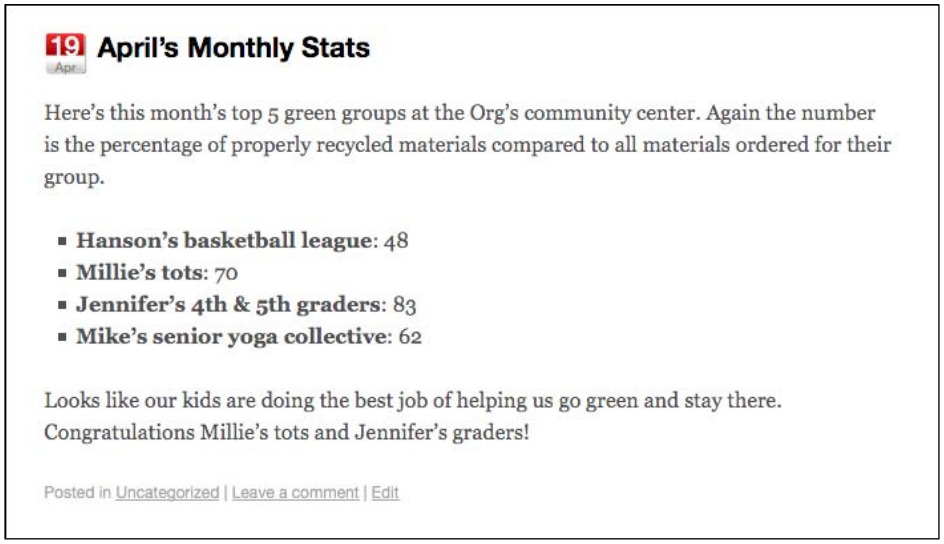
The alert posts are very noticeable now and the organization can't thank you enough! Which is more than enough for your few minutes worth of work.

Creating easy, animated graphs

The non-profit organization was so impressed with your alert sticky post solution, they've allocated some funds together and have got another request for you. They noticed how you fixed up the alert icon using Inkscape and they've asked you how much trouble it would be to generate a monthly graph for them for another post they put up. The post is their top five stats from their green recycle program.

While the bulleted list is super easy for the site administrator to implement, people don't really notice or remember the information, so they were thinking of posting graphs to the site, but need someone to draw them or generate them in some way.

Looking through their site, you notice the editor always consistently formats the information posted. All post titles have "**... Monthly Stats**" in them, all the information is in bullets, and the percentage number always comes *after* a colon "**:**". It is great that the admin has been so consistent. It will make it very easy to work out a solution that allows the editor to continue doing what they've always done. The posts currently look like this:



You let the admin know as long as he/she continues to consistently format the posts in this way, you can write up a jQuery script that will draw the chart for them. They almost don't believe you, and are happy to have you proceed.

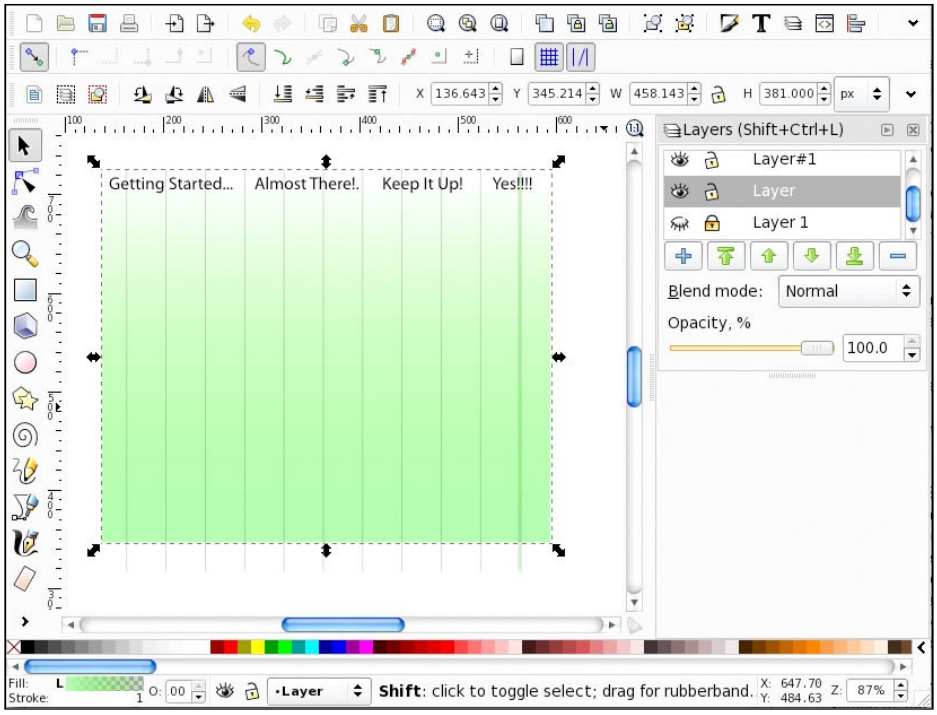
To get started, we'll need to first make sure we're targeting the correct posts for **Monthly Stats** only. We'll do this by setting up a jQuery selector like so:

```
...
jQuery('
.post h2:contains(Monthly Stats)')
.siblings('.entry-content')
.children('ul:first')
.css({background: '#ccffee'});
...
```

As we can see, this little "test" selection grabs all `h2` tags which are inside `.posts` that only contain the text "`Monthly Stats`". We then move along the DOM and target the `.entry-content` div and then the **first** `ul` inside that. We can see the previous code is properly targeting what we want in these posts by changing the background color, as shown in the next screenshot:

Now that we can target the specific posts we want, without changing the theme's output or making our client do any work, let's get to work on the rest of the graph!

First up, because we'll be loading a background image, and those just load a lot nicer from a theme's stylesheet (it's easier to target the images), let's use Inkscape again to help us create a basic background about 450 pixels wide that shows the progression from "Just getting started" to "Yes!", like so:



Let's export a PNG of that graphic and add it to the image directory of our client's theme. Then, using jQuery, let's dynamically add a class to all our targeted `ul`:

```
...
jQuery('.post h2:contains(Monthly Stats)')
  .siblings('.entry-content').children('ul').addClass('greenStats');
...
```

We can now go into the client's theme stylesheet, and just as we did for the sticky alert posts, create custom CSS rules for our new class. Open up the theme's `style.css` stylesheet and add these rules at the end:

```
...
.entry-content .greenStats{
  margin: 0;
  background:url(images/greenBackground.png) no-repeat;
  border: 1px solid #006633;
  padding: 40px 20px 5px 20px;
}

.entry-content .greenStats li:before{content:none;}
.entry-content .greenStats li{padding-left: 10px; margin: 0}
...
```

The first rule adds our new `greenBackground.png` chart image and sets some basic properties so that the list items can start accommodating our upcoming jQuery additions. The next two rules after that fix specific `.entry-content li` issues that the client's theme (in this case, the default theme) places on every `li` element inside an `.entry-content` div. We don't want the "little squares" before our chart items, and we want the padding on each `li` moved in about `10px` more. Again, we only want to affect the `.entry-content` li items if jQuery has added our `.greenStats` class so we're sure to add that class name into the CSS rule.

Now, we're ready for some serious jQuery magic. I hope you've been getting really comfortable working with selectors and traversing the DOM until now. We're going to have to put a fair amount of that knowledge to work to accomplish the next few tasks.

We want to place an `.each()` function item on our targeted `li` and begin manipulating the content inside of them.

We'll start by setting up this jQuery statement:

```
...
jQuery('.post h2:contains(Monthly Stats)')
  .siblings('.entry-content').children('ul').children('li')
  .each(function(){

    //code here

  });//end jQ li
...
```

Next, *inside* our `.each()` function, we'll place code that start's manipulating the HTML and text inside each `li` object. We want to look for the colon `":"` and use that as a point to wrap a `div` around the number that comes after it. After that, we'll look for the ending `</li>` tag and use that as a point to close our `div` that we started. We'll accomplish this by using the `.text()` and `.replace()` functions like so:

```
...
var string1 =
  jQuery(this).text().replace(': ',':<div class="nVal">');
var string2 = string1.replace('</li>','</div></li>');

//place back into the li element as html markup and text:
jQuery(this).html(string2);
...
```

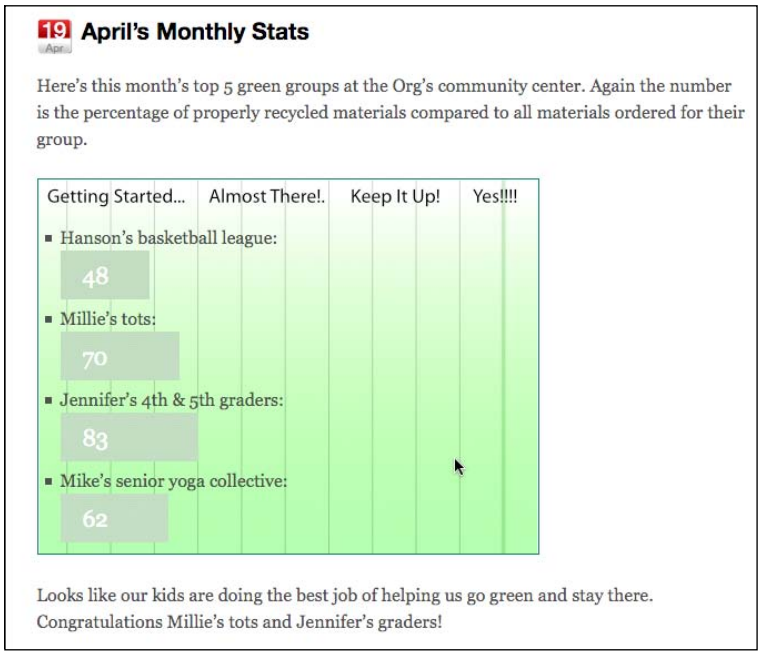
That previous code snippet now gives us custom `div` with the class `.nVal` that we can start working with. The `.nVal` divs are ultimately going to be the "bars" in our green stats graph! Underneath the previous code, we'll continue to flesh-out our `.each()` function with the following code, again *inside* the `.each()` function:

```
...
//set the default css values of each nVal div:
jQuery(this).children('.nVal').css({width: '0',
  padding: '10px 0 10px 20px', fontSize: '130%',
  color: '#ffffff', marginBottom: '5px'});

//retrieve the number text from inside the nVal div:
var nVar = jQuery(this).children('.nVal').text();

//animate the nVal divs with the nVar values:
jQuery(this).children('.nVal').delay(600)
  .animate({backgroundColor: '#006600', width: nVar*(3.8)}, 2000);
...
```

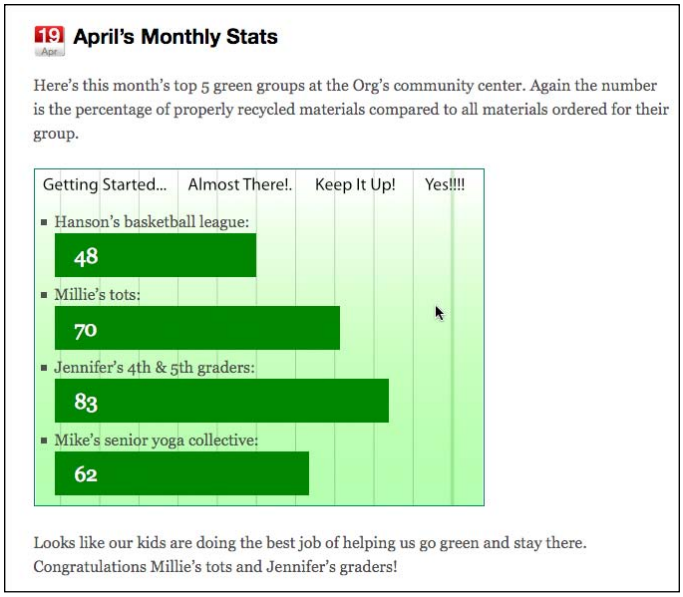
In the previous code snippet, note that I used the `.delay()` function. That function is optional if you're not using jQuery 1.4.2 or a higher library. I just think a good half second or so pause to make sure that users notice the animation is helpful.

We again used the `.text()` function to pull the text from inside the `.nVal` divs and use it in a mathematical equation to calculate the `width` of the divs with the `.animate()` function. We multiply `nVar` by `3.8` because within our chart design, a div width of about 380 pixels wide would be the equivalent of 100 percent. If your chart has different dimensions, you'd change those accordingly to have the chart bars extend out properly.

The result looks great! Here's our chart animation as it starts off:



And here it is at its completion, a fun visually clear display of the organization's shining greensters:



Delving deeper into animation

Thanks to your animated green stats chart, you're ready to take on some slightly more complicated requests: clients *insisting* on Flash development. As someone who got into web development through Flash in the 90s, a request for Flash development is no problem. Complain all you want, you have to admit, Flash sure can animate.

However, Flash does require a plugin and despite being the most popular desktop browser plugin out there, it's not always a good way to display core content you want to ensure everyone can see, much less essential elements such as site navigation. Plus, while it's the most popular plugin for *desktop/laptop browsers*, Flash is a "no-go" in Safari Mobile for iPhone and other WebKit-based browsers for most SmartPhones.

With all the advances of CSS and JavaScript support in browsers these days (especially mobile browsers), my first question for Flash requests is always: "Sure. First, tell me exactly what you'd like done, and we'll see". Sure enough, our client wants their main navigation panel animated.

Flash can certainly do this, but then so can jQuery, and when JavaScript is not an option, it will elegantly degrade into nicely styled CSS elements and in the worst case, with no CSS, the page will load our WordPress theme's clean, semantic XHTML into a text-only browser.

While there are plenty of ways to serve up Flash content and applications so they degrade elegantly into compliant HTML alternatives (and you should *always* offer these alternatives when working with the Flash player), why add that extra layer of development and complexity if it's not essential? The client doesn't need to serve up streaming video, or have custom cartoon characters animated, much less want an in-depth, multimedia slathered and mashed up **Rich Interface Application** (**RIA**). So, let's leave Flash for what Flash does best and use jQuery to enhance what our client's WordPress site already does best.

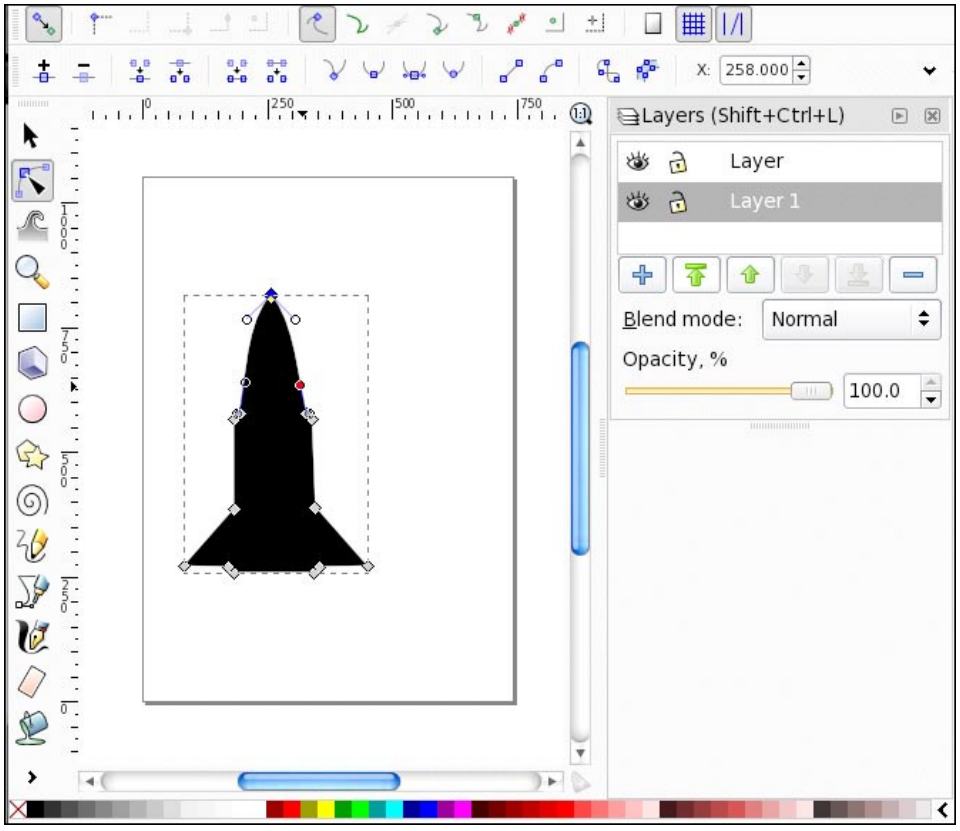
Luckily, the client is reasonable and willing to see what jQuery can do before we resort to Flash. Let's show them what their WordPress site is made of with a little jQuery inspiration.

Project: Creating snazzy navigation

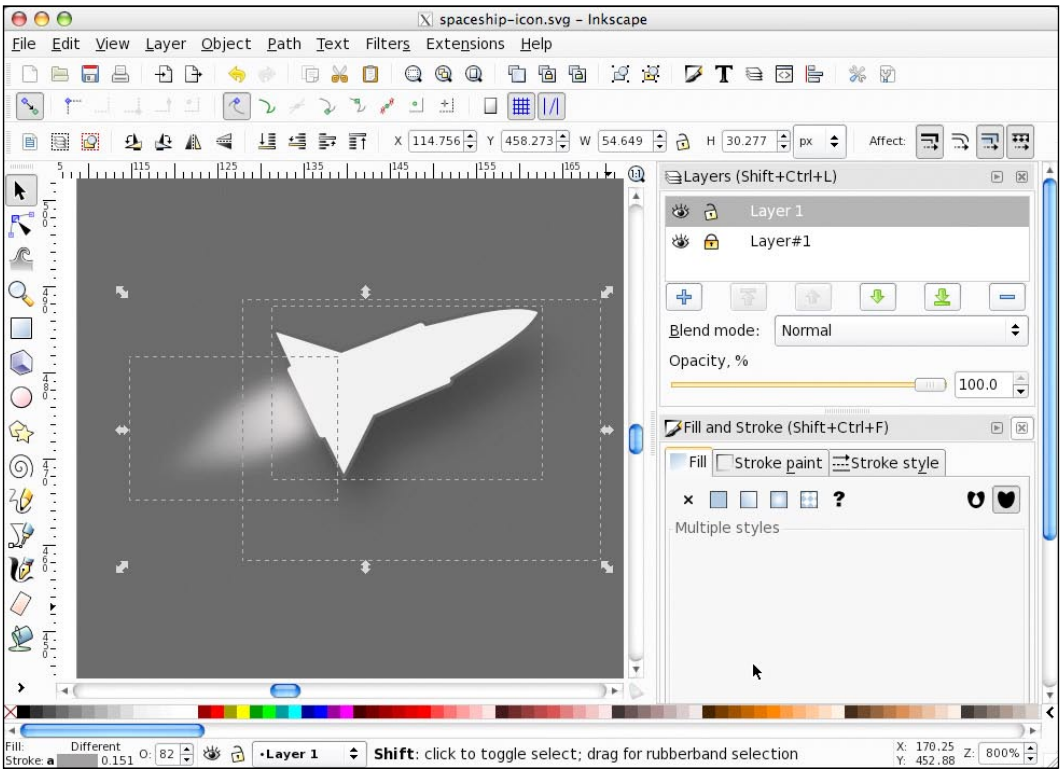
Our Flash-inspired client has a business that recycles and refits materials that NASA and other space agencies have junked and sold. They'd like their navigation panel to represent this futuristic (and at the same time, retro) feel to their users and provide a page navigation that, according to the client: "has a smooth animation and our logo/icon rocket as a pointer".

Let's go ahead and prep the theme so that we can get started. We'll continue to use the Default Theme with the Page Navigation CSS changes that we made in *Chapter 2*, *Working with jQuery in WordPress*. We'll be enhancing the navigation with a smooth indent and release animation that triggers on hovering on and off the menu items. We'll top it off with a cool floating point selector (which also happens to be the site's space ship icon).

First up, we'll need to trace the client's space ship icon used in their logo, into a basic silhouette form so that we can create a floating pointer with it. Again, this is easily done using Inkscape:



We'll take an extra step here and rotate the ship, and since it's going to be a transparent PNG file, add a nice drop shadow and afterburn gloss to give it some depth:



We'll export this image as a 37 pixel wide transparent `.png`. Next up, we'll need to prep our theme's stylesheet to accept this background image. We'll be creating a `div` called `#shipSlide` in jQuery to hold the image so our stylesheet needs to accommodate that `id` name:

```
...
#shipSlide{
  position: absolute; margin-top: 12px; margin-left: -7px;
    width: 37px; height: 20px;
  background: url(images/spaceship-icon.png) no-repeat;
}
...
```

Again, as with many examples in this book, to keep the process concise and easy to understand, we'll be doing things as directly as possible, but not necessarily as optimized as possible. In a real-world project you may want to create a separate stylesheet for any project like this or wrap your jQuery work into a plugin or even in a WordPress plugin using the techniques we covered in *Chapter 3*, *Digging Deeper: Understanding jQuery and WordPress Together*. This all depends on how flexible and portable you'd like the jQuery enhancement to be afterwards.

Now, we'll get to work in jQuery. As usual, for every project you'll make sure that jQuery is included into the theme, and that you have a `custom-jquery.js` file included and set up to work in. Also, for this navigation, we'll be using the Color and Easing plugin. You can register the bundled Color plugin, but you'll need to download and include the custom Easing plugin into your theme manually. Get it from: `http://gsgd.co.uk/sandbox/jquery/easing/`.

In our particular default theme, we'll start off with some jQuery to make it a little clearer what our nav will do.

Our first bit of jQuery looks like this:

```
...
      //this adds our #shipSlide div
      //under the twenty ten theme's menu header div
       jQuery('.menu-header').prepend('<div id="shipSlide"> </div>');

      //this fades the ship div to 40%
     jQuery('#shipSlide').fadeTo('slow', 0.4);
...
```

Before I fade the `#shipSlide` div with jQuery's `.fadeTo()` function, I did load it up into the browser to check and make sure the background image was loading in from the CSS. The following screenshot shows the ship image loaded in and faded by our budding jQuery script:

OK, next up, let's set up a basic animation that pushes the navigation `li.page_item` objects in from the left, 35 pixels, relative to where they are. We'll also then target the tags and change their background color. We'll use the `.hover` function to make sure this happens on rollover and rollout of the `li.page_item` objects:

```
...
      jQuery('li.menu-item')
      .hover(function() {
      //animates each menu item to the right (from the left)
      jQuery(this).animate({paddingLeft: '+=25px'}, 400, 'swing');

      //this over rides the style sheet's background on hover
      jQuery(this).find('a').css('background','none');

      //ship move code will go here

      }, function(){

      //returns the menu item to it's location
      jQuery(this).animate({paddingLeft: '-=25px'}, 400, 'swing');


        });//end hover
...
```

Finally, *inside* the first hover function, just *below* the a object's color animation, we'll add in the following code snippet, which will move the `#shipSlide` object to the position of the `li.item_page` (note the bold code only):

```
...
//this custom moves the ship image
      var p = jQuery(this);
      var position = p.position();
      jQuery("#shipSlide").fadeTo('slow', 1)
        .animate({marginLeft: position.left-175},
          {duration: 600, easing: 'easeOutBack', queue: false});
...
```

Here, we've set up a variable we named `position` and also used a function called `.position()` to be able to pull an array of information from the `li.page_item` objects.

The `#shipSlide` object's animate function moves the `marginLeft` of the ship left to the `position.left` of the `page_item`, minus 175 pixels.

You'll also notice in the previous code snippet's `animate` function that we set the `queue` to `false` and that we're using the `easeOutBack` easing method that's only available to us because we included the Easing plugin.

The very last bit of code we need, *below* the `li.page_item .hover()` code is another jQuery selection and `.hover()` function, which will fade the `#shipSlide` object in and out on hover of the `#mainNav` object. Again, just place this jQuery below all the other navigation code:

```
...
//this fades and moves the ship back to it's starting point
  jQuery('.menu-header').hover(function(){
      jQuery("#shipSlide").fadeIn(1000);
    }, function(){
      jQuery("#shipSlide").fadeTo('slow', .4)
        .animate({marginLeft: '-5px'},
          {duration: 600, easing: 'easeOutBack', queue: false});

    });//end hover
...
```

The final result looks great, the ship and menu item animation is smooth, and the client is very happy with their new snazzy navigation.

Project: Creating rotating sticky posts

Earlier we discovered that working with WordPress sticky posts is pretty easy! That's good to know because our Mr. "I want Flash" client has now requested an additional enhancement solution. They are using WordPress sticky posts to make site viewers aware of the products that they're featuring. Making the posts sticky works great keeping their product mentions up top (usually two to four at a time), while their regular news posts and updates flow below the product features.

However, when they have more than two products to feature, (especially when they have three or more products to feature) their current posts get pushed down, sometimes way below the fold. They're worried that people just glancing at the site from time to time may feel it's stale if they don't take the time to scroll down and see the current posts.

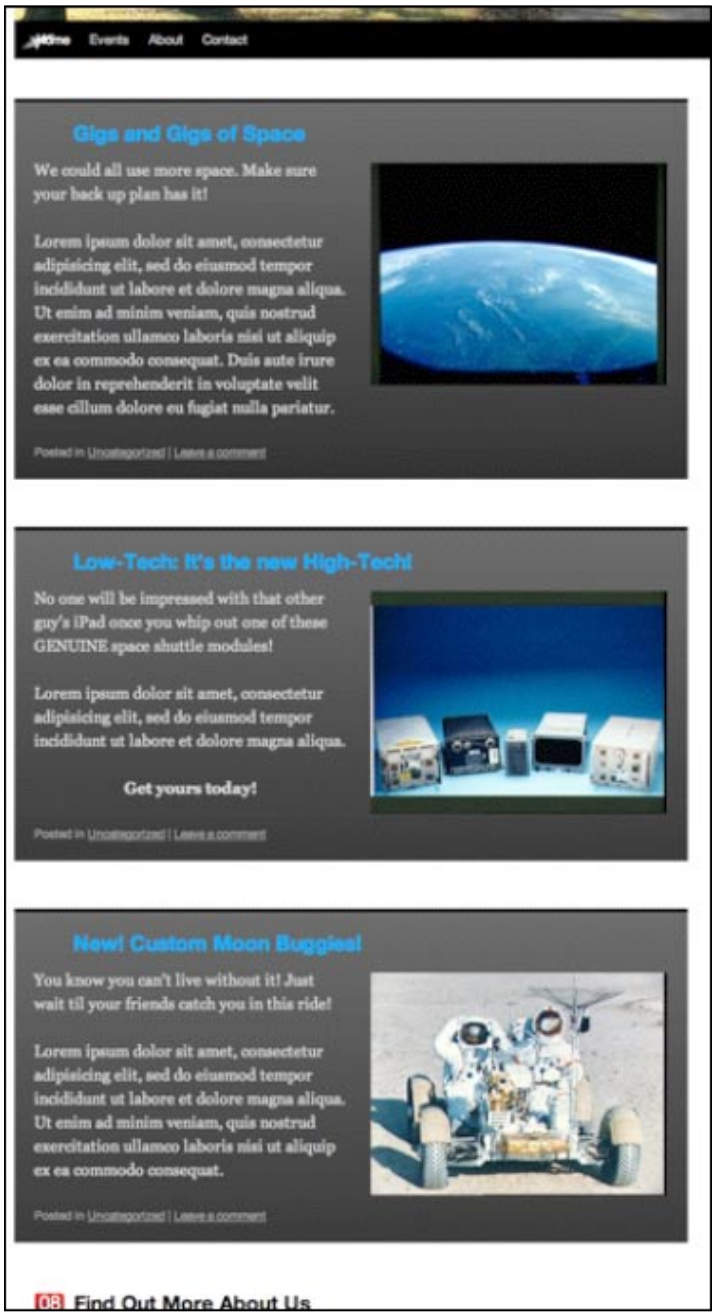
They've seen plenty of examples of sites that have really cool image rotators with slide or cross-fade effects up on top of featured items and they'd like to work something like that into their site. They originally thought they'd do this in Flash and give up convenience, but since the jQuery navigation panel turned out so well, they'd like to create a solution that:

- Conserves space, so other posts don't get pushed "below the fold"
- Looks really nice and draws attention to the sticky feature posts
- Means it's still easy for their marketing administrator to implement new featured items (as easy as just creating a post and marking it "sticky"!)

This client's theme already has the sticky post's CSS changed slightly, in that there's a simple background that makes the posts have a dark gradation as well as some font color changes. You'll find these CSS rules at the bottom of their theme's `style.css` stylesheet:

```
...
.sticky { background: #000 url(images/sticky-background.png)
  repeat-x; color: #ccc;}
.sticky small.date{display:none;}

.sticky h2 a{color: #0099ff;}
...
```

The result looks like this, and you can see how just three sticky posts leave NO room for checking out the current posts below those, and leave the user with quite a bit of scrolling to do:



Essentially, we'll want to collapse these stickies on top of themselves, maybe make them a little shorter if possible, hide all of them except the *first* sticky post, and then proceed to fade in the remaining posts *over* the first one.

First up, it seems obvious, but again, make sure that you've registered and included jQuery into the theme along with the Color and Easing plugins discussed earlier. You can include jQuery however you wish, but I'll be using 1.4.2 from the WordPress 3.0 bundle as discussed in *Chapter 2*, *Working with jQuery in WordPress*. And per usual, you'll also want to be sure to include a `custom.js` file to the theme so that you can keep your jQuery code out of the WordPress `header.php` template (this is also covered in *Chapter 2*, *Working with jQuery in WordPress*).

Once jQuery and your plugins are included in the theme, we'll get to work with jQuery. Because the site is functional the way it is, and the client is OK with this as an alternative view, we'll leave the theme and `style.css` alone and make sure *all* our enhancements are done with jQuery.

Again, the following code may not be the most elegant way to achieve the client's goals, but it's written to make sure each step of what's happening is clear.

Let's start by changing the CSS properties of the sticky posts so that they all stack up on top of each other. The easiest way to do this? Make the `.sticky` class `position: absolute`. Let's also go ahead and make the width and the height correct and that any overflow is hidden like so:

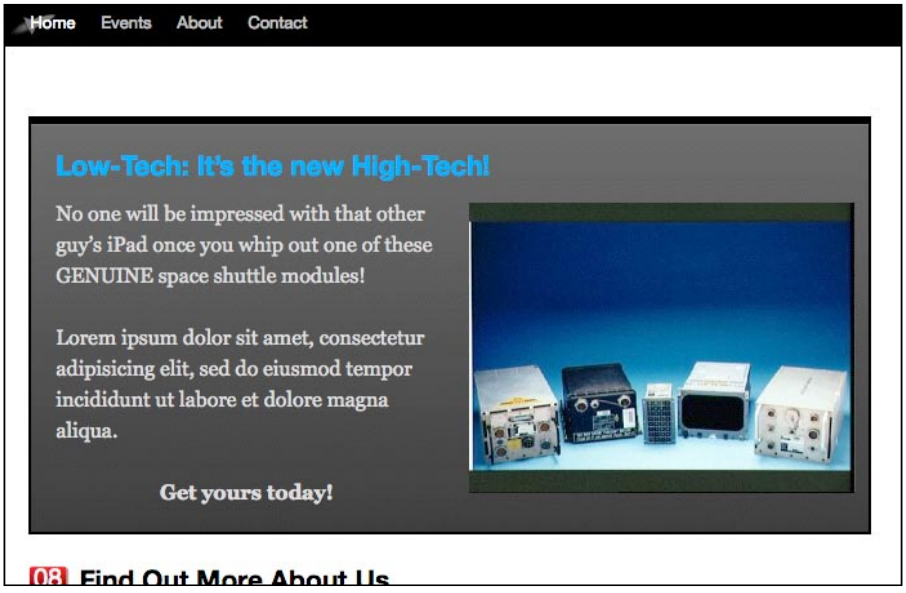
```
jQuery(function(){
jQuery(".sticky")
        .css({
        position: 'absolute',
        top: '0',
        margin: '0',
        width: '650px',
        height: '320px',
        overflow: 'hidden'
        });
...
```

Next up, we'll move the `h2` header up a bit and most importantly, as our actual posts are *under* the positioned absolute `.sticky` posts, we'll move those down so they show up under our soon-to-be-animated sticky posts. We'll also adjust the image's right-hand side margin a bit for placement.

```
...
//move the header back over if it's affected by the css
//you could also do this in the CSS directly
    jQuery('.sticky h2').css({margin: '0', padding: '0'});

//move the margin over a bit
//you could also do this in the CSS directly
    jQuery('.sticky img').css('marginRight','30px');

//this pushes the other posts down out of the way
    jQuery('.post:not(.sticky):first').css('margin-top','360px');
...
```

Pay special attention to the bold jQuery selector in the previous code snippet. You can refer to *Chapter 3, Digging Deeper: Understanding jQuery and WordPress Together* for more on using selectors if you need to refresh your knowledge. Essentially, we're targeting the *first* `.post` div that does *not* have the `.sticky` class assigned to it. Nice!

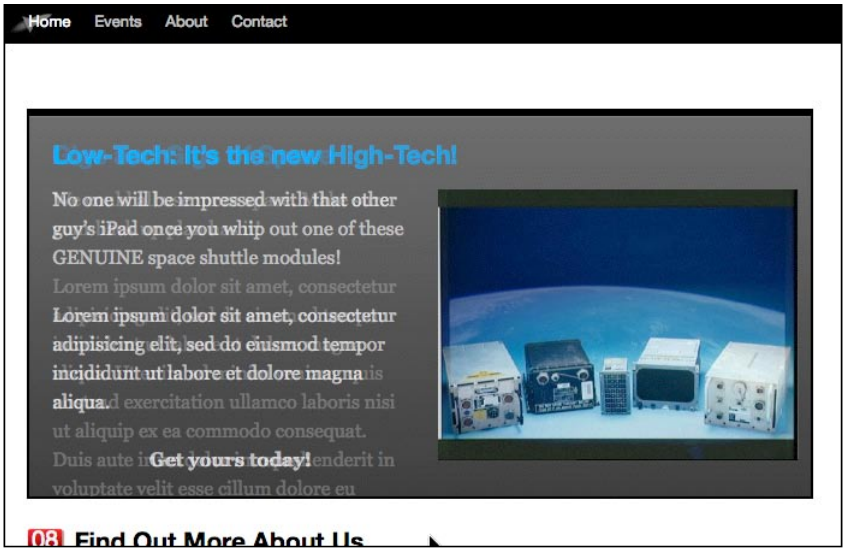
The result looks like this:



OK! jQuery has that really nice function we've looked at previously called `.each`, which will run additional functions on every object in a wrapper set. If all we wanted to do was run through each item one time, we could use this bit of code:

```
...
jQuery('.sticky')
  .hide()/*hide each post*/
  .each( function (i){
    /*i = numeric value that will increase with each loop*/
      jQuery(this)
    /*make sure each div is on it's own z-index*/
      .css('z-index','i+10')
    //using the animate function to fade in each div
    //3 seconds apart*/
    .animate({'backgroundColor': '#000000'}, i*3000, function(){
        /*actual div fade in*/
    jQuery(this).fadeIn('slow');
    }
    );//end animate
  });//end each
...
```

This looks good! However, once the last `div` has faded in, it stops and doesn't continue.



Nope, there's no super slick jQuery way to keep the `.each()` function going. Yet, an `.each` function is so easy to set up, it's a shame not to leverage them, even for "infinite loops".

Now, a quick explanation here: you can do a Google search for "`infinite animation loops jquery`", if you dare, and see that for all ten-thousand-some results, there appears to be about that many ways JavaScript developers like to set up repeating, or infinite loops, and each developer seems to feel (of course!) that their method is the best method available. My preference is to resort to regular JavaScript, and use a `setInterval` function and some custom variables set up in a way that makes it very easy to leverage my existing jQuery `.each()` statement and functions.

To get started creating our loop, we'll take our existing jQuery statement and place it *inside* its own function. You'll need to make sure this function is **outside** your main `jQuery(function(){...` document ready function. Otherwise, the `setInterval` function will not launch it properly.

Let's call our new function `loopStickies`. You'll find it familiar, aside from the first statement:

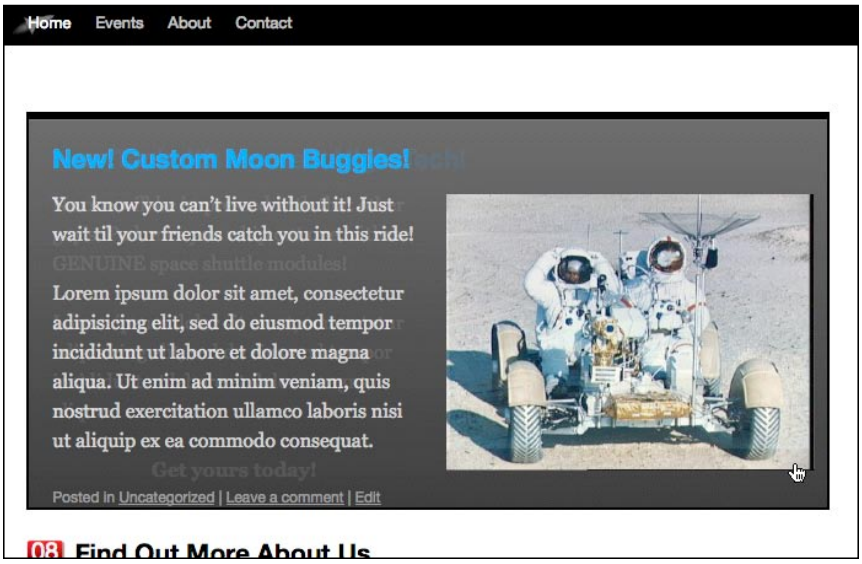
```
...
function loopStickies(duration){
/*note the variable "duration" being passed*/

    ///we'll need to make sure everything fades out
    //except the first sticky post*/

     jQuery('.sticky:not(:first)').fadeOut();
     /*this should look almost the same*/
     jQuery('.sticky')
       .each( function (i){
          /*i = numeric value that will increase with each loop*/
          jQuery(this)
          /*make sure each div is on it's own z-index*/
          .css('z-index','i+10')

          /*using the animate function & "duration" var for timing*/
          .animate({'backgroundColor': '#000000'}, i*duration,
            function(){
              jQuery(this).fadeIn('slow');
            }
          );//end animate
       }); //end each

}//end loopStickies
```

OK, that's just the start, now that we have our `loopStickies` function, located *outside* the jQuery document ready function, let's place the rest of our code, back **inside** the `jQuery(function(){...` document ready function. Follow along with the comments in bold:

```
...
/*set the stickies in a wrapper set to overflow hidden*/
    jQuery('.sticky').wrapAll('<div id="stickyRotate"
      style="position: absolute; padding: 0; margin-top: 5px;
      width: 650px; height: 320px; border: 2px solid #000;
      overflow:hidden;"></div>');

    //make sure the first .sticky post fades in:
        jQuery('.sticky:first').fadeIn();

    //set the "duration" length to 6 seconds for each slide:
     //(this is the var our function uses)
    var duration = 6000;

    /*create the interval duration length, based on the duration:*/
    var intervalDuration = duration * jQuery('.sticky').length;

  /*the function needs to run once before the setInterval kicks in*/
    loopStickies(duration);

    //the setInterval will kick off loopStickies in
    //18 seconds: (6secs x number of sticky posts) */
        setInterval( 'loopStickies("'+duration+'")', intervalDuration
);
...
```

The way this works is, our original jQuery statement and `.each()` function runs through each sticky post in the jQuery selection by evoking the `loopStickies` function. At the *same time*, the `setInterval` function is kicked off, but because we have the `intervalDuration` variable set to calculate our `duration` variable times the number of sticky posts, it's not going to kick off for 18 seconds. Just in time for our original function to complete! The `setInterval` function will take it from there and loop our sticky posts into infinity.

OK, let's take a look; we now have a very nice set of sticky posts, holding for six seconds and then crossfading to the next post!



Putting in a little extra effort: Adding a loop indicator

The rotating stickies are great! Yet, while the client will only have three or four stickies rotating at any given time, it's a good practice to at least let a user know about how long a view they're in for should they decide to look at all the rotations. Most rotating slide shows have an indicator somewhere to let a user know how many panels are going to be displayed and allowing the user to navigate around the panels.

Let's see about adding this functionality to our rotating posts. First up, we'll need to create a little interface. Inside our `#stickyRotate` wrapper that we created in the previous code, after the last sticky post object, I'll add in a `div` with inline styles. Again, this is not necessarily ideal for a working project, but I want to make each step clear. In reality, you'll probably create custom stylesheets or amend the theme you're working on. At any rate, here's our interaction holder. I've placed this code at the bottom of my previous code inside the jQuery document ready function:

```
...
jQuery('.sticky:last')
.after('<div id="stickyNav"
  style="position: absolute; padding: 10px 0 0 0; margin-top: 280px;
  height: 25px; width: 650px; color: #eee; background: #000;
  text-align: center"></div>');
...
```

And below that code, we'll add some more jQuery which will insert numbers for each sticky post into the `#stickyNav` div we just created:

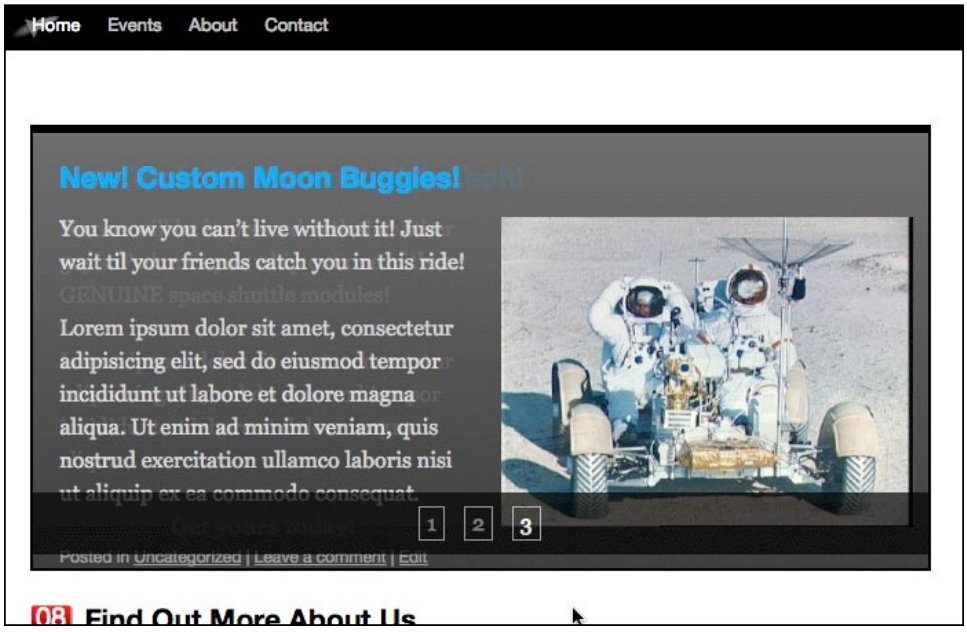
```
...
jQuery('.sticky')
       .each( function (i){
          jQuery('#stickyNav').fadeTo(0, 0.8)
          .append("<div class='sN'
            style='display:inline; margin: 0 5px;
            border: 1px solid #999;
            padding: 2px 5px;'>"+(i+1)+"</div> ");
    });
...
```

This code uses another `each` function, but we only need, and want, it to run once, and append the numbers 1 through 3 (or however many sticky posts we have) to the `#stickyNav` div.

Last, to really finalize this effect, we'll need to dip back inside our `loopStickies` function. Inside the `.animate` function's call back function, we'll add the following code that's in bold:

```
...
jQuery('.sticky')
     .each( function (i){
        /*i = numeric value that will increase with each loop*/

        jQuery(this)
        /*make sure each div is on it's own z-index*/
        .css('z-index','i+10')
        /*using the animate function for timing*/
       .animate({'backgroundColor': '#000000'}, i*duration,
function(){
          jQuery(this).fadeIn('slow');

          //interactivity
          jQuery("#stickyNav .sN").css('color','#666666');
          jQuery('#stickyNav .sN:eq('+i+')').css('color','#ffffff');
        }
        );//end animate
     }); //end each
...
```

Using the `:eq()` selector in the previous code, we're able to target the corresponding number in the interface display and make it stand out compared to the other numbers. This allows users to see how many slides there are, and which slide they're on.



Summary

You're now a pro at handling animations with jQuery! From these examples you can probably recognize several ways to incorporate further enhancements into your WordPress sites. You should now know how to:

- Use animation to direct a user's attention to key information
- Generate animated bar graphs
- Create some really slick, animated page navigation
- Develop rotating sticky posts

Up next, let's take a look at the jQuery UI plugin and the many ways it can benefit a WordPress site.

6
WordPress and jQuery's UI

We're now ready to take a look at jQuery's most popular plugin: UI. UI of course, stands for **User Interface**. The jQuery UI plugin takes many of the most popular tasks that the developers have already made simple through jQuery, and makes them even simpler. I know, it's hard to imagine it getting any easier, but that's exactly what this plugin does. Most importantly, while the enhanced effects are nice, the UI plugin provides interface widgets and an easy way to style or "theme" them without the need for coding up specific interface elements such as tabs, dialog boxes, and more.

In this chapter, we'll:

- Take a look at the UI plugin and how to get started with it quickly
- Learn how to apply jQuery UI widgets to our WordPress site, make it more intuitive, easier to understand content, and encourage users to take action
- Learn how to implement popular UI features and widgets with common WordPress features

Let's get started.

Getting to know jQuery's UI plugin

You can take a tour of the jQuery UI plugin by heading on over to `http://www.jqueryui.com`.

The UI plugin offers a set of standardized widgets, interactions, and effects. Let's take a look at each type of offering in detail.

Widgets

The term "widget" within jQuery is a bit different from a WordPress widget, which is a small plugin designed to sit nicely in a sidebar of a theme. Within jQuery's UI plugin, widgets describe a set of fully-featured, user interface controls that are commonly needed in projects and created by jQuery developers. The UI widgets save jQuery developers a lot of time writing jQuery statements and chaining functions together to create the same interface and effect. Here are the interface widget's jQuery UI offers:

- **Accordion**: This widget expands and collapses content that is broken into logical sections by clicking on the headers of each section. Only one section can be opened at any given time.

- **Autocomplete** (1.8+): This is a new feature available in version 1.8. The **Autocomplete** widgets provide suggestions while you type into the field. The suggestion source is provided as a basic JavaScript array.
- **Button** (1.8+): Also new to 1.8 is the **Button** widget. This lets you take different types of markup and apply the UI's button styling and functionality to it.
- **Datepicker**: The **Datepicker** widget can be applied a standard form input field. Focus on the input field opens an interactive calendar in a small overlay.
- **Dialog**: This widget is an overlay positioned within the page. It has a title bar and a content area, and can be moved, resized, and closed with the 'x' icon by default or by additional button parameters passed to it.
- **Progressbar**: The **Progressbar** widget is designed to simply display the current percentage complete for a process passed to it through a value parameter. It scales to fit inside its parent container by default.
- **Slider**: The jQuery UI **Slider** widget turns objects, such as empty `div` tags into sliders. There are various options such as multiple handles, and ranges that can then be passed to other objects and functions. You can mouse or use the arrow keys to change the slider's position.
- **Tabs**: The **Tabs** widget is used to break content into multiple sections that can be swapped by clicking the tab header to save space, much like an accordion.

Interactions

jQuery UI interactions takes a collection of the more common complex jQuery behaviors that developers need to create, most often for projects, and packages them into convenient and easy-to-use functions as follows:

- **Draggable**: This interaction makes the selected elements draggable by mouse.
- **Droppable**: This interaction works with the draggable elements and makes the selected ones droppable (meaning that they accept being dropped on by the draggable elements).
- **Resizable**: This interaction makes the selected elements resizable by adding visual "handles" to the object. You can specify one or more handles as well as min and max width and height.
- **Selectable**: This interaction allows elements to be selected by dragging a "lasso" or box with the mouse over the elements.
- **Sortable**: This makes the selected elements sortable by dragging with the mouse.

Effects

The main feature is the `.effect()` function, but the standard animation functions and shortcuts that are available in jQuery are enhanced with the jQuery UI plugin's "effects core". This core also includes the ability to color, animate, and also include additional easing options; so, if you include it into your project, you won't need the Color or Easing plugins that we've been working with previously. The jQuery effects comprise:

- **Effect:** This function allows you to assign an effect from a set of 15 to any object.
- **Show**: This enhanced show method optionally accepts jQuery UI advanced effects.
- **Hide**: This enhanced hide method optionally accepts jQuery UI advanced effects.
- **Toggle**: This enhanced toggle method optionally accepts jQuery UI advanced effects.
- **Color animation**: The Color plugin that we learned about in *Chapter 5*, *jQuery Animation within WordPress*, is included into the jQuery UI effects core. Again, it simply extends the `animate` function to be able to animate colors as well.
- **Add class**: Adds the specified class to each of the set of matched elements with an optional transition between the states.
- **Remove class**: Removes all or a specified class from each of the set of matched elements with an optional transition between the states.
- **Toggle class**: Adds the specified class if it is not present, and removes the specified class if it is present, using an optional transition.
- **Switch class**: Switches from the class defined in the first argument to the class defined as second argument, using an optional transition.

jQuery UI plugin versions bundled in WordPress

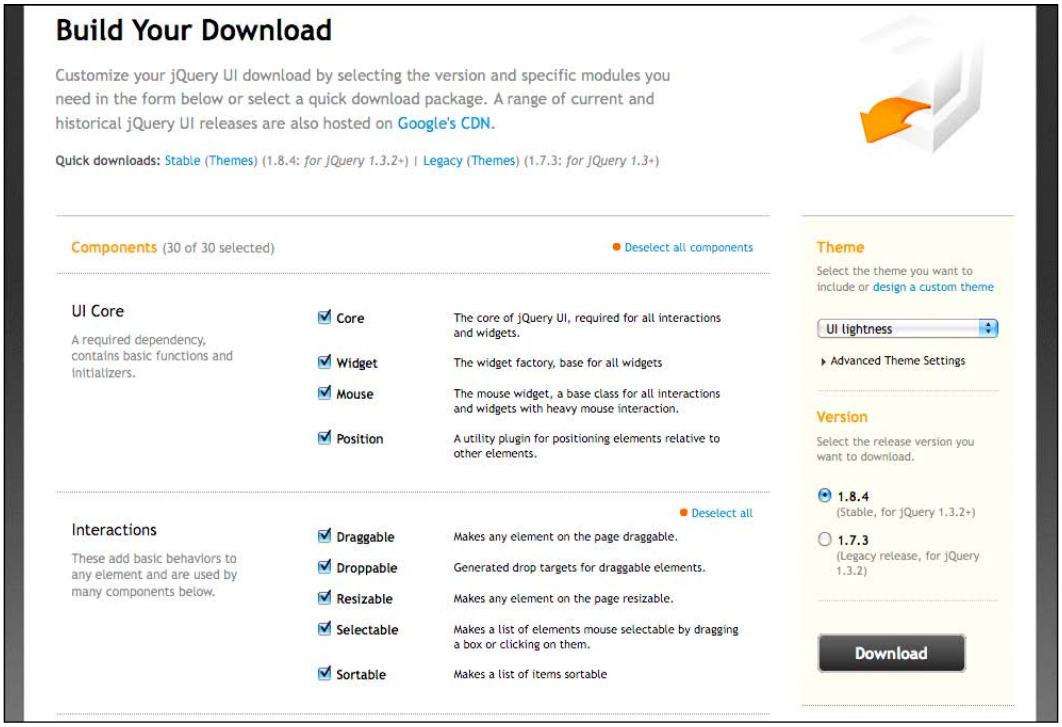
Most of the jQuery UI plugin's main **widget** and **interaction** cores are available bundled into your WordPress installation. If you're using WordPress 2.9.2, you've got jQuery 1.3.2 bundled in and the UI plugin core is 1.7.1 and you've also got the following jQuery UI widgets and interactions available: **Dialog**, **Draggable**, **Droppable**, **Resizable**, **Selectable**, **Sortable**, and **Tabs**.

If you're using WordPress 3.0+, you've got jQuery 1.4.2 bundled in with your installation with the UI core 1.7.3 bundled in. Again, this is with the same widgets and interactions as mentioned in the previous paragraph.

If you'd like to take advantage of the UI plugin's *effects* or, if you're using jQuery 1.4.2 and want to take advantage of the UI plugin's 1.8+ features, you'll need to include a copy of the UI plugin version 1.8+ separately through your own download from the jQuery's UI site or through Google's CDN.

Picking and choosing from the jQuery's UI site

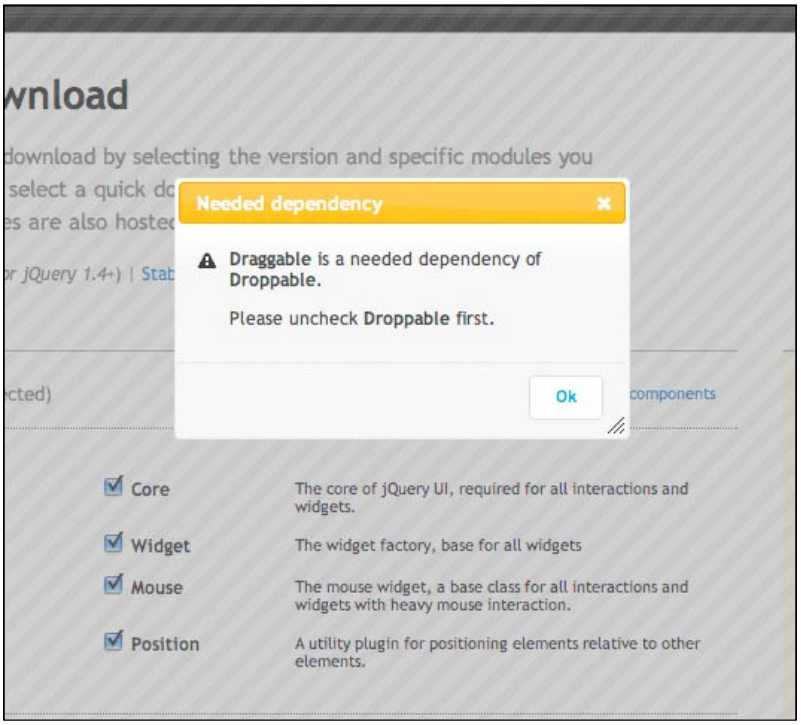
The advantage of downloading from the jQuery's UI site is you can pick and choose only what you need for your project. If you go to the download page at `http://www.jqueryui.com/download`, you'll see on the right-hand side that you can pick version **1.7.3** or **1.8.4** and click on the **Download** button; this will give you everything.



For development purposes, you can just download the whole thing. The ZIP file is over 4 MB but that includes a development bundle directory chock full of examples and documentation; none of that would get loaded into your project.

With all options selected, the actual UI plugin's `.js` file you'll load into your WordPress project is about 200 KB, and you can count on adding about another 100 KB to the project for your CSS theme depending on what you choose from the site or how you rolled your own. If you know exactly what features you're using, you can shave off some kilobytes by only selecting what you want to use.

The download page is great because it won't let you deselect anything that is dependent on another feature you've selected, and that you'd like to use. This is a screenshot of an alert for selecting something you need:



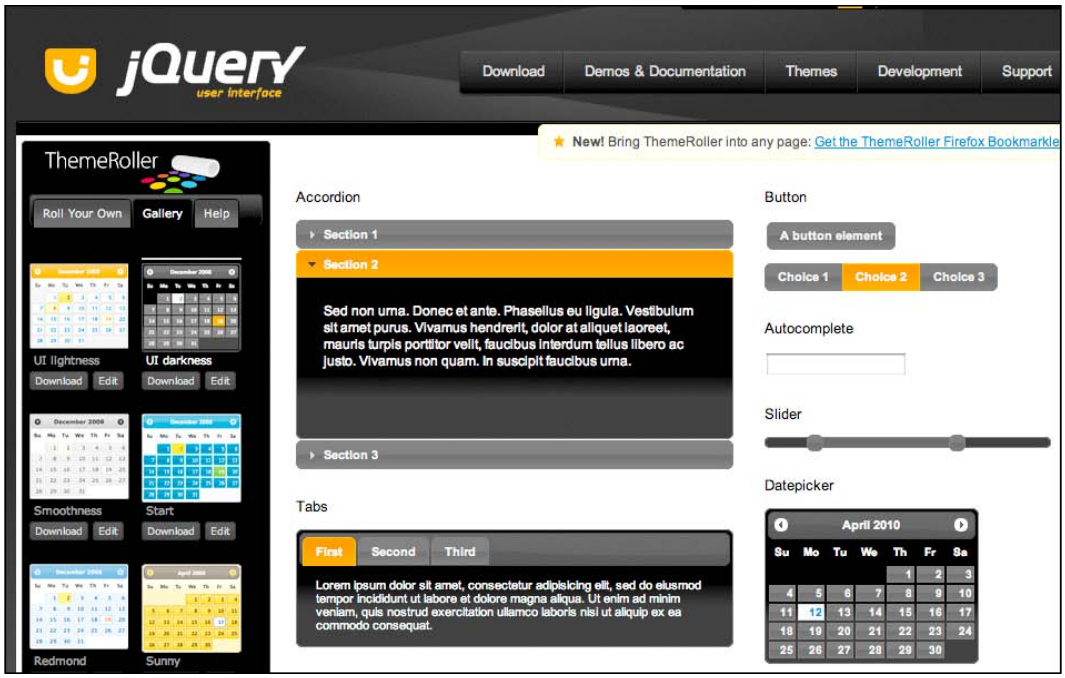
Make sure you download the correct UI plugin version for your version of jQuery!

If your project is using WordPress 2.9.2, the bundled version is of jQuery 1.3.2, so you'll want to make sure you download the UI plugin version 1.7.3. If you're using the Google CDN or your own jQuery download version 1.4+, you can download and work with the jQuery UI plugin version 1.8+.

Making it look right: Easy UI theming

No matter where you're pulling in the UI plugin from, your own download, the Google CDN, or the WordPress bundled UI options, you'll need to provide your own styles for it. You can include one of many great themes into your project or easily "roll" your own to best match your site's design.

On the jQuery's UI site select **Themes** from the navigation bar, or go to: `http://jqueryui.com/themeroller/`.



You can tweak the resulting theme's CSS stylesheet directly or by simply loading the jQuery UI stylesheet up *before* your WordPress stylesheet. Using the WebDeveloper's **Toolbar** or **Firebug** in Firefox, it's very easy to see what styles the UI is producing and overwrite them in your main WordPress stylesheet.

Including the jQuery UI plugin features into your WordPress site

By now, you should be pretty comfortable including jQuery plugins into your WordPress sites. Because *specific components* of the UI plugins are available bundled in WordPress, we'll review getting them into your project in a few different ways.

Including jQuery's UI from WordPress' bundle

The jQuery's UI plugin bundled into WordPress is separated out into individual `.js` files. You'll have to register the UI core file in your project first, as well as each widget or specific interaction that you'd like to include in your project. Again, the only widgets and interactions available are: **Dialog**, **Draggable**, **Droppable**, **Resizable**, **Selectable**, **Sortable**, and **Tabs**.

To register the core in your WordPress theme:

```
...
<?php
  if (!is_admin()) {//checking for is_admin makes sure that the UI
doesn't load in admin
   //adding array('jquery') means the ui-core requires jquery
   wp_enqueue_script("jquery-ui-core", array('jquery'));
  }//end of is_admin
?>
...
```

Then, register a particular widget you want:

```
...
<?php
   if (!is_admin()) {//checking for is_admin makes sure that the UI
doesn't load in admin
   //requires jquery AND the ui-core
   wp_enqueue_script("jquery-ui-dialog",
     array('jquery','jquery-ui-core'));
   }//end of is_admin()
?>
...
```

Just repeat the above code for additional widgets. The widget `.js` file names are as follows:

```
jquery-ui-tabs
jquery-ui-sortable
jquery-ui-draggable
jquery-ui-droppable
jquery-ui-selectable
jquery-ui-resizable
jquery-ui-dialog
```

Again, the full list of bundled JavaScripts for WordPress can be found in the codex: `http://codex.wordpress.org/Function_Reference/wp_enqueue_script`.

Including from the Google CDN

You can include jQuery's UI plugin very similarly to including jQuery via the Google CDN. The UI plugin path is: `http://ajax.googleapis.com/ajax/libs/jqueryui/1.8.0/jquery-ui.min.js`. Note the bold version number here. You can change it to the version of the UI plugin that you require. If you're using jQuery version 1.3.2, be sure to target 1.7.2. If you're using 1.4.2, you can target 1.8.0.

Let's refresh on how to use `wp_register_script` to call up a script available from the bundle from Google's CDN:

```
...
if (!is_admin()) {//checking for is_admin makes sure that UI doesn't
load in admin
wp_deregister_script( 'jquery-ui-core' );
wp_register_script(   'jquery-ui-core', 'http://ajax.googleapis.com/
ajax/libs/jqueryui/1.8.0/jquery-ui.min.js');
/*this brings over the entire 1.8 core and all widgets, interactions
and effects from the Google CDN*/
}//end of is_admin
...
```

You should note that although we're deregistering the bundled `jquery-ui-core` file, what we're loading in from the Google CDN is the *complete* jQuery UI plugin with access to all its widgets, interactions, and effects. It might be wise to add a comment in your code so that the other developers will know that they don't need to register individual widgets and interactions from the bundle into the project.

Loading up your own custom download from your theme or plugin directory

If you've included the UI into your theme or a plugin directory you'll load it up, again using `wp_enqueue_script`, using the following methods:

Including a local copy of the UI plugin from a theme:

```
...
if (!is_admin()) {//checking for is_admin() makes sure that UI doesn't
load in admin
wp_enqueue_script('jquery-ui-1.8.custom.min', get_
bloginfo('stylesheet_directory') . '/js/jquery-ui-1.8.custom.min.js',
array('jquery'), '20100410' );
}//end of is_admin()
...
```

Again, by adding `array('jquery')` at the end of the script, this lets WordPress know that jQuery is required, just in case it hasn't already been registered.

To include a local copy of the UI plugin from a WordPress plugin use the `wp_register_script` function as follows:

```
...
function myPluginFunction(){
   if (!is_admin()) {//checking for is_admin makes sure that the UI
doesn't load in admin
  wp_register_script('jquery-ui-1.8.custom.min',
    WP_PLUGIN_URL . '/js/jquery-ui-1.8.custom.min.js');
       }//end of is_admin
}//end of myPluginFunction()
add_action('wp_head', 'myPluginFunction');
...
```

Don't forget your styles!

No matter where you're pulling the UI plugin from, WordPress, Google's CDN, or your own download, you'll need to include CSS styles for the UI plugin. If you didn't play around with the theme roller earlier, go back now and do so. Select a theme or amend one of the themes with the theme roller or just roll your own from scratch to create widgets that look great with your site's existing design.

Once you've done that, you can take your selected theme or custom rolled theme and place it in your theme or a plugin directory. Make sure to include the images directory that comes with the theme. You can then include it using a direct link into your `header.php` theme file or use the `wp_enque_style` function we've used before to include it into a plugin or your theme through the `functions.php` page:

To include a UI theme directly in your WordPress theme, by linking to it directly, use the following:

```
...
<link rel="stylesheet" href="<?php bloginfo('stylesheet_directory');
?>/js/smoothness/jquery-ui-1.8.custom.css" type="text/css"
media="screen" />
...
```

Include a UI theme into a WordPress theme from the theme's `functions.php` page using `wp_enqueue_style`:

```
...
<?php
function addUIstyles(){
wp_enqueue_style('ui-theme', bloginfo('stylesheet_directory')
  '/js/smoothness/jquery-ui-1.8.custom.css', array('style'), '1.0',
'screen');
}

add_action('init', 'addUIstyles');
?>
...
```

Including a UI theme into a WordPress plugin using `wp_enqueue_style`, is similar to the above example, but be sure to use `WP_PLUGIN_DIR` to target your plugin directory.

```
...
wp_enqueue_style('ui-theme', WP_PLUGIN_DIR .
  .'/js/smoothness/jquery-ui-1.8.custom.css',
  array('style'), '1.0', 'screen');
...
```

Enhancing effects with jQuery UI

You'd think that after taking the time to select a theme or roll our own, we'd jump into putting widgets to use. We will! But first, while animations and interactions are still fresh in our mind from *Chapter 5, jQuery Animation within WordPress* (though, don't worry if you're skipping around), you'll be interested to learn that it's as easy as setting up most of those animations and effects, things can be made quite a few times snazzier with the UI plugin.

First up, these effects *are not bundled* with WordPress at the time of this writing. So, in order to use these UI effects, you'll need to include the UI plugin through your own download or from the Google CDN.

Effects made easy

What the UI plugin does is add in a single new function called `.effect()` that offers, 15 or so, new and slick animation effects. Most notably, `blind`, which rolls things up like a blind; `shake`, which adds a little shake; and `explode`, which manages to "break" the object up and shoots pieces of it out in several directions.

Let's apply the `shake` effect to headers in our posts when we mouse-over them. In addition to registering and/or including the necessary jQuery and jQuery UI plugin files in our WordPress project, you should have also included a `custom-jquery.js` file to your theme to work with. Once you've done that, include the following code:

```
jQuery(function(){
  jQuery(".post h2").hover(function(){
    jQuery(this).effect('shake', 200);
  }, function(){
      jQuery(this).effect('shake', 200);
  });
});
```

You can (sort of) see this effect in action in the following screenshot:

Easing is just as easy

Beyond the `.effects` function, the UI plugin then *extends* jQuery's existing `.animate` function as well as shortcut functions such as `.hide`, `.show`, `.toggle`, `.addClass`, `.removeClass`, and `.toggleClass`. with the great Easing plugin (introduced by *Robert Penner*) we looked at in *Chapter 5, jQuery Animation with WordPress*. So, if you're using the jQuery UI plugin and have included the effects core into your download, there's no need to include the Easing plugin separately into your project.

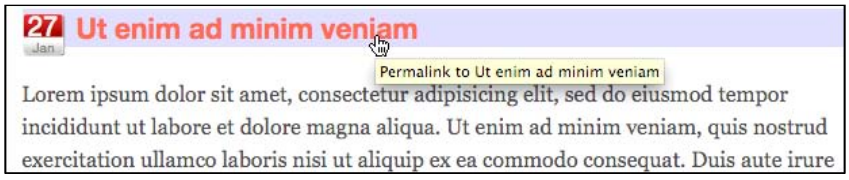
Color animation with jQuery UI

In addition to the Easing plugin included, jQuery UI also has the Color animation plugin already built into it. In *Chapter 5*, *jQuery Animation within WordPress*, we used the Color plugin that was bundled in with our WordPress installation. However, if you're going to use the downloaded version or Google CDN version of the UI plugin anyway, as with the Easing plugin, you just saved yourself the need to use it separately or register it from the WordPress bundle.

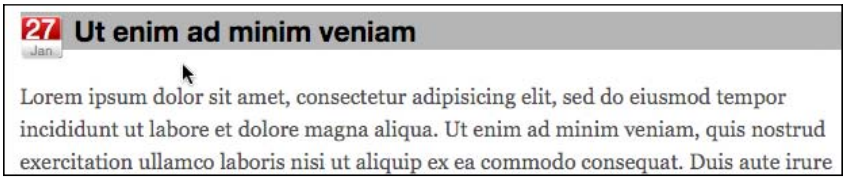
To test it out in our project, which doesn't have the Color plugin registered but is referencing our downloaded UI plugin version 1.8, let's animate the colors of our header text with the `easeOutBounce` easing option applied:

```
...
jQuery(".post h2").hover(function(){
    jQuery(this).animate({'backgroundColor':'#ccccff'}, 2000,
      'easeOutBack');
  }, function(){
      jQuery(this).animate({'backgroundColor': '#999999'}, 2000,
        'easeOutBack');
  });
...
```

It then animates up to a light blue color, as seen in the next screenshot:



Then, back down to grey:



You've noticed that using the color animation and easing features in the jQuery UI plugin's effects core is no different than using it as the separate Color animation or Easing plugins. Again, there shouldn't be any difference other than which version, the standalone plugin or the UI plugin, is more convenient and useful to your WordPress project.

Enhancing the user interface of your WordPress site

We can see that the UI demos at jQueryUI.com certainly look cool, but now that we've got the UI plugin loaded up in to our project, how do we go about really putting these features to use on a WordPress project? Different types of interfaces can help us organize and relate to various types of information more easily and reduce confusion. WordPress' theme API allows for various types of information to be displayed within the site's design in logical chunks, mostly posts and lists. Let's see if we can enhance any of that information with UI features.

We've seen that the UI plugin offers: accordions, tabs, dialog boxes, date pickers, as well as easy ways to implement drag-and-drop and sorting. Also, if you're using the latest version, 1.8 or higher (as the examples in this chapter are), there are cool widgets such as **Autocomplete** and **Button**. Let's pick up another hypothetical client and see how some minor interface enhancements can help their site out.

Project: Turning posts into tabs

You've probably seen tabs being used more and more in sites lately. The main reason for using tabs within your site, is that it allows users to easily see a set of related content, one chunk at a time (this is why "tab style" site navigation is also popular). It also allows you, as a designer, to contain the content into a convenient module, saving valuable screen space.

In our *Chapter 5*, *jQuery Animation with WordPress*, we learned how to stack up sticky posts so they rotated, slide-show style. While animating posts works well with the unrelated content that you want to ensure, everyone gets a glimpse at, loading content up into tabs means the content is somehow related, and yes, you also want to conserve space, perhaps getting that information above the fold so that the user is more likely to take it in.

Your newest hypothetical client has three pieces of information that are related to understanding their company. This content doesn't change much, but he would like the site's users to be able to get an overview of the information, along with the option to download a white paper up front, without scrolling.

The client already has this content on his site. The posts are assigned to a unique category called **Our Structure**. The posts are rather old by now and don't even show up on the site's main page, so the client has been manually linking to the perma-links for the posts in various other pages on the site.

To get started, we decide that it would benefit us to leverage a little help from the WordPress theme.

Setting up custom loops in the WordPress theme

Let's start by going into the client's theme and setting a loop that pulls only from the **Our Structure** category. Then, using jQuery UI we'll display those posts in a set of tabs that is viewable mostly "above the fold" ensuring site visitors get an overview of the organization's most important information up front and general post items will flow below.

First up, in the `index.php` page, we'll create a new loop, above the existing `loop.php` include that only displays the **Our Structure** category. Before we do this though, we'll head over to the jQuery UI site and take a look at the demo of how tabs are set up: `http://jqueryui.com/demos/tabs/`.

Essentially we see that demo tabs have a `ul` that lists the titles, wrapped in `href` calls to `id` anchors that point to the content's `div`. This means our theme actually will require *two* custom WordPress loops to accommodate this widget.

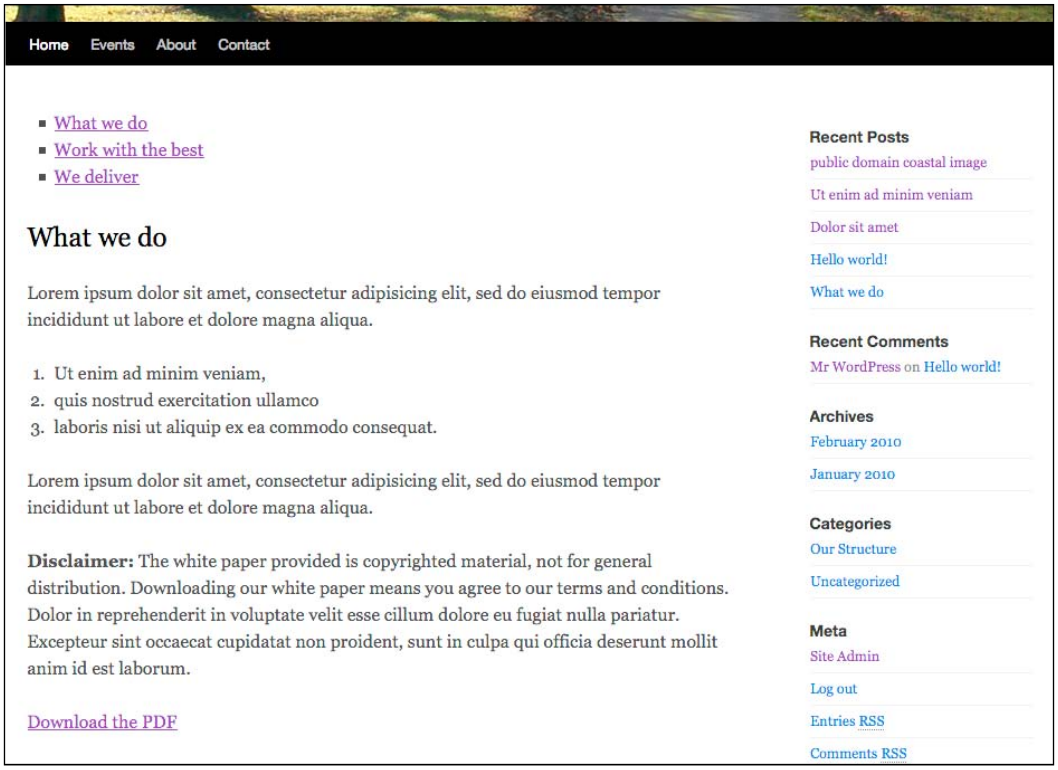
We'll set them up in our `index.php` template file, right above our main content `loop.php` include, *inside* the `#content` div in the theme we're using, which is the default theme. The first loop will set up our custom `#ourStructure` div with the `ul` list of titles:

```
...
<div id="ourStructure">
  <ul>
 <?php//start custom loop
      //get posts in the proper category
     $postList = get_posts('category=4');
     foreach($postList as $post):
        setup_postdata($post);
        ?>
        //set up a list item with a unique anchor link
        <li>
          <a href="#post-<?php the_ID(); ?>">
          <?php the_title(); ?></a>
        </li>
 <?php endforeach; ?>
    </ul>

    <!--//second loop goes here-->

  </div><!--//end of ourStructure-->
...
```

Next, under the previous loop, but still inside the `#ourStructure` div, we'll run the loop one more time, focusing now on the post's titles and content as follows:

```
...
<!--//second loop goes here-->
   <?php
   //again, call correct category
   $postContent = get_posts('category=4');
   foreach($postContent as $post):
      setup_postdata($post);
      ?>
      //assign a unique ID to div
      <div id="post-<?php the_ID(); ?>">
        <h2><?php the_title(); ?></h2>
         //add content:
        <div class="entry">
        <?php the_content('Read the rest of this entry &raquo;'); ?>
    </div>

      </div>
<?php endforeach; ?>

</div><!--//end of ourStructure-->
...
```

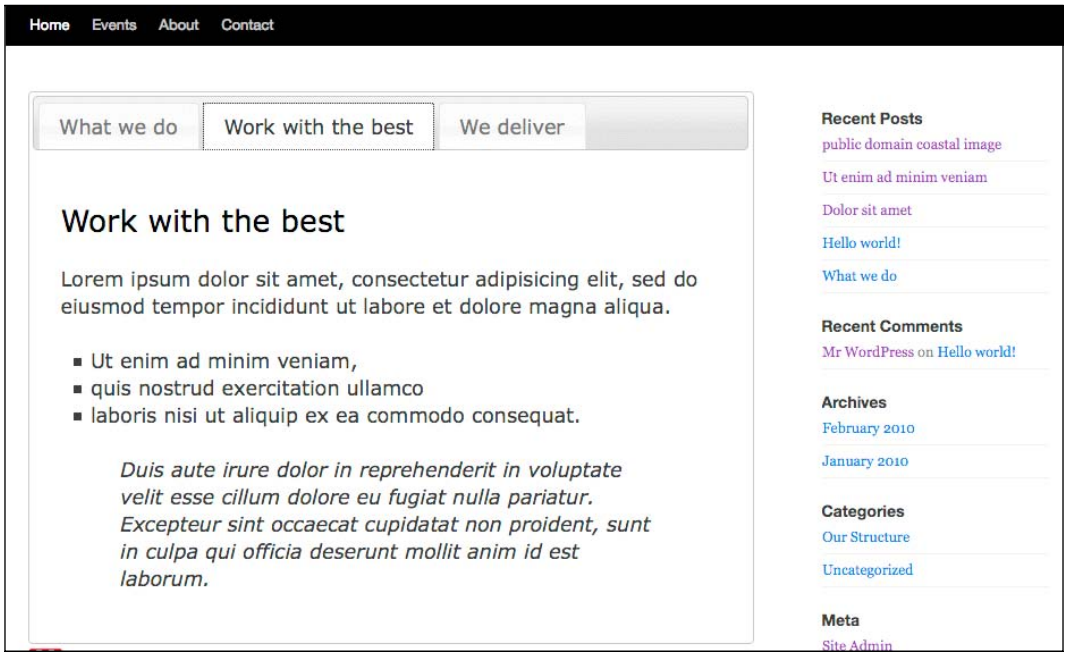
This gives us a result that looks like the next screenshot. While not super pretty, it's functional, and it certainly gets that important info up there and allows the user to link down to the `id` instance's anchor name.



We'll then enhance that markup with jQuery's UI tabs like so, by targeting the `#ourStructure` div, in our `custom.js` file we set up the following jQuery statement:

```
...
jQuery("#ourStructure").tabs();
...
```

Yes. Hard to believe, but thanks to the flexibility of WordPress and the work we got the theme to do for us, that's *all* the jQuery we need!



Not bad! The content is now contained up top using the jQuery UI theme we chose, called "Smoothness" to compliment our WordPress theme best (again, we're using the default WordPress theme that comes with 3.0 as of the writing of this book). Let's look at some other uses for the UI plugin.

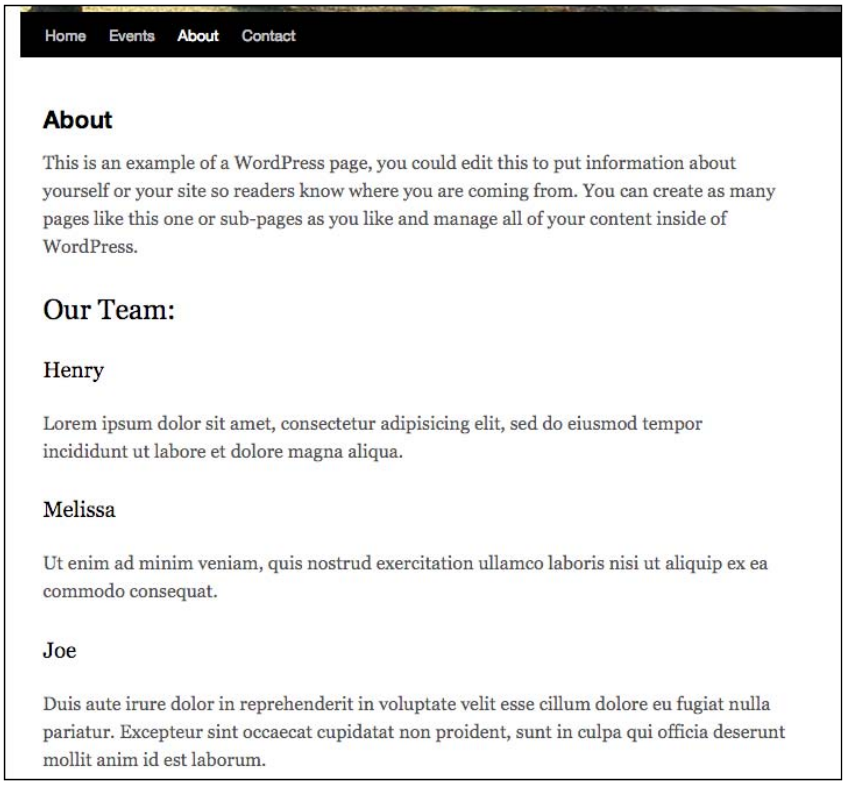
Implementing tabs entirely with jQuery

We achieved the above tab scenario by tweaking the WordPress theme to include a `ul` list of titles in HTML and then the post content within div tags below. This worked well as it generated a `ul` list with `href` links to anchor names that would still present the content and work functionally in a non-JavaScript enabled browser.

However, for other situations where WordPress is already presenting the content you need (for example, a list of `h2` or `h3` headings and content already tucked inside a *single* post or page), or you just don't have access to edit the theme, it might be easier to generate the DOM objects needed for the UI `.tab` feature by applying a little jQuery beforehand.

For a list of `h3` headers and `p` paragraph tags added to a single page or WordPress post, we can still wrap that content in the UI tab widget.

The next screenshot depicts the **About** page, which already has all the content inside it; we just need to "massage" it to best meet the jQuery UI tab requirements:



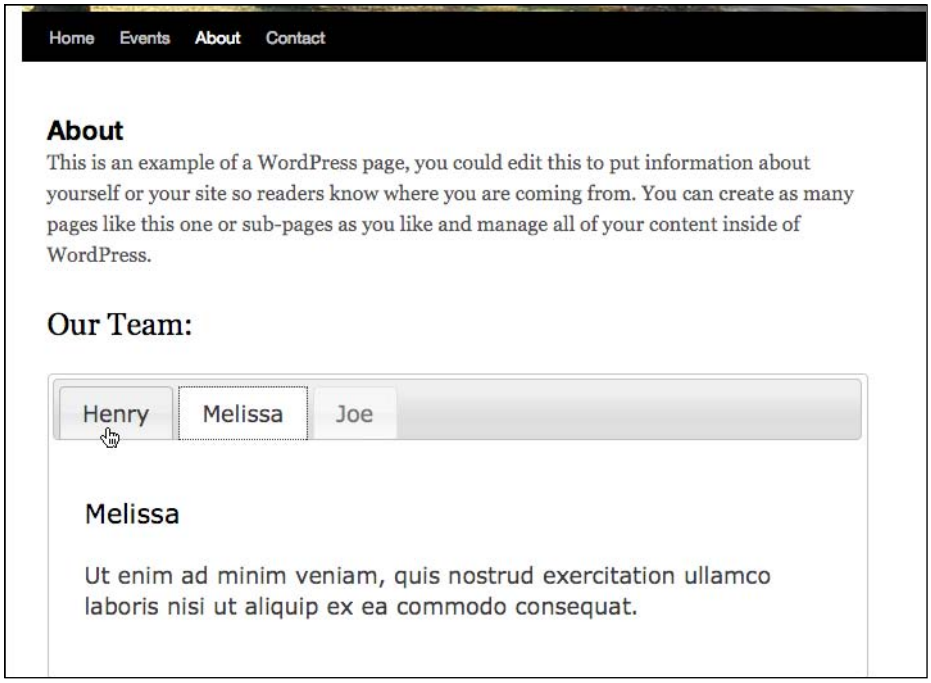
First, we'll target the specific page. WordPress can output unique IDs to pages as well as a host of class names; you'll have to **View Source** on the HTML output of your WordPress theme to the browser and see if the theme leverages this feature (most good WordPress themes will). This ability can help us target only the content we want to affect. For example, if all we want to enhance is our **About** page, we can view source and see that the post's unique ID is `#post-104`. This allows us to target the post we want to add tabs to, by first prepending a `ul` list of `h3` titles.

Once we have the `ul` list, we'll need to wrap everything in a new, selectable div with an ID of `#aboutUs`. Then, we'll cycle through each `h3` item to create individual `li` list items with anchor links and wrap each following `h3` and `p` tag with an anchor-named `id` div of their own.

Read the bold comments in the code to follow along:

```
...
//add in a ul list on the About page only, before the first h3
  jQuery("#post-104 h3:first").before("<ul></ul>");

//select the ul, the h3's AND the h3's p tags
//and wrap them in a new div
  //use the .add() function to make sure everything is selected
  jQuery("#post-104 ul").add("#post-104 h3")
    .add("#post-104 h3+p").wrapAll("<div id='aboutUs'></div>");


//for EACH h3 item:
  jQuery("#post-104 h3").each(function(i){
      //add text to the ul list w/ anchor links
      var titleTxt = jQuery(this).text();
      var htmlTxt = "<li>
        <a href='#name-"+i+"'>"+titleTxt+"</a></li>";
      jQuery("#post-104 ul").append(htmlTxt);

      //wrap each h3 AND p in a div with anchor names
      //this time, use .andSelf to make sure everything is selected
      jQuery(this).next("p").andSelf()
        .wrapAll("<div id='name-"+i+"'></div>");
  });

  //remove .entry class so list items don't have right quotes
  //this is a list style in the default theme
  jQuery("#post-104 .entry").removeClass('entry');


  //Last, create the tabs widget
  jQuery("#post-104 #aboutUs").tabs();
...
```

Refreshing the page now displays this:



Again, the more you understand about your WordPress theme and jQuery, the more power you have to decide which route is quicker or better in terms of deciding whether to manipulate the theme to aid your jQuery enhancement, or if it's better to just use pure jQuery.

Project: Accordion-izing the sidebar

Accordions pretty much have the same functionality as tabs. Mostly they're just vertical rather than horizontal. As with tabs, you'll want to use them to "group" similar information together into a tidier space, allowing the site user to take in the information in logical chunks and not have to wander down through the site or scroll.

In the default theme that we've been working with, our page navigation on the sidebar has some information that we'd like people to be able to see at a glance and not have the headings pushed down past the fold where they may miss them. By grouping sections into accordions that drop down and display additional information and links, we save some room and ensure when a page loads that users can at least, see the important organizational headers and know that there is more information they may want to expand and view.

The accordion widget works great with lists, which is what the sidebar is. The widget also, as you can tell by the example code at `http://jQueryUI.com/demos/accordion`, recognizes and works with headers and paragraph or `div` tags set in a consistent, hierarchical order. You can also use various options to set specific DOM objects as headers and navigation elements.

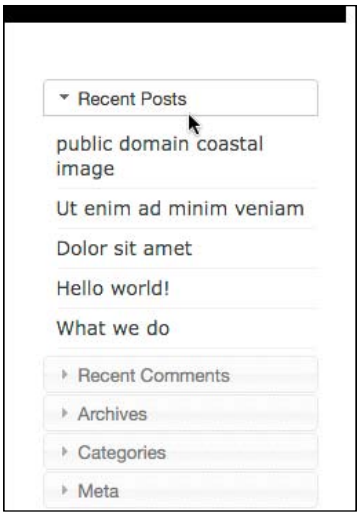
Our default theme's WordPress sidebar is one big `ul` list inside a `div`. Perfect for the accordion widget, but since we set up some custom CSS to make the page list display more like navigation buttons, we want to target the next two lists in the list *below* the page navigation list items. Not to worry, it's easy to target and select the following list items and apply the accordion widget to them as follows:

```
...
//select the proper li level and exclude the inner ul lists then wrap
in a targetable div
  jQuery(".xoxo ul li:gt(10)").not(".xoxo ul li ul li")
    .wrapAll('<div id="sideAccordion"></div>');

  //select the new target and assign the widget
  jQuery('.xoxo').accordion().css({'marginTop':'30px'});
...
```

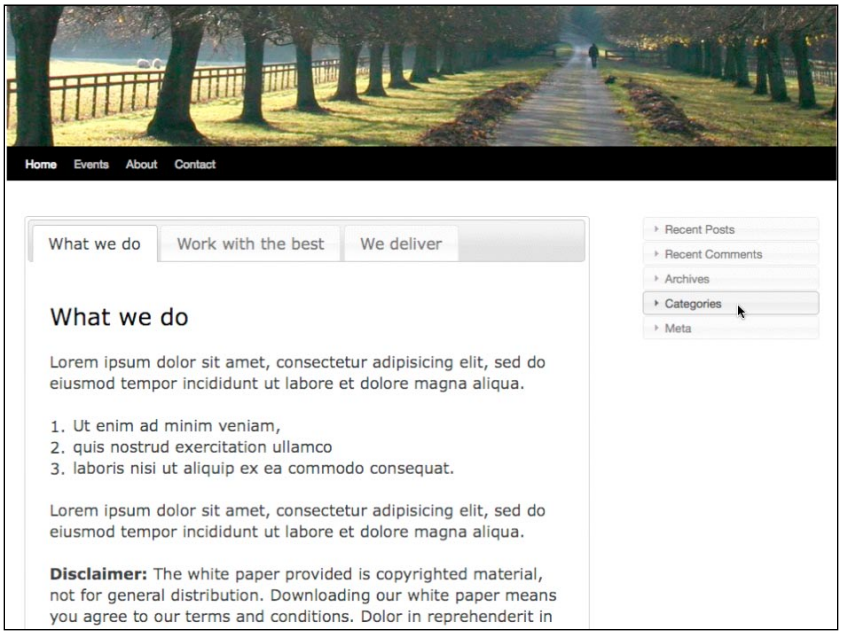
The widget's default state is to display the top accordion open. The client would like it to be completely closed. To achieve this, we'll add some parameters to the widget, including `active: -1`, which is normally used to select which bar to open, but by setting it to `-1`, they'll all be closed:

```
...
jQuery('.xoxo')
      //options for the accordion
      .accordion({header: 'h2', collapsible: true, active: -1})
      .css({'marginTop':'30px'});

  //last, some extra styles to the headers and ul lists
  //to line them up
  jQuery(".xoxo h3")
    .css({'padding':'5px 0 5px 25px', 'height':'15px'});
  jQuery(".xoxo ul").css({'height': 'auto', 'margin': '0px',
    'paddingLeft': '25px', 'paddingTop': '5px',
    'paddingBottom': '5px'});
...
```

Our sidebar under our page navigation is now accordion-ized in a nice style that matches our page's tabs.



These accordion headers are closed when the page loads, making it easy for the site user to chose which one to explore.



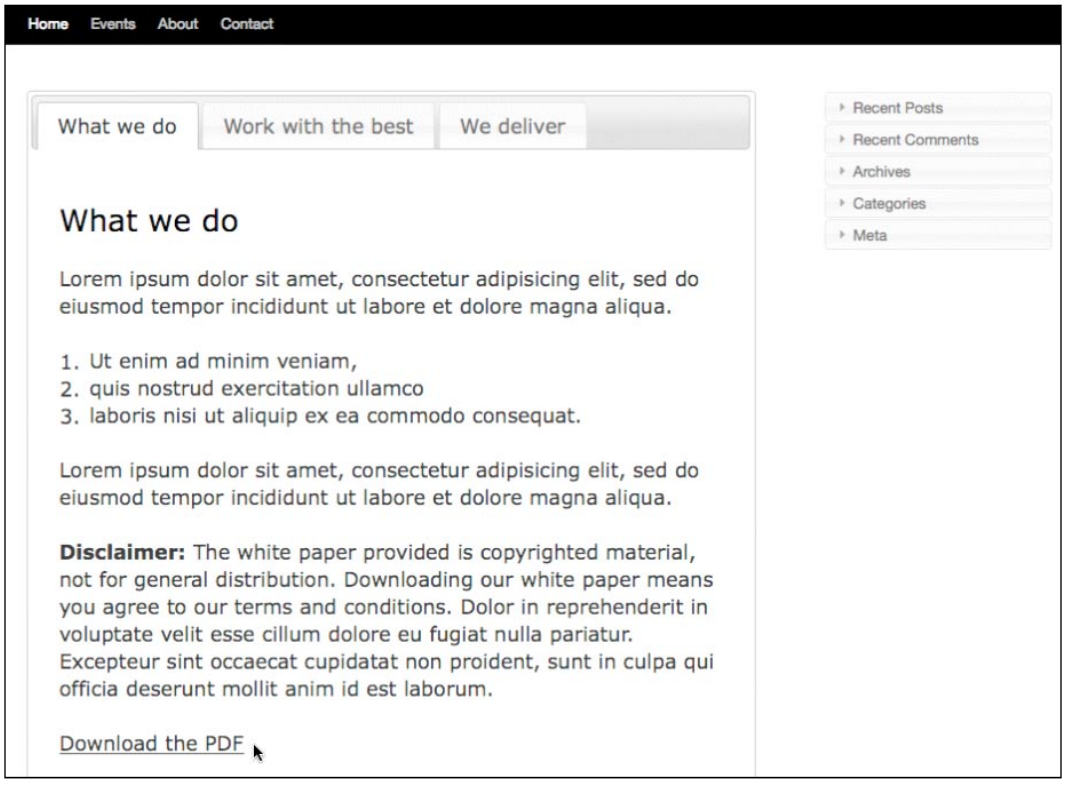
Let's now move on to making our client's last enhancement.

Project: Adding a dialog box to a download button with icons

Dialog boxes are great ways to alert and direct people's attention to really important information, making sure they understand the next steps that they need to take, as well as confirming an action.

Our client is very happy with the tabbed information on the home page and the condensed accordion side bar. They just need one more enhancement. The first tab on the home page offers a PDF download of a white paper that contains information about their methodology, products, and their various uses. As you can see by the next screenshot, the client wants users to understand they're downloading copyrighted information and that the document can not be freely distributed.

As you can see in the following screenshot, they've placed some disclaimer language right before the download link to the PDF file:

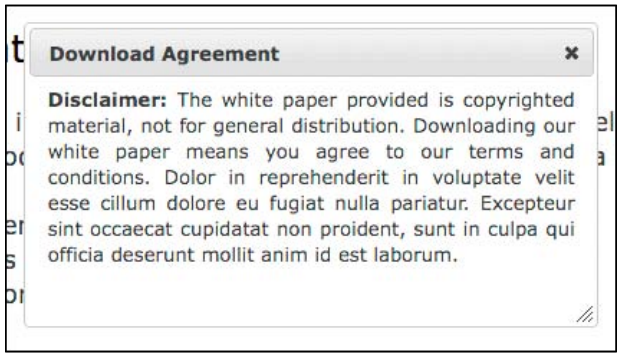


In general, that's all their legal department claims they need, but they'd like it to be a little more clear. We can enhance this download process further by making the download link more prominent using the button widget, and turning the previous **Disclaimer** text into a dialog box. The user will then have to select **I Agree** on the dialog box to get the PDF download to continue and the client can rest assured that the majority of the people downloading their white paper through a JavaScript-enabled browser are definitely aware of the disclaimer.

First, let's set up that **Disclaimer** text to go inside our dialog box. We'll target the paragraph and apply the dialog widget as follows:

```
...
//select p that contains the disclaimer text
  jQuery("#post-98 p:contains(Disclaimer:)")
    .wrapAll("<div id='disclaimer'></div>");

//create the disclaimer dialog widget
jQuery("#disclaimer").dialog();
...
```

If you reload your page, you'll see that the **Disclaimer** text now appears in a dialog box as follows:



The dialog box's default is to align the text "center". This is great for one line of text, but our paragraph looked a little strange so we've added a style to our `.wrapAll` HTML as follows:

```
...wrapAll("<div id='disclaimer' style='text-align:justify'></
div>");...
```

Next, we really don't want the dialog box to appear immediately, so we'll set its option of `autoOpen` to `false`. We also want confirmation buttons to appear, as well as a title in the dailog's top bar. The dialog widget can also accommodate buttons, so we'll add them in, along with their functionality as follows:

```
...
  //create the disclaimer dialog widget
  jQuery("#disclaimer").dialog({
     //set the dialog to close
       autoOpen: false,
      //set the title
       title: 'Download Agreement',
      // set up two buttons
       buttons: {
           //activates the URL placed in the a href
           "I Agree": function() {
                //get the URL of the PDF
              var pdfFile = jQuery("#post-98 a").attr('href');
              //direct the browser to that URL
              window.location.href = pdfFile;
                },
            //close the dialog box
            "Close" : function() {
                   jQuery(this).dialog("close");
                     }
              },
   });
...
```

The above works great—or at least we think it does. Now that the dialog's `autoOpen` option is set to `false`, we can't tell! We'll need the **Download PDF** link to kick-off the dialog box and while we're at it, we'll need to make sure that the link's `href` doesn't kick-off the PDF download.

If you've been paying attention, you're probably ready to use the `.removeAttr()` function to remove the `href` attribute from the link and render it powerless. That's a good idea; however, in the previous code snippet, we reference the `href` attribute of the link. That reference doesn't kick-off until after the box has appeared, which would be *after* we removed it from the object, which means our `window.location.href` JavaScript won't have a clue where to go.

Our best bet is to use another great function called `preventDefault()`, which will leave all the attributes of the link intact, but prevent it from acting like a clicked link. Let's add in this new link functionality:

```
...
 jQuery("#post-98 a")
      //set up a click function on the link
      .click(function(event){
        //open the dialog box
        jQuery("#disclaimer").dialog("open");
        //ensures that the link to the href is disabled
        event.preventDefault();

   });
...
```

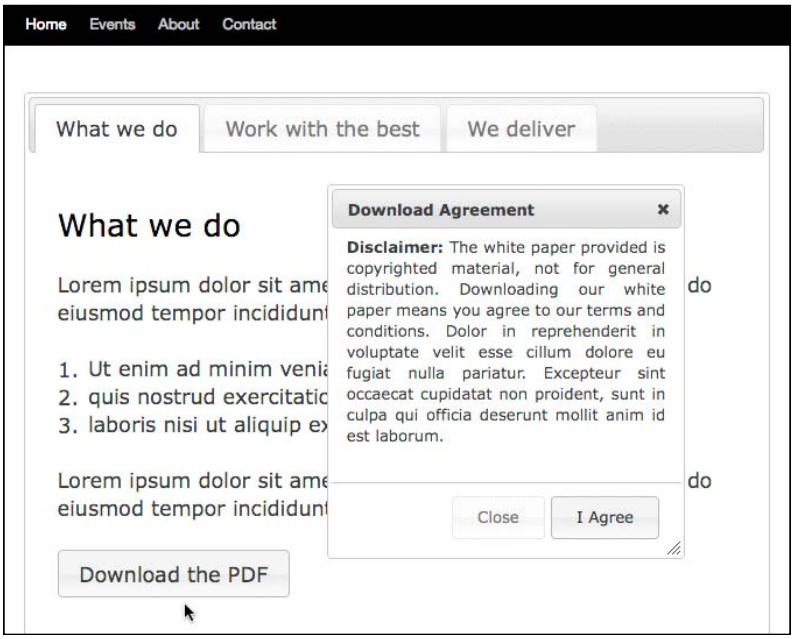
Last, before we refresh our page and take a peek, let's go ahead and make the PDF download link look a little more "clickable". Because we're using jQuery version 1.4.2 from the Google CDN, and the 1.8 version of the jQuery UI plugin, we can do this by selecting the link and adding button widget to it.

If you're not using version 1.8 of the UI plugin, this step is optional. You can simply use CSS styles or the `.css()` function to style the link to your liking.

We'll simply **chain** the `.button()` widget function on to our existing link selection, *after* the `.click()` function as follows:

```
...
jQuery("#post-98 a")
      //set up a click function on the link
      .click(function(event){
        //open the dialog box
        jQuery("#disclaimer").dialog("open");
        //ensures that the link to the href is disabled
        event.preventDefault();

        })
      //add the button widget
      .button();
...
```

You can refresh your page and check out the new button, as shown in the next screenshot:



As great as the button-ized link looks, it doesn't take much to go one step further and add a few icons so it's clear what clicking on the button will get people, and encourage them to take action.

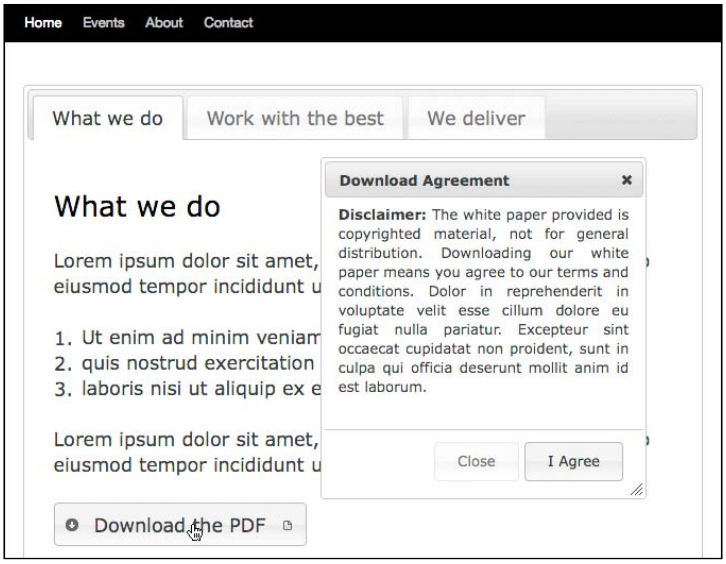
The jQuery UI plugin themes come with a host of **framework icons**. If you included the `image` directory relative to your jQuery UI stylesheet, you have access to them.

The button widget allows for icons to be placed in a "primary" and "secondary" position. The primary position is to the left of the button, and the secondary is to the right, after any button text. Let's add the "circle-arrow-s" icon and the "document" icon to our button as follows:

```
...
 jQuery("#post-98 a")
      //set up a click function on the link
      .click(function(event){
        //open the dialog box
        jQuery("#disclaimer").dialog("open");
        //ensures that the link to the href is disabled
        event.preventDefault();
```

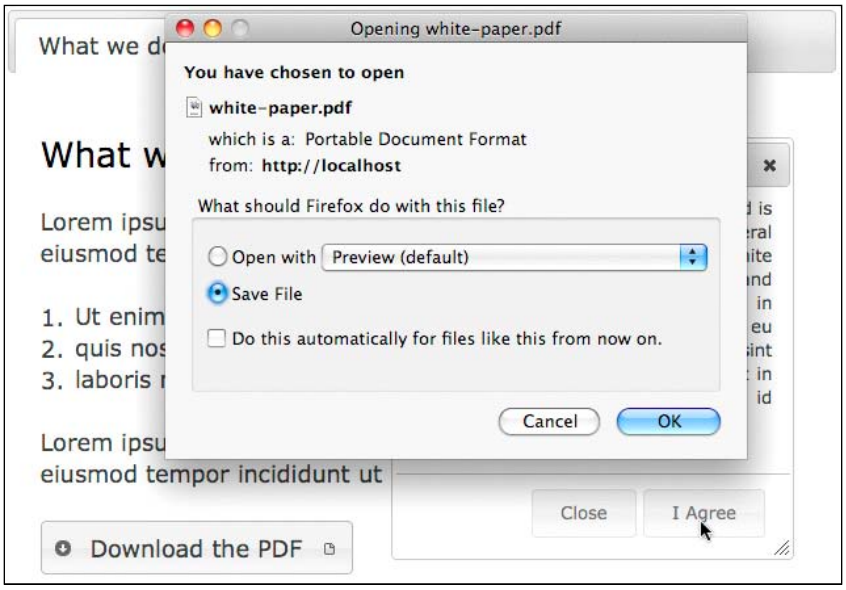
```
        })
      //add the button widget
      .button({
         //add the icons
         icons: {primary:'ui-icon-circle-arrow-s',
                 secondary:'ui-icon-document'}
      });
...
```

Here's our "iconic" button and dialog box once people hit the button:



Want to find out what icons are available for widgets? Check out the theme roller: `http://jqueryui.com/themeroller/`. At the bottom of the page you'll see all the framework icons. Rolling over them will display their `title` tag info which contains the name you want to place in your jQuery statements to reference them.

The last thing to test with this enhancement, is that clicking on **I Agree** kicks-off the download, which as you can see by the following screenshot, works!



This is actually an impressive amount of interactivity to add to a site, and yet at the same time, it degrades and works just fine the way it was without JavaScript. It's a really great use of jQuery and the jQuery UI plugin.

Summary

That's our look at the jQuery UI plugin and just a few of the ways it can really benefit a WordPress site. There are dozens, possibly hundreds of them and more, it just depends on your site or project and its needs.

Remember, jQuery runs on the client-side, in the browser, and WordPress serves up the finished HTML pages to the browser. This means that you have the power to not only enhance WordPress content, but also most WordPress plugins, such as cforms II, and most sidebar widgets should be easy to enhance with jQuery and the jQuery UI plugin.

In this chapter, we had a look at:

- The UI plugin and various ways to include it and get started with it in WordPress
- Understanding how applying UI elements to our WordPress site makes it more intuitive, easier to understand, and encourages users to take action
- Common ways to implement popular UI features with common WordPress features

Let's now move on to the next chapter and see about using jQuery to help us create AJAX interactions.

7
AJAX with jQuery and WordPress

AJAX is an acronym that *Jesse James Garrett*, a user-experience expert who founded `www.AdaptivePath.com`, coined back in 2005. It quickly morphed into a buzzword who's descriptiveness (and verby-ness) as we'll see, goes way beyond its actual acronym definition. We'll take a quick look at what AJAX really is and how easy it is to implement, not to mention cook up a few more cool solutions for our "hypothetical" clients.

In this chapter, we're going to take a look at:

- The basics of using jQuery's AJAX `.load()` function and the more robust `.ajax()` function
- Working with JSON and hooking into other site's APIs
- Creating a custom AJAX enhanced home page and comment form
- Refining that functionality using animation and events

Let's get started by taking a look at what jQuery does for AJAX.

What AJAX is and isn't: A quick primer

To start, if you're new to AJAX, I'll just point out that **AJAX** is actually *not* a technology or language! The acronym stands for **Asynchronous JavaScript and XML**. It's the *technique* of using JavaScript and XML to send and receive data between a web browser and a web server. The most obvious (and cool) use of this technique means you can dynamically update a piece of content on your web page with a call to the server, without forcing the entire page to reload.

The implementation of this technique has made it obvious to many web developers that they can start creating advanced web applications (sometimes called **Rich Interface Applications**(**RIAs**)) that work and feel more like desktop software applications, instead of like web pages.

As eluded to above, the word AJAX is starting to have its own meaning (as you'll also note its occasional use in this book and others, as well as all over the web as a proper noun: "Ajax", rather than an all-cap acronym). For example, a web developer using predominately Microsoft technology may develop their site using a browser scripting language called VBScript instead of JavaScript, to sort and display content transformed into a lightweight data format called JSON instead of XML. You guessed it, that developer's site would still be considered an AJAX site, rather than an "AVAJ" site (let's face it, AJAX simply sounds cooler).

In fact, as we noted in *Chapter 5*, *jQuery Animation within WordPress*, it's getting to the point where just about anything on a website (that isn't in Flash) that slides, moves, fades, or pops up without rendering a new browser window is considered an "Ajaxy" site. In truth, most of these sites don't truly qualify as using AJAX and if you use just a few of the jQuery examples from this book in your WordPress site, it will probably be considered Ajaxy, despite not calling asynchronously to the server. But after this chapter, it will.

AJAX: It's better with jQuery

In the past, when writing up introductions to AJAX or going over the pros and cons of using AJAX with my clients for their projects, I used to give long, in-depth disclaimers and warnings for using AJAX techniques: regaling tales of worst-case scenarios and horror stories of lost browser functionality, not-to-mention ruined accessibility for special needs users. While some of those concerns are still valid, much of the "implementation dread" has pretty much ended with jQuery.

As with all things jQuery that we've learned so far, the point is to create great *enhancements* that degrade gracefully down to basic, working HTML functionality. You'll find the same holds true for AJAX techniques so long as they're thoughtfully implemented with jQuery. If the core content or functionality of your site can be accessed and retrieved without JavaScript enabled in the browser, you'll find that all your users, no matter what their browser or accessibility requirements are, should be able to enjoy your content and effectively use your site. The majority of your users will get to use your site with slick, visually appealing enhancements that make the site easier to use and can even aid in understanding the content.

Assessing if AJAX is right for your site—a shorter disclaimer

Sure, accessibility and compliance aside, there are still some considerations to make for your site's users. Most notably, as you start to realize the power that AJAX techniques can bring to your site, you'll want to make an effort to stay within the *conventions of standard web practices*. Essentially, most web users expect web pages, even really cool web pages, to simply act like web pages!

That doesn't mean you can't break standard conventions, especially if your site is more of an RIA than a pure content site. Just make sure that you inform your users of what to expect. For example, if the navigation panel is not at the top of the site or sidebar, you'll need to find some way to tell people up-front where it is and why you think it's more conveniently located where you put it. If you use a different indicator other than underlines and button boxes for click-able objects, tell people what to look for so they know what's click-able and what's not.

With that said, let's take a look at what our latest crop of hypothetical clients have to ask of us and get to work.

Getting started with jQuery's AJAX functionality

At the heart of jQuery's AJAX functionality is the `.ajax()` function. This little guy allows you to do some heavy lifting and has everything you need for all your **XML HTTP Requests** (**XHR**) needs.

For those of you with a little AJAX experience under your belts, you'll be pleased to find that in true jQuery form, this function eliminates the need for setting up the traditional `if/else` statement to test for support for the `XMLHTTPRequest` object and if not then, the `ActiveXObject` (for IE browsers).

Using the .ajax() function

Let's take a quick look at *some* of the functionality available in the `.ajax` call:

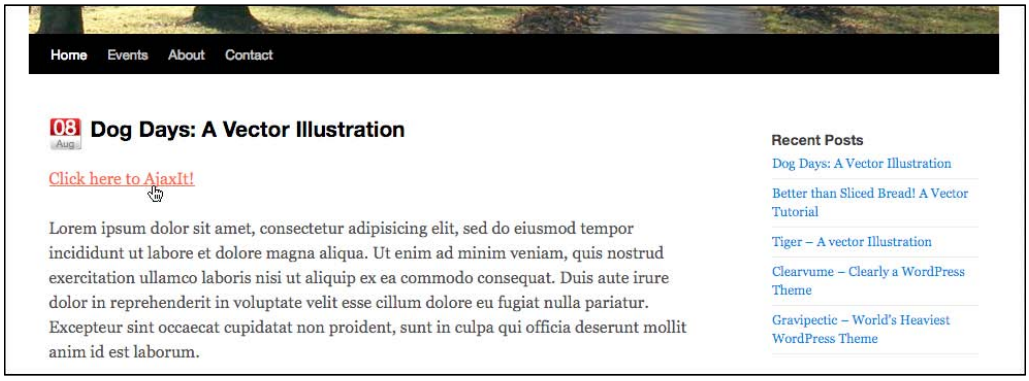
```
jQuery.ajax({
type: //"GET" or "POST",
url: //"url/to/file.php",
dataType: //"script", "xml", "json", or "html"
data: //a query string "name=FileName&type=PDF"
beforeSend://a callback function
```

```
        function(){
          alert("Starting Request");
         }
    success: //a callback function
         function(){
            alert("Request successful");
         }
    complete: //a callback function
        function(){
            alert("Request complete");
         }
    error: //a callback function
        function(){
            alert("Request returned and error!");
         }
    });
    ...
```

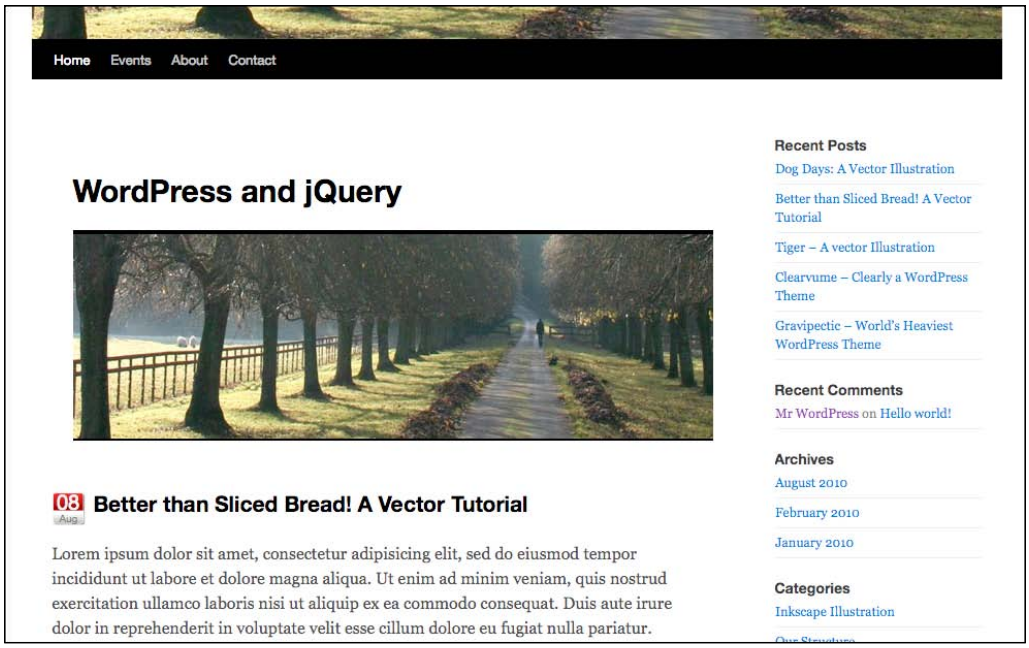
For example, implemented within WordPress, an `.ajax()` call might look something like this:

```
    ...
    jQuery(".ajaxIt").click(function(){
      //.ajaxIt is a class assigned to link in the first post

        jQuery.ajax({
            //url to the about page:
          url: "/wp-jquery/about/",
          data: "html",
          success: function(data){
          //limit the overflow and height on the first post
                 jQuery('.post:first')
                   .css({overflow: "hidden", height: "310px"})
                     //add in the data
                   .html(data);
                   //alert just shows the function kicked off
                  alert('loaded up content');
                 }
        });

      });
    ...
```

In the given code, when the user clicks on the `.ajaxIt` object jQuery selector, as seen in the next screenshot, the `.ajax` function loads the *whole* **About** page into the first post's `.post` div:



By changing the CSS properties on the div to hide the overflow and set the height, we can keep it from looking *too* messy:



There you have it! Your first use of AJAX within WordPress! However, you're probably thinking: "That's a fair bit of work for something that I'd never really want to do in real life. (Reloading in the whole site into a div *including* the header? Yuk!)"

You're right. Let's take a look at shortcutting-in some more accessible and useful functionality.

Taking shortcuts

You can see the `.ajax()` function is quite robust and flexible. As cool as that is, you're probably already hoping for a shortcut. Never fear, similar to the `.animate()` function we've already worked with, jQuery has nicely broken down a few of the more "routine" tasks into bite size functions that are much easier to use and leverage. Here are the most important for WordPress users:

- `.load`—you can call through POST and GET with this function and pull specific, jQuery-selected content and tuck it a lot more easily into other jQuery selected areas.
- `.get`—like `.load`, but only does get requests.
- `.post`—like `.load`, but focuses on post requests.
- `.getJSON`—allows you to pull JSON data (this is a good way to go if you're cross site scripting—that is, pulling data in from another URL, such as `twitter.com` for example).
- `.getScript`—allows you to kick off the actions tucked in a script that's not attached to your WordPress theme. (Very useful if you want to add functionality that you don't want other people to be able to easily find and comb through, and you can also pull in JavaScripts from other domains to work with.)

In most WordPress projects, you'll find that you won't need to use the `.ajax()` function at all. You'll use `.load`, `.post` or `.get`, sometimes `.getJSON` or `.getScript`. But, like the `.animate()` function, you'll occasionally come up with scenarios where the flexibility and *granular control* of the `.ajax` function is handy.

The most useful of all of these shortcut functions and the one we'll focus on the most is the `.load` function.

Specifying where to .load() it

We can achieve the exact same effect we got from our full `.ajax()` function with the parred-down code here:

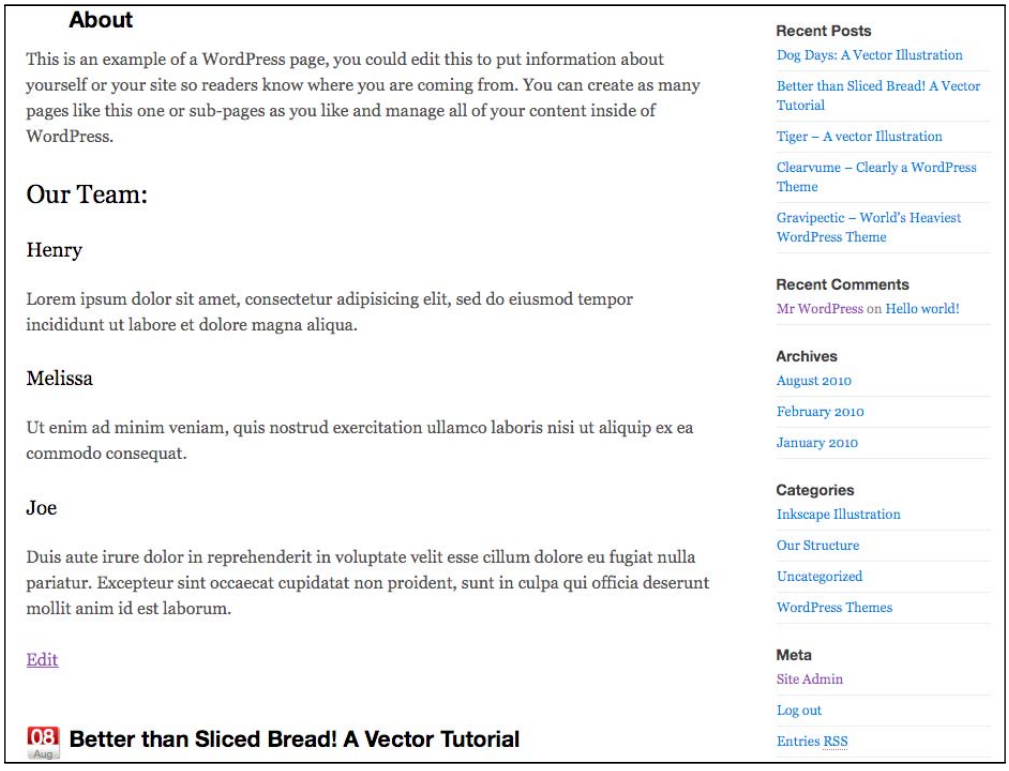
```
...
jQuery('.post:first').css({overflow: "hidden",
  height: "310px"}).load('about-2/');
...
```

Again, kinda cool, in that the code snippet is a lot simpler. It's AJAX; the page itself isn't reloading, but why would you *want* to do that? (Again, to keep the example from being too messy, I used the `.css` function to change the CSS properties and hide the overflow and lock the height of the `.post` div.)

It does seem rare that this would be useful for a project (and if it was useful, an `iframe` would achieve the same effect). What we really want to do is be able to load in *key pieces* of content from another page into our current page. The good news is, we can achieve that easily:

```
...
jQuery('.post:first').load('about-2/ #post-104');
...
```

By extending the `url` parameter of the `.load` function, the given snippet of code will replace our *first* `.post` div with content from the `#post-104` div on the **About** page. The result is this:

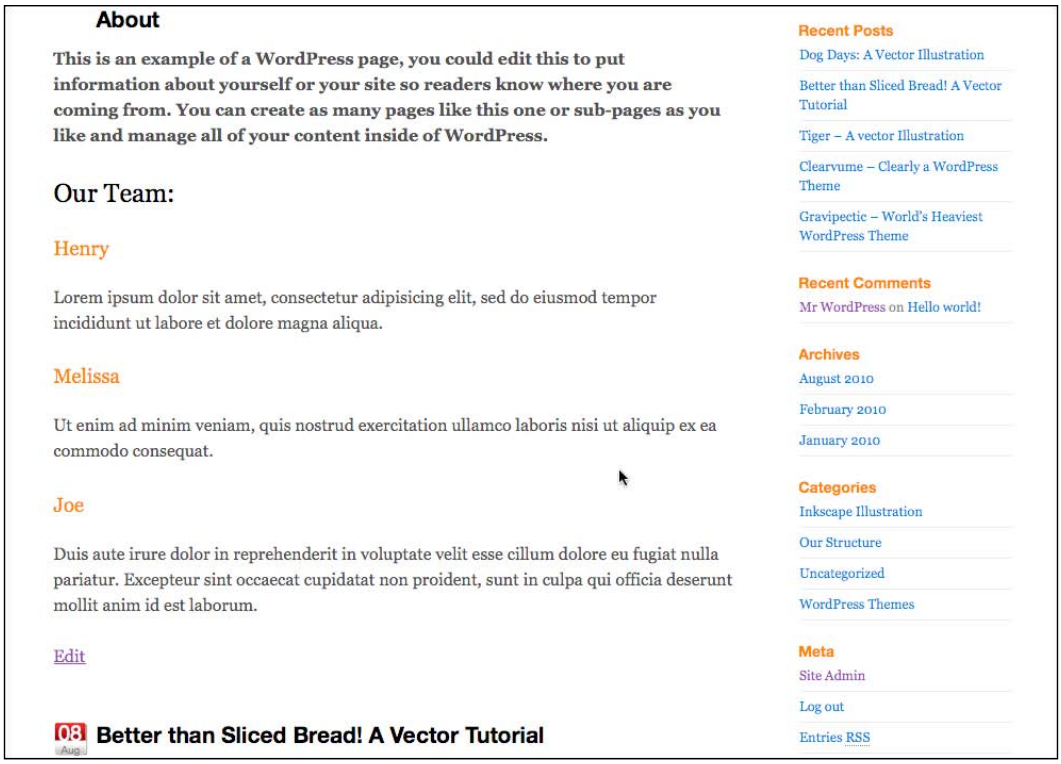


You'll also note that I was able to remove the `.css` function because only useful content is loaded in, nice and clean.

Transforming loaded content

Let's say we need to transform some of the content that we load in. Not a problem. There's a basic "success" callback function available. We can take advantage of it like so:

```
...
jQuery('.post:first').load('about-2/ #post-104', function(){
        jQuery('h3').css("color","#ff6600");
        jQuery('#post-104 p:first').css("font-weight","bold");
  });
...
```



As you can see, the content is now "part" of our page, and a set of DOM objects as our h3s in the ajaxed content changed along with other selected matches on the page. Now this seems a lot more useful. I bet you can think of a lot of uses for functionality like this! Guess what—so can our "clients".

Project: Ajaxifying posts

Lets assume you've got a client (relax, this is the *last* hypothetical client!) who's an "open source media designer" and would like a very clean and sparse home page. So sparse, they'd like only a list of the titles of the top, most current posts from two specific categories to appear. (In an ideal world, a decision like this would ensure their site's awesome design could sink in on the user before bombarding them with content.)

They'd of course like it to be slick. When you click on the title for a post, it loads in through AJAX, nice n' smooth. There's no reloading over to a single content page.

To get started on this request, we'll have to reference what we understand of the Template Hierarchy and custom loops. We'll create a `home.php` template page that will become the default home page which only displays the five most recent posts for the "WordPress Design" and "Inkscape Illustration" categories. Sounds straightforward enough, so let's get started.

First create a new custom template page called `home.php` and insert your `#content` div markup as well as the theme's header and footer (and anything else you want).

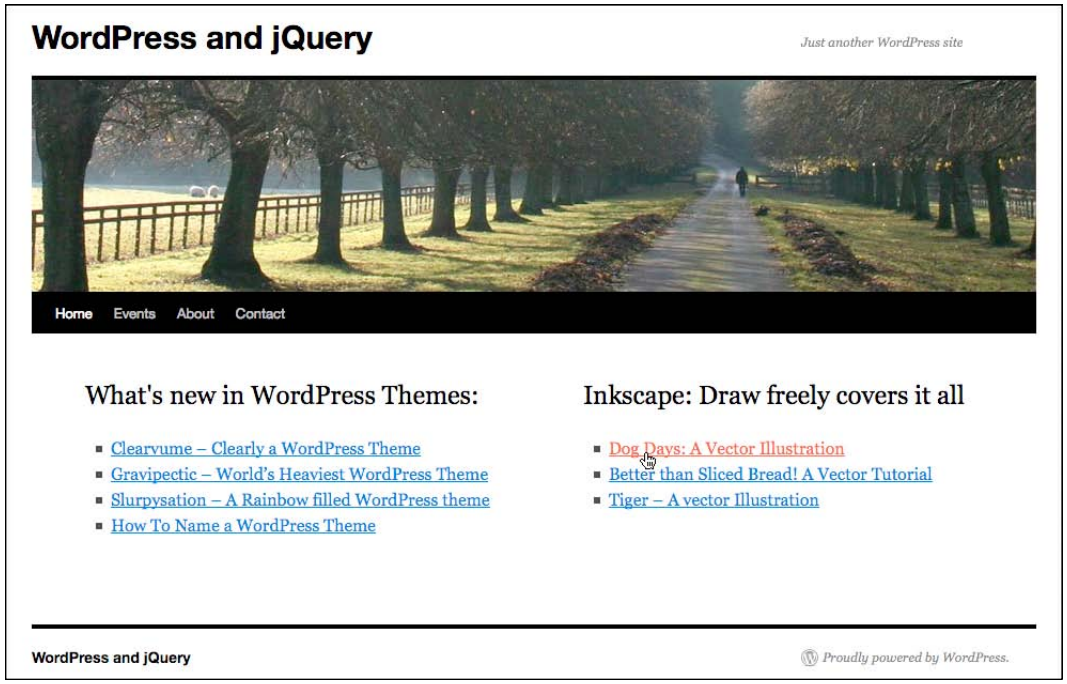
```
<?php get_header(); ?>

  <div id="content" role="main">

  </div><!--//content-->

<?php get_footer(); ?>
```

Next, inside our `#content` div, we'll place in our custom loops which load up the "WordPress Themes" and "Inkscape Illustration" categories. We know that the categories IDs are `5` and `6` so our custom "mini loops" look like this:

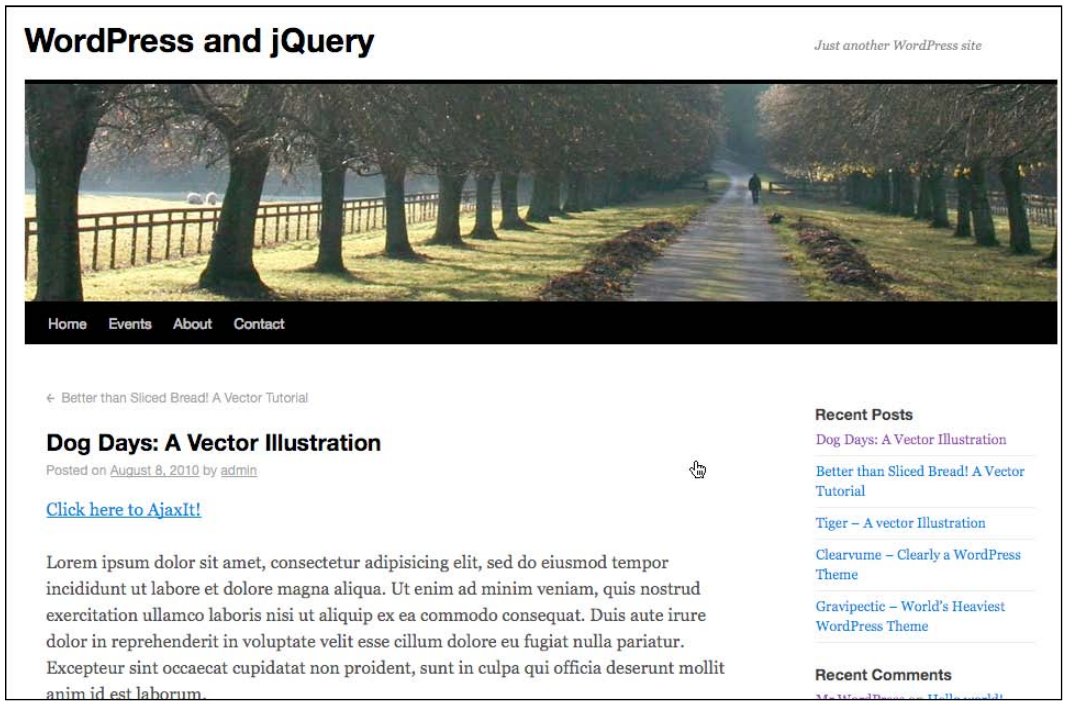
```
...
  <div style="float:left; width: 380px;">
  <h2>What's new in WordPress Themes:</h2>
  <ul>
  <?php global $post;
 $wpposts = get_posts('numberposts=5&category=6');
 foreach($wpposts as $post):
      setup_postdata($post);?>
      <li><a href="<?php the_permalink() ?>">
          <?php the_title(); ?></a></li>
 <?php endforeach; ?>
  </ul>
  </div>
```

```
 <div style="float:right; width: 380px;">
 <h2>Inkscape: Draw freely covers it all</h2>
 <ul>
 <?php global $post;
$inkposts = get_posts('numberposts=5&category=7');
foreach($inkposts as $post):
     setup_postdata($post);?>
     <li><a href="<?php the_permalink() ?>">
         <?php the_title(); ?></a></li>
<?php endforeach; ?>
 </ul>
 </div>

 <div style="clear:both;"> </div>
...
```

The custom loops will result in a page that appears like this:



Because we set up our loops to display the title *inside* an `href` link to the single page layout, if we check what we've got so far in WordPress, we'll see the post titles, and if we click on them, we'll be taken to the full post page, as seen in the next screenshot:



That's what we want. If the user doesn't have JavaScript enabled for whatever reason, the site will still work and give them the info they want. This is always the point we want to start from when working with jQuery: basic, working HTML and CSS. The goal is always to *enhance*, and not exclude, people who don't use one of the latest browsers for various reasons, or have one of the cool JavaScript enabled, smartphones.

At this point we're going to leverage a technique that we got a little taste of in *Chapter 6* with the PDF download enhancement. We're going to "hijack" the link to the post (this technique is often called "hijax") and use the URL to our advantage in the jQuery `.load` command.

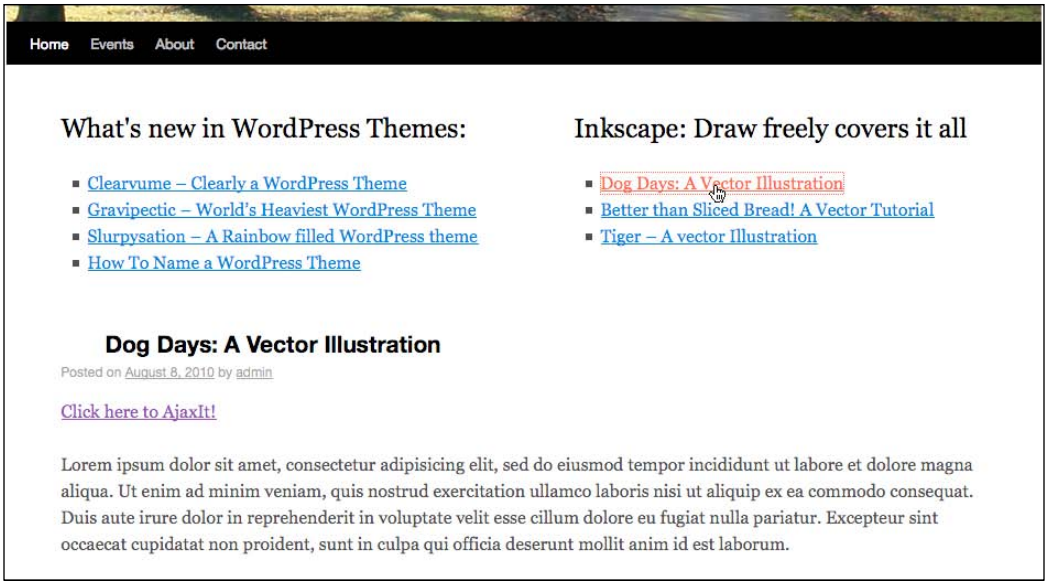
First up, we'll need something to load the content into, so in our `custom-jquery.js` file, we'll `.append` a new `div` to the bottom of the `#content` div.

```
...
jQuery('.home #content').append('<div class="displayPost"></div>');
...
```

Now, as we saw in our previous examples, we certainly don't want *everything* from the opening body tag to the closing to load in! We really only want the `.post div`. So let's set up our `.load` function and narrow down what loads in as follows:

```
...
jQuery('#content li a').click(function(event){
     //This keeps the href from reloading the page
     event.preventDefault();
     //grab the page link
     var page = jQuery(this).attr('href');

     jQuery('.displayPost')
       //use the grabbed link in the load function
       .load(page+' .post')
       .fadeOut()//fade out the previous content
       .slideDown(2000);//slide in the new content

});
...
```

Can you believe how simple that is? Any link that's clicked on will *fade* out the content that's loaded and *slide* in the new content! We now have a super slick effect that uses AJAX super simply, on our home page.



.getJSON: The littlest birds get the most re-tweets

Twitter is ridiculously popular these days, as a result, there are tons of great jQuery plugins for connecting to it already. My personal favorite is: *Damien du Toit 's* **jQuery Plugin for Twitter**: `http://coda.co.za/blog/2008/10/26/jquery-plugin-for-twitter`. If you really want nice control over your twitter displays, you can't go wrong with this plugin.

However, the Twitter Search and User Timeline API methods for JSON are pretty straightforward; thus, it makes for a great "quick tutorial" on how to use jQuery's `.getJSON` function.

Plus, you guessed it, our hypothetical client thinks the initial home page layout might be "too sparse", asking if we can just add in the three latest tweets from their username.

JSON and jQuery basics

Before we dive into Twitter and other services, let's go over the basics of JSON and how to use it with jQuery.

JSON (pronounced often like the name Jason) is an acronym for **JavaScript Object Notation**. Essentially, it's a simple machine-readable data-interchange format, which makes constructing and working with API applications in JavaScript a snap (and it can be used with other programming languages). If you're into learning the history of it, you can take a look at `http://json.org` to find out more.

What JSON looks like

You'll be pleasantly surprised to find that JSON markup syntax looks the same as most parameter/values syntax you've already been using so far in jQuery, or with CSS. It is based on most C language object notations such as Java and JavaScript, so it makes things quite nice and handy when dealing with in JavaScript and jQuery.

For example, jQuery's `.css()` function can have multiple values values passed within `{}` brace brackets, like so:

```
.css({background: '#ff6600', color: '#333333', height: '300px'});
```

In the same manner, JSON data can be set up as such:

```
{"results":[{"text":"text string here",
  "to_user_id":0001,"user_name":"ThunderCat"}]}
```

Pretty similar all right! Let's take a look at using it within jQuery.

Using JSON in jQuery

Let's take a closer look at the `.getJSON` function.

```
jQuery.getJSON(
    url, //the location of the data
     data, //if you need to send anything to the service POST
     function(){
       //callbackfunction
     }
);
...
```

The first parameter of this function is just like the `.load` function; you'll place in the the URL that you are planning to read. The `data` parameter is used if you need to POST data to the URL (you can do this in a query string or array object). The call back function is not required, unless you're calling a URL from a server other than your own.

Let's now take a look at putting `.getJSON` to use in our WordPress site.

Using .getJSON with Twitter

First up, when dealing with other service APIs, there's no excuse for not reading and using their documentation. Services often update their APIs to make them better and faster, but then the methods used to connect to and work with them change from time to time. It can sometimes take quite a bit of diligence to keep your code up-to-date with an API. Twitter's API documentation can be found here: `http://apiwiki.twitter.com/Twitter-API-Documentation`.

Also, many API services require that you sign up as a developer and use OAuth to use some or all of their services (or their own authenticating system to protect your user login and data).

What's OAuth?

OAuth is an open standard that allows users to hand out tokens instead of usernames and passwords to their hosted data by a given service provider. Many API service providers use it and you can find out more from their site: `http://oauth.net/about/`.

In this section, I'll cover the basics of connecting to the user timeline method in the twitter API. This method doesn't require OAuth so long as the user has a publicly viewable twitter stream, so you don't need to register for an OAuth application (but it certainly doesn't hurt to sign up).

Using Twitter's user timeline method

The URL parameter in our `.getJSON` function will contain the following API, formatted URL:

```
http://api.twitter.com/1/statuses/user_timeline/username.format
```

You can choose from the following formats (but guess which one we'll be using!):

- atom
- **json**
- rss
- xml

First up, we'll need to place our tweets on the home page.

We have two options here, we can go into the `home.php` template file and create an "actual" `div` and `ul` list, or we can create it entirely with jQuery.

Honestly, a call like this is just up to you. At this point in the book, you should be plenty comfortable editing and tweaking your theme files or generating useful DOM objects with jQuery.

Because the tweets are completely dependent on JavaScript being enabled, and we aren't trying to custom display any WordPress content with template tags, I'm happy to do all the work in jQuery.

We'll start off in our `custom-jquery.js` file, inside the document ready statement, create the space for the tweets like so:

```
...
//we'll want to make sure we add our div to the home page only,
//referencing the WordPress body class .home (make sure your theme is
//using the template tag body_class() in the body HTML tag!)
jQuery('.home #content')
//this .append string is a div, h2 heading, and three list items
//in a ul with a Follow Us link:
    .append('<div class="tweets"><h2>Our Tweets:</h2>
      <ul><li></li><li></li><li></li></ul>
        <p>
          <a href="http://twitter.com/ozoopa">Follow Us!</a>
        </p></div>');
...
```

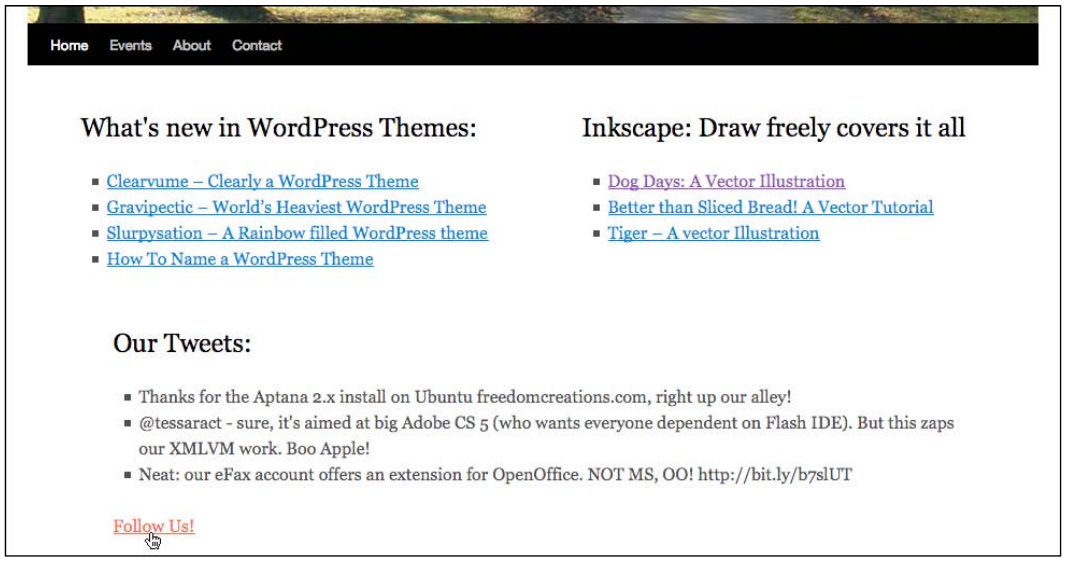
Next we'll set up the Twitter API URL as a variable with our "clients" twitter user name (we'll use one of mine: ozoopa).

```
...
var tweetURL = 'http://api.twitter.com/1/statuses/user_timeline/
ozoopa.json?callback=?';
...
```

We can now make our `.getJSON` call. Follow along in the bold comments:

```
  jQuery.getJSON(tweetURL, function(twitter){
     //'twitter' is the callback function that returns the tweets
     //for each li in the twees class we'll drop in the text
     jQuery('.tweets li').each(function(i){
        //we only want the tweet text, nothing else
      jQuery(this).html(twitter[i].text);
    });
  });
...
```

As you can see in the next screenshot, our tweets are showing up just great!



What's Twitter sending back?

You'll note that we focused in on just getting the "tweet" text itself back. Here's a taste of what twitter is actually sending back through JSON in that URL (the bold part is what we actually used):

```
[{"coordinates":null,"in_reply_to_screen_name":null,"geo":
null,"favorited":false,"truncated":false,"in_reply_to_status_id":
null,"source":"web","in_reply_to_user_id":null,"contributors":
null,"user":{"profile_background_image_url":"http://s.twimg.
com/a/1274899949/images/themes/theme1/bg.png","profile_link_col
or":"0000ff","url":"http://ozoopa.com","description":"","follow
ers_count":14,"profile_background_tile":false,"profile_sidebar_
fill_color":"e0ff92","location":"","notifications":null,"friends_
count":3,"profile_image_url":"http://s.twimg.com/a/1274899949/images/
default_profile_3_normal.png","statuses_count":10,"profile_sidebar_
border_color":"87bc44","lang":"en","favourites_count":0,"screen_name"
:"ozoopa","contributors_enabled":false,"geo_enabled":false,"profile_
background_color":"9ae4e8","protected":false,"following":
null,"time_zone":"Central Time (US & Canada)","created_at":"Tue Sep
15 21:54:45 +0000 2009","name":"ozoopa open source","verified":
false,"profile_text_color":"000000","id":74567461,"utc_offset":-
21600},"created_at":"Tue May 11 19:34:09 +0000 2010","id":138053
49673,"place":null,"text":"Thanks for the Aptana 2.x install on
Ubuntu freedomcreations.com, right up our alley!"}, ...//more tweets
follow...]
```

As you can see, you're given back a lot of data to work with! Again, it pays to dig through the API and see what's available to leverage; you can also have hours of fun just experimenting with displaying all of the various items available in the JSON feed.

Using getJSON with Flickr

The client likes it! And of course, they now think the home page is now "too text heavy". What about adding in the six latest images from Flickr images tagged "`wordpress theme`" in the sidebar? That should balance it out.

Fortunately, this is not a problem either.

Again, your first stop should be the Flickr API documentation: `http://www.flickr.com/services/api/`.

But we'll go ahead and get started, again, creating a little space in the home page's sidebar for the images:

```
...
jQuery('.home).append('<div class="flickr">
  <h2>Latest Flickr:</h2></div>');
...
```

Here with their public photo stream method URL:

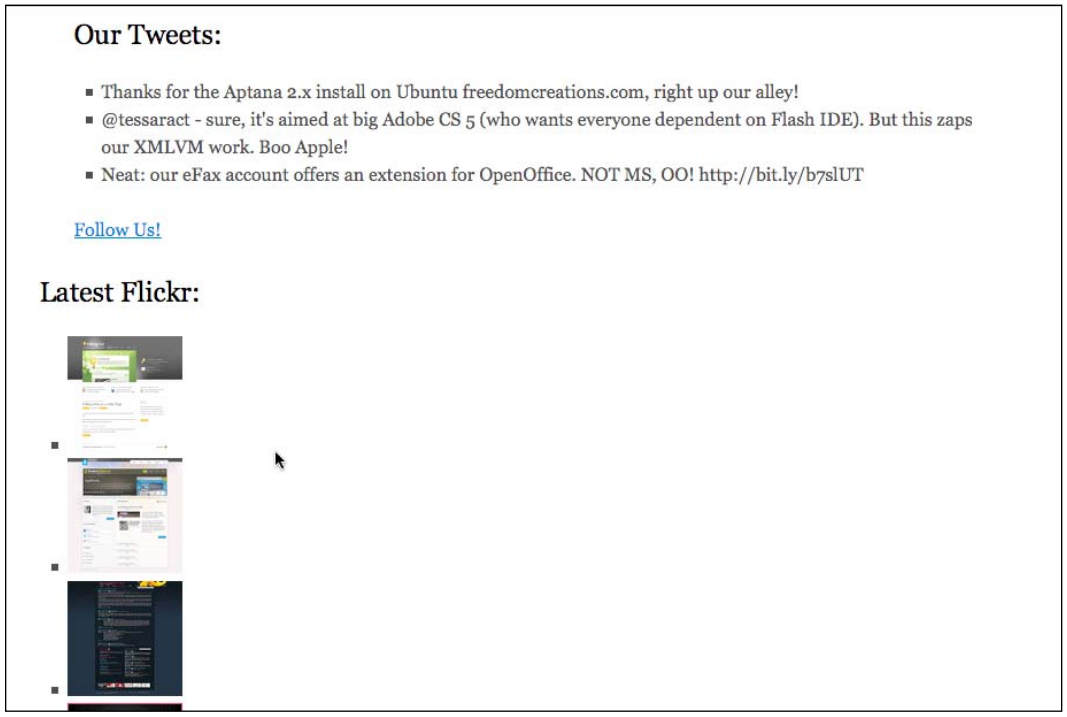
```
...
var flickrURL = 'http://api.flickr.com/services/feeds/photos_public.
gne?tags=wordpress,themes&tagmode=all&format=json&jsoncallback=?';
...
```

And now we can set up our `getJSON` call:

```
...
  jQuery.getJSON(flickrURL, function(flickrImgs){
    jQuery('.flickr li').each(function(i){
        jQuery(this)
         .html('<img src='+flickrImgs.items[i].media.m+'
           width="100" height="100" />');
    });
  });
...
```

The Flickr JSON string returns an array called `items` that offers all sorts of data. You'll notice that it's a little different when targeting the information we want compared to the twitter API. By pulling the `media.m` url to a thumbnail we're able to create a quick list of images.

It looks like this under **Our Tweets**:



Other popular services that offer APIs with JSON format

The fun doesn't have to stop there! Now that you're familiar with using `.getJSON`, your world is open to implement all sorts of custom cross-site mashups and solutions in your WordPress sites. Understanding JSON and the `.getJSON` function also makes you more adept at being able to "massage" a good WordPress or jQuery Plugin into handling your custom needs better.

The following popular services offer APIs with JSON support:

- YouTube: `http://code.google.com/apis/youtube/2.0/developers_guide_json.html`
- Netflix: `http://developer.netflix.com/`
- delicious: `http://delicious.com/help/api`
- bitly: `http://code.google.com/p/bitly-api/wiki/ApiDocumentation`
- goodreads: `http://www.goodreads.com/api`
- LibraryThing: `http://www.librarything.com/api`

Look around! If you use a great service that offers any kind of "social" capability, they might offer an API that serves up data in the JSON format. You may need to register as a developer with that service in order to authenticate your requests (usually using OAuth) but if the end result you get back is a JSON string, you're good to go with jQuery and your WordPress project!

Project: Ajax-izing the built-in comment form

From the working samples we've done so far with `.load` and `.getJSON`, you can probably think of many extremely cool ways to implement AJAX in your WordPress site. The most useful application of this is the comment form.

First up, we don't even need to amend any template page HTML or WordPress Template Tag, PHP code. This is great as again, as often as possible (all the time really) we always want our site to work without the jQuery enhancement.

Ajaxing the WordPress comment form is deceptively simple. And for you "premium" theme developers, it's a great way to entice people to download your theme: "Built in AJAX comments!". It is something that we'd like full control over, so we'll be using the `.ajax()` function instead of `.load` (see, I told you `.ajax` would come in handy every now and then).

First off, in experimenting with the comment form, we'll be wanting to change its CSS properties to alert users to errors. I've found it's just better to set the form's CSS to something consistent that we can then change easily in jQuery for other uses. Add the following code to your `custom-jquery.js` file to change the CSS properties of the default theme's comment form styles.

```
...
jQuery('#commentform input')
  .css({border: '1px solid #ccc', padding: '5px'});
```

```
jQuery('#commentform textarea')
  .css({border: '1px solid #ccc', padding: '5px'});
...
```

We're now ready to "take control" of the form. Upon submit, we want our jQuery to do the talking, not the form's "action" attribute. So we'll use a handy function called `.submit()` like so:

```
jQuery('#commentform').submit(function(){

    //turns all the form info into an object
    var formData = jQuery("#commentform").serialize();

    //so we can display the comment back to the user
    var comment = jQuery('textarea#comment').val();


  });
...
```

Note our use of another handy, little known jQuery function called `.serialize()`. This takes all the data in our `#commentform` form and upon submit, turns it into a handy object that we can now pass on in our `.ajax` function.

Inside the `.submit` function, *under* the comment variable, let's add in our `.ajax` call. We'll be using this function because we need a little extra control and will be taking advantage of its `success:` and `error:` callback functions. Read through the code's bold comments to follow along:

```
...
jQuery.ajax({
      type: "POST",
        //this is the script that the comment form submits to:
      url: "/wp-jqury/wp-comments-post.php",
        //formData is our serialized content object
      data: formData,
      success: function(){
        //on success load content and fade in:
      },
      error: function(){
        //on error, inform user of what to do:
      }
    });
      //this makes sure the page doesn't reload!
    return false;
...
```

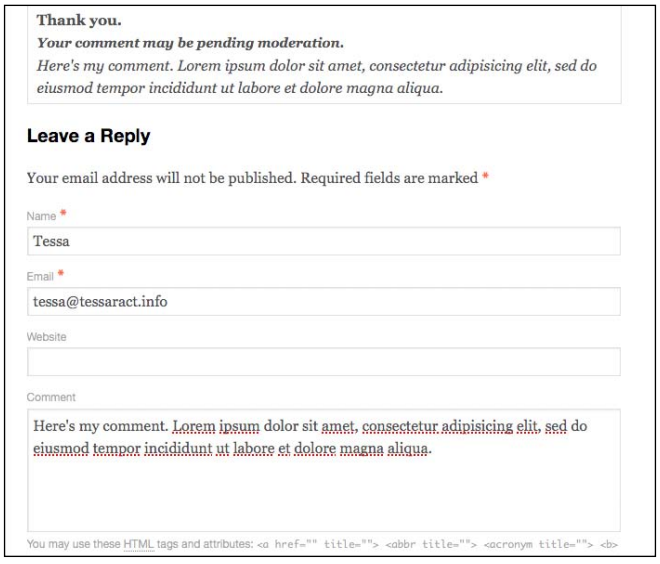
That's the gist. We're now ready to get down to work by setting up the `success:` and `error:` functions. Let's start with the `success:` function.

We'll first want to create a `div` that will contain a message. We'll then add our message to that div along with the `comment` variable that we set up earlier (under our `formData` serialized object) to pull the comment entered in the form into our code.

We'll also be sure to add in a little jQuery "shine" and leverage some of those animation skills from *Chapter 5, jQuery Animation within WordPress* to make sure the `success` response loads in nice and smooth. *Inside* the `success: function()` brace brackets, insert the following code:

```
...
//on success load content and fade in:

//create the div that the message goes in
jQuery('#respond').prepend('<div class="message"></div>');

jQuery('#respond .message')
      .html("<div style='border: 1px solid #ccc; padding: 5px 10px'>
        <b>Thank you.</b><br/>
          <span style='font-size: 90%;'>
            <i>Your comment may be pending moderation.</i>
          </span><br/> "+comment+"</div>")
      .hide() //then hide it!
      .fadeIn(2000); //then fade it in nicely
...
```

When the Form is properly filled out, the end result is this message that fades in:



We're now ready to tackle the people who don't fill the form out properly. The `wp-comments-post.php` file does throw an error back if the required fields are not filled out. We can use this to our advantage by just checking for an error using the `error:` function.

Nice, we just created some slick commenting functionality for our WordPress site using AJAX!

Shouldn't some of these examples be WordPress plugins?

As mentioned in *Chapter 3*, *Digging Deeper: Understanding jQuery and WordPress Together*, if you create a jQuery enhancement that doesn't require any tweaks or edits to the WordPress theme, and will work with most themes released for public use, you may want to consider wrapping up your scripts into a separate WordPress plugin.

This is a handy practice if you're busy and don't want to amend a new theme with all your custom jQuery scripts every time you swap themes, or if you're part of a larger project with lots of people or if you just simply want to share your jQuery work with less technical WordPress users. Follow the steps in *Chapter 3*, to wrap your jQuery scripts and plugins into simple WordPress plugins so that any less-technical administrators can easily add and remove them from their projects.

Also remember, *Chapter 3*, walks you through creating a jQuery plugin as well. You'll probably be able to condense and clean up your code by placing it into a jQuery plugin that you then wrap into a WordPress plugin. This should also make creating updates and enhancements of your scripts easier to manage. You'll then have better organized code that you can document and share with both worlds: jQuery developers and WordPress enthusiasts.

Think about it though: if a jQuery enhancement *is dependent* on any custom, special markup that you've edited a theme to generate (such as our post list example at the beginning of this chapter), it's better to leave that jQuery script as part of the theme, as it won't work outside of it. This is a good thing for super-custom or premium themes. By making your enhancements part of your theme, you can entice people to download it because it offer features they don't need to then go out and find separate WordPress Plugins for.

Summary

Who knew AJAX was so darn easy these days? As you can see, leveraging the strengths of WordPress themes and jQuery's AJAX events and requests, it's very easy to make some mighty dynamic sites. In this chapter we took a look at:

- Creating custom loading content and hijacking (hijaxing) links to do with as we please
- Working with `.getJSON` and other site's APIs
- Creating our own custom AJAX loading comment form (probably one of the most popular enhanced theme features and plugins sought after by WordPress site owners)
- Further enhancing our AJAX work with simple jQuery animation features

You now understand a lot about applying jQuery to specific enhancements and features to WordPress sites. We've started off with the basics and really learning how to leverage selectors so that your WordPress editor's workflow doesn't have to be interrupted and applied that to some very exciting enhancements that include slick animation, the UI plugin and AJAX. We also covered getting those solutions into your WordPress site's theme, a WordPress Plugin as well as jQuery Plugins. For the majority of your WordPress and jQuery development needs, you are all set!

In the next and final chapter, we'll take a look at some tips and tricks for working with jQuery and WordPress plus; the final appendix of this book is a condensed "cheat sheet" of reference information for key jQuery functions as well as important WordPress function and template tags and classes, all to aid you in your jQuery and WordPress development.

8

Tips and Tricks for Working with jQuery and WordPress

You're now ready to take your jQuery knowledge to the world of WordPress. But first up, let's take a look at what we'll cover in this chapter:

- Tips and tricks to properly load our jQuery scripts and making sure that they are compatible with other scripts, libraries, and plugins
- Some tips and tricks for using Firefox and Firebug to speed and aid in your jQuery development
- The virtues of valid WordPress markup and how you can make it easy on the site's content editors

The following are the tips and tricks required for working with jQuery and WordPress.

Keep a code arsenal

A "snippet collection" or, what I call my "code arsenal" will go a long way to help you out, not just with jQuery and WordPress code, but also with the general HTML markup and even CSS solutions you create, not to mention any other code language you work in.

I'm terrible at remembering syntax for code, markup, and CSS. I often know what I need, but can never quite recall exactly how it's supposed to be typed. I used to spend hours going through various stylesheets, markup, and codes from previous projects to copy into my current project as well as googling (and "re-googling") web pages that had samples of the syntax I needed.

If you often find yourself in a similar situation, using the **Snippets** or **Clip** features that are usually available in good HTML/Code editors will free you from this mundane (and very time consuming) task. You simply type or paste the WordPress template tags, functions, PHP code, key CSS rules, and jQuery functions (and any other code syntax, whatever you find you need to use the most), into the **Snippets** or **Clips** panel available in your editor, and the application saves it for you, for future use.

As you work on different projects, you'll come up with solutions that you'll probably want to use again in the future, say, a set of CSS rules for unordered lists that make a nice gallery view, or a very clever use of two jQuery functions together. Every time you create something you think may come in handy (and a lot of it will come in handy again), be sure to save it right then and there, for future reference.

Good editors such as Dreamweaver, HTML-Kit, and Coda usually have the ability to organize snippets and keep them logically grouped so they're easy to access. Some editors will even let you assign custom "key shortcuts" and/or drag-and-drop to your clips right into your working file. How easy is that?

Free your arsenal

Once you discover how handy this is, you might want to have your arsenal available to other programs you work with, especially if you switch between multiple editors and authoring environments. I suggest you invest in a multi- paste/clip board application that lets you save and organize your code snippets. When I was on a PC, I used a great little app called Yankee Clipper 3 (which is free and is available at `http://www.intelexual.com/products/YC3/`), and now on the Mac, I use iPaste (which has a modest price; go to `http://www.iggsoftware.com/ipaste/`). In addition to having your arsenal handy from any application, being able to go back through the last 10 or so items you copied to the clip board is a real time saver when you're working on a project.

Your arsenal on-the-go

Last, I find I like to take most of my arsenal with me. If you use a handheld device or have a phone with a note app that lets you categorize and search for notes (especially the one that will let you sync from your desktop or a web service), you'll probably find it useful to keep some or all of your arsenal in it so you can easily look up syntax from your arsenal at any time. I occasionally freelance at places that require me to use one of their computers and not my laptop, so having access to my arsenal on my device is very useful.

Palm's native note app suited me great in this capacity for years and years; I now keep a large part of my arsenal in Google docs and use a little desktop app called NoteSync, which lets you write and view Google docs notes quickly (they'll have an Android app out soon, but in the meantime I use Gdocs on my Android device to see my notes). I have many friends who swear by EverNote's system (though, their mobile app only works offline on the iPhone and not on Android—as of yet).

Once all your often used and creative one-off solutions are all located in a convenient (hopefully categorized and key-word-searchable) place, you'll be amazed at the amount of speed your development picks up and how much more relaxing it is.

jQuery tips and tricks for working in WordPress

Let's start-off with some of my favorite jQuery tips and tricks, before focusing on WordPress. Most of these items have been covered in detail in the book and this is to remind you that they're important (in a way, that's the first "tip", don't skimp on the essentials). There are also a few nuggets in here that haven't been covered as yet and that will help you speed up your jQuery development.

Try to use the latest version of jQuery

This is one of the drawbacks to using the bundled WordPress version: it may get a little behind the current version of jQuery until the next version of WordPress comes out. I'm all for staying on top of the current version as jQuery's top goals for version releases are not just to provide new functionality, but continually streamline and improve the performance and speed of the existing functionality. If the latest version of jQuery available on CDN is greater than the version that's bundled, be sure to `deregister` jQuery first or restrict your newer version with the `if else` statements we learned in *Chapter 3*, *Digging Deeper: Understanding jQuery and WordPress Together*, so it loads on the front end of the site on required pages only. Otherwise, you may create problems with plugins using the bundled version of WordPress.

Stay current with the Google CDN

The best way to stay current is to simply use Google's CDN. I covered this in *Chapter 2*, *Working with jQuery in WordPress*, and *Appendix A*, *jQuery and WordPress Reference Guide*, has a reference of this as well. There are additional advantages to loading up from Google's CDN instead for your project's hosted server. Instead of having to load JavaScript's, libraries and assets one by one from your server, your site can simultaneously load the main library from the Google CDN in addition to other local jQuery scripts and collateral. The bonus is that jQuery will be cached for users who've visited other sites that load it up from Google's CDN. Be sure to check out *Appendix A*, for a complete reference on `wp_enque_script`.

Stay in No Conflict mode

The great thing about WordPress is that a site can have so many people contributing to it in lots of different ways: writing content, working on the theme, and adding WordPress plugins. One of the worst things about WordPress is that so many people can easily contribute who knows what to a site, depending on their admin status, some other collaborator could add to them, or what plugins they could install.

Staying in **No Conflict** mode is a must for WordPress. This in conjunction with using the `wp_enque_script` to load in WordPress will ensure that jQuery doesn't get "pushed out" if anyone loads up any other plugin that uses say MooTools or Scriptaculous, or even just an older version of jQuery.

It's easy to stay in `noConflict` mode. The easiest is what we've been doing throughout this whole book! Just use `jQuery` instead of the shortcut dollar sign (`$`) in front of your scripts.

```
jQuery('.selector').function();
```

Make sure other scripts in the theme or plugin use the Script API

If you're using a theme or a plugin from a third party, take a look through the theme's `header.php` file or the plugin's PHP pages and double-check that all scripts have been loaded in using the `register` and `wp_enqueu_script` methods. I've had a few instances that were rather frustrating and caused some hair-pulling, as we tried to figure out why my jQuery scripts were not working or wondering how I "broke" them porting them over to the live site. Turns out, the live site had a plugin installed that my sandbox site didn't, and you guessed it, that plugin was including an older version of jQuery and a custom script file using hard-coded `script` tags instead of the `wp_enqueue_script` method. Once this was figured out and straightened up, setting everything into `noConflict` mode, everything worked fine again!

Check your jQuery syntax

This one always gets me. You write up a nifty little jQuery chain, add a few tweaks to it, and the darn thing just stops working. And you know it's right! Well, at least, you think it's right. Right? This is where a great code editor comes in handy. You'll want some nice **find** features that let you step through and look at each returned **find**, as well as let you run a find not just on the whole document, but on individual selections. I like to select the just the "offending chain" and run the following **find** features on it to see what comes up.

Colons and semicolons

Do a **find** for `:` (colons); you'll probably find a few that are accidentally set up as `;` (semicolons) in your function's various object parameters, or you may have typed a colon where a semicolon should have been there.

Closing parenthesis

I'll also run a **find** on closing parenthesis, `)`, and make sure each one that comes up is part of a continuing chain or the end of the chain marked with a `;`.

Mismatched double and single quotes

Last, a quick check for matched-up single and double quotes sometimes shows me where I've messed up. Panic's Coda lets you place in "wild cards" into the **find** so a search for `"*'` or `'*"` usually turns up a pesky problem.

Most good code editors have color-coded syntax, which really helps in recognizing when something isn't right with your syntax, such as not having a closing quote mark at all or parenthesis. But, the issues above are tricky as they'll still often display as proper color coded syntax, so you don't know until you run your script that something's wrong.

Use Firefox and Firebug to help with debugging

Firebug has a feature called "console logging". This is one of many great features of Firebug in my opinion. For years I often resorted to using JavaScript's "alert" statement to try and show me what was going on "inside" my work but the Firebug console handles so much more than that. This is really useful because sometimes you have to debug a "live" site and setting up JavaScript alerts is a little risky as you may confuse visitors to the site. Using Firebug's console logging eliminates that.

First up, there's the `console.log` and `console.info` statements which you can add to your jQuery scripts to pass info to and have a plethora of useful (and sometimes not-so-useful, but interesting) information about your script returned.

`console.profile` and `console.time` are great for measuring how fast you scripts are being processed by the browser.

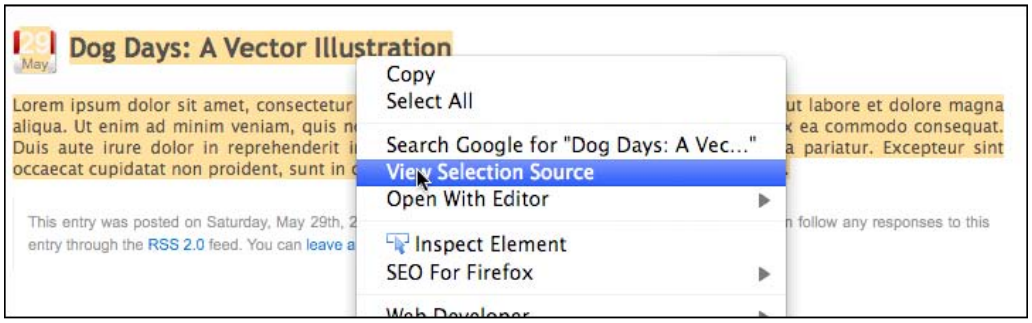
For a complete overview of everything Firebug's console can do, check out: `http://www.getfirebug.com/logging`.

Know what jQuery is doing to the DOM

Another reason to love Firefox, as much as I love Opera and Chrome, when I can't select text and objects on the page and right-click on **View Selected Source** I'm at a loss and feel blind.

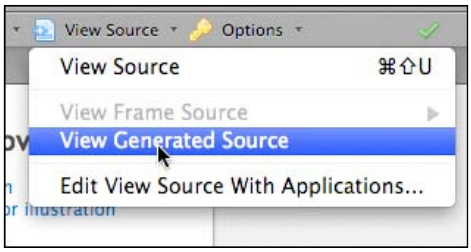
If your jQuery script has created new DOM objects on-the-fly or is manipulating objects, right-clicking **View Page Source** will only show you what the server served up and not what jQuery and JavaScript cooked up in your browser.

This is a great, quick, and an easy way to see if jQuery added that class, or wrapped those selected elements in your new div. Select what's generated by jQuery or should be affected by your jQuery script and right-click **View Selected Source** to see what's actually in the DOM.



Web Developer's Toolkit: View Generated Source

If you find having to make a selection confining and would like to see what the entire "generated" source is, you can use the Web Developer's Toolkit to see the page as affected by jQuery.



Seeing what Firebug sees

The most robust look at your generated HTML objects in the DOM, comes from using Firebug's **HTML** view. By selecting the **HTML** tab as well as the **Click an element in the page to inspect** tab, you can essentially run your mouse over any element and get an instant view of what it looks like in nested drop-down objects in the HTML view.

At first, I found this view a bit cumbersome as I was usually just trying to confirm the presence of a new object or manipulated attribute, but I quickly became accustomed to exactly how powerful it can be in helping me debug my jQuery scripts as we'll see in the next tip, where we'll even write selectors.

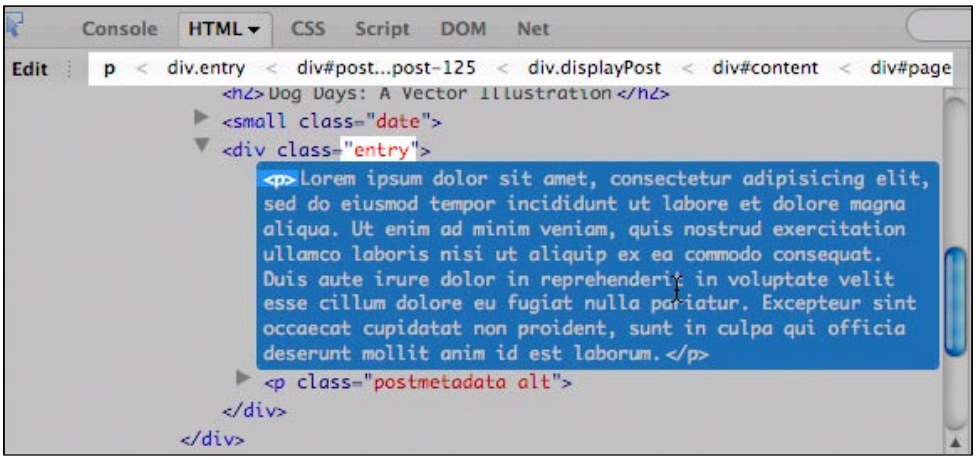
Tips for writing great selectors

If by any chance you skimmed or skipped *Chapter 2, Working with jQuery in WordPress (or haven't read it yet),* you'll want to go back and review it in detail. You'll also find that the next appendix has the top "cheat-sheet" selector filter highlights that will be helpful once you have an understanding of the fundamentals of selectors.

Having a handle on your selectors means you'll be able to do anything you want with jQuery. Literally! I have yet to come up with a problem that I've had to push back onto the WordPress content editor. But sometimes when it comes to starting my jQuery scripts, targeting the selectors I need can prove a tad challenging, especially when working with an unfamiliar, custom theme.

Again, Firebug to the rescue. Remember our previous tip where we used the HTML view? You can use that view to select what you want to affect with jQuery and easily see how to construct a selector for it.

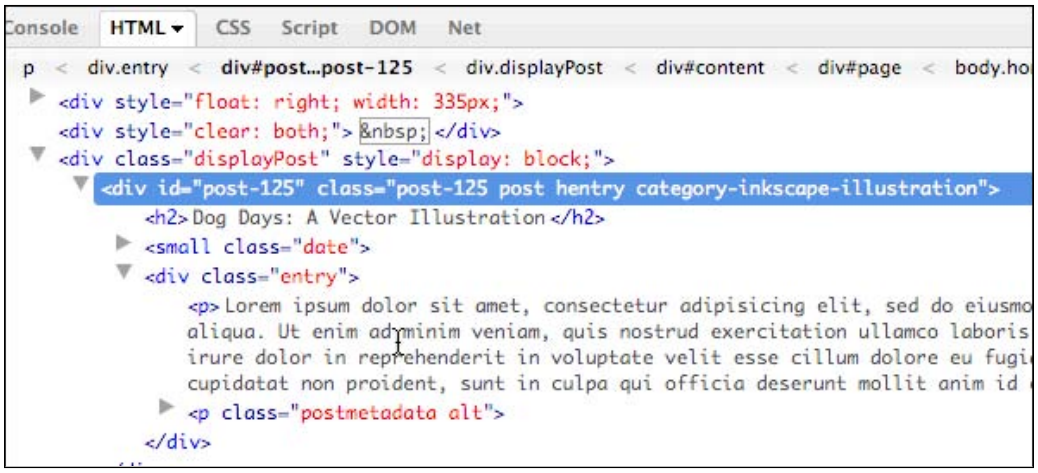
For example, take a look at the following screenshot's highlighted areas:



If we want to select that paragraph `<p>` tag, it becomes apparent that we simply write our jQuery selector for:

```
jQuery('.entry p')...
```

We can also see that we can get much more specific and target the id `#post-125` if we only want to affect `<p>` tags in that particular post. By clicking in that top bar area that's displaying the hierarchy of ID and class names, on a particular class or ID, it will expand the object with that class or ID so we can fully see what our options are. For instance, we could also target paragraphs in the `category-inkscape-illustration`.



Don't forget about your selection filters!

Remember: sometimes it's easier to tell jQuery what you don't want to select using the `:not` filter or what you specifically want to select, such as the `:first` or `:has()` filters. The reference from *Appendix A, jQuery and WordPress Reference Guide*, has a great overview on the best selection filters to use in WordPress and of course, *Chapter 2, Working with jQuery in WordPress*, has a comprehensive list and example set.

There you have it. jQuery selectors made easy! The more experienced you are in making jQuery selections, the easier you'll find it to generate your own HTML and objects to aid in your jQuery enhancements. This is useful because our next tip is all about making it simple for the site's editors.

Keep the WordPress editor's workflow "flowing"

A few years ago, when I first started using other well-known JavaScript libraries, I found them incredibly useful for my own hand-coded projects or frontend interface projects, but implementing them and their plugins on a CMS site such as WordPress was often disappointing. Most scripts relied on adding special markup or attributes to the HTML. This meant site editors had to know how to add that markup into their posts and pages if they wanted the feature and most of them just couldn't do it, leaving me with frustrated clients who had to defer back to me or other web admins just to implement the content.

Also, it puts more work back on me, eating up time that I could have been using to code up some other features for the site (entering content into the site's CMS, is not my favorite part of web development). jQuery changes all that and makes it very easy to write enhancements that can easily work with just about any HTML already on the page!

Despite just about everything being online "in the cloud" these days, most people don't have a knack of HTML. In fact, as we move full-on through Web 2.0 into Web 3.0, and beyond, less and less people will know any HTML, or ever need to, because of the great web-based applications such as WordPress and all the various social networking platforms that take the user's raw information and organize it as well as style and present it to the world for them.

If your enhancement requires the user to flip over into the **HTML** view and manually add in special tags or attributes, that's not an effective jQuery enhancement! Don't do it!

The user should be able to add in content and format it with the built in, **Visual**, WYSIWYG editor. You, the great jQuery and WordPress developer, will develop a solution that works with the available HTML instead of imposing requirements on it, and your clients and editors will be wowed and love you forever for it.

But my jQ script or plugin needs to have specific elements!

As we've seen in several chapters of this book, it's true, your jQuery plugin may require certain elements to be present in the DOM to transform content into a widget or interaction.

Here's the thing to remember: *if HTML elements can be constructed to make the enhancement work at all, you can create those HTML elements, within the DOM, on the fly with jQuery*. You don't have to force your client to create them in the editor!

Take a look at our work in *Chapter 6*, *WordPress and jQuery's UI,* with the UI plugin where we took simple, basic `h3` headers and paragraphs and wrapped them dynamically in the proper markup for the jQueryUI tab widget. Or heck, even before that in *Chapter 5*, *jQuery Animation with WordPress*, where we took a client's unique post text (didn't have anything to do with HTML!) and were able to construct a lovely animated chart out of it.

jQuery is all about selectors and it's true, sometimes, to get started you need something clear and unique to select in the first place! Work with the site's editors when coming up with enhancements. It's much easier for most content editors to wrap their head around having to simply apply a unique category or tag to certain posts in order for the enhancement to take effect, or even manually adding in keywords to a post's header or formatting content in a specific way (like the chart example in *Chapter 5*, *jQuery Installation within WordPress*). Look at all of these options first, with a site's editor, to make sure the enhancement is really an enhancement, for everyone.

WordPress tips and tricks for optimal jQuery enhancements

Just because you're up to speed with jQuery doesn't mean that you can neglect what's happening on the server-side with your WordPress installation. Let's take a look at a few key things to remember when dealing with WordPress

Always use wp_enqueue_script to load up jQuery and wp_register_script for plugins for custom scripts.

We went over this in detail in *Chapter 3*, *Digging Deeper: Understanding jQuery and WordPress Together*, but again, you'll want to make sure you use `wp_enqueue_script` for all your jQuery loading-up needs. `wp_enqueue_script` and `wp_register_script` is WordPress' solution to keeping multiple versions of scripts from needlessly loading or canceling other versions out.

You can use the `wp_enqueue_script` to easily load up jQuery and plugins that come bundled with WordPress and even from the Google CDN. If you have your own custom script, you can use `wp_register_script` to register your custom script with WordPress and you can then use `wp_enqueue_script` to load it up, making it dependent on jQuery or some other jQuery plugin or JavaScript library.

Appendix A, *jQuery and WordPress Reference Guide*, shows you quick and easy ways of how to use `wp_enqueue_script` for all your top jQuery and WordPress implementations.

Always start with a basic, working, "plain HTML" WordPress site

I've said this several hundred times by now (or so it seems) but the name of the game is enhancement. Don't write anything that will break if JavaScript is disabled or unavailable in some way. Most WordPress themes out there already work this way, displaying content and links that use simple `http` calls to link over to additional content pages or anchor names. If you're developing a theme from scratch that will be enhanced by jQuery, develop it as completely as possible so it works with basic `http` functionality before adding in your jQuery enhancements. This will ensure your content is seen no matter what browser or device it's accessed with.

There are more and more premium themes that come with built in jQuery and other JavaScript enhancements. You'll want to turn off JavaScript in your browser and see how that content is handled without access to the enhancement. If the site completely appears to "break" and not properly display the content without JavaScript, depending on what types of devices you're planning on deploying to, you might not want to use that theme!

Validate, validate, validate!

It's hard for jQuery to make those selections if your HTML is not well formed or broken. Often the fix comes from repairing HTML markup that's broken.

The easiest way to validate is to go to `http://validator.w3.org/` and if your file is on a server, you can just enter in the URL address to it. If you're working locally, from your browser, you'll need to choose **Save Page As** and save an HTML file of your project's WordPress output and upload that full HTML file output to the validator using the upload field provided.

Also, Firebug's main console view automatically validates markup loaded onto a page. What's great about Firebug, you can select the error and be taken right to the offending line of code. I also find Firebug's explanation of what's wrong with it a little more understandable than some of the W3C's site, but then, Firebug also finds all sorts of "little things" that W3C doesn't and as far as I can tell, doesn't affect my jQuery development, so it's often a little simpler to use the W3C validator.

Check your PHP syntax

Just as with jQuery syntax, small, simple PHP syntax errors and typos always get me, even after all these years of experience.

If you're getting a PHP error, most of the time PHP will simply not render the entire page and display an error message with a note to the script page and number line in the code that's offensive. That makes it really easy to find and fix PHP issues.

Still it is possible to have a PHP syntax problem that doesn't throw an error. If you've checked everything else, take a quick run through the following common problems that happen with PHP.

PHP shorthand

Double check that you're not using any PHP shorthand. Make sure you have opening and closing `<?php ?>` brackets and make sure that you have `php` after the first bracket. Some systems don't have shorthand turned on and what works on your sandbox's hosted server or local server may not work on the live server. Avoid this by avoiding PHP shorthand syntax.

Check for proper semicolons

As with jQuery, because the syntax is rather simliar, you'll do well to use the **find** feature on your code editor and double check that statement closing semicolons are not written as colons `:` or commas `,` or missing altogether.

Concatenations

This gets tricky when going from JavaScript and jQuery to PHP; the syntax is very similar! However, concatenations in PHP are handled with a `.` (period) and not a `+` (plus) sign. It's easy to work with JavaScript and jQuery for a while and then try to work on the WordPress theme and keep using JavaScript syntax.

Summary

There you have it. I hope this list of tips and tricks for both jQuery and WordPress helps you out. We took a look at:

- Best ways to integrate jQuery with WordPress for maximum compatibility with other scripts, libraries, and plugins
- All the different ways that Firfox and Firebug are your development friend
- Tips and tricks for making sure you keep your WordPress user's job easy and your WordPress HTML valid and easy to work with

Up next our last and final chapter! If you even want to call it a "chapter", *Appendix A, jQuery and WordPress Reference Guide,* which provides a chock full of quick and easy reference lookups of the top jQuery and WordPress know-how and syntax that you'll need for most of your jQuery enhancement work.

jQuery and WordPress Reference Guide

OK! Welcome to the final part of this book! This appendix isn't like the others, in that it's really intended to be a quick reference guide to help you out now that you've read and understood the underlying principles, applications, and best practices of using jQuery within WordPress sites. Think of this chapter as your "Cheat Sheet".

Where applicable, I'll point you back to the location in the book where the function or technique was described in detail, as well as point you out to any other useful links if you're interested in more detail.

In this appendix, the topics that we'll cover include:

- Top, essential jQuery selectors and functions
- Working with the template hierarchy, and key template tags and WordPress functions
- WordPress shortcodes

jQuery reference for WordPress

In the next few sections, we'll take a look at the top references you'll need for jQuery development within WordPress. Let's get started with staying in `noConflict` mode and looking at the most useful `selector` filters.

noConflict mode syntax

The simplest is to just use the jQuery variable in all your selection statements:

```
jQuery('.selector').function();
```

You can also set up your own variable:

```
<script type="text/javascript">
var $jq = jQuery.noConflict();
 $jq(document).ready(function() {
   $jq("p").click(function() {
     alert("Hello world!");
   });
 });
</script>
```

You can even safely use the `$` variable if you set it up correctly:

```
jQuery(function ($) {
        /* jQuery only code using $ can safely go here */
      $("p").css('border','#ff6600');
});
```

Useful selector filters for working within WordPress

Remember: Sometimes it's easier to *exclude* what you don't want in a selection set, rather than select for everything you do want.

Selection filter syntax

Here's the basic syntax for working with selector filters:

```
jQuery('.selector:filter(params if any)').function();
```

Selector filters

Here are the top selector filters that you'll find most useful with WordPress (`:not` is my personal favorite):

Example	Syntax	Description
:not(selector)	`jQuery(".post img:not(.pIcon)").jqFn();`	Filters out all elements matching the given selector.
:header	`jQuery(".post :header").jqFn();`	Filters down to all elements that are headers, such as h1, h2, h3, and so on.
:first	`jQuery(".post:first").jqFn();`	Filters down to the first selected element only.
:last	`jQuery(".post:last").jqFn();`	Filters down to the last selected element only.
:even	`jQuery(".post:even").jqFn();`	Filters down to even elements only. Note: Arrays are zero-indexed! Zero is considered an even number, so your first item will be selected!
:odd	`jQuery(".post:odd").jqFn();`	Filters down to odd elements only. Note: Arrays are zero-indexed! Zero is considered an even number, so your second item will be selected!
:eq(number)	`jQuery(".post:eq(0)").jqFn();`	Filters down to a single element by its index, which again is zero-indexed.
:gt(number)	`jQuery(".post:gt(0)").jqFn();`	Filters down to all elements with an index **above** the given one, again this is zero-indexed.
:lt(number)	`jQuery(".post:lt(2)").jqFn();`	Filters all elements with an index below the given one.
:animated	`jQuery(".post:animated").jqFn();`	Filters down to all elements that are currently being animated (we'll get to animation later in this chapter).

Content filter syntax

After the regular selector filters, you'll find these content filters very useful (especially `:has()`).

```
jQuery(".selector:content-filter(params if any)").function();
```

Content filters

Pretty much all the content filters come in handy with WordPress. They help you work with what the Page and Post WYSIWYG editor's output very well.

Example	Syntax	Description
:has(selector)	`jQuery(".post:has(.entry)").css("background", "#f60");`	Filters down to elements that have at least one of the matching elements inside it.
:contains(text)	`jQuery(".post:contains('Hello world')").css("background", "#f60");`	Filters down to elements that contain the specific text. Note: This is **case sensitive**!
:empty	`jQuery(":empty')").css("background", "#f60");`	Filters down to elements that have no children. This includes text nodes.
:parent	`jQuery(":parent')").css("background", "#f60");`	Filters down to elements that are the parent of another element. This includes text nodes.

Child filter syntax

Here's the basic syntax for using child filer syntax:

```
jQuery(".selector:child-filter(params if any)").function();
```

Child filters

You'll find child filters will come in most handy when working with the various list tags that WordPress puts out. Categories, pages, gallery pages, you'll be able to control them and select specifics using these filters.

Example	Syntax	Description
:nth-child(number/even/odd)	`jQuery(".linkcat li:nth-child(1)").css("background", "#f60");`	Filters down to the elements that are the "nth" child of it's selector. Note, this is **not** zero-indexed! 1 and odd selects the **first** element.
:first-child	`jQuery(".linkcat li:first-child").css("background", "#f60");`	Filters down to the elements that are the first child of their parent.
:last-child	`jQuery(".linkcat li:last-child").css("background", "#f60");`	Filters down to the elements that are the last child of their parent.
:only-child	`jQuery(".pagenav li:only-child").css("background", "#f60");`	Filters down to the elements that are only-children of their parent. If a parent has more than one child, no elements are selected.

Form filter syntax

Here's the form filter syntax:

```
jQuery(":form-filter").function();
```

Form filters

WordPress natively has a simple content form and a single input field. However, the WordPress Cforms II plugin is quite invaluable for most projects, and if you're using that plugin, you'll find most of these filters helpful:

Example	Syntax	Description
:input	`jQuery("form:input"). css("background", "#f60");`	Filters to all input, textarea, select and button elements.
:text	`jQuery("form:text"). css("background", "#f60");`	Filters to all input elements that are type text.
:password	`jQuery("form:password"). css("background", "#f60");`	Filters to all input elements that are type password.
:radio	`jQuery("form:radio"). css("background", "#f60");`	Filters to all input elements that are type radio.
:checkbox	`jQuery("form:checkbox"). css("background", "#f60");`	Filters to all input elements that are type checkbox.
:submit	`jQuery("form:submit"). css("background", "#f60");`	Filters to all input elements that are type submit.

Example	Syntax	Description
:image	`jQuery("form:image"). css("background", "#f60");`	Filters to all image elements (classified as a form filter, but useful for regular images).
:reset	`jQuery("form:reset"). css("background", "#f60");`	Filters to all input elements that are type reset.
:button	`jQuery("form:button"). css("background", "#f60");`	Filters to all input elements that are type button.
:file	`jQuery("form:file"). css("background", "#f60");`	Filters to all input elements that are type file.

jQuery: Useful functions for working within WordPress

While I've recapped most of the selector filters as they're just that useful, in this next section I'll go over the syntax and usage for the top functions that you'll find you use the most in your WordPress projects.

Never fear, you can skim through *Chapter 2, Working with jQuery in WordPress* for a complete listing as well as usage of functions not covered here.

Working with classes and attributes

One of the most simple yet powerful things you can do quickly with jQuery is transform objects by changing their CSS properties.

Example	Syntax	Description
.css('property', 'value')	`jQuery(".post") .css("background", "#f60");`	Adds or changes the CSS properties of the selected elements.
.addClass('className')	`jQuery(".post") .addClass("sticky");`	Adds listed class(es) to each of the selected elements.
.removeClass('className')	`jQuery(".post") .removeClass("sticky");`	Removes listed class(es) from each of the selected elements.
.toggleClass('className', switch-optional)	`jQuery(".post") .toggleClass("sticky");`	Toggles listed class(es) from each of the selected elements based on their current state. If the class is there, it's removed; if it's not, it's added.

Example	Syntax	Description
.hasClass('className')	`jQuery(".post").hasClass("sticky");`	Returns true or false if listed class(es) from each of the selected elements exist.
.attr	`jQuery(".post").attr();`	Retrieves the attribute's value for the first element of the selected elements.

Traversing the DOM

`.append` and `.prepend` are going to be your most used DOM functions. However, you'll find `.wrapAll` invaluable for helping contain any new elements you create.

Example	Syntax	Description
.append(html & text)	`jQuery(".post").append("<b>post ends here</b>");`	Inserts content in the parameter, to the end of each selected element.
.appendTo(selector)	`jQuery("<b>post ends here</b>").appendTo(".post");`	Does the same thing as append, just reverses the element selection and content parameter.
.prepend(html & text)	`jQuery(".post").prepend("<b>post starts here</b>");`	Inserts content in the parameter, to the beginning of each selected element.
.prependTo(selector)	`jQuery("<b>post starts here</b>").prependTo(".post");`	Does the same thing as prepend, just reverses the element selection and content parameter.
.after(string)	`jQuery(".post").after("<b>This goes after</b>");`	Inserts content in the parameter, after and outside of each selected element.
.insertAfter(selector)	`jQuery("<b>This goes after</b>").insertAfter(".post");`	Does the same thing as after, just reverses the element selection and content parameter.
.before(html & text)	`jQuery(".post").before("<b>This goes before</b>");`	Inserts content in the parameter, before and outside of each selected element.
.insertBefore(selector)	`jQuery("<b>This goes before</b>").insertBefore("class");`	Does the same thing as before, just reverses the element selection and content parameter.

Example	Syntax	Description
.wrap(html or functionName)	`jQuery(".post").wrap("<div class=".fun" />");`	Wraps an HTML structure around each selected element. You can also construct a function that will wrap each element in HTML.
.wrapAll(html)	`jQuery(".post") .wrapAll("<div class=" .fun" />");`	Similar to wrap, but places the HTML structure around all of the elements together, not each individual element.
.wrapInner(selector)	`jQuery(".post") .wrapInner("<div class=" .fun" />");`	Similar to wrap, but it places the HTML structure *inside* each of the selected elements around any text or child elements of each selected element.
.html(html & text)	`jQuery(".post") .html("<h2>Replacement Text</h2>");`	Replaces any content and child elements of selected items with the content in the parameter.
.text(text only– html chars will be escaped)	`jQuery(".post") .text("Replacement Text");`	Similar to HTML, but text only. Any HTML characters will be escaped as ascii codes.

Important jQuery events

Most of the time in WordPress, it's all about `.click` and `.hover` but `.toggle` and `.dbclick` will come in handy as well.

Example	Syntax	Description
.click(functionName)	`jQuery(".post") .click(function(){// code});`	Binds a function to the click event type, executed on a single click.
.dbclick(functionName)	`jQuery(".post") .dbclick(function(){// code});`	Binds a function to the click event type, executed on a double click.
.hover(functionName1, functionName2)	`jQuery(".post") .hover(function(){// code});`	Works with the mouseenter/ mouseleave event types and binds just two functions to the selected elements, to be executed on mouseenter and mouseleave.
.toggle(functionName1, functionName2, functionName3, …)	`jQuery(".post") .toggle(function(){// code});`	Works with the click event type and binds two or more functions to the selected elements, to be executed on alternate clicks.

Animation at its finest

Anything that animates is going to look cool. Make sure that you know how to handle these functions for some top-notch jQuery enhancements.

Example	Syntax	Description
.slideUp(speed, functionName)	`jQuery(".post") .slideUp('slow', function() { // code });`	Slides the selected element up from bottom to top until it is hidden. Speed can be "fast" or "slow" or in milliseconds. A function can be called when the animation is finished.
.slideDown(speed, functionName)	`jQuery(".post") .slideDown('slow', function() { // code });`	Slides a hidden selected element down from top to bottom until it is defined size. Speed can be "fast" or "slow" or in milliseconds. A function can be called when the animation is finished.
.slideToggle()	`jQuery(".post") .slideToggle('slow', function() { // code });`	Toggles the visibility of the selected element using the slide animation. Speed can be "fast" or "slow" or in milliseconds. A function can be called when the animation is finished.
.fadeOut(speed, functionName)	`jQuery(".post") .fadeOut("slow", function(){//code});`	Fades a selected element that's visible or alpha is 1 to 0.
.fadeIn(speed, functionName)	`jQuery(".post") .fadeIn("slow", function(){//code});`	Fades a selected element who's visibility is hidden or alpha is set as 0 to 1.
.fadeTo(speed, alpha, functionName)	`jQuery(".post") .fadeTo("slow", .3, function(){//code});`	Fades a selected element to a specific alpha from 0 to 1.
.animate(css properties, duration, easing, functionName)	`jQuery(".post") .animate({width: 200, opacity: .25}, 1000, function(){//code});`	Creates a custom transition of CSS properties on the selected elements.
.stop()	`jQuery(".post") .stop();`	Stops an animation on a selected element.

Getting the most out of WordPress

Those are the top elements that you'll need to know for jQuery, now lets take a look at what can be done to keep things running smooth on the WordPress side. First up, the more you know how to leverage your theme's hierarchy the more easily you can create views and pages to leverage with jQuery.

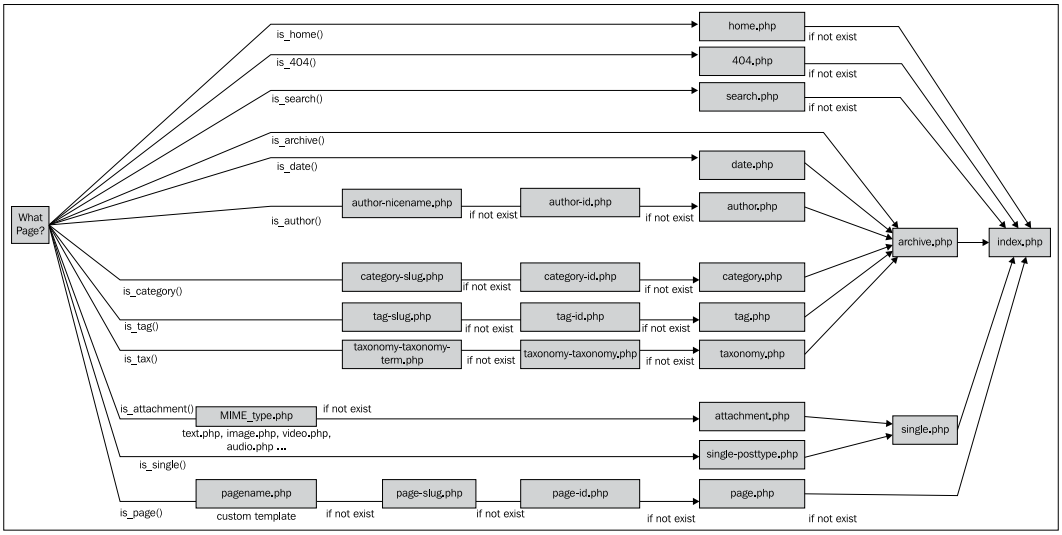
The WordPress template hierarchy

Need to work with the theme a bit? Understanding the Template Hierarchy can really help you create the view you need with minimal programming headaches. The following list contains the general template hierarchy's rules. The absolute simplest theme you can have must contain an `index.php` page. If no other specific template pages exist, then `index.php` is the default.

You can then begin expanding your theme by adding the following pages:

- `archive.php` trumps `index.php` when a category, tag, date, or author page is viewed.
- `home.php` trumps `index.php` when the home page is viewed.
- `single.php` trumps `index.php` when an individual post is viewed.
- `search.php` trumps `index.php` when the results from a search are viewed.
- `404.php` trumps `index.php`, when the URI address finds no existing content.
- `page.php` trumps `index.php` when looking at a static page.
    - A custom **template** page, such as: `page_about.php`, when selected through the page's **Administration** panel, trumps `page.php`, which trumps `index.php` when that particular page is viewed.
- `category.php` trumps `archive.php`. This then trumps `index.php` when a category is viewed.
    - A custom **category-ID** page, such as: `category-12.php` trumps `category.php`. This then trumps `archive.php`, which trumps `index.php`.

- `tag.php` trumps `archive.php`. This in turn trumps `index.php` when a tag page is viewed.
    - A custom **tag-tagname** page, such as: `tag-reviews.php` trumps `tag.php`. This trumps `archive.php`, which trumps `index.php`.
- `author.php` trumps `archive.php`. This in turn trumps `index.php`, when an author page is viewed.
- `date.php` trumps `archive.php`, This trumps `index.php` when a date page is viewed.



You can learn more about the WordPress theme template hierarchy here: `http://codex.wordpress.org/Template_Hierarchy`.

Top WordPress template tags

The following are the top WordPress template tags that I find most useful for helping out with jQuery and theme development:

Template Tag	Description	Parameters
`bloginfo()` Sample: `bloginfo('name');`	Displays your blog's information supplied by your user profile and general options in the Administration Panel. **More Info:** `http://codex.wordpress.org/Template_Tags/bloginfo`.	Any text characters you want to appear before and after the tags, as well as to separate them—`name`, `description`, `url`, `rdf_url`, `rss_url`, `rss2_url`, `atom_url`, `comments_rss2_url`, `pingback_url`, `admin_email`, `charset`, `version`. Default: No parameters will display anything. You must use a parameter.
`wp_title()` Sample: `wp_title('——',true,'');`	Displays the title of a page or single post. **Note:** Use this tag anywhere outside The Loop. **More Info:** `http://codex.wordpress.org/Template_Tags/wp_title`.	Any text characters you want to use to separate the title—`("--")`. You can set up a Boolean to display the title— `("--", "false")`. As of version 2.5+: You can decide if the separator goes before or after the title—`("--", true, "right")`. Default: No parameters will display the page title with a separator if a separator is assigned its default to the left.

Template Tag	Description	Parameters
`the_title()` Sample: `the_title('<h2>', '</h2>');`	Displays the title of the current post. **Note:** Use this tag in The Loop. (See *Chapter 3, Digging Deeper: Understanding jQuery and WordPress Together* for how to set up The Loop.) **More Info:** `http://codex.wordpress.org/Template_Tags/the_title`.	Any text characters you want to appear before and after the title—`("<h2>", "</h2>")`. You can also set a Boolean to turn the display to false—`("<h2>", "</h2>", "false")`. Default: No parameters will display the title without a markup.
`the_content()` Sample: `the_content( 'more_link_text', strip_teaser, 'more_file' );`	Displays the content and markup you've edited into the current post. **Note:** Use this tag in The Loop. (See *Chapter 3, Digging Deeper: Understanding jQuery and WordPress Together* for how to set up The Loop.) **More Info:** `http://codex.wordpress.org/Template_Tags/the_content`.	As you can add text to display the "more link", a Boolean to show or hide the "teaser text", there is a third parameter for more_file that currently doesn't work—`("Continue reading" . the_title())`. You can also set a Boolean to turn the display to false—`("<h2>", "</h2>", "false")`. Default: No parameters will display the content for the post with a generic "read more" link.
`the_category()` Sample: `the_category(', ');`	Displays a link to the category or categories a post is assigned to. **Note:** Use this tag in The Loop. (See *Chapter 3, Digging Deeper: Understanding jQuery and WordPress Together* for how to set up The Loop) **More Info:** `http://codex.wordpress.org/Template_Tags/the_category`.	You can include text separators in case there's more than one category—`('>')`. Default: No parameters will display a comma separation if there is more than one category assigned.

Template Tag	Description	Parameters
`the_author_meta()` Sample: `the_author_meta();`	Displays the author of a post or a page. **Note:** Use this tag in The Loop. (See *Chapter 3, Digging Deeper: Understanding jQuery and WordPress Together* for how to set up The Loop.) **More Info:** `http://codex.wordpress.org/Template_Tags/the_author_meta`.	This tag accepts a large amount of parameters. They are covered in the previous sections you can also check out the codex.

`wp_list_pages()`
Sample:
`wp_list_pages(`
`'title_li=');`
	Displays a list of WordPress pages as links.
	More Info:
	`http://codex.wordpress.org/Template_Tags/wp_list_pages`.
		`title_li` is the most useful as it wraps the page name and link in list tags `<li>`.
		The other parameters can be set by separating with an "`&`": `depth, show_date, date_format,`
		`child_of, exclude, echo, authors,`
		`sort_column.`
		Default: No parameters will display each title link in an `<li>` list and include an `<ul>` tag around the list (not recommended if you want to add your own custom items to the page navigation).

`wp_nav_menu();`
Sample:
`wp_nav_menu(`
`array('menu' => 'Main Nav' ));`
	Displays a list of menu items assigned to a WordPress 3.0+ menu, as links.
	More Info:
	`http://codex.wordpress.org/Function_Reference/wp_nav_menu`.
		This tag accepts a large amount of parameters, the most common parameter is the name of the menu set up in the menu tool in the Administration Panel.
		If no menu is available, the function will default to the `wp_list_pages();` tag.
		Please see the codex for more parameters.

Template Tag	Description	Parameters
`next_post_link()` Sample: `next_post_link( '<strong> %title </strong>');`	Displays a link to the next post which exists in chronological order from the current post. **Note:** Use this tag in The Loop. (See *Chapter 3, Digging Deeper: Understanding jQuery and WordPress Together* for how to set up The Loop.) **More Info:** `http://codex.wordpress.org/Template_Tags/next_post_link`.	Any markup and text characters you want to appear—`(<strong>%title</strong>)`. `%link` will display the permalink, `%title` the title of the next post. Default: No parameters will display the next post title as a link followed by angular quotes (>>).
`previous_post_link()` Sample: `previous_post_link( '<strong> %title </strong>');`	Displays a link to the previous post which exists in chronological order from the current post. **Note:** Use this tag in The Loop. (See *Chapter 3, Digging Deeper: Understanding jQuery and WordPress Together* for how to set up The Loop.) **More Info:** `http://codex.wordpress.org/Template_Tags/previous_post_link`.	Any markup and text characters you want to appear—`(<strong>%title</strong>)`. `%link` will display the permalink, `%title` the title of the next post. Default: No parameters will display the previous post title as a link preceded by angular quotes (<<).

Template Tag	Description	Parameters
`comments_number()` Sample: `comments_number( 'no responses', 'one response', '% responses');`	Displays the total number of comments, Trackbacks, and Pingbacks for a post. **Note:** Use this tag in The Loop. (See *Chapter 3*, *Digging Deeper: Understanding jQuery and WordPress Together* for how to set up The Loop.) **More Info:** `http://codex.wordpress.org/Template_Tags/comments_number`.	Lets you specify how to display if there are 0 comments, only 1 comment, or many comments— `('no responses','one response','% responses')`. You can also wrap items in additional markup— `("No Comments","<span class="bigNum">1</span> response","<span class="bigNum">%</span> Comments")`. Default: No parameters will display: No comments, or 1 comment, or ? comments.
`comments_popup_link()` Sample: `comments_popup_link( 'Add Your Thoughts' );`	If the comments_popup_ script is not used, this displays a normal link to comments. **Note:** Use this tag in The Loop. (See *Chapter 3*, *Digging Deeper: Understanding jQuery and WordPress Together* for how to set up The Loop.) **More Info:** `http://codex.wordpress.org/Template_Tags/comments_popup_link`.	Lets you specify how to display if there are 0 comments, only 1 comment, or many comments— `("No comments yet", "1 comment so far", "% comments so far (is that a lot?)", "comments-link", "Comments are off for this post")`. Default: No parameters will display the same default information as the `comments_number()` tag.

Template Tag	Description	Parameters

`edit_post_link()`
 	
Sample:
`edit_post_link(`
`'edit',`
`'<p>',`
`'</p>'`
`);`
	If the user is logged in and has permission to edit the post, this displays a link to edit the current post.
	
	Note: Use this tag in The Loop. (See *Chapter 3, Digging Deeper: Understanding jQuery and WordPress Together* for how to set up The Loop.)
	
	More Info:
	
	`http://codex.wordpress.org/Template_Tags/edit_post_link`.
		Any text that you want to be in the name of the link, plus markup that you'd like to come before and after it—`("edit me!", "<strong>", "</strong>")`.
		
		Default: No parameters will display a link that says "edit" with no additional markup.

`the_permalink()` Sample: `the_permalink();`	Displays the URL for the permalink to the current post. **Note:** Use this tag in The Loop. (See *Chapter 3, Digging Deeper: Understanding jQuery and WordPress Together* for how to set up The Loop.) **More Info:** `http://codex.wordpress.org/Template_Tags/the_permalink`.	This tag has no parameters.
`the_ID()` Sample: `the_ID();`	Displays the numeric ID of the current post. **Note:** Use this tag in The Loop. (See *Chapter 3, Digging Deeper: Understanding jQuery and WordPress Together* for how to set up The Loop.) **More Info:** `http://codex.wordpress.org/Template_Tags/the_ID`.	This tag has no parameters.

Template Tag	Description	Parameters
wp_get_archives() Sample: `wp_get_archives( 'type=monthly');`	Displays a date-based archives list. **More Info:** `http://codex.wordpress.org/Template_Tags/wp_get_archives.`	You can set parameters by separating them with an "&"—`('type=monthly&limit=12').` The other parameters are `type, limit, format, before, after, show_post_count.` Default: No parameters will display a list of all your monthly archives in HTML format without before or after markup and `show_post_count` set to false.
get_calendar() Sample: `get_calendar(false);`	Displays the current month/year calendar. **More Info:** `http://codex.wordpress.org/Template_Tags/get_calendar.`	A Boolean value can be set which will display a single-letter initial `(S = Sunday)` if set to true. Otherwise, it will display the abbreviation based on your localization `(Sun = Sunday)`—`(true)` Default: No parameters will display the single-letter abbreviation.

Conditional tags

The conditional tags can be used in your template files to change what content is displayed and how that content is displayed on a particular page depending on what conditions that page matches. For example, you might want to display a snippet of text above the series of posts, but only on the main page of your blog. With the `is_home()` conditional tag, that task is made easy.

There are conditional tags for just about everything, out of all of them, these are the seven I find I need most in my theme development:

- `is_admin()`
- `is_page()`
- `is_single()`
- `is_sticky()`
- `is_home()`
- `is_category()`
- `in_category()`

All of those functions can take the following parameters: either the `post ID` or `page ID` number, the post or page `title`, or the post or page slug.

The first conditional tag, `is_admin()`, you'll notice we used throughout this title, along with `is_home()` to load up our `wp_enqueue_scripts` so that we can avoid having the script load if we're looking at some aspect of the theme from the admin panel (reviewing theme's for instance). The scripts could conflict with scripts in the admin panel so it's best to make sure they only load when not loading the theme from the admin panel.

Also, if you have any jQuery scripts that only affects the home page, say, a "sticky post rotator" script or something similar, you might want to consider placing that `wp_enqueue_script` call inside an `if(is_home()){wp_enqueue_script(//)}` call. This way, the script will only load up on the page you need it, and not on every page of the site, even when it's not being used.

As for the rest of the conditional tags, as great as theme's are, I'm sure you've run into the conundrum that you or your client doesn't want the exact same sidebar on every single page or post.

I use these conditional tags so that specific pages can have particular styles or divs of content turned on and off, and display or not display specific content. These seven tags really help give my client's custom themed sites a true, custom website feel and not that standard: "nice design, but every page has the exact same sidebar, this is probably another WordPress site" feel.

The conditional tag fun doesn't end there. There are many more that you may find invaluable in aiding your theme's customization, listed here: `http://codex.wordpress.org/Conditional_Tags`.

Quick overview of loop functions

All those template and conditional tags are one thing, it's another to apply them within the loop. In quite a few chapters of this book, we had to amend the loop in a theme's template file or create a custom one. The following table contains a description for each part of The Loop.

Loop functions	Description
`<?php if(have_posts()) : ?>`	This function checks to make sure that there are posts to display. If so, the code continues onto the next function below.
`<?php while(have_posts()) : the_post(); ?>`	This function shows the posts that are available and continues onto the next function below.
`<?php endwhile; ?>`	This function closes the `while(have_posts...` loop that was opened above once the available posts have been displayed.
`<?php endif; ?>`	This function ends the `if(have_posts...` statement that was opened above once the `while(have_posts...` loop has completed.

Setting up WordPress shortcodes

This whole appendix has been about useful references. We should take a quick look at shortcodes. They were first introduced in version 2.5. If you're comfortable with writing functions in WordPress, shortcodes can help you take longer bits of code (such as custom loops and complex template tag strings) or even just markup and text that you feel you'd use a lot in your theme (or plugin) and allow you to compress them in to cleaner, simpler bit of reusable code. You can add shortcodes to your theme's `functions.php` file.

You're probably familiar with shortcodes and may not realize it. If you've ever taken a look at how WordPress's media manager inserts captions into images, you've probably noticed something like:

```
...
[caption id="attachment_12" align="alignleft" width="150"
  caption="this is my caption"]<img src.../>[/caption]
...
```

That's a built in shortcode for captions and alignment in WordPress.

To create a shortcode, you do need to create a PHP function in your theme's `functions.php` file. If your theme does not have a `functions.php` file, simply create a new file and name it `functions.php` and place it in the root of your theme's directory.

Creating a basic shortcode

We'll start off by opening up our `functions.php` file and at the end of it, create a simple function that returns a string of text and markup for our shortcode like so:

```
<?php
...

function quickadd() {

   //code goes here
  $newText = 'This page is brought to you by
    <a href="#">the letter Z</a>';

  return $newText;
}

?>
```

Now, to really take advantage of shortcodes, you do need to know some PHP which, to fully cover, is a bit beyond the scope of this title. But even without much PHP experience, if you follow this example, you'll start to see how flexible this WordPress feature is in saving you time, not just in your theme, but in your day-to-day use of WordPress.

In the previous sample, inside our function brackets {}, I set up a very basic variable **`$donateText`** and assigned it a string of text and markup.

The `return` statement is a very basic PHP function that will make sure our `quickadd` function passes back whatever has been assigned to that variable.

We're now ready to use WordPress' `add_shortcode()` function by adding it just *underneath* our `quickadd` function that we previously set up. The `add_shortcode` function has two parameters. For the first parameter, you'll enter in a reference name for your shortcode and in the second, you'll enter in the name of the function that you'd like your shortcode to call, like so:

```
...

add_shortcode('broughtby', 'quickadd');

?>
```

Now the fun part: Pick any template page in your theme and use the `broughtby` shortcode by simply adding in:

```
...

[broughtby]

...
```

Wherever you paste that `[broughtby]` shortcode in your theme's template files, the **This page is brought to you by the letter Z** text, with a link to the letter's page, will appear!

Bonus: You are not restricted to using this shortcode in just your template files! Paste it directly into a post or page through the Administration panel, you'll get the same result. And, you guessed it, the output from shortcodes are easily leveraged and enhanced by jQuery!

If you have an enhancement that might take a bit more HTML markup than the WYSIWYG editor in WordPress can handle, and the site's content editor is overwhelmed by switching over to the HTML view, creating a solution with a shortcode may be just what you need! For example, it would be much easier for your client to add a set of square brackets with some parameters than mark up a definition list, which the basic WYSIWYG editor doesn't allow for.

This would turn:

```
...
<dl>
  <dt><a href='#'>Event Name and Location</a></dt>
  <dl>Event description</dl>
</dl>
...
```

Into a simpler:

```
...
[event title="Event Name and Location"
  description="Event description" url="#"]
...
```

In addition to helping out WordPress content editors with markup, shortcodes are a wonderful way to shortcut your time if you're a busy WordPress content author. Even if you're not creating your own theme from scratch, you can easily add your own shortcodes to any theme's `functions.php` file and ramp up your productivity.

Those of you more comfortable with PHP can take a look at WordPress' Shortcode API and see how to extend and make your shortcodes even more powerful by adding parameters to them: `http://codex.wordpress.org/Shortcode_API`.

Summary

Hopefully after taking a run through this appendix, you've dog-eared it and trust that you'll be back anytime you have a quick question about usage or syntax with the top jQuery and WordPress issues most WordPress developers have regarding jQuery. We also took a quick turn through the "underbelly" of WordPress' core functions and shortcodes which hopefully open you up to the endless possibilities of creating useful enhancements for WordPress sites. I hope you've enjoyed this book and found it useful in aiding your WordPress sites with jQuery creations and enhancements.

Index

Symbols

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Thank you for buying

WordPress 3.0 jQuery

About Packt Publishing

Packt, pronounced 'packed', published its first book "*Mastering phpMyAdmin for Effective MySQL Management*" in April 2004 and subsequently continued to specialize in publishing highly focused books on specific technologies and solutions.

Our books and publications share the experiences of your fellow IT professionals in adapting and customizing today's systems, applications, and frameworks. Our solution based books give you the knowledge and power to customize the software and technologies you're using to get the job done. Packt books are more specific and less general than the IT books you have seen in the past. Our unique business model allows us to bring you more focused information, giving you more of what you need to know, and less of what you don't.

Packt is a modern, yet unique publishing company, which focuses on producing quality, cutting-edge books for communities of developers, administrators, and newbies alike. For more information, please visit our website: `www.packtpub.com`.

About Packt Open Source

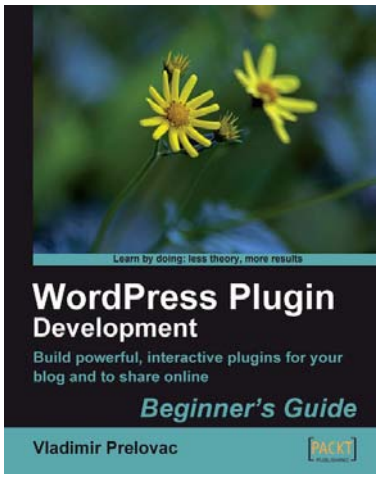
In 2010, Packt launched two new brands, Packt Open Source and Packt Enterprise, in order to continue its focus on specialization. This book is part of the Packt Open Source brand, home to books published on software built around Open Source licences, and offering information to anybody from advanced developers to budding web designers. The Open Source brand also runs Packt's Open Source Royalty Scheme, by which Packt gives a royalty to each Open Source project about whose software a book is sold.

Writing for Packt

We welcome all inquiries from people who are interested in authoring. Book proposals should be sent to author@packtpub.com. If your book idea is still at an early stage and you would like to discuss it first before writing a formal book proposal, contact us; one of our commissioning editors will get in touch with you.

We're not just looking for published authors; if you have strong technical skills but no writing experience, our experienced editors can help you develop a writing career, or simply get some additional reward for your expertise.

[PACKT] open source
community experience distilled
PUBLISHING



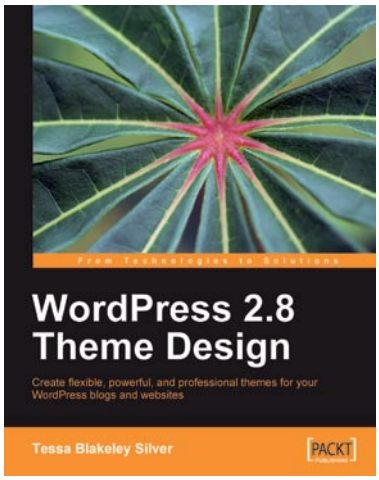
WordPress Plugin Development: Beginner's Guide

ISBN: 978-1-847193-59-9 Paperback: 296 pages

Build powerful, interactive plug-ins for your blog and to share online

1. Everything you need to create and distribute your own plug-ins following WordPress coding standards
2. Walk through the development of six complete, feature-rich, real-world plug-ins that are being used by thousands of WP users
3. Written by Vladimir Prelovac, WordPress expert and developer of WordPress plug-ins such as Smart YouTube and Plugin Central

---



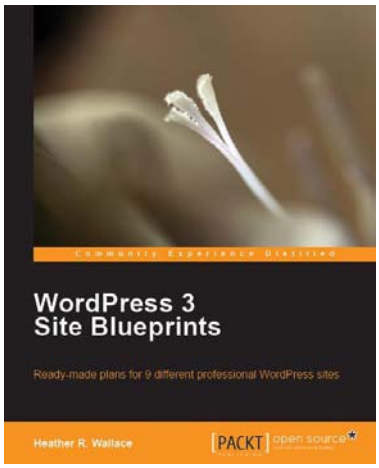
WordPress 2.8 Theme Design

ISBN: 978-1-849510-08-0 Paperback: 292 pages

Create flexible, powerful, and professional themes for your WordPress blogs and web sites

1. Take control of the look and feel of your WordPress site by creating fully functional unique themes that cover the latest WordPress features
2. Add interactivity to your themes using Flash and AJAX techniques
3. Expert guidance with practical step-by-step instructions for custom theme design
4. Includes design tips, tricks, and troubleshooting ideas

[PACKT] open source
community experience distilled
PUBLISHING



WordPress 3 Site Blueprints

ISBN: 978-1-847199-36-2 Paperback: 230 pages

Ready-made plans for 9 different professional WordPress sites

1. Everything you need to build a varied collection of feature-rich customized WordPress websites for yourself
2. Transform a static website into a dynamic WordPress blog
3. In-depth coverage of several WordPress themes and plugins
4. Packed with screenshots and step-by-step instructions to help you complete each site

---

jQuery Plugin Development Beginner's Guide

ISBN: 978-1-849512-24-4 Paperback: 308 pages

A practical straightforward guide for creating your first jQuery plugin

1. Utilize jQuery's plugin framework to create a wide range of useful jQuery plugins from scratch
2. Understand development patterns and best practices and move up the ladder to master plugin development
3. Discover the ins and outs of some of the most popular jQuery plugins in action

Please check **www.PacktPub.com** for information on our titles