



# WordPress 3.0 jQuery

**Tessa Blakeley Silver**



## Chapter No. 3

### "Digging Deeper: Understanding jQuery and WordPress Together"

## In this package, you will find:

A Biography of the author of the book

A preview chapter from the book, Chapter NO.3 "Digging Deeper: Understanding jQuery and WordPress Together"

A synopsis of the book's content

Information on where to buy this book

## 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.

**For More Information:** [www.packtpub.com/wordpress-30-jquery/book](http://www.packtpub.com/wordpress-30-jquery/book)

---

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).

---

**For More Information:** [www.packtpub.com/wordpress-30-jquery/book](http://www.packtpub.com/wordpress-30-jquery/book)

# WordPress 3.0 jQuery

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.

|  |
|--|
| For More Information: <a href="http://www.packtpub.com/wordpress-30-jquery/book">www.packtpub.com/wordpress-30-jquery/book</a> |
|--|

*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.

*Chapter 7, 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, especially 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.

**For More Information:** [www.packtpub.com/wordpress-30-jquery/book](http://www.packtpub.com/wordpress-30-jquery/book)

# 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).

For More Information: [www.packtpub.com/wordpress-30-jquery/book](http://www.packtpub.com/wordpress-30-jquery/book)

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.

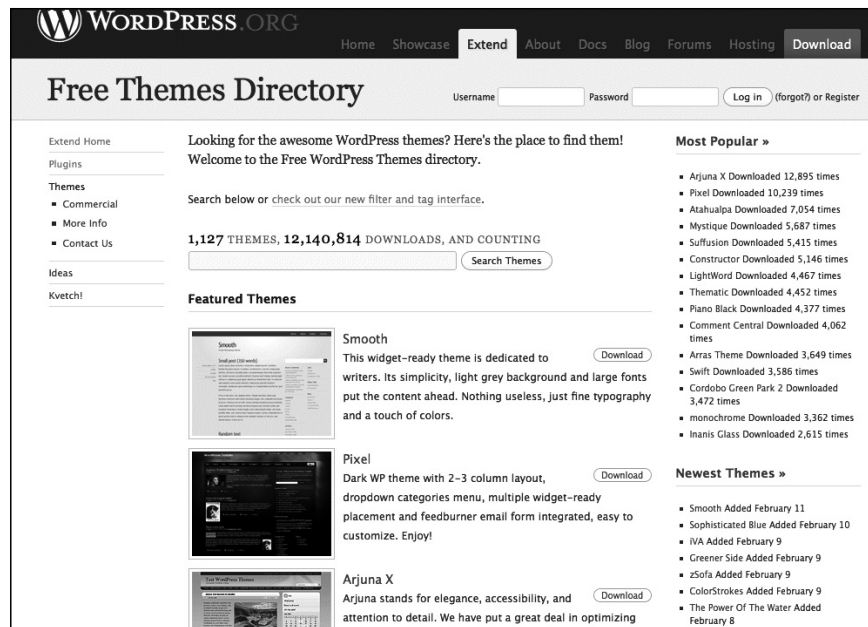
Speaking of jQuery enhancements, jQuery scripts can be turned into their own type of plugins, not to be confused with WordPress plugins. This makes the work you do in jQuery easily portable to different projects and uses.

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.

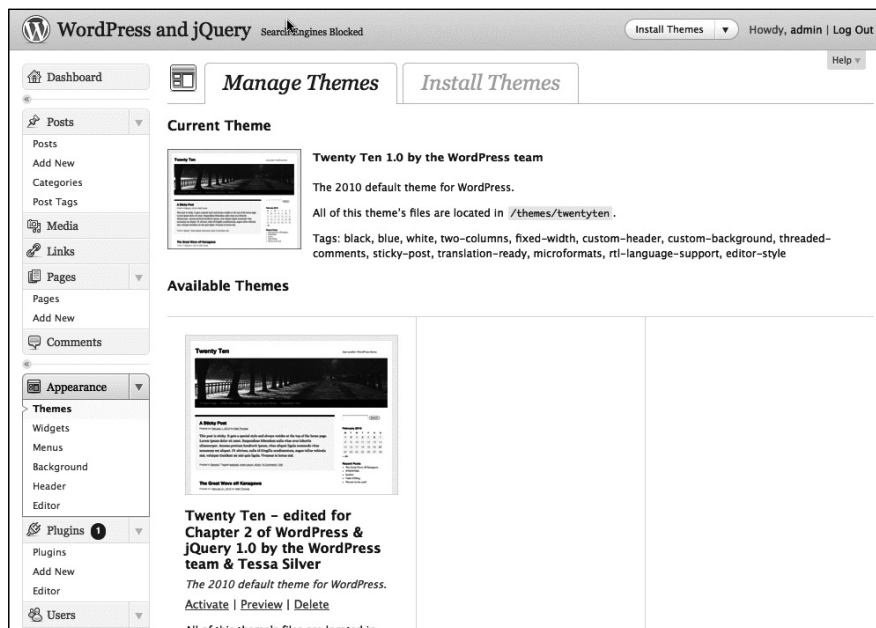
By understanding the basics of editing themes and creating your own WordPress and jQuery plugins, you'll be ready to traverse the world of third-party creations and find the best solutions for your projects. You'll also be able to determine if it's better or faster to work with another developer's themes,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

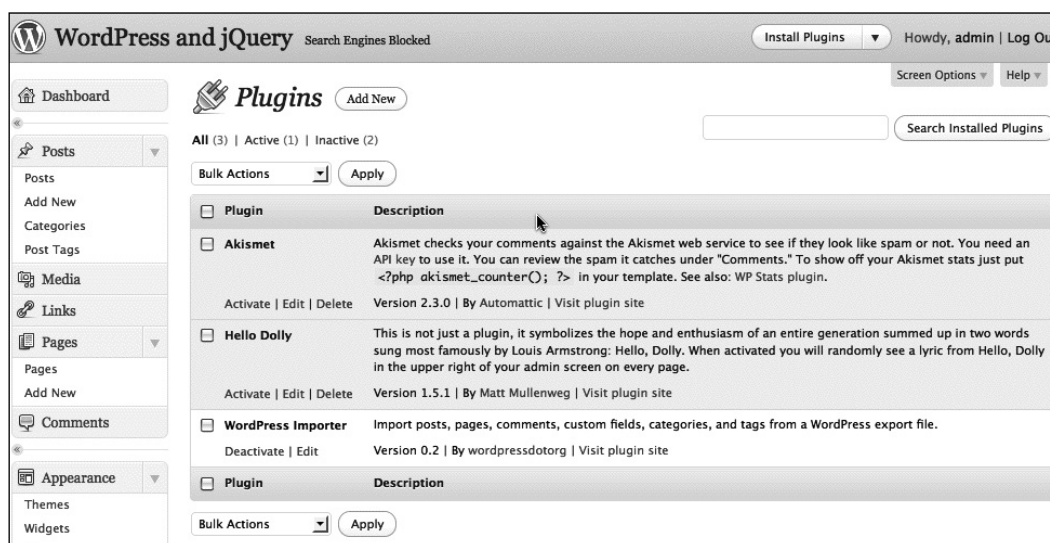
## WordPress plugins overview

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:

The screenshot shows the WordPress.org Plugin Directory page. At the top is the WordPress logo and the text 'WORDPRESS.ORG'. Below this is a navigation bar with links: Home, Showcase, Extend (active), About, Docs, Blog, Forums, Hosting, and Download. A search bar is located in the top right corner. The main heading is 'Plugin Directory'. Below the heading is a login section with fields for Username and Password, and a 'Log in' button, with links for '(forgot?)' and 'Register'. The page is divided into several sections: 'Extend Home' with links to Plugins, Themes, Ideas, and Kvetch!; 'Popular Tags' with a list of tags like widget (1316), Post (1017), plugin (800), admin (757), posts (739), sidebar (676), comments (587), images (446), links (413), and page (408); 'Featured Plugins' with a list of plugins including PollDaddy Polls & Ratings, WP Super Cache, and IntenseDebate Comments, each with a 'Download' button; 'Most Popular' with a list of popular plugins and their download counts; and 'Newest Plugins' with a list of recently added plugins.

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:

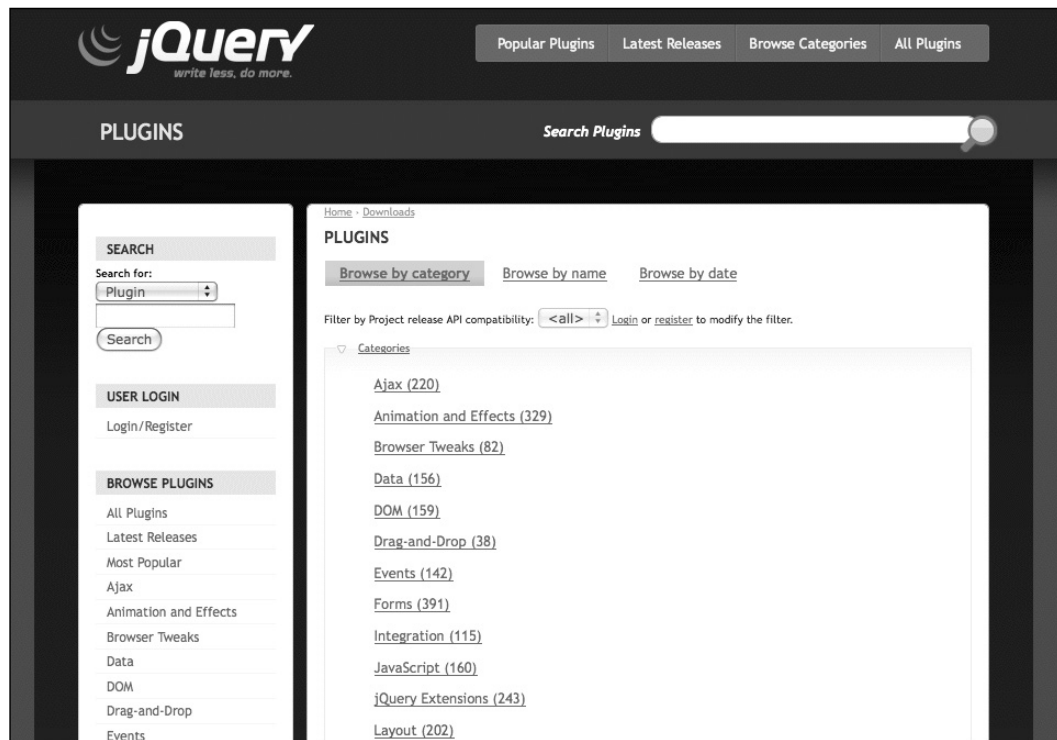


So how does a WordPress plugin differ from a jQuery plugin? In theory and intent, not that much, but in practice, there are quite a few differences. Let's take a look at jQuery plugins.

## 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.

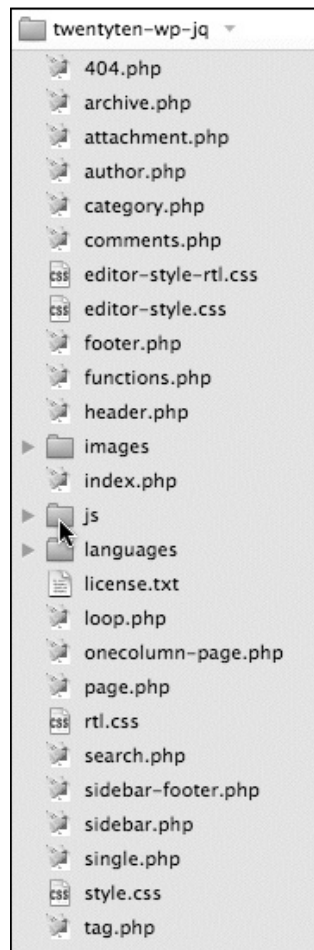
## Understanding the template's hierarchy

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**:

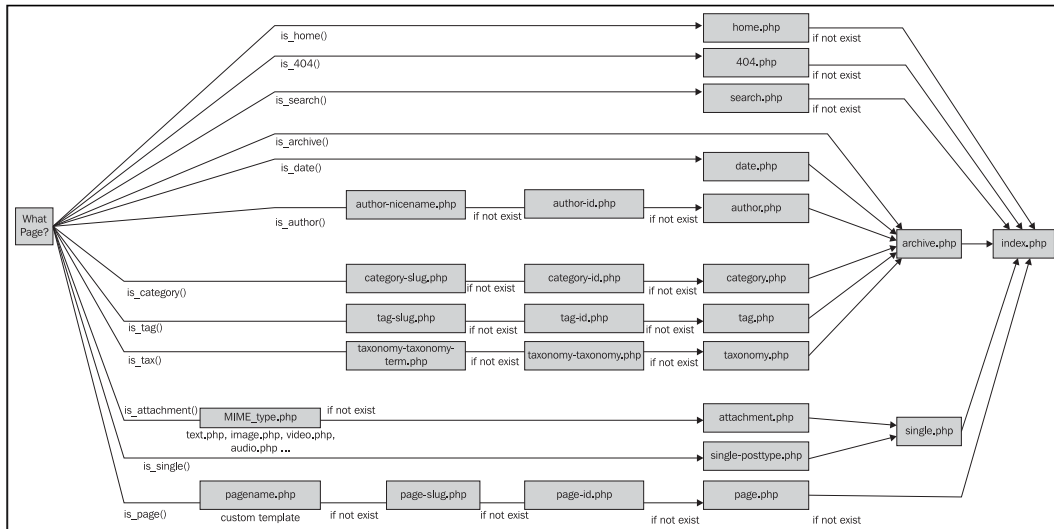



The next list contains the general template hierarchy rules. The absolute simplest theme you can have must contain an `index.php` page. If no other specific template pages exist, then `index.php` is the default.

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.

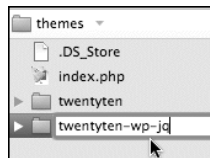


[  You can learn more about the WordPress theme template hierarchy here: [http://codex.wordpress.org/Template\\_Hierarchy](http://codex.wordpress.org/Template_Hierarchy). ]

## 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:

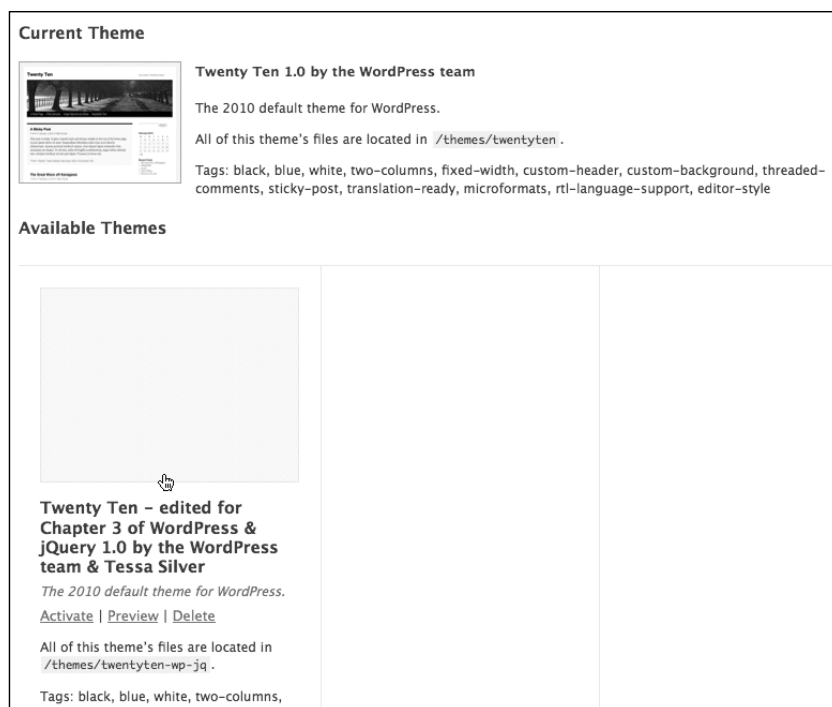


I'll then open up the `style.css` file and modify the information at the beginning of the CSS file:

```
/*
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 `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](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](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](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](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](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](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](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](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.

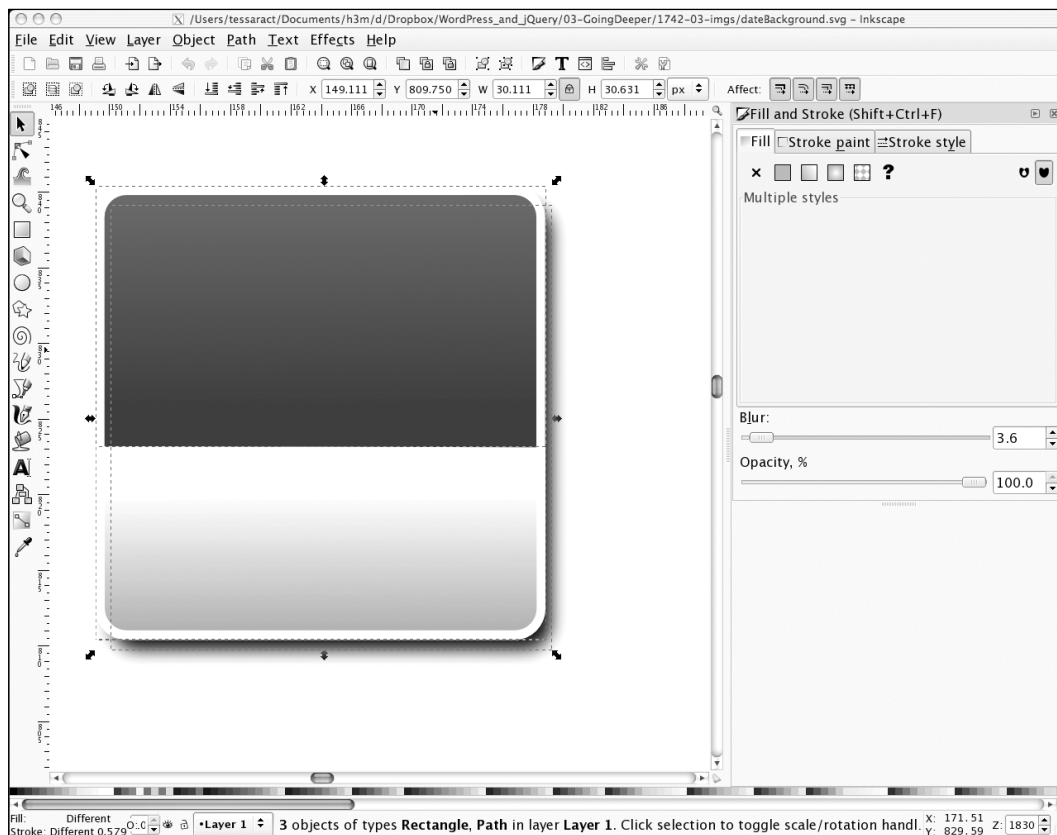
Granted, in *Chapter 2, Working with jQuery in WordPress*, I strongly advocated that you learn how to handle jQuery's selectors, inside and out, specifically so that you would be able to enhance any WordPress site without having to edit its theme. While you should know your jQuery selectors and filters 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 `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.

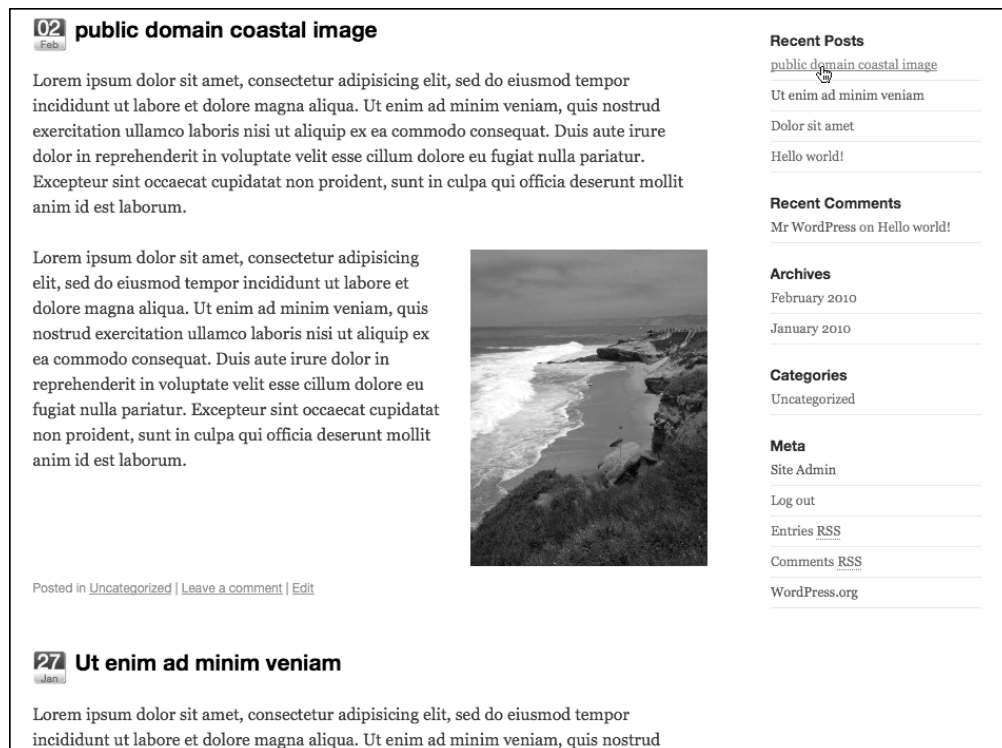
```
...
<ul class="xoxo currentsidebar">
...
<ul class="xoxo currentsidebar">
...
```

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 `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](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 `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 `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](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' );
    }
}

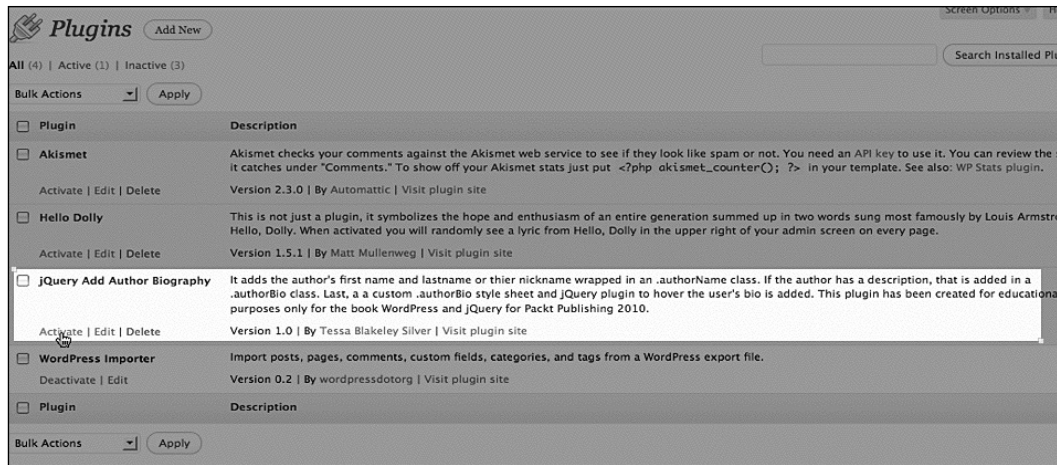
//evoke the authorCSS function on WordPress initialization
add_action('init', 'authorCSS');
```

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

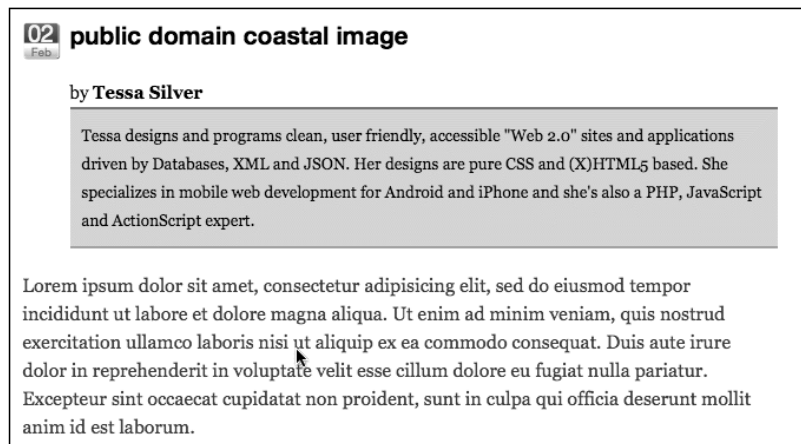
## Activating our plugin in WordPress

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");

    });
}
};
};

```

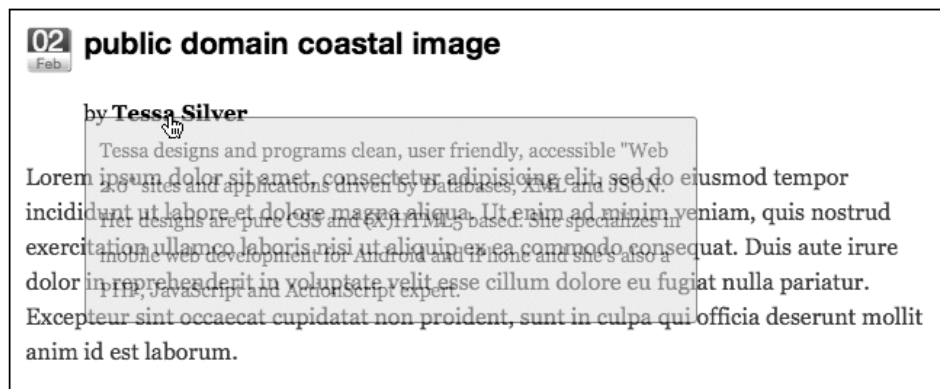
That's all it takes. Now that we've created a jQuery plugin script, let's quickly test it out in our theme first. All we have to do is embed our new jQuery plugin named `jquery.authover.js` to our theme, *under* the `wp_enqueue_script` call, *below* the `wp_head` hook and evoke it with a simple script:

```

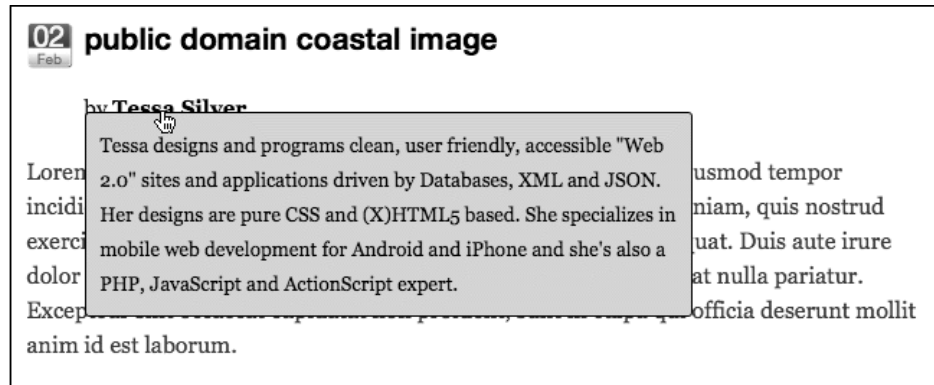
...
<script type="text/javascript">
jQuery(function() {
    jQuery(".authorName").authorHover();
});
</script>
...

```

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_enqueue_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:

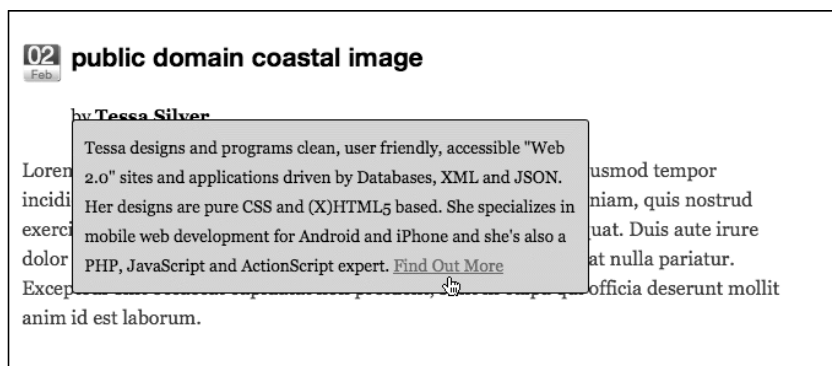
```
...
function addAuthorHover() {
    echo '<script type="text/javascript">
jQuery(function() {
    jQuery(".authorName").authorHover();
});
</script>';
}
...
```

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.

## Where to buy this book

You can buy WordPress 3.0 jQuery from the Packt Publishing website:

<https://www.packtpub.com/wordpress-30-jquery/book>

Free shipping to the US, UK, Europe and selected Asian countries. For more information, please read our [shipping policy](#).

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.



[www.PacktPub.com](http://www.PacktPub.com)

For More Information: [www.packtpub.com/wordpress-30-jquery/book](http://www.packtpub.com/wordpress-30-jquery/book)