From Technologies to Solutions

Learning jQuery 1.3

Better Interaction Design and Web Development with Simple JavaScript Techniques

Foreword by John Resig, Creator of jQuery

Jonathan Chaffer **Karl Swedberg**

PACKT PUBLISHING

Learning jQuery 1.3

Better Interaction Design and Web Development with Simple JavaScript Techniques

Jonathan Chaffer

Karl Swedberg

PACKT PUBLISHING

BIRMINGHAM - MUMBAI

Learning jQuery 1.3

First published: February 2009

Production Reference: 1040209

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

ISBN 978-1-847196-70-5

www.packtpub.com

Cover Image by Karl Swedberg (karl@englishrules.com)

Credits

Authors

Jonathan Chaffer

Karl Swedberg

Reviewers

Akash Mehta

Dave Methvin

Mike Alsup

Senior Acquisition Editor

Douglas Paterson

Development Editor

Usha Iyer

Technical Editor

John Antony

Editorial Team Leader

Akshara Aware

Production Editorial Manager

Abhijeet Deobhakta

Project Team Leader

Lata Basantani

Project Coordinator

Leena Purkait

Indexer

Rekha Nair

Proofreader

Jeff Orloff

Production Coordinator

Aparna Bhagat

Cover Work

Aparna Bhagat

Foreword

I feel honored knowing that Karl Swedberg and Jonathan Chaffer undertook the task of writing Learning jQuery. As the first book about jQuery, it set the standard that other jQuery — and, really, other JavaScript books in general — have tried to match. It's consistently been one of the top selling JavaScript books since its release, in no small part due to its quality and attention to detail.

I'm especially pleased that it was Karl and Jonathan who wrote the book since I already knew them so well and knew that they would be perfect for the job. Being part of the core jQuery team, I've had the opportunity to come to know Karl quite well over the past couple years, and especially within the context of his book writing effort. Looking at the end result, it's clear that his skills as both a developer and a former English teacher were perfectly designed for this singular task.

I've also had the opportunity to meet both of them in person, a rare occurrence in the world of distributed Open Source projects, and they continue to be upstanding members of the jQuery community.

The jQuery library is used by so many different people in the jQuery community. The community is full of designers, developers, people who have experience programming, and those who don't. Even within the jQuery team, we have people from all backgrounds providing their feedback on the direction of the project. There is one thing that is common across all of jQuery's users, though: We are a community of developers and designers who want JavaScript development to be made simple.

It's almost a cliché, at this point, to say that an open source project is community-oriented, or that a project wants to focus on helping new users get started. But it's not just an empty gesture for jQuery; it's the liquid-oxygen fuel for the project. We actually have more people in the jQuery team dedicated to managing the jQuery community, writing documentation, or writing plugins than actually maintaining the core code base. While the health of the library is incredibly important, the community surrounding that code is the difference between a floundering, mediocre project and one that will match and exceed your every need.

How we run the project, and how you use the code, is fundamentally very different from most open source projects — and most JavaScript libraries. The jQuery project and community is incredibly knowledgeable; we understand what makes jQuery a different programming experience and do our best to pass that knowledge on to fellow users.

The jQuery community isn't something that you can read about to understand; it's something that you actually have to participate in for it to fully sink in. I hope that you'll have the opportunity to partake in it. Come join us in our forums, mailing lists, and blogs and let us help guide you through the experience of getting to know jQuery better.

For me, jQuery is much more than a block of code. It's the sum total of experiences that have transpired over the years in order to make the library happen. The considerable ups and downs, the struggle of development together with the excitement of seeing it grow and succeed. Growing close with its users and fellow team members, understanding them and trying to grow and adapt.

When I first saw this book talk about jQuery and discuss it like a unified tool, as opposed to the experiences that it's come to encapsulate for me, I was both taken aback and excited. Seeing how others learn, understand, and mold jQuery to fit them is much of what makes the project so exhilarating.

I'm not the only one who enjoys jQuery on a level that is far different from a normal tool-user relationship. I don't know if I can properly encapsulate why this is, but I've seen it time and time again — the singular moment when a user's face lights up with the realization of just how much jQuery will help them.

There is a specific moment where it just clicks for a jQuery user, when they realize that this tool that they were using was in fact much, much more than just a simple tool all along — and suddenly their understanding of how to write dynamic web applications completely shifts. It's an incredible thing, and absolutely my favorite part of the jQuery project.

I hope you'll have the opportunity to experience this sensation as well.

John Resig
Creator of jQuery

About the Authors

Jonathan Chaffer is the Chief Technology Officer of Structure Interactive, an interactive agency located in Grand Rapids, Michigan. There, he oversees web development projects using a wide range of technologies, and continues to collaborate on day-to-day programming tasks as well.

In the open-source community, Jonathan has been very active in the Drupal CMS project, which has adopted jQuery as its JavaScript framework of choice. He is the creator of the Content Construction Kit, a popular module for managing structured content on Drupal sites. He is responsible for major overhauls of Drupal's menu system and developer API reference.

Jonathan lives in Grand Rapids with his wife, Jennifer.

I would like to thank Jenny for her tireless enthusiasm and support, Karl for the motivation to continue writing when the spirit is weak, and the Ars Technica community for constant inspiration toward technical excellence.

Karl Swedberg is a web developer at Fusionary Media in Grand Rapids, Michigan, where he spends much of his time implementing design with a focus on "web standards"—semantic HTML, well-mannered CSS, and unobtrusive JavaScript. A member of the jQuery Project Team and an active contributor to the jQuery discussion list, Karl has presented at workshops and conferences and provided corporate training in Europe and North America.

Before his current love affair with web development, Karl worked as a copy editor, a high-school English teacher, and a coffee house owner. His fascination with technology began in the early 1990s when he worked at Microsoft in Redmond, Washington, and it has continued unabated ever since.

Karl would rather be spending time with his wife, Sara, and his two children, Benjamin and Lucia.

I wish to thank my wife, Sara, for her steadfast love and support. Thanks also to my two delightful children, Benjamin and Lucia. Jonathan Chaffer has my deepest respect for his programming expertise and my gratitude for his willingness to write this book with me.

Many thanks to John Resig for creating the world's greatest JavaScript library and for fostering an amazing community around it. Thanks also to the folks at Packt Publishing, the technical reviewers of this book, the jQuery Cabal, and the many others who have provided help and inspiration along the way.

About the Reviewers

Akash Mehta is a web application developer, technical writer and business consultant based in Brisbane, Australia. His past projects include brochure websites, e-learning solutions and information systems. He has written web development articles for several of publishers in print and online, is a regular speaker at local conferences, and contributes to prominent PHP blogs.

As a student, Akash maintained PHP web applications and built user interfaces using the jQuery toolkit. While pursuing a degree in both commerce and IT, Akash develops web applications on PHP and Python platforms. After hours, he organizes his local PHP user group.

Akash develops applications on a wide range of open source libraries. His toolbox includes a number of application frameworks, including the Zend Framework, CakePHP and Django; Javascript frameworks like jQuery, Prototype and Mootools, platforms such as Adobe Flash/Flex, and the MySQL and SQLite database engines.

Currently, Akash provides freelance technical writing and web development through his website, `http://bitmeta.org`.

Dave Methvin has more than 25 years of software development experience in both the Windows and Unix environments. His early career focused on embedded software in the fields of robotics, telecommunications, and medicine. Later, he moved to PC-based software projects using C/C++ and web technologies.

Dave also has more than 20 years of experience in computer journalism. He was Executive Editor at PC Tech Journal and Windows Magazine, covering PC and Internet issues; his how-to columns on JavaScript offered some of the first cut-and-paste solutions to common web page problems. He was also a co-author of the book "Networking Windows NT" (John Wiley & Sons, 1997).

Currently, Dave is Chief Technology Officer at PC Pitstop, a web site that helps users fix and optimize the performance of their computers. He is also active in the jQuery community.

Mike Alsup has been involved with the jQuery project since near its inception and has contributed many popular plugins to the community. He is an active participant in the jQuery Google Group where he frequently provides support to new jQuery users.

Mike lives in upstate NY with his wife, Diane, and their triplet teenage sons. He is a Senior Software Developer at Click Commerce, Inc. where he focuses on Java, Swing, and web application development.

His jQuery plugins can be found at `http://jquery.malsup.com/`

Table of Contents

Preface

It began as a labor of love back in 2005 by John Resig, a JavaScript wunderkind who now works for the Mozilla Corporation. Inspired by pioneers in the field such as Dean Edwards and Simon Willison, Resig put together a set of functions to make it easy to programmatically find elements on a web page and assign behaviors to them. By the time he first publicly announced his project in January 2006, he had added DOM modification and basic animations. He gave it the name jQuery to emphasize the central role of finding, or "querying," parts of a web page and acting on them with JavaScript. In the few short years since then, jQuery has grown in its feature set, improved in its performance, and gained widespread adoption by some of the most popular sites on the Internet. While Resig remains the lead developer of the project, jQuery has blossomed, in true open-source fashion, to the point where it now boasts a core team of top-notch JavaScript developers, as well as a vibrant community of thousands of developers.

The jQuery JavaScript Library can enhance your websites regardless of your background. It provides a wide range of features, an easy-to-learn syntax, and robust cross-platform compatibility in a single compact file. What's more, hundreds of plug-ins have been developed to extend jQuery's functionality, making it an essential tool for nearly every client-side scripting occasion.

Learning jQuery provides a gentle introduction to jQuery concepts, allowing you to add interactions and animations to your pages—even if previous attempts at writing JavaScript have left you baffled. This book guides you past the pitfalls associated with AJAX, events, effects, and advanced JavaScript language features, and provides you with a brief reference to the jQuery library to return to again and again.

What this book covers

In Chapter 1 you'll get your feet wet with the jQuery JavaScript library. The chapter begins with a description of jQuery and what it can do for you. It walks you through downloading and setting up the library, as well as writing your first script.

In *Chapter 2* you'll learn how to use jQuery's selector expressions and DOM traversal methods to find elements on the page, wherever they may be. You'll use jQuery to apply styling to a diverse set of page elements, sometimes in a way that pure CSS cannot.

In *Chapter 3* you'll use jQuery's event-handling mechanism to fire off behaviors when browser events occur. You'll see how jQuery makes it easy to attach events to elements unobtrusively, even before the page finishes loading. And, you'll be introduced to more advanced topics, such as event bubbling, delegation, and namespacing.

In *Chapter 4* you'll be introduced to jQuery's animation techniques and see how to hide, show, and move page elements with effects that are both useful and pleasing to the eye.

In *Chapter 5* you'll learn how to change your page on command. This chapter will teach you how to alter the very structure of an HTML document, as well as its content, on the fly.

In *Chapter 6* you'll discover the many ways in which jQuery makes it easy to access server-side functionality without resorting to clunky page refreshes.

In the next three chapters (7, 8, and 9) you'll work through several real-world examples, pulling together what you've learned in previous chapters and creating robust jQuery solutions to common problems.

In *Chapter 7*, "Table Manipulation," you'll sort, sift, and style information to create beautiful and functional data layouts.

In *Chapter 8*, "Forms with Function," you'll master the finer points of client-side validation, design an adaptive form layout, and implement interactive client-server form features such as autocompletion.

In *Chapter 9*, "Shufflers and Rotators," you'll enhance the beauty and utility of page elements by showing them in more manageable chunks. You'll make information fly in and out of view both on its own and under user control.

Chapters 10 and 11 take you beyond the core jQuery methods to explore third-party extensions to the library, and show you various ways you can extend the library yourself.

In *Chapter 10*, "Using Plug-ins," you'll examine the Form plug-in and the official collection of user interface plug-ins known as jQuery UI. You'll also learn where to find many other popular jQuery plug-ins and see what they can do for you.

In *Chapter 11*, "Developing Plug-ins," you'll learn how to take advantage of jQuery's impressive extension capabilities to develop your own plug-ins from the ground up. You'll create your own utility functions, add jQuery object methods, write custom selector expressions, and more.

In Appendix A, "Online Resources," you'll find recommendations for a handful of informative websites on a wide range of topics related to jQuery, JavaScript, and web development in general.

In Appendix B, "Development Tools," you'll discover a number of useful third-party programs and utilities for editing and debugging jQuery code within your personal development environment.

In *Appendix C*, "JavaScript Closures," you'll gain a solid understanding of closures—what they are and how you can use them to your advantage.

In *Appendix D,* "Quick Reference," you'll get a glimpse of the entire jQuery library, including every one of its methods and selector expressions. Its easy-to-scan format is perfect for those moments when you know what you want to do, but you're just unsure about the right method name or selector.

What you need for this book

In order to both write and run the code demonstrated in this book, you need the following:

- A basic text editor.
- A modern web browser such as Mozilla Firefox, Apple Safari, or Microsoft Internet Explorer.
- The jQuery source file, version 1.3.1 or later, which can be downloaded from `http://jquery.com/`.

Additionally, to run the AJAX examples in *Chapter 6*, you will need a PHP-enabled server.

Who is this book for

This book is for web designers who want to create interactive elements for their designs, and for developers who want to create the best user interface for their web applications. Basic JavaScript programming knowledge is required. You will need to know the basics of HTML and CSS, and should be comfortable with the syntax of JavaScript. No knowledge of jQuery is assumed, nor is experience with any other JavaScript libraries required.

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 will be set as follows:

```
<html>
  <head>
    <title>the title</title>
  </head>
  <body>
    <div>
      <p>This is a paragraph.</p>
      <p>This is another paragraph.</p>
      <p>This is yet another paragraph.</p>
    </div>
  </body>
</html>
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items will be made bold:

```
$(document).ready(function() {
  $('a[href^=mailto:]').addClass('mailto');
  $('a[href$=.pdf]').addClass('pdflink');
  $('a[href^=http][href*=henry]')
    .addClass('henrylink');
});
```

Any command-line input and output is written as follows:

```
outerFn():
Outer function
Inner function
```

New terms and **important words** are introduced in a bold-type font. Words that you see on the screen, in menus or dialog boxes for example, appear in our text like this: "Note the PDF icon to the right of the **Hamlet** link, the envelope icon next to the **email** link, and the white background and black border around the **Henry V** link.".

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 drop an email to `feedback@packtpub.com`, making sure to mention the book title in 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 email `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 the book

Visit `http://www.packtpub.com/files/code/6705_Code.zip` to directly download the example code.

The downloadable files contain instructions on how to use them.

Errata

Although we have taken every care to ensure the accuracy of our contents, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in text or code—we would be grateful if you would report this to us. By doing this you can save other readers from frustration, and help to improve subsequent versions of this book. If you find any errata, report them by visiting `http://www.packtpub.com/support`, selecting your book, clicking on the **let us know** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata added to the list of existing errata. The 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 the location address or website name immediately so 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 some aspect of the book, and we will do our best to address it.

1
Getting Started

Today's World Wide Web is a dynamic environment, and its users set a high bar for both style and function of sites. To build interesting, interactive sites, developers are turning to JavaScript libraries such as jQuery to automate common tasks and simplify complicated ones. One reason the jQuery library is a popular choice is its ability to assist in a wide range of tasks.

It can seem challenging to know where to begin because jQuery performs so many different functions. Yet, there is a coherence and symmetry to the design of the library; most of its concepts are borrowed from the structure of **HTML** and **Cascading Style Sheets** (**CSS**). The library's design lends itself to a quick start for designers with little programming experience since many web developers have more experience with these technologies than they do with JavaScript. In fact, in this opening chapter we'll write a functioning jQuery program in just three lines of code. On the other hand, experienced programmers will also be aided by this conceptual consistency, as we'll see in the later, more advanced chapters.

So let's look at what jQuery can do for us.

What jQuery does

The jQuery library provides a general-purpose abstraction layer for common web scripting, and is therefore useful in almost every scripting situation. Its extensible nature means that we could never cover all possible uses and functions in a single book, as plugins are constantly being developed to add new abilities. The core features, though, address the following needs:

- **Access elements in a document**. Without a JavaScript library, many lines of code must be written to traverse the **Document Object Model** (**DOM**) tree, and locate specific portions of an HTML document's structure. A robust and efficient selector mechanism is offered in jQuery for retrieving the exact piece of the document that is to be inspected or manipulated.

- **Modify the appearance of a web page**. CSS offers a powerful method of influencing the way a document is rendered, but it falls short when web browsers do not all support the same standards. With jQuery, developers can bridge this gap, relying on the same standards support across all browsers. In addition, jQuery can change the classes or individual style properties applied to a portion of the document even after the page has been rendered.
- **Alter the content of a document**. Not limited to mere cosmetic changes, jQuery can modify the content of a document itself with a few keystrokes. Text can be changed, images can be inserted or swapped, lists can be reordered, or the entire structure of the HTML can be rewritten and extended—all with a single easy-to-use **Application Programming Interface** (**API**).
- **Respond to a user's interaction**. Even the most elaborate and powerful behaviors are not useful if we can't control when they take place. The jQuery library offers an elegant way to intercept a wide variety of events, such as a user clicking on a link, without the need to clutter the HTML code itself with event handlers. At the same time, its event-handling API removes browser inconsistencies that often plague web developers.
- **Animate changes being made to a document**. To effectively implement such interactive behaviors, a designer must also provide visual feedback to the user. The jQuery library facilitates this by providing an array of effects such as fades and wipes, as well as a toolkit for crafting new ones.
- **Retrieve information from a server without refreshing a page**. This code pattern has become known as **Asynchronous JavaScript And XML** (**AJAX**), and assists web developers in crafting a responsive, feature-rich site. The jQuery library removes the browser-specific complexity from this process, allowing developers to focus on the server-end functionality.
- **Simplify common JavaScript tasks**. In addition to all of the document-specific features of jQuery, the library provides enhancements to basic JavaScript constructs such as iteration and array manipulation.

Why jQuery works well

With the recent resurgence of interest in dynamic HTML comes a proliferation of JavaScript frameworks. Some are specialized, focusing on just one or two of the above tasks. Others attempt to catalog every possible behavior and animation, and serve these all up pre-packaged. To maintain the wide range of features outlined above while remaining compact, jQuery employs several strategies:

- **Leverage knowledge of CSS**. By basing the mechanism for locating page elements on **CSS selectors**, jQuery inherits a terse yet legible way of expressing a document's structure. The jQuery library becomes an entry point for designers who want to add behaviors to their pages because a prerequisite for doing professional web development is knowledge of CSS syntax.
- **Support extensions**. In order to avoid "feature creep", jQuery relegates special-case uses to **plugins**. The method for creating new plugins is simple and well-documented, which has spurred the development of a wide variety of inventive and useful modules. Even most of the features in the basic jQuery download are internally realized through the plugin architecture, and can be removed if desired, yielding an even smaller library.
- **Abstract away browser quirks**. An unfortunate reality of web development is that each browser has its own set of deviations from published standards. A significant portion of any web application can be relegated to handling features differently on each platform. While the ever-evolving browser landscape makes a perfectly browser-neutral code base impossible for some advanced features, jQuery adds an **abstraction layer** that normalizes the common tasks, reducing the size of code, and tremendously simplifying it.
- **Always work with sets**. When we instruct jQuery, *Find all elements with the class* `collapsible` *and hide them*, there is no need to loop through each returned element. Instead, methods such as `.hide()` are designed to automatically work on sets of objects instead of individual ones. This technique, called **implicit iteration**, means that many looping constructs become unnecessary, shortening code considerably.
- **Allow multiple actions in one line**. To avoid overuse of temporary variables or wasteful repetition, jQuery employs a programming pattern called **chaining** for the majority of its methods. This means that the result of most operations on an object is the object itself, ready for the next action to be applied to it.

These strategies have kept the jQuery package slim—under 20 KB compressed—while at the same time providing techniques for keeping our custom code that uses the library compact, as well.

The elegance of the library comes about partly by design, and partly due to the evolutionary process spurred by the vibrant community that has sprung up around the project. Users of jQuery gather to discuss not only the development of plugins, but also enhancements to the core library. Appendix A details many of the community resources available to jQuery developers.

Despite all of the efforts required to engineer such a flexible and robust system, the end product is free for all to use. This open-source project is dually licensed under the **GNU Public License** (appropriate for inclusion in many other open-source projects) and the **MIT License** (to facilitate use of jQuery within proprietary software).

History of the jQuery project

This book covers the functionality and syntax of **jQuery 1.3.x**, the latest version at the time of writing. The premise behind the library—providing an easy way to find elements on a web page and manipulating them—has not changed over the course of its development, but some syntax details and features have. This brief overview of the project history describes the most significant changes from version to version.

- **Public Development Phase**: John Resig first made mention of an improvement on Prototype's "Behaviour" library in August of 2005. This new framework was formally released as **jQuery** on January 14, 2006.
- **jQuery 1.0** (August 2006): This, the first stable release of the library, already had robust support for CSS selectors, event handling, and AJAX interaction.
- **jQuery 1.1** (January 2007): This release streamlined the API considerably. Many rarely-used methods were combined, reducing the number of methods to learn and document.
- **jQuery 1.1.3** (July 2007): This minor release contained massive speed improvements for jQuery's selector engine. From this version on, jQuery's performance would compare favorably to its fellow JavaScript libraries such as Prototype, Mootools, and Dojo.
- **jQuery 1.2** (September 2007): **XPath** syntax for selecting elements was removed in this release, as it had become redundant with the CSS syntax. Effect customization became much more flexible in this release, and plugin development became easier with the addition of **namespaced events**.
- **jQuery UI** (September 2007): This new plugin suite was announced to replace the popular but aging Interface plugin. A rich collection of prefabricated widgets was included, as well as a set of tools for building sophisticated elements such as drag-and-drop interfaces.
- **jQuery 1.2.6** (May 2008): The functionality of Brandon Aaron's popular **Dimensions** plugin was brought into the main library.
- **jQuery 1.3 (January 2009)**: A major overhaul of the selector engine (**Sizzle**) provided a huge boost to the library's performance. **Event delegation** became formally supported.

Release notes for older jQuery versions can be found on the project's web site at `http://docs.jquery.com/History_of_jQuery`.

Our first jQuery-powered web page

Now that we have covered the range of features available to us with jQuery, we can examine how to put the library into action.

Downloading jQuery

The **official jQuery website** (`http://jquery.com/`) is always the most up-to-date resource for code and news related to the library. To get started, we need a copy of jQuery, which can be downloaded right from the home page of the site. Several versions of jQuery may be available at any given moment; the most appropriate for us as site developers will be the latest uncompressed version of the library. This can be replaced with a compressed version in production environments.

No installation is required. To use jQuery, we just need to place it on our site in a public location. Since JavaScript is an interpreted language, there is no compilation or build phase to worry about. Whenever we need a page to have jQuery available, we will simply refer to the file's location from the HTML document.

Setting up the HTML document

There are three pieces to most examples of jQuery usage: the HTML document itself, CSS files to style it, and JavaScript files to act on it. For our first example, we'll use a page with a book excerpt that has a number of classes applied to portions of it.

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

    <title>Through the Looking-Glass</title>

    <link rel="stylesheet" href="alice.css"
      type="text/css" media="screen" />

    <script src="jquery.js" type="text/javascript"></script>
```

```
    <script src="alice.js" type="text/javascript"></script>
  </head>

  <body>
    <h1>Through the Looking-Glass</h1>
    <div class="author">by Lewis Carroll</div>

    <div class="chapter" id="chapter-1">
      <h2 class="chapter-title">1. Looking-Glass House</h2>
      <p>There was a book lying near Alice on the table,
        and while she sat watching the White King (for she
        was still a little anxious about him, and had the
        ink all ready to throw over him, in case he fainted
        again), she turned over the leaves, to find some
        part that she could read, <span class="spoken">
        "&mdash;for it's all in some language I don't know,"
        </span> she said to herself.</p>
      <p>It was like this.</p>
      <div class="poem">
        <h3 class="poem-title">YKCOWREBBAJ</h3>
        <div class="poem-stanza">
          <div>sevot yhtils eht dna ,gillirb sawT'</div>
          <div>;ebaw eht ni elbmig dna eryg diD</div>
          <div>,sevogorob eht erew ysmim llA</div>
          <div>.ebargtuo shtar emom eht dnA</div>
        </div>
      </div>
      <p>She puzzled over this for some time, but at last
        a bright thought struck her. <span class="spoken">
        "Why, it's a Looking-glass book, of course! And if
        I hold it up to a glass, the words will all go the
        right way again."</span></p>
      <p>This was the poem that Alice read.</p>
      <div class="poem">
        <h3 class="poem-title">JABBERWOCKY</h3>
        <div class="poem-stanza">
          <div>'Twas brillig, and the slithy toves</div>
          <div>Did gyre and gimble in the wabe;</div>
          <div>All mimsy were the borogoves,</div>
          <div>And the mome raths outgrabe.</div>
        </div>
      </div>
    </div>
  </body>
</html>
```

The actual layout of files on the server does not matter. References from one file to another just need to be adjusted to match the organization we choose. In most examples in this book, we will use relative paths to reference files (`../images/foo.png`) rather than absolute paths (`/images/foo.png`). This will allow the code to run locally without the need for a web server.

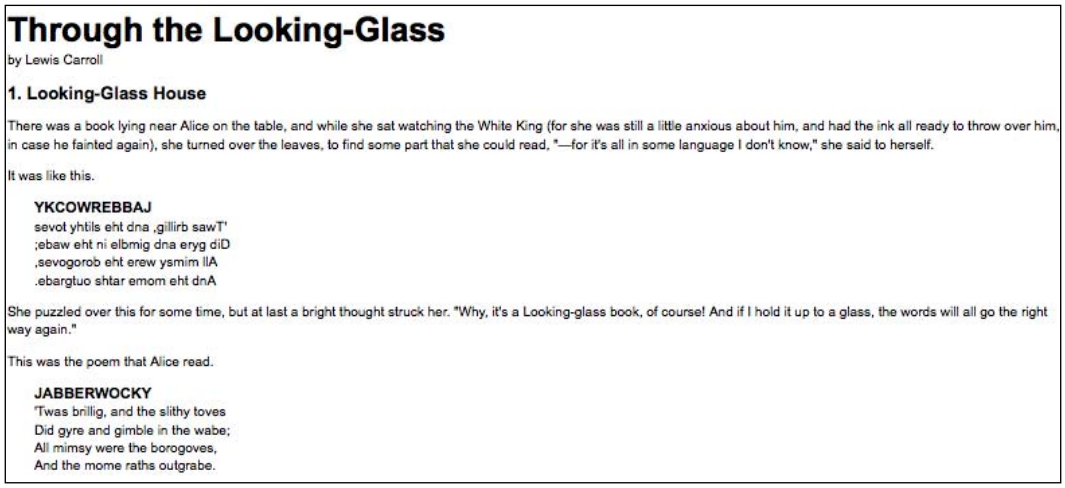
Immediately following the normal HTML preamble, the stylesheet is loaded. For this example, we'll use a spartan one.

```
body {
  font: 62.5% Arial, Verdana, sans-serif;
}
h1 {
  font-size: 2.5em;
  margin-bottom: 0;
}
h2 {
  font-size: 1.3em;
  margin-bottom: .5em;
}
h3 {
  font-size: 1.1em;
  margin-bottom: 0;
}
.poem {
  margin: 0 2em;
}
.highlight {
  font-style: italic;
  border: 1px solid #888;
  padding: 0.5em;
  margin: 0.5em 0;
  background-color: #ffc;
}
```

After the stylesheet is referenced, the JavaScript files are included. It is important that the script tag for the jQuery library be placed *before* the tag for our custom scripts; otherwise, the jQuery framework will not be available when our code attempts to reference it.

Throughout the rest of this book, only the relevant portions of HTML and CSS files will be printed. The files in their entirety are available from the book's companion website `http://book.learningjquery.com` or from the publisher's website `http://www.packtpub.com/support`.

Now we have a page that looks like this:



Through the Looking-Glass

by Lewis Carroll

1. Looking-Glass House

There was a book lying near Alice on the table, and while she sat watching the White King (for she was still a little anxious about him, and had the ink all ready to throw over him, in case he fainted again), she turned over the leaves, to find some part that she could read, "—for it's all in some language I don't know," she said to herself.

It was like this.

YKCOWREBBAJ
sevot yhtils eht dna ,gillirb sawT'
;ebaw eht ni elbmig dna eryg diD
,sevogorob eht erew ysmim llA
.ebargtuo shtar emom eht dnA

She puzzled over this for some time, but at last a bright thought struck her. "Why, it's a Looking-glass book, of course! And if I hold it up to a glass, the words will all go the right way again."

This was the poem that Alice read.

JABBERWOCKY
'Twas brillig, and the slithy toves
Did gyre and gimble in the wabe;
All mimsy were the borogoves,
And the mome raths outgrabe.

We will use jQuery to apply a new style to the poem text.

This example is to demonstrate a simple use of jQuery. In real-world situations, this type of styling could be performed purely with CSS.

Adding jQuery

Our custom code will go in the second, currently empty, JavaScript file, which we included from the HTML using `<script src="alice.js" type="text/javascript"></script>`. For this example, we only need three lines of code:

```
$(document).ready(function() {
  $('.poem-stanza').addClass('highlight');
});
```

Finding the poem text

The fundamental operation in jQuery is selecting a part of the document. This is done with the `$()` construct. Typically, it takes a string as a parameter, which can contain any CSS selector expression. In this case, we wish to find all parts of the document that have the `poem-stanza` class applied to them, so the selector is very simple. However, we will cover much more sophisticated options through the course of the book. We will step through the different ways of locating parts of a document in Chapter 2.

The `$()` function is actually a factory for the **jQuery object**, which is the basic building block we will be working with from now on. The jQuery object encapsulates zero or more DOM elements, and allows us to interact with them in many different ways. In this case, we wish to modify the appearance of these parts of the page, and we will accomplish this by changing the classes applied to the poem text.

Injecting the new class

The `.addClass()` method, like most jQuery methods, is named self-descriptively; it applies a CSS class to the part of the page that we have selected. Its only parameter is the name of the class to add. This method, and its counterpart, `.removeClass()`, will allow us to easily observe jQuery in action as we explore the different selector expressions available to us. For now, our example simply adds the `highlight` class, which our stylesheet has defined as italicized text with a border.

Note that no iteration is necessary to add the class to all the poem stanzas. As we discussed, jQuery uses **implicit iteration** within methods such as `.addClass()`, so a single function call is all it takes to alter all of the selected parts of the document.

Executing the code

Taken together, `$()` and `.addClass()` are enough for us to accomplish our goal of changing the appearance of the poem text. However, if this line of code is inserted alone in the document header, it will have no effect. JavaScript code is generally run as soon as it is encountered in the browser, and at the time the header is being processed, no HTML is yet present to style. We need to delay the execution of the code until after the DOM is available for our use.

The traditional mechanism for controlling when JavaScript code is run is to call the code from within **event handlers**. Many handlers are available for user-initiated events, such as mouse clicks and key presses. If we did not have jQuery available for our use, we would need to rely on the `onload` handler, which fires after the page (along with all of its images) has been rendered. To trigger our code from the `onload` event, we would place the code inside a function:

```
function highlightPoemStanzas() {
  $('.poem-stanza').addClass('highlight');
}
```

Then we would attach the function to the event by modifying the HTML `<body>` tag to reference it:

```
<body onload="highlightPoemStanzas();">
```

This causes our code to run after the page is completely loaded.

There are drawbacks to this approach. We altered the HTML itself to effect this behavior change. This tight coupling of structure and function clutters the code, possibly requiring the same function calls to be repeated over many different pages, or in the case of other events such as mouse clicks, over every instance of an element on a page. Adding new behaviors would then require alterations in multiple places, increasing the opportunity for error and complicating parallel workflows for designers and programmers.

To avoid this pitfall, jQuery allows us to schedule function calls for firing once the DOM is loaded—without waiting for images—with the `$(document).ready()` construct. With our function defined as above, we can write:

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

This technique does not require any HTML modifications. Instead, the behavior is attached entirely from within the JavaScript file. We will learn how to respond to other types of **events**, divorcing their effects from the HTML structure as well, in Chapter 3.

This incarnation is still slightly wasteful, though, because the function `highlightPoemStanzas()` is defined only to be used immediately, and exactly once. This means that we have used an identifier in the global namespace of functions that we have to remember not to use again, and for little gain. JavaScript, like some other programming languages, has a way around this inefficiency called **anonymous functions** (sometimes also called **lambda functions**). Using anonymous functions, we can write the code as it was originally presented:

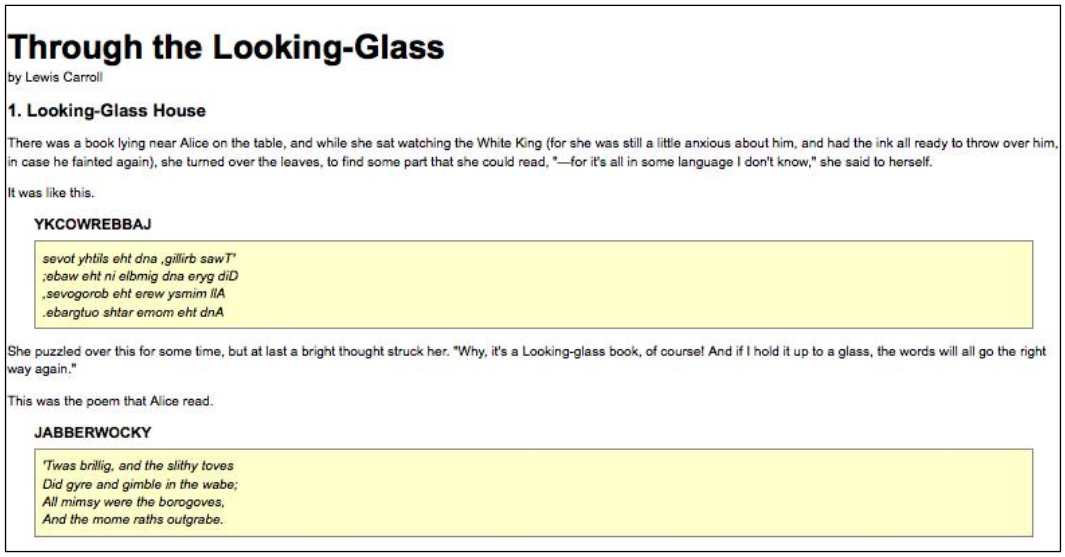
```
$(document).ready(function() {
  $('.poem-stanza').addClass('highlight');
});
```

By using the `function` keyword without a function name, we define a function exactly where it is needed, and not before. This removes clutter and brings us down to three lines of JavaScript. This idiom is extremely convenient in jQuery code, as many methods take a function as an argument and such functions are rarely reusable.

When this syntax is used to define an anonymous function within the body of another function, a **closure** can be created. This is an advanced and powerful concept, but should be understood when making extensive use of nested function definitions as it can have unintended consequences and ramifications on memory use. This topic is discussed fully in Appendix C.

The finished product

Now that our JavaScript is in place, the page looks like this:



The poem stanzas are now italicized and enclosed in boxes, as specified by the `alice.css` stylesheet, due to the insertion of the `highlight` class by the JavaScript code.

Summary

We now have an idea of why a developer would choose to use a JavaScript framework rather than writing all code from scratch, even for the most basic tasks. We also have seen some of the ways in which jQuery excels as a framework, and why we might choose it over other options. We also know in general which tasks jQuery makes easier.

In this chapter, we have learned how to make jQuery available to JavaScript code on our web page, use the `$()` factory function to locate a part of the page that has a given class, call `.addClass()` to apply additional styling to this part of the page, and invoke `$(document).ready()` to cause this code to execute upon the loading of the page.

The simple example we have been using demonstrates how jQuery works, but is not very useful in real-world situations. In the next chapter, we will expand on the code here by exploring jQuery's sophisticated selector language, finding practical uses for this technique.

2
Selectors

The jQuery library harnesses the power of Cascading Style Sheets (CSS) selectors to let us quickly and easily access elements or groups of elements in the Document Object Model (DOM). In this chapter, we will explore a few of these selectors, as well as jQuery's own **custom selectors**. We'll also look at jQuery's **DOM traversal methods** that provide even greater flexibility for getting what we want.

The Document Object Model

One of the most powerful aspects of jQuery is its ability to make selecting elements in the DOM easy. The Document Object Model is a family-tree structure of sorts. HTML, like other markup languages, uses this model to describe the relationships of things on a page. When we refer to these relationships, we use the same terminology that we use when referring to family relationships—parents, children, and so on. A simple example can help us understand how the family tree metaphor applies to a document:

```
<html>
  <head>
    <title>the title</title>
  </head>
  <body>
    <div>
      <p>This is a paragraph.</p>
      <p>This is another paragraph.</p>
      <p>This is yet another paragraph.</p>
    </div>
  </body>
</html>
```

Here, `<html>` is the **ancestor** of all the other elements; in other words, all the other elements are **descendants** of `<html>`. The `<head>` and `<body>` elements are not only descendants, but **children** of `<html>`, as well. Likewise, in addition to being the ancestor of `<head>` and `<body>`, `<html>` is also their **parent**. The `<p>` elements are children (and descendants) of `<div>`, descendants of `<body>` and `<html>`, and **siblings** of each other. For information on how to visualize the family-tree structure of the DOM using third-party software, see Appendix B.

An important point to note before we begin is that the resulting set of elements from selectors and methods is always wrapped in a jQuery object. These jQuery objects are very easy to work with when we want to actually do something with the things that we find on a page. We can easily bind **events** to these objects and add slick **effects** to them, as well as **chain** multiple modifications or effects together. Nevertheless, jQuery objects are different from regular DOM elements or node lists, and as such do not necessarily provide the same methods and properties for some tasks. In the final part of this chapter, therefore, we will look at ways to access the DOM elements that are wrapped in a jQuery object.

The $() factory function

No matter which type of selector we want to use in jQuery, we always start with the dollar sign and parentheses: `$()`. Just about anything that can be used in a stylesheet can also be wrapped in quotation marks and placed inside the parentheses, allowing us to apply jQuery methods to the matched set of elements.

Making jQuery Play Well with Other JavaScript Libraries

In jQuery, the dollar sign `$` is simply an "alias" for `jQuery`. Conflicts could arise if more than one of these libraries were being used in a given page because a `$()` function is very common in JavaScript libraries. We can avoid such conflicts by replacing every instance of `$` with `jQuery` in our custom jQuery code. Additional solutions to this problem are addressed in Chapter 10.

Three building blocks of these selectors are **tag name**, **ID**, and **class**. They can be used either on their own or in combination with other selectors. Here is an example of what each of these three selectors looks like on its own:

Selector	CSS	jQuery	Description
Tag name	**p**	**$('p')**	**Selects all paragraphs in the document**
ID	**#some-id**	**$('#some-id')**	**Selects the single element in the document that has an ID of some-id**
Class	**.some-class**	**$('.some-class')**	**Selects all elements in the document that have a class of some-class**

As mentioned in Chapter 1, when we attach methods to the `$()` factory function, the elements wrapped in the jQuery object are looped through automatically and implicitly. Therefore, we can usually avoid **explicit iteration**, such as a `for` loop, that is so often required in DOM scripting.

Now that we have covered the basics, we're ready to start exploring some more powerful uses of selectors.

CSS selectors

The jQuery library supports nearly all of the selectors included in CSS specifications 1 through 3, as outlined on the *World Wide Web Consortium*'s site: `http://www.w3.org/Style/CSS/#specs`. This support allows developers to enhance their websites without worrying about which browsers (particularly Internet Explorer 6) might not understand advanced selectors, as long as the browsers have JavaScript enabled.

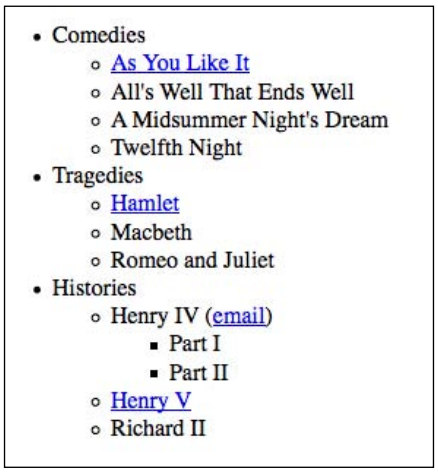
Responsible jQuery developers should always apply the concepts of **progressive enhancement** and **graceful degradation** to their code, ensuring that a page will render as accurately, even if not as beautifully, with JavaScript disabled as it does with JavaScript turned on. We will continue to explore these concepts throughout the book.

To begin learning how jQuery works with CSS selectors, we'll use a structure that appears on many websites, often for navigation—the nested, unordered list.

```
<ul id="selected-plays">
  <li>Comedies
    <ul>
      <li><a href="/asyoulikeit/">As You Like It</a></li>
      <li>All's Well That Ends Well</li>
      <li>A Midsummer Night's Dream</li>
      <li>Twelfth Night</li>
    </ul>
  </li>
  <li>Tragedies
```

```
      <ul>
        <li><a href="hamlet.pdf">Hamlet</a></li>
        <li>Macbeth</li>
        <li>Romeo and Juliet</li>
      </ul>
    </li>
    <li>Histories
      <ul>
        <li>Henry IV (<a href="mailto:henryiv@king.co.uk">email</a>)
           <ul>
             <li>Part I</li>
             <li>Part II</li>
           </ul>
        <li><a href="http://www.shakespeare.co.uk/henryv.htm">
                                                    Henry V</a></li>
        <li>Richard II</li>
      </ul>
    </li>
  </ul>
```

Notice that the first `<ul>` has an ID of `selected-plays`, but none of the `<li>` tags have a class associated with them. Without any styles applied, the list looks like this:



The nested list appears as we would expect it to—a set of bulleted items arranged vertically and indented according to their level.

Styling list-item levels

Lets suppose that we want the top-level items, and *only* the top-level items, to be arranged horizontally. We can start by defining a `horizontal` class in the stylesheet:

```
.horizontal {
  float: left;
  list-style: none;
  margin: 10px;
}
```

The `horizontal` class floats the element to the left of the one following it, removes the bullet from it if it's a list item, and adds a 10 pixel margin on all sides of it.

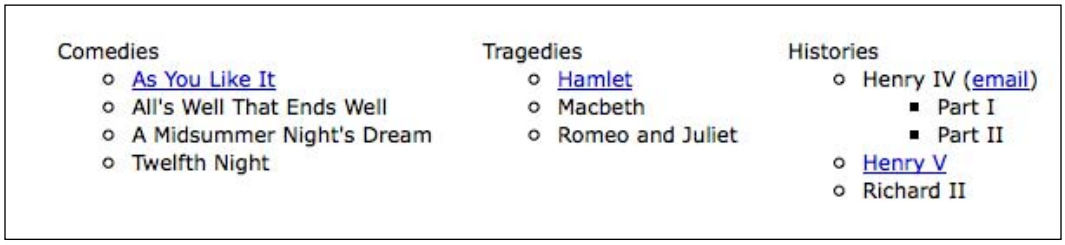
Rather than attaching the `horizontal` class directly in our HTML, we'll add it dynamically to the top-level list items only—**Comedies**, **Tragedies**, and **Histories**—to demonstrate jQuery's use of selectors:

```
$(document).ready(function() {
  $('#selected-plays > li').addClass('horizontal');
});
```

As discussed in Chapter 1, we begin the jQuery code with the `$(document).ready()` wrapper, that runs as soon as the DOM has loaded.

The second line uses the **child combinator** (`>`) to add the `horizontal` class to all top-level items only. In effect, the selector inside the `$()` function is saying, *find each list item* (`li`) *that is a child* (`>`) *of the element with an ID of* `selected-plays` (`#selected-plays`).

With the class now applied, our nested list looks like this:



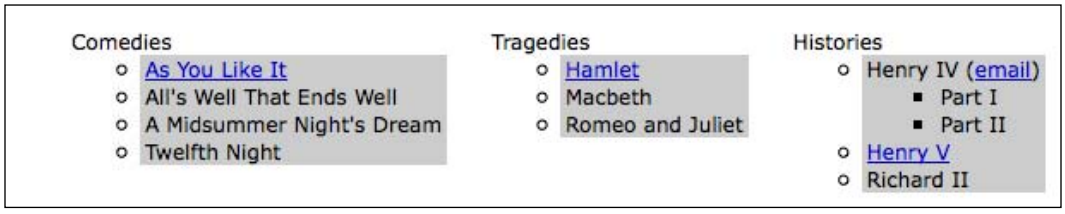
Styling all of the other items—those that are *not* in the top level—can be done in a number of ways. Since we have already applied the `horizontal` class to the top-level items, one way to select all sub-level items is to use a **negation pseudo-class** to identify all list items that do *not* have a class of `horizontal`. Note the addition of the third line of code:

```
$(document).ready(function() {
  $('#selected-plays > li').addClass('horizontal');
  $('#selected-plays li:not(.horizontal)').addClass('sub-level');
});
```

This time we are selecting every list item (`li`) that:

1. Is a descendant of the element with an ID of `selected-plays` (`#selected-plays`)
2. Does not have a class of `horizontal` (`:not(.horizontal)`)

When we add the `sub-level` class to these items, they receive the shaded background defined in the stylesheet. Now the nested list looks like this:



Attribute selectors

Attribute selectors are a particularly helpful subset of CSS selectors. They allow us to specify an element by one of its HTML properties, such as a link's `title` attribute or an image's `alt` attribute. For example, to select all images that have an `alt` attribute, we write the following:

```
$('img[alt]')
```

In versions prior to 1.2, jQuery used **XML Path Language** (**XPath**) syntax for its attribute selectors and included a handful of other XPath selectors. While these basic XPath selectors have since been removed from the core jQuery library, they are still available as a plugin: `http://plugins.jquery.com/project/xpath/`