VISUAL QUICKSTART GUIDE

jQuery and jQuery UI

JAY BLANCHARD

LEARN THE QUICK AND EASY WAY!

VISUAL QUICKSTART GUIDE

jQuery and jQuery UI

JAY BLANCHARD

Peachpit Press

Visual QuickStart Guide
jQuery and jQuery UI
Jay Blanchard

Peachpit Press
www.peachpit.com.

To report errors, please send a note to errata@peachpit.com.
Peachpit Press is a division of Pearson Education.

Acquisitions and Project Editor: Rebecca Gulick
Developmental Editor: Dave Awl
Copy Editor: Liz Welch
Technical Reviewer: Jesse Castro
Production Coordinator: Myrna Vladic
Compositor: David Van Ness
Proofreader: Patricia Pane
Indexer: Valerie Haynes Perry
Cover Design: RHDG / Riezebos Holzbaur Design Group, Peachpit Press
Interior Design: Peachpit Press
Logo Design: MINE™ www.minesf.com

ISBN-13: 978-0-321-88514-2
ISBN-10: 0-321-88514-7

9 8 7 6 5 4 3 2 1

Printed and bound in the United States of America

Dedication

Dedicated to the memory of Mr. Coy Watkins, who instilled a love of science and exploration in hundreds of students...especially me.

Special Thanks to:

It is not possible to embark upon the journey of creating a book without a lot of talented and supportive people by your side. I love that they get me, warts and all, even if it means that I say, “I can’t…I have to write.”

To my daughters Brittany and Kaitlyn, thank you for inspiring me and cheering me on. You make life worth every single moment. I love you to the moon and back…infinity!

To my Dad, I could not be here without the life lessons, support, and love that you have so graciously shared with a hardheaded kid. Thank you for being my Dad. I love you.

To Rebecca Gulick, the journey gets more interesting as we go. Thank you for your patient and firm guidance while maintaining a sense of humor. You’re truly a treasure.

To Dave Awl, thank you for steering the boat through some choppy seas. You’ve taught me a lot about that.

To Jesse Castro, what can I say? Your depth and breadth of knowledge always astounds me. Your ability to keep me on point is amazing. Your friendship and quirky sense of humor make me smile. Thank you!

To the Peachpit/Pearson team, thank you for your patience and your ability to turn sows’ ears into silk purses. Your wow factor is way off the top of the scale.

To my valued mentor Larry Ullman, you know how to cut to the heart of the matter and remind me what the value of humbleness is.

To my friends who have supported me and carried me through when the pressure was on and time was short, I appreciate you beyond the mere measure of words on a page.

To the jQuery team and community, thank you for your dedication to bringing a great product to the masses, your encouragement, and your steadfast vision of what a JavaScript library should look like. Your willingness to share your knowledge and to bring people into the fold is tremendously valuable and welcomed.

Contents at a Glance

	Introduction	xi
Chapter 1	Using Selectors	1
Chapter 2	Handling Events	15
Chapter 3	Getting and Setting DOM Attributes	35
Chapter 4	Manipulating DOM Elements	55
Chapter 5	Harnessing Advanced Selectors	81
Chapter 6	Traversing the DOM Tree	95
Chapter 7	Using Ajax	109
Chapter 8	Creating Captivating Effects	127
Chapter 9	Turning on jQuery's Utilities	141
Chapter 10	Using Plugins	151
Chapter 11	Introducing jQuery UI	161
Chapter 12	Fixing Common Problems	179
Appendix A	jQuery vs. Other Technology	187
Appendix B	An Active jQuery Website	189
	Index	193

This page intentionally left blank

Table of Contents

Introduction xi

Chapter 1 Using Selectors 1

Using Basic Selectors 2
Applying Filters to Selectors 6
Rewind and Review 14

Chapter 2 Handling Events 15

Attaching Event Handlers 16
Calling Document Events 17
Trapping Mouse Events 19
Capturing Keyboard Events 25
Forging Form Events 30
Rewind and Review 34

Chapter 3 Getting and Setting DOM Attributes 35

Changing Properties 37
Managing Values 43
Using and Updating Attributes 45
Handling CSS Classes 50
Rewind and Review 54

Chapter 4 Manipulating DOM Elements 55

Inserting Elements 56
Creating a Lightbox Effect 57
More Fun with DOM Manipulators 67
Getting and Setting Measurements 69
Cloning 72
Changing an Input Element 76
Rewind and Review 80

Chapter 5 Harnessing Advanced Selectors 81
Forming Attribute Selectors 82
Combining Selectors 91
Rewind and Review 94

Chapter 6 Traversing the DOM Tree 95
Traversing the Tree 96
Applying Traversal Filters 102
Rewind and Review 108

Chapter 7 Using Ajax 109
Applying Shorthand Methods 110
Working with JSON 122
Rewind and Review 126

Chapter 8 Creating Captivating Effects 127
Controlling Visibility 128
Managing Motion 131
Composing Custom Animations 133
Rewind and Review 140

Chapter 9 Turning on jQuery's Utilities 141
Looping Through Elements 143
Setting and Getting Data 146
Rewind and Review 150

Chapter 10 Using Plugins 151
Working with Plugins 152
Rewind and Review 160

Chapter 11 Introducing jQuery UI 161
Riding the ThemeRoller Coaster 162
Exploring Popular UI Widgets 166
Adding a Dialog Widget 174
Rewind and Review 178

Chapter 12 Fixing Common Problems 179
Nothing Works! 180
Nothing Comes Back from Ajax! 181
jQuery Doesn't Work in Ajax-Loaded Content 183
Loading Two Libraries 185
Rewind and Review 186

Appendix A jQuery vs. Other Technology 187

Appendix B An Active jQuery Website 189

Index 193

This page intentionally left blank

Introduction

Welcome to jQuery!

You are on the verge of learning how to add the world's most popular JavaScript library to your websites and web applications. You'll be joining the ranks of developers for companies like Google, ESPN, and Best Buy in applying jQuery to your web development arsenal.

Originally developed in 2005 by John Resig, the jQuery library has grown into a mature and powerful tool for enhancing web pages. Developers worldwide took up the banner and began developing widgets (called plugins) using the jQuery library for the foundation. The jQuery group joined in the plugin craze, absorbing some very popular plugins into its framework and adding some widgets to an additional library, jQuery UI, in 2007.

Since that time, the jQuery Foundation has continued to enhance and rework the library to make it more robust and efficient. New versions of the library with new features and enhancements are rolled out at an incredible pace to keep up with the changing landscape of web development.

Why Use This Book?

The goal of this book is to introduce you to the concepts of the jQuery and jQuery UI libraries, as well as how to use those concepts in practical examples. You'll be able to use the concepts right away in your web projects.

What Is jQuery?

There is a very simple answer to that question: jQuery is JavaScript. To be more specific, jQuery is a free, open source library of code written in the JavaScript programming language.

Because jQuery is JavaScript, it can take advantage of many of JavaScript's functions and concepts, like object notation, and in many cases make them easier to work with.

At the heart of jQuery is its selector engine, Sizzle, which is also written in JavaScript. The Sizzle selector engine makes it easy to use CSS (Cascading Style Sheet) selectors as jQuery objects to work with specific web page elements or groups of elements.

Variables and functions and arguments, oh my!

If this is your first foray into programming languages, let's cover some of the basic concepts of putting together a working piece of software. You'll see examples of these concepts in nearly every exercise.

The first concept is that of variables. Variables are essentially containers for pieces of information that can come in the form of values, arrays (indexed collections of items), or objects (items that have a collection of properties). The following will declare a variable and set its value to be blank:

```
var myVariable = '';
var myNewVariable = '';
```

Learning jQuery

Learning how to use jQuery is an organic process. For example, knowing how to create selectors is no fun if you don't do something with the items you select.

You'll be introduced early on to many of jQuery's methods and functions even if the concept has not yet been covered in the book. You'll be guided to chapters where more information concerning those methods and functions can be found at the end of each exercise.

Starting in Chapter 2, "Handling Events," you'll begin building jQuery functions within the framework of a website's templates. Once complete, the web pages will have used very many of the jQuery methods available, including some of the jQuery UI plugin widgets (widgets and plugins are small, stand-alone tools that are easily added to websites). The markup and code for the completed web pages (including the PHP and MySQL queries) are provided for you in Appendix B, "An Active jQuery Website."

All of the code used in the book is available for download from www.jayblanchard.net. Just look for the jQuery and jQuery UI Visual QuickStart Guide download link.

The variable is now ready to be used, and you can populate it with any value that you need just by referring to it:

```
myVariable = 1234;
myNewVariable = $('#id');
```

One thing that you'll need to be aware of with variables: They have a scope. In other words, a variable may only be available within the confines of a function and not available anywhere else in your code. JavaScript variable scope has been a popular topic of discussion, and you'll find many good references online.

TIP Give your variables and functions meaningful names, because doing so will help you maintain your code later as well as provide clues for how the variable or function is used.

The next concept is designed for testing a condition and then reacting to that condition. Conditional statements come in the form of **if** (if this, then that) or **while** (while condition exists, do this):

```
if(myNewVariable == 1234) {
    // then do this
} else {
    // then do that
}
```

The **while** condition typically starts a loop (there is also a **for** condition that sets up a loop). Here you'll test the value of **myVariable**, print out the value of the variable, and then reduce the variable's value by 1. The loop will run as long as **myVariable**'s value is greater than 0.

```
var myVariable = 10;
while(myVariable > 0) {
    // do something
    document.write(myVariable +
"<br>");
    myVariable--;
}
```

A little more difficult concept is the function. A *function* is typically a group of instructions that allow you to perform a set of actions just by calling the function name. Then you can return the result of the function's actions.

To declare a function, you do this:

```
function myFunction() {
    // place instructions here
    var result = 1 + 1; //
    → instruction
    return result;
}
```

continues on next page

To use the function, you might do something like this:

```
var myVariable = myFunction();
    //myVariable now equals 2
```

Sometimes you'll want to send information to a function. This is known as passing arguments. Revamp your function to look like this:

```
function myFunction(argument) {
    var result = argument + 1;
    return result;
}
```

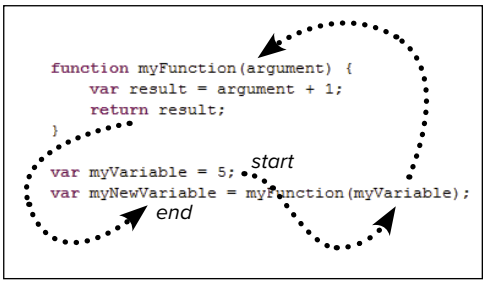
Now you can pass an argument to the function:

```
var myVariable = 5;
var myNewVariable =
→ myFunction(myVariable);
```

You pass **myVariable** to **myFunction**. Then **myVariable** becomes **argument**. You add 1 to **argument** and return the result. **myNewVariable** is now equal to 6 A.

Functions are great when you need to perform the same action over and over again while maybe passing different arguments. You can also set up functions to receive and process multiple arguments. Functions will become a key tool in your development toolbox.

This is nowhere near an exhaustive introduction to programming, but it should be enough to get your feet wet and give you the understanding that you need to work through the jQuery exercises in the book. There are many good resources for learning how to program with JavaScript and many other languages where the concepts are the same, only the syntax is changed.



A The transformation of an argument.

Functions or Methods?

One concept that may be a little vague is the difference between functions and methods. It seems, on the surface, that they're the same thing. In reality they are not.

A *function* is a piece of code that's called by name. You can pass data to a function and the function will operate on that data. You can also return data from a function. Any data passed to the function is passed explicitly—you choose to pass data to a function.

A *method* is a piece of code that is normally associated with an object, like when a selector is bound to a jQuery method. You still call a method by name, and in most respects, it's identical to a function except for two key differences: A method is implicitly passed the object for which it was called and the method is able to operate on data that's contained within the object.

The water becomes muddy when you begin to understand that functions may contain methods and in a similar fashion you may create methods, like jQuery plugins and extensions, that contain functions. If you remember how information is passed, either explicitly or implicitly, you'll be able to keep the differences straight.

Learning the Basics

In many cases, your first decision as a web developer using jQuery is deciding whether to download the jQuery core to make it available from your web server or use a hosted service known as a CDN (Content Delivery Network). Both have advantages and disadvantages.

The single largest advantage of using a CDN, like Google's, is that its distributed network almost always uses servers closest to the website visitor to deliver the jQuery library. Once the library from a CDN is cached by the browser, it doesn't have to be downloaded again (as long as the browser cache isn't cleared), which makes site loading faster. The largest disadvantage is that you'll have to rely on a third party to be available when your site is requested by a first-time visitor.

If you decide to host the jQuery library yourself, your biggest advantage is that you'll be in control. If someone can reach your site, they can get all of the files needed to use your site. Once it's cached from your site, returning visitors gain the same advantage they'd get if you were using a CDN. You can also create jQuery applications that require no connection to the Internet if the application has no requirement for a remote data source. The biggest disadvantage is that some browsers limit the number of connections they can make to a server simultaneously, so getting everything downloaded quickly may be difficult.

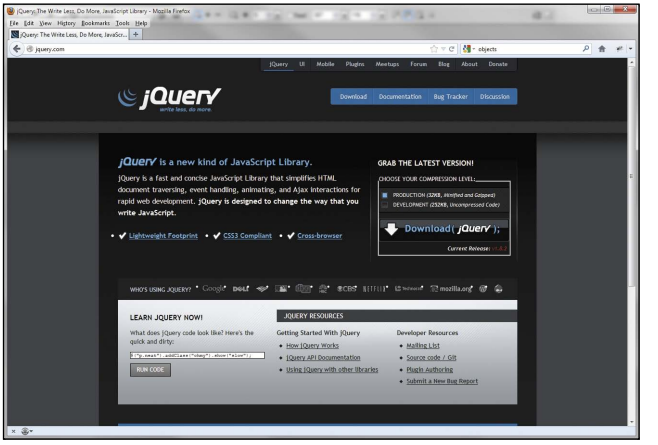
For most people in this day and age bandwidth is not a concern, but you may want to consider those do who have bandwidth limitations or who are using browsers that place low limits on connections to servers.

Your best bet may be using a CDN with a fallback to a local (on your web server) copy of the jQuery library. Let's prepare for creating a fallback by downloading a copy of jQuery first.

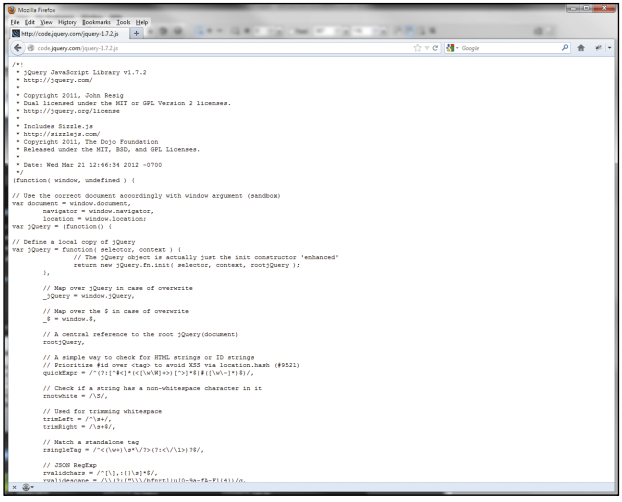
To download jQuery from jquery.com:

1. Open a browser and visit jquery.com A.
2. Choose the version of jQuery you'd like to download, either production or development. The production version is minified (white spaces and comments stripped) to provide the smallest possible footprint and overhead.
3. The jQuery file will appear in your browser B.
4. Save the file to your computer and then move it to the proper place on your web server.

TIP In order to facilitate offline development, you'll want to download a copy of jQuery to host on your local machine.



A The jQuery website. The links to download the code are in the upper-right-hand corner of the site.



B The raw JavaScript code for the jQuery library.

Once you've downloaded the file, you can include it in your web projects. Let's set up a fallback method to use with Google's CDN.

To use jQuery in your projects:

- Add the following code within the **<head></head>** tags in your web pages:

```
<script type="text/javascript"
→src="https://ajax.googleapis.com/
→ajax/libs/jquery/1.7.2/jquery.
→min.js">

</script>

<script type="text/javascript">

if (typeof jQuery ==
→'undefined') {

document.write(unescape("%3Cscript
→src='path/to/jquery-1.7.2.min.js'
→type='text/javascript'%3E%3C/
→script%3E"));

}

</script>
```

The first script is an attempt to include the jQuery core in your web project from the CDN. The second script then tests to see if the jQuery object is available. If the jQuery object isn't available, a line is written to the HTML file to load the jQuery file from the local source.

TIP If you're using HTML5, it isn't necessary to include the type attribute in your script tags.

TIP You have choices when it comes to CDNs. Microsoft, Google, and jQuery all offer CDNs.

Minifying your code

As you're developing your markup, style sheets, and jQuery code, you'll leave a lot of whitespace, tabs, and comments in your files to make the files easy to maintain and read. This is great during development, but for production you should consider minifying your code.

The process of minifying code removes all of the unnecessary whitespace, comments, and formatting characters, which makes the download of the code much faster. Some minifying applications will even check your code for errors and perform other actions to reduce your code to the smallest possible size without losing functionality.

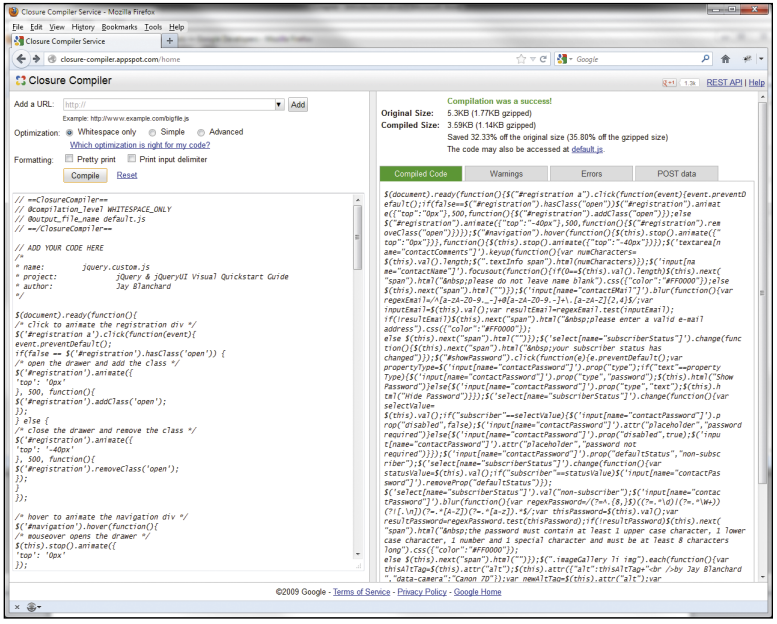
My favorite application for minifying code is the Google Closure Compiler C.

To minify your code with the Google Closure Compiler:

1. Go to http://closure-compiler.appspot.com/home to access the application.
2. Modify the line of code in the left-hand pane of the compiler containing the directive **`@output_file_name`**. Use the name of the file you wish to save your minified code to:

```
// ==ClosureCompiler==
// @compilation_level
    → WHITESPACE_ONLY
// @output_file_name
    → jquery.custom.min.js
// ==/ClosureCompiler==
```

3. Copy your jQuery code into the space below the directives.
4. Click the Compile button in the upper-left pane of the compiler.



C The Google Closure Compiler interface.

Once you've completed those steps, you'll see the compiled code in the right pane of the application. In the upper-right section of the interface, you can get statistics about the original size of the code, followed by the compiled size of the code to give you an idea of how much compression occurred.

Also in the upper-right section of the compiler is a link to your minified code, using the name you specified in the directives. Clicking on the link will open your raw, minified JavaScript code in your browser window. Save the code to your computer and move it to the proper location on your web server.

TIP **Always keep a development version of your code containing all of the comments and whitespace for readability and minify only for your production environment.**

Performing Progressive Enhancement

One of the major benefits of using the jQuery library is that you can use it on any website without having to modify any of your HTML or CSS.

Most websites are developed using a similar workflow. The HTML markup and CSS are developed first along with any artwork to give the website its look and feel. Once complete, the website may be fully functioning. Most developers keep their style sheets separate from their markup, choosing to include the CSS in their projects via link tags in the head of HTML documents. It's just good organization. Maintenance is easier and much more efficient.

Because jQuery has the ability to interact with the full range of CSS selectors and HTML elements, it can be kept apart from the markup, whereas many JavaScript calls have to be written inline with the markup. For instance, to capture a click event from a link tag you'd have to include a call to JavaScript's **`onClick()`** method within the anchor tag:

```
<a href="some.html"
→ onClick="jsFunction">
```

It can be a lot of work to go back to a website you've developed to add JavaScript interaction.

TIP **When planning new websites and applications that you'll be using jQuery on, be sure to add classes and IDs that will assist you in the development process.**

With jQuery you'd include your jQuery file in a script tag, just as you did with the jQuery core earlier in "Learning the Basics." Then you could write the click event handler into your jQuery code file:

```
$('a[href="some.html"]').click(...
```

This means you never have to touch your existing markup. The concept of keeping everything separate is known as *progressive enhancement*.

The basic rules for progressive enhancement are simple. Develop your markup, add your styles, and then enhance with jQuery—with each of those being in standalone files. In Chapter 1, "Using Selectors," progressive enhancement isn't used—all of the HTML, CSS, and jQuery (except the jQuery core) you'll write are in one file. This was done for simplicity's sake. Once you start developing jQuery in earnest (beginning with Chapter 2, "Handling Events"), the markup, styles, and jQuery code are kept in separate files and included in the HTML as needed.

Tracking Down Tools

All you need other than the jQuery core file is a good text editor. Every developer has his or her preferences, so I'm going to give you a couple of recommendations. I encourage you to experiment with a few different text editors until you find one you're comfortable with. Additionally, **Table I.1** contains some recommendations for other tools that will help you to become a better jQuery developer and a better web developer.

As your skills and abilities grow, you may find that other tools will enhance your workflow and make your development process more efficient.

Rewind and Review

Take a few moments to reflect on what you've learned in this introduction:

- Where do you get the jQuery library?
- Is it better to host your own jQuery code or rely on a Content Delivery Network (CDN)?
- What is jQuery?
- What is the advantage of progressive enhancement?

TABLE I.1 Suggested Tools

Name	Information
Eclipse IDE	General-purpose text editor for Windows or Mac. Available from www.eclipse.org.
Sublime Text 2	General-purpose text editor for Windows, Mac, and Linux. Available from www.sublimetext.com.
Firebug	DOM inspector and troubleshooting tools for Firefox on Windows and Mac. Available from http://getfirebug.com.
Developer Tools	DOM inspector and troubleshooting tools available with Internet Explorer 9 on Windows. Just press F12.
Developer Tools	DOM inspector and troubleshooting tools for Google Chrome. It comes bundled with Chrome: Select Tools > Developer Tools.
XAMPP	A personal web server you can install on your computer for local development. Includes PHP and MySQL, and runs on Windows. Available from www.apachefriends.org.
MAMP	A personal web server you can install on your Mac. Includes PHP and MySQL. Available from www.mamp.info.

Manipulating DOM Elements

Adding, changing, and removing elements from your web pages based on user interactions is one of the coolest things you can do with jQuery. The library is deep, with lots of functions you can apply to your web pages to achieve dramatic effects.

In many of the exercises prior to this chapter, you've used the **`css()`** method to create styles on the fly. In this chapter, you'll take it a couple of steps further using CSS height, width, and position properties. You've also used the **`html()`** method to add error messages when form elements were not filled out correctly. You'll explore using **`html()`** further while employing some custom HTML attributes.

The DOM manipulators don't stop there. You'll learn how to copy, add, change, and remove DOM elements to enhance your website visitors' experience. Let's get started!

In This Chapter

Inserting Elements	56
Creating a Lightbox Effect	57
More Fun with DOM Manipulators	67
Getting and Setting Measurements	69
Cloning	72
Changing an Input Element	76
Rewind and Review	80

Inserting Elements

You'll find there are times when you need to add and remove elements from your web pages. The jQuery library provides a number of methods for performing these manipulations, allowing you a great deal of flexibility when creating specific interactions (see **Table 4.1**).

Of special note are the methods that perform the same function but use a different syntax. Take, for example, **`before()`** and **`insertBefore()`**. Both allow you to place content into your page in the same way but with the syntax flipped:

```
$('element')
    .before('<p>before element</p>');
$('<p>before element</p>')
    .insertBefore('element');
```

The paragraph containing "before element" will be inserted before "element" in either case—the choice for you is a stylistic one. Many say using **`insertBefore()`** is more easily read because it reads left to right and is easier to understand. On the other hand, many like the syntax of the first example even if it's another case where jQuery sounds a little like Yoda. Methods that perform the same task are pointed out in their descriptions.

TABLE 4.1 DOM Insertion Manipulators

Method	Use It To...
`after()`	Insert content after each of the selected elements.
`insertAfter()`	Perform the same action as **`after()`**; requires a different syntax.
`append()`	Insert content at the end of each selected element.
`appendTo()`	Perform the same task as **`append()`**, but the syntax is flipped.
`prepend()`	Insert content at the beginning of each selected element.
`prependTo()`	Perform the same task as **`prepend()`**, but the syntax is reversed.
`before()`	Insert content before each of the selected elements.
`insertBefore()`	Perform the same action as **`before()`**, but the syntax is flipped.
`clone()`	Create a deep copy (copies all of the descendants) of the set of selected elements.
`detach()`	Remove the set of matched elements from the DOM and keep the data for later reinsertion.
`empty()`	Remove all child nodes of the set of selected elements from the DOM.
`remove()`	Remove the set of selected elements from the DOM.
`removeAttr()`	Remove an attribute from each of the selected elements.
`replaceAll()`	Replace each target element with the set of selected elements.
`replaceWith()`	Replace each element in the set of selected elements with new content.
`wrap()`	Wrap an HTML structure around each element in the set of selected elements.
`unwrap()`	Remove the parents of the set of selected elements from the DOM.
`wrapAll()`	Wrap an HTML structure around all elements in the set of selected elements.
`wrapInner()`	Wrap an HTML structure around the content of each element in the set of selected elements.

Creating a Lightbox Effect

In the next few exercises you'll use several of the DOM manipulators to create a lightbox effect. Many websites use a lightbox effect to show enlargements of photographs centered and highlighted on web pages. You'll be able to use the effect on your web pages too, once you've learned how to put together the function.

Some of the manipulators that you'll use during the exercise are specifically designed for getting or modifying information about CSS. These are described in **Table 4.2**.

The first order of business is covering the current page with a translucent background on which the photograph will be displayed.

TABLE 4.2 DOM CSS Manipulators

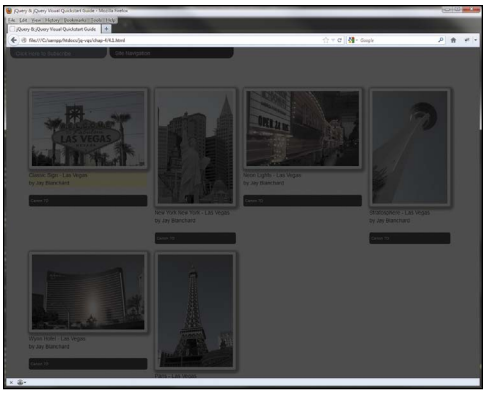
Method	Use It To...
`css()`	Get or set the value of a style property for the first element in the set of selected elements.
`height()`	Get the current computed height for the first element in the set of selected elements.
`innerHeight()`	Get the current computed height for the first element in the set of selected elements, including the padding but not the border.
`outerHeight()`	Get the current computed height for the first element in the set of selected elements, including padding, border, and optionally, margin.
`width()`	Get the current computed width for the first element in the set of selected elements.
`innerWidth()`	Get the current computed width for the first element in the set of selected elements, including the padding but not the border.
`outerWidth()`	Get the current computed width for the first element in the set of selected elements, including padding and border.
`offset()`	Get the current coordinates of the first element in the set of selected elements, relative to the document.
`position()`	Get the current coordinates of the first element in the set of selected elements, relative to its parent element.
`scrollLeft()`	Get the current number of pixels hidden from view to the left of any scrollable area for the first element in the set of selected elements.
`scrollTop()`	Get the number of pixels hidden above any scrollable area for the first element in the set of matched elements.
`remove()`	Remove the set of selected elements from the DOM.
`removeAttr()`	Remove an attribute from each of the selected elements.
`replaceAll()`	Replace each target element with the set of selected elements.
`replaceWith()`	Replace each element in the set of selected elements with new content.
`wrap()`	Wrap an HTML structure around each element in the set of selected elements.
`unwrap()`	Remove the parents of the set of selected elements from the DOM.
`wrapAll()`	Wrap an HTML structure around all elements in the set of selected elements.
`wrapInner()`	Wrap an HTML structure around the content of each element in the set of selected elements.

To use the append() method to display a translucent shade:

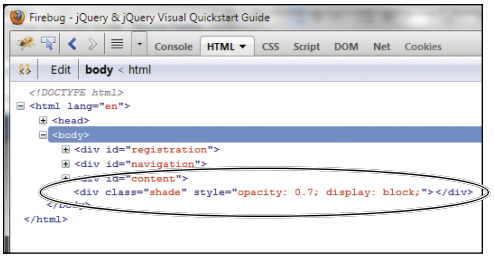
1. Open **gallery.html** in your text editor and add the **data-photo** attribute to each of the list items (Script **4.1.html**):

```
<li>
<img src="images/thumb_lv01.jpg"
→ data-photo="images/lv01.jpg"
→ alt="Classic Sign - Las Vegas" />
</li>
<li>
<img src="images/thumb_lv02.jpg"
→ data-photo="images/lv02.jpg" alt=
→ "New York New York - Las Vegas" />
</li>
<li>
<img src="images/thumb_lv03.jpg"
→ data-photo="images/lv03.jpg"
→ alt="Neon Lights - Las Vegas" />
</li>
<li>
<img src="images/thumb_lv04.jpg"
→ data-photo="images/lv04.jpg" alt=
→ "Stratosphere -  Las Vegas" />
</li>
<li>
<img src="images/thumb_lv05.jpg"
→ data-photo="images/lv05.jpg"
→ alt="Wynn Hotel - Las Vegas" />
</li>
<li>
<img src="images/thumb_lv06.jpg"
→ data-photo="images/lv06.jpg"
→ alt="Paris - Las Vegas" />
</li>
```

2. Save the **gallery.html** file and upload it to your web server.



A The backdrop is in place for the lightbox.



B The backdrop **div** element is the last element within the body tags.

3. Edit **jquery.custom.js** and insert the following code to add a translucent background to the browser window:

```
$('.imageGallery li img')
.click(function() {
$('body').append
('<div class="shade"></div>');
$('.shade')
.css('opacity', 0.7).fadeIn();
});
```

4. Save the jQuery file and upload it to your server.
5. Click on any of the images in the photo gallery and the background should appear. There's no way to get rid of it at this point without reloading the page. You'll add code to remove it later A.

For the backdrop to appear, you have to append a **div** to the body element of your page and declare the shade class on the **div** (the shade class is already defined for you in **css/base.css**). At this point, you apply a CSS opacity property (to make the backdrop translucent) and use **fadeIn()** to bring the backdrop into view. (More on **fadeIn()** and other effects in Chapter 8, "Creating Captivating Effects.")

Take a look at your DOM inspection tool while you have the **div** applied to the body. Notice that the backdrop **div** is the last element within the body tags because **append()** inserts content at the end of the selected element B.

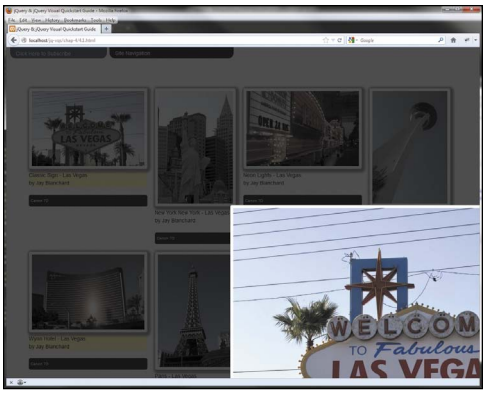
With the backdrop in place, it's time to add the photo. There will be two things you'll have to take care of: preloading all the full-sized images and placing the image centered on the browser window.

continues on next page

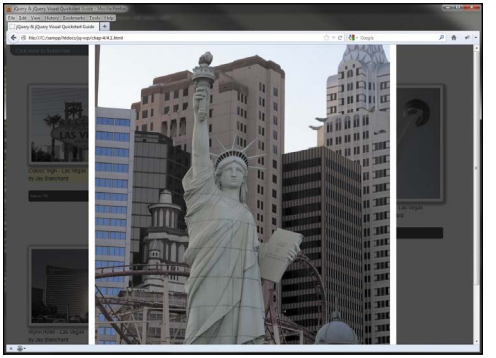
The reason for preloading the images is to ensure that the lightbox function can properly measure the image and know how to place it within the window. The jQuery methods can't get the height and width of an element that isn't currently available in the DOM. Failing to perform this step results in the image not being centered properly C.

You'll also use the browser window's height to set the size of the image to make sure the photo is fully displayed within the boundaries of the browser window. Many of the full-sized images in the example are either taller or wider than the browser window D.

As a matter of organization, most developers will group functions like the image preloader near the top of their jQuery file. In this case, the preloader needs to have completed its job before the lightbox function is called, so let's put the preloading function together first.



C The top-left corner of the image is centered on the screen rather than the whole photo.



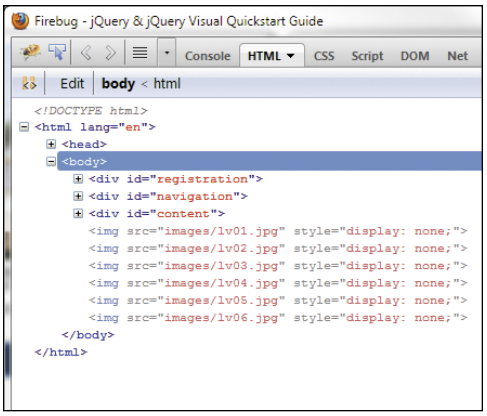
D You can never tell how tall the Statue of Liberty is until you try to fit her in a browser window.

To create an image preloader using appendTo():

1. Edit your jQuery file to add the image preloader:

```
function preload(arrayOfImages) {
$(arrayOfImages).each(function(){
$('<img />')
.attr('src',this)
.appendTo('body')
.css('display','none');
});
}
```

   You start by declaring a function named **preload**. The function is given the argument of **arrayOfImages**. Once the



E The new image tags have been appended to the body.

function is called, the jQuery method **each()** loops through each item in the array that you'll pass to the function. (More on **each()** in Chapter 9, "Turning on jQuery's Utilities.")

For each image, you append an image tag to the body. Then you set the **src** attribute for the image tag to the current image information. Finally, you make sure the images are not visible until you need them by setting their CSS display method to **none**.

Using **appendTo()** here makes perfect sense because it allows you to specify attributes more easily for each image tag prior to the tag being added to the page.

2. Create the array inside a function call to **preload**:

```
preload([
    'images/lv01.jpg',
    'images/lv02.jpg',
    'images/lv03.jpg',
    'images/lv04.jpg',
    'images/lv05.jpg',
    'images/lv06.jpg'
]);
```

The square brackets indicate a JavaScript array using JavaScript Object Notation. The path for each full-sized image has been specified in a comma-separated list. Have a look at your DOM inspector and you should see the image tags just before the closing body tag E.

Now let's add further to the lightbox function.

To use height() and width() to set an element's size and position:

1. Open **jquery.custom.js** in your text editor.
2. Add the following jQuery code to create an image tag:

```
var imgSRC = $(this)
→.attr('data-photo');

var imgTAG =
→'<img src="'+ imgSRC + '" />';
```

   This code should be added immediately after the line where you applied **fadeIn()** to the backdrop.

3. Continue the function by adding the following code to append the modal window to the body and the image tag to the modal window:

```
$('body')
.append('<div class="photoModal">
</div>');
$('.photoModal').html(imgTAG);
$('.photoModal')
    .fadeIn('slow')
    .append('<div>
<a href="#" class="closePhoto">
Close X</a></div>');
```

   The additional **append()** method adds an anchor tag to the modal, which will be used for closing the photo.

4. Enter the code to check the window's height and apply the height to the image:

```
var windowHeight =
→ $(window).height();
$('.photoModal img')
.css('height',
    (windowHeight - 200));
```

You've subtracted 200 pixels from the window's height to ensure that the image will fit in the browser window.

5. Save information about the modal's current height and width to two variables.

These two variables will be applied to the modal to center it horizontally and vertically within the browser window:

```
var modalTopMargin =
→ ($('.photoModal')
    .height() + 20) / 2;
var modalLeftMargin =
→ ($('.photoModal')
    .width() + 20) / 2;
```

The reason you add 20 to the height and width is because the CSS specified for the modal window has a border of 10 pixels per side F.

continues on next page

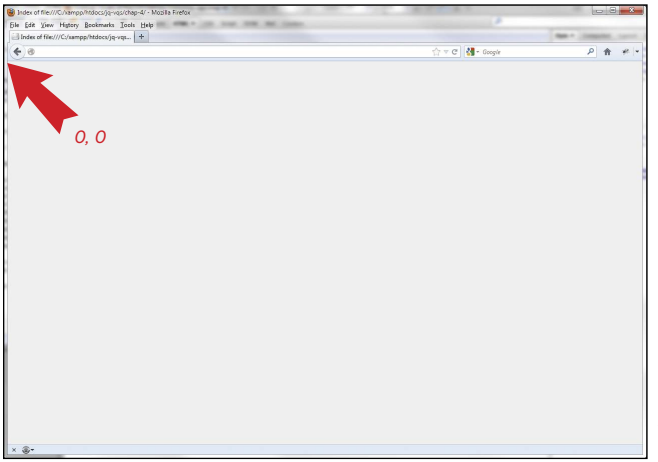
F Take note of the border to make sure the photo is perfectly centered.

6. Add the code to apply the CSS to the modal:

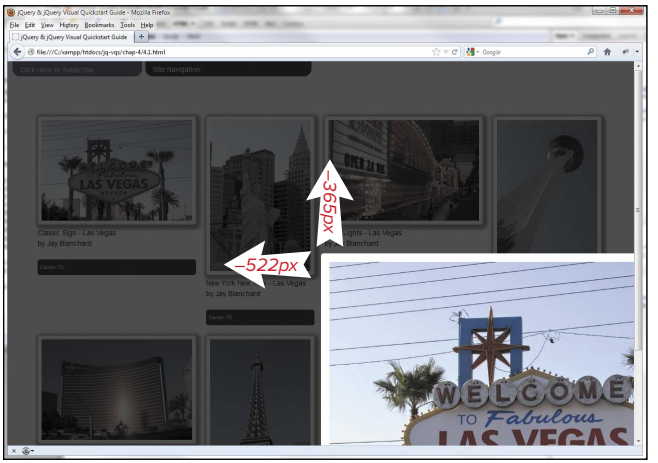
```
$('.photoModal')
.css({
→'margin-top' : -modalTopMargin,
→'margin-left': -modalLeftMargin
→});
```

In the original CSS (see **css/base.css**) for the modal, the top-left corner is originally set to be in the center of the screen. The top-left corner of the browser window is at coordinates 0, 0 G.

To make sure the photo is centered, you apply negative measurements from the photo's top-left corner to move it into position H.



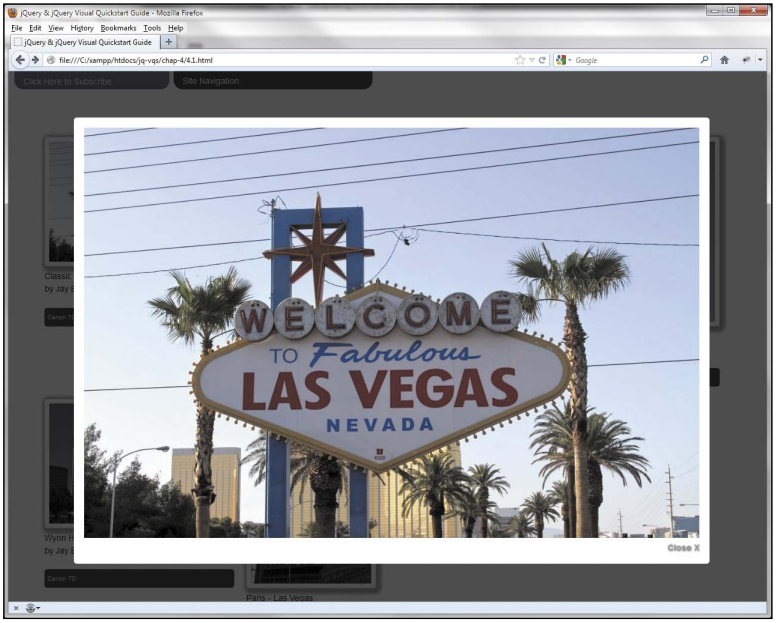
G The base coordinates for the browser window start at the upper-left corner.



H By calculating the photo's height and width, you can apply negative numbers to move it into position.

7. Save the jQuery file and upload it to your web server. Reload gallery.html into your web page and click on one of the pictures I.

There's only one problem at this point: you can't close a picture once you've opened it. Because you've added elements to the DOM that were not previously there, you'll have to use a special way to attach event handlers to account for the new elements.



I The picture is sized and presented!

To close the picture using remove():

1. Reload **jquery.custom.js** into your text editor.
2. Add the following function to the file:

```
$('body')
.on('click', '.closePhoto',
→ function(e){
e.preventDefault();
$('.photoModal, .shade')
    .fadeOut(function(){
        $(this).remove();
    });
});
```

   The **on()** method accounts for elements either in the DOM now or added in the future. You use it to bind event handlers to items within a selected element. In the exercise, you attached the click event handler to the body and specified that the handler should answer to any item having the **closePhoto** class. You'll recall that you appended an anchor tag having the class **closePhoto** in the previous exercise.

   Once the tag is clicked, the photo modal and the backdrop are faded and then removed using the **remove()** method, allowing the lightbox function to be reset for its next performance.

   You have undoubtedly noticed the **preventDefault()** method used here. You passed the click event *e* to the function:

```
function(e){...}
```

   To keep the link from acting normally, which is typically navigating to another page, you applied the **preventDefault()** method to the event, which does what it says—it prevents the default event action from occurring.

3. Save the file and upload it to your web server.
4. Reload the **gallery.html** page. Click on an image and then click on the "Close X" link at the bottom right of the image. Your gallery page has returned to normal.

More Fun with DOM Manipulators

Let's look at more ways to use some of the other DOM manipulators.

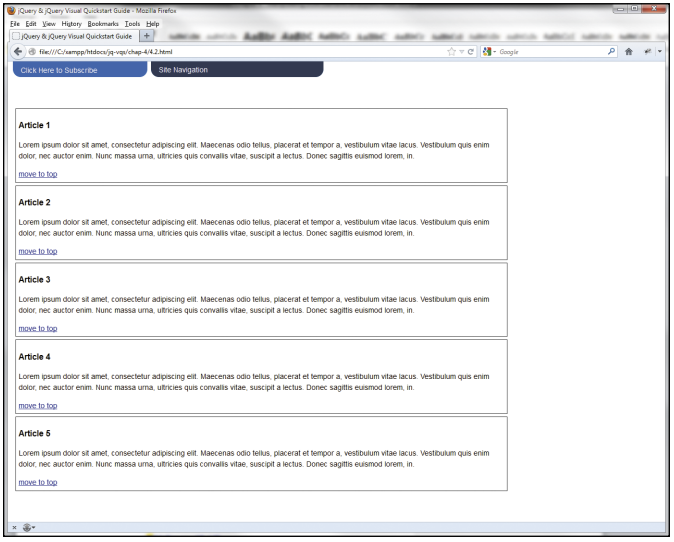
To use before() to rearrange order:

1. Open a new copy of the HTML5 boilerplate in your text editor and add the following markup (Script **4.2.html**):

```
<div id="content>
<div class="article">
<h3>Article 1</h3>
<p>Lorem ipsum...</p>
<a href="" class="mover">
move to top</a></div>
<div class="article">
<h3>Article 2</h3>
<p>Lorem ipsum... </p>
<a href="" class="mover">
move to top</a></div>
<div class="article">
<h3>Article 3</h3>
<p>Lorem ipsum...</p>
<a href="" class="mover">
move to top</a></div>
<div class="article">
<h3>Article 4</h3>
<p>Lorem ipsum...</p>
<a href="" class="mover">
move to top</a></div>
<div class="article">
<h3>Article 5</h3>
<p>Lorem ipsum...</p>
<a href="" class="mover">
move to top</a></div>
</div>
```

2. Save the file as **article.html** and upload it to your web server. Load the page into your browser A.

continues on next page



A The list of articles in their normal order.

3. Modify **jquery.custom.js** with the following code to move an article to the top of the list of articles:

```
$('.mover').click(function(e) {
e.preventDefault();
$('#content div:first')
.before($(this).parent('div'));
});
```

4. Save the jQuery file and upload it to your web server.
5. Reload **article.html** in your web browser and click on any of the "move to top" links. The article moves to the top of the list B.

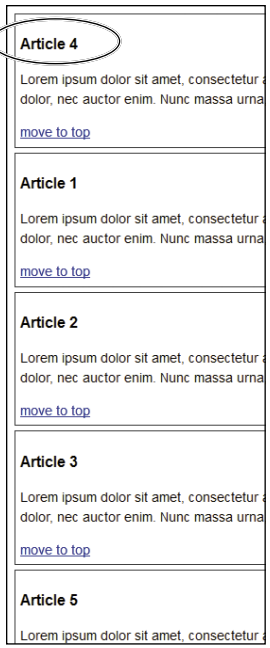
Let's look at what's in play here. Using **before()** makes things read backward so the selector selects the first **div**, using a jQuery selector extension (more about those in Chapter 5, "Harnessing Advanced Selectors"), **:first**. The first **div** in the group is the **div** you'll insert your chosen **div** before. Whew. Then you invoke the **before()** method to carry your chosen **div** to the first spot in the group C.

To get the chosen **div**, you get the parent **div** of the clicked link. The **parent()** method is a DOM traversal method you'll see again in Chapter 6, "Traversing the DOM Tree."

The **before()** method has a counterpart that performs the same job exactly, the **insertBefore()** method. The **insertBefore()** method has one huge advantage: It's much easier to read:

```
$(this).parent('div')
.insertBefore($('#content div:first'));
```

This line of code says to take the clicked element's parent **div** and insert it before the first **div** in the selected group of **div**s. Which one should you use? As mentioned earlier, it's a matter of personal preference.



B You can rearrange the list to bring an article to the top.

C Clicking the third **div** moves it to the top of the list.

Getting and Setting Measurements

There are some DOM manipulators whose sole purpose is to help you get and set measurements. Let's use a couple of those to help animate a floating menu.

To use scrollTop() and offset() to create a floating menu:

1. Modify the markup of **article.html** first by adding a **div** to wrap the **div** with an **id="content"** (Script **4.3.html**):

```
<div class="pageWrapper">
<div id="content">
// all of the content is here
</div>
</div>
```

2. Add the following markup to define the floating menu. It must be within the **pageWrapper div**:

```
<div class="sidebar">
<a href="article.html">
Articles</a><br />
<a href="gallery.html">
Photo Gallery</a>
</div>
```

3. Save **article.html** and upload it to your web server.

4. Create the jQuery function to make the menu float in **jquery.custom.js**:

```
var sidebarOffset =
→$('.sidebar').offset();
var paddingTop = 10;
$(window).scroll(function() {
if ($(window).scrollTop() >
→sidebarOffset.top) {
$('.sidebar').stop()
.animate({
marginTop: $(window).scrollTop() -
→sidebarOffset.top + paddingTop
});
} else {
$('.sidebar').stop()
.animate({
marginTop: $(window).scrollTop()
});
}
});
```

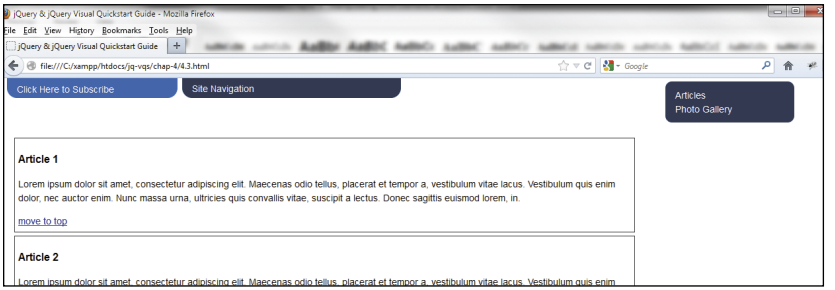
5. Save **jquery.custom.js** and upload it to your web server.

continues on next page

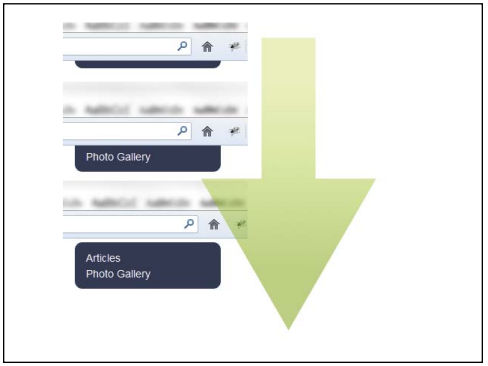
6. Load **`article.html`** into your web browser and you'll see the floating menu on the right-hand side of the page A.

   As you scroll down the page, the menu will float into place B.

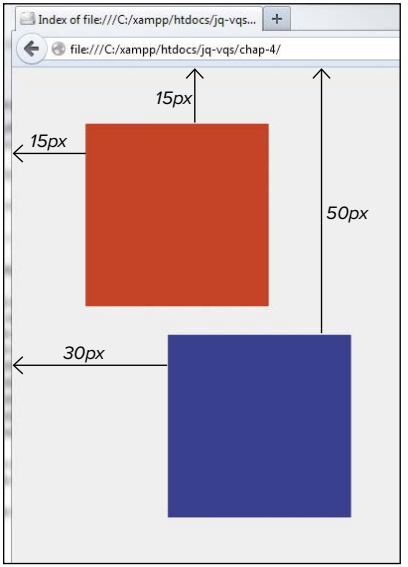
Let's see what's behind the curtain on making this little function work. First you save the sidebar's offset to a variable. The **`offset()`** function gets the current position of an element relative to the document. The function returns an object having two properties: **`top`** and **`left`** C.



A The new menu is ready to go.



B The menu floats into view as lightly as a cloud.

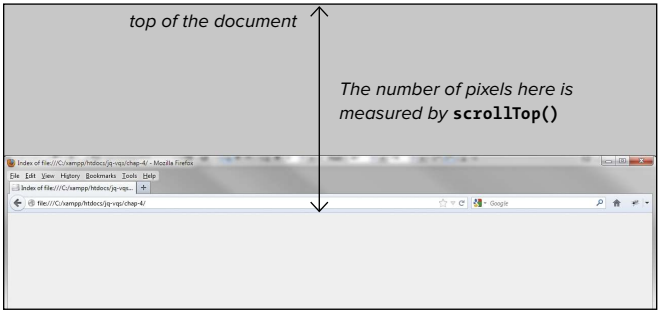


C An example of offset for two elements.

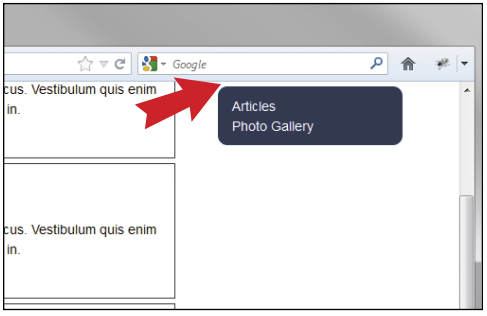
The other measurement you took is the window's **`scrollTop()`** amount. This measurement, in pixels, is the number of pixels hidden from view above the browser window as you scroll down the page D.

TIP You can determine the bottom and right properties of an object by using `offset()` and a little math.

Once the scroll event takes place, all you have left to do is to animate the **`div`** into its new position (you'll read more about animations in Chapter 8). You'll do that based on the measurement provided by **`scrollTop()`**. Then you do a little math to subtract the original offset amount and add in a padding value to make sure the **`div`** is 5 pixels below the edge of the top of the browser window E.



D The **`scrollTop()`** method measures what you can't see.



E The original gap is maintained after the animation.

Cloning

You should be getting pretty comfortable manipulating elements in the DOM. You've learned how to add and remove elements, get measurement information, and set measurement information. Let's turn our focus to duplicating elements on a page, a little thing jQuery calls *cloning*.

On the surface, cloning a group of elements on a page looks pretty simple. All you have to do is use the **`clone()`** function and you're all set, right? Let's dig a little further.

When you use the **`clone()`** method, you're making a copy of the selected elements and all of their descendants and any text nodes contained within the selected items and the descendants. This is known as a *deep copy*.

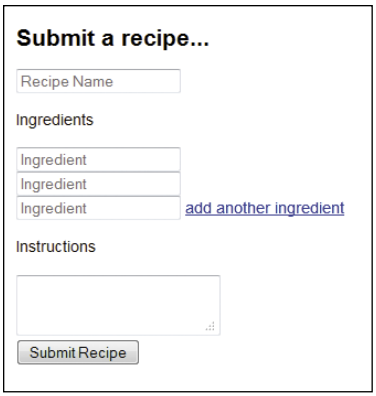
You can also copy the entire set of event handlers bound to the selected elements, ensuring that your functions will continue to work even though you're adding new elements to the DOM. You do this by setting the **`withDataAndEvents`** and **`deepWithDataAndEvents`** properties of **`clone()`** to **`true`**.

To demonstrate **`clone()`**, you'll create a new form for the website that allows visitors to submit recipes. Some recipes have more ingredients than others, but you don't want to clutter up the page with input elements. You'll use the **`clone()`** method to allow form users to add as many ingredient fields as they need.

To use clone() to add form elements:

1. Use a fresh copy of the HTML5 boilerplate in your text editor and add the following markup to create a recipe form (Script **`4.4.html`**):

```
<div id="content">

<h2>Submit a recipe...</h2>

<form name="recipe" action="inc/
→ php/recipe.php" method="post">

<input name="recipeName"
→ placeholder="Recipe Name"/>

<div id="ingredients">

<p>Ingredients</p>

<span class="inputSpan">

<input name="recipeIngredient[]"
→ placeholder="Ingredient" />

<br /></span>

<span class="inputSpan">

<input name="recipeIngredient[]"
→ placeholder="Ingredient" />
→ <br /></span>

<span class="inputSpan">

<input name="recipeIngredient[]"
→ placeholder="Ingredient" /> 

<a href="newIngredient">add
→ another ingredient</a>

<br /></span>

</div>

<p>Instructions</p>

<textarea name=
→ "recipeInstructions">

</textarea><br />

<input type="submit" name="submit"
→ value="Submit Recipe" />

</form>

</div>
```



A The new recipe form is almost ready to go.

Take note of the span tags surrounding the inputs for ingredients. These are used to make writing your code much easier and more compact. Additionally, each ingredient tag is named with square brackets (**[]**) to make them each part of an array that can be handled more easily by server-side languages like PHP.

2. Save the file as **recipe.html** and upload it to your web server. When loaded into a browser, it looks like A.

3. Add a function to **jquery.custom.js** to clone the last recipe ingredient span:

```
$('a[href="newIngredient"]')
.click(function(e){
e.preventDefault();
var clonedInput =
→$('.inputSpan').filter(':last')
.clone(true, true);
```

Using a class on the span tags surrounding it helps to keep your selector short. Be sure to set the **clone()** function's properties to **true, true** so event handlers are copied.

4. Get the current value of the last input. You'll use this value to make sure you don't lose any ingredients:

```
var lastInputData = $('input[name=
→"recipeIngredient[]"]')
.filter(':last').val();
```

continues on next page

5. Set the last ingredient input's HTML to get rid of the link and to ensure that it retains its current value:

```
$('.inputSpan').filter(':last')
.html('<span class="inputSpan">
→<input name="recipeIngredient[]"
→placeholder="Ingredient"
→value="' + lastInputData + '" />
→<br /></span>');
```

Resetting the HTML of the element prevents it from creating the "add another ingredient" link again and again B.

6. Append the cloned input to the ingredients **div**:

```
$('#ingredients')
.append(clonedInput);
```

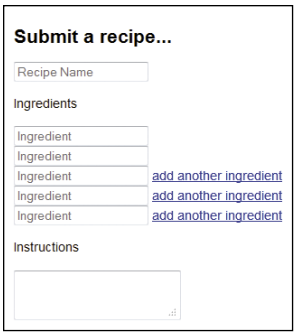
7. Clean up the new input by setting its value to be blank and then placing the focus on the new input:

```
$('input[name=
→"recipeIngredient[]"]')
.filter(':last').val('');
$('input[name=
→"recipeIngredient[]"]')
.filter(':last').focus();
});
```

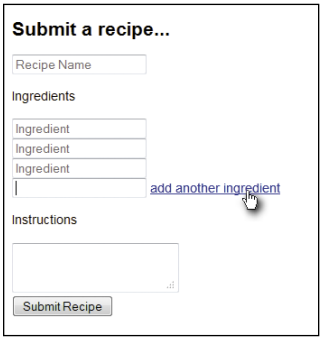
Setting the focus into the new input is a convenience for users. It allows them to just start typing when the new element is added.

8. Save the jQuery file and upload it to your server. Reload **recipe.html** and click the "add another ingredient" link C.

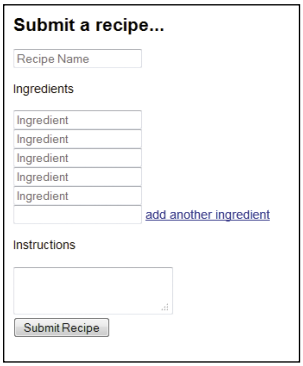
If you keep clicking the link, the click event handler is triggered each time without you having to resort to changing the event handler D.



B Duplicating the links not only looks bad, but also it's confusing to the user.



C A new ingredient field has been added and now has the focus.



D The click event is still triggered each time because you set up **clone()** to copy the event handlers for the form.

Chaining jQuery Methods

In many of the exercises in this book, you've used several jQuery methods on a single selector. Using more than one function on a selector is known as *chaining*.

Chaining is beneficial for two reasons. First, you don't have to reselect elements to add another function to them, thus saving time. Second, you can use chaining to make your code much more readable. You're allowed to place line breaks between each function:

```
$('input[name="recipeIngredient[]"]')
    .filter(':last')
    .val('')
    .focus();
});
```

The only caveat with chaining you need to be aware of is function order. Be sure to add functions in the order you wish them to be executed or the results may not be what you expect.

Keep in mind that chaining jQuery functions is not suitable for every situation. There may be times when you need to reselect elements because of the function's length or the order in which you need things to occur. In those cases, you may want to cache a selector by holding it in a variable:

```
var ingredients = $('input[name="recipeIngredient[]"]');
```

The selector **`$(ingredients)`** is now reusable:

```
$(ingredients)
    .filter(':last')
    .val('')
    .focus();
```

Caching the selector also prevents jQuery from having to reselect the elements each time it's used, providing enhanced performance—especially when there is a large group of elements defined by one selector.

Changing an Input Element

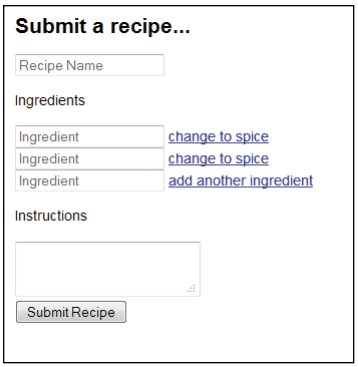
Let's make one other change to the form. Assume the user wants to designate that an ingredient is really a spice. To accomplish the change, you'll use the **replaceWith()** manipulator to change the input.

To use replaceWith() to change an element:

1. Modify the first two ingredient inputs of **recipe.html** to create a link that will trigger the change to a spice, as seen in the following highlighted markup (Script **4.5.html**):

```
<span class="inputSpan">

<input name="recipeIngredient[]"
→ placeholder="Ingredient" />
→  <a href="makeSpice"
→ class="ingredientType">change
→ to spice</a><br /></span>
```

2. Save **recipe.html** and upload it to your web server A.



A The new links have been added to the input boxes.

3. Open **jquery.custom.js** and modify the function you created in the previous exercise to account for the additional link. The section you need to add is highlighted:

```
$('.inputSpan')
.filter(':last')
.html('<span class="inputSpan">
→<input name="recipeIngredient[]"
→placeholder="Ingredient" value=
→"' + lastInputData + '" />
→ <a href="makeSpice"
→class="ingredientType">change
→to spice</a><br /></span>');
```

By making this modification, you ensure that the link to change the input element is available.

4. Create a new function in **jquery.custom.js**. The function will determine what kind of input is available and make the needed change:

```
$('#ingredients').on('click',
→'.ingredientType', function(e) {
e.preventDefault();
var ingredientType =
→$(this).attr('href');
if('makeSpice' == ingredientType)
{
```

5. Get the existing value of the input:

```
var oldElement =
→$(this).closest('span');
var oldElementValue =
→$(this).closest('span')
.find('input').val();
```

6. Create the new input element and assign it to a variable to be used later in the function:

```
var newElement =
→'<span class="inputSpan">
→<input name="recipeSpice[]"
→placeholder="Spice" value="' +
→oldElementValue + '" />
 <a href="makeIngredient"
→class="ingredientType">
→change to ingredient</a>
→<br /></span>';
```

7. Replace the old input with the new input:

```
$(oldElement)
.replaceWith(newElement);
```

8. Set the new input's value based on what was already entered in the form:

```
$(newElement).find('input')
.val(oldElementValue);
```

continues on next page

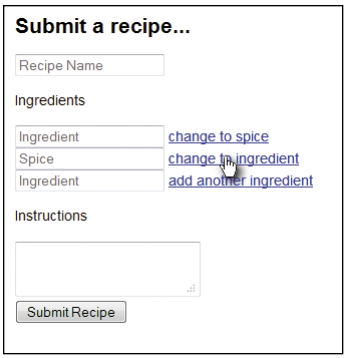
9. Add the remainder of the function to change the input back to an ingredient input if requested:

```
else {
var oldElement =
→$(this).closest('span');
var oldElementValue =
→$(this).closest('span')
.find('input').val();
var newElement =
'<span class="inputSpan">
→<input name="recipeIngredient[]"
→placeholder="Ingredient"
→value="' + oldElementValue
→+ '" /> 
→<a href="makeSpice"
→class="ingredientType">change
→to spice</a>
<br /></span>';
$(oldElement)
.replaceWith(newElement);
$(newElement).find('input')
.val(oldElementValue);
}
});
```

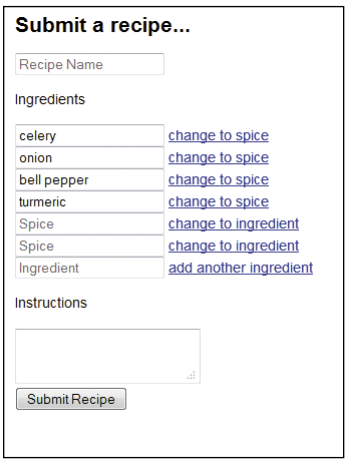
The functionality of the **else** condition is exactly the same as the **if** condition, except that it changes the input type back to ingredient.

10. Save **jquery.custom.js** and upload it to your web server. Reload **recipe.html** in your browser and click one of the "change to spice" links B.

TIP When you add an ingredient, the creation of another input element occurs and has the "change to spice" link added C.



B The input has been changed to reflect its status as a spice.



C New inputs have been added, ready to take their place in the recipe as an ingredient or a spice.

Take a look at your DOM inspection tool, and you'll see the changes you put into place D.

There should be no doubt that jQuery makes it very easy to manipulate DOM elements. The library excels at adding, removing, measuring, and changing elements on every web page you create. The library's ability to modify markup is a valuable set of tools that lets you worry less about existing markup—giving you the power to make sensible changes that will allow your functions to operate effectively.

You also learned how to chain jQuery methods to create complex functions. Chaining makes your code compact, easy to read, and efficient.

Next you're going to ramp up your understanding and use of selectors. Selector boot camp ahead!

D The replaced input elements are evident when inspecting the DOM.

Rewind and Review

Take a few moments to reflect on what you've learned in this chapter:

- How many of jQuery's DOM manipulators have counterpart functions that perform the same action?
- Are there advantages to using functions that perform the same action but have a different syntax? What are the advantages?
- Why is it a good idea to preload images?
- What is the benefit of using a DOM inspection tool like Firebug?
- What is **`preventDefault()`** used for?
- What is the difference between **`postion()`** and **`offset()`**?
- How do you preserve event handlers on cloned elements?
- How does the term deep copy apply to cloned elements?
- When should you cache a selector?
- How many jQuery functions can you chain together?
- Are line breaks allowed when chaining jQuery methods?

Index

Symbols

{} (curly brackets), using with attributes, 49

" (quotes), using with attribute selectors, 82

A

accordion widget, 169–171

`:active` CSS pseudo-class selectors, 7

`add()` DOM tree traversal method, 103

`addClass()` function, 36, 50–51

- usage, 36
- using, 50–51

`:after` CSS pseudo-class selectors, 7

`after()` DOM insertion manipulator, 56

AJAX (Asynchronous JavaScript and XML). *See also* JSON (JavaScript Object Notation)

- attaching **`on()`** method, 117–119
- event bubbling, 117–118
- execution of requests, 110
- **`get()`** shorthand method, 110
- **`getJSON()`** shorthand method, 110, 123–125
- **`getScript()`** shorthand method, 110
- **`load()`** shorthand method, 110–116
- managing loaded content, 117–118

AJAX *(continued)*

- PHP and MySQL, 120
- **`post()`** shorthand method, 110, 119–121
- shorthand methods, 110
- testing server-side scripts, 121

Ajax requests, troubleshooting, 181–182

Ajax scripts, testing, 182

`all` selector, 2, 4

`alt` attribute, using, 47

`andSelf()` DOM tree traversal method, 103

`animate()` custom animation method, 133, 135–139

animations, easing, 130. *See also* custom animation methods

`append()` DOM insertion manipulator, 56, 58–60

`appendTo()` DOM insertion manipulator, 56, 60–61

applications, planning, xix

arguments

- explained, xii–xiv
- passing to functions, xiv
- transformation of, xiv
- as windows, 142

array elements, finding, 142
array-like objects, converting to JavaScript arrays, 142. *See also* objects
arrays
  identifying arguments in, 142
  iterating over, 142
  merging contents of, 142
  searching for values in, 142
  translating items in, 142
`article.html` file, saving, 67
articles
  with bylines and post times, 84
  displaying with tags, 88
  highlighting, 85
  placing focus on information in, 90
  selecting titles in, 93
`attr()` function
  adding attributes, 48–49
  setting attributes, 48–49
  usage, 36
  using, 46–47
attribute methods
  `addClass()` function, 36, 50–51
  `attr()` function, 36, 46–47
  `hasClass()` function, 36, 52
  `prop()` function, 36
  `removeAttr()` function, 36
  `removeClass()` function, 36, 51–52
  `removeProp()` function, 36
  `toggleClass()` function, 36, 53
  `val()` function, 36, 43–44
attribute selectors. *See also* selectors
  combining for OR condition, 91–93
  finding substrings, 83–85
  `[name*]`, 82
  `[name*="value"]`, 83–85
  `[name~=value]`, 85–88
  `[name!=value]`, 89–90
attribute selectors *(continued)*
  quotes (") used with, 82
  using, 12–13
attributes
  changing, 49
  data elements, 37
  using curly brackets ({}) with, 49

B

balloon example, 105
`basic_filter.html` file, saving, 8
`basic.html` file, saving, 4
`:before` CSS pseudo-class selectors, 7
`before()` DOM insertion manipulator, 56, 67–68
`blur()` event handler, using, 30–32
browser features, determining presence of, 142

C

CDNs (Content Delivery Networks), xv, xvii
`change()` event handler, using, 30, 32–33
`:checked` CSS pseudo-class selectors, 7
`children()` DOM tree traversal method, 96
`class` selector, 2
[[Class]] of object, determining, 142
classes, assigning to elements, 2
`clearQueue()` custom animation method, 133
click event handler
  animation triggered by, 21
  generating, 22
  using, 20–22
`clone()` function, 56, 72–74
`closest()` DOM tree traversal method, 96, 100–101
code
  documenting, 22
  downloading for chapters, 191
  minifying, xviii–xix

conditional statements
for, xiii
while, xiii
console.log, including, 180
content filters, using, 9
contents() DOM tree traversal method, 103
css() method, 57
CSS pseudo-class selectors. *See also* selectors
:active, 7
:after, 7
:before, 7
:checked, 7
:disabled, 7
:empty, 7
:enabled, 7
:first-child, 7
:first-letter, 7
:first-line, 7
:focus, 7
:hover, 7
:last-child, 7
:link, 7
:not(selector), 7
:nth-child(n), 7, 12
:nth-last-child(n), 7
:only-child, 7
:root, 7
:visited, 7
curly brackets ({}), using with attributes, 49
custom animation methods. *See also* animations
animate(), 133, 135–139
clearQueue(), 133
delay(), 133
dequeue(), 133
outerHTML() helper function, 134
queue(), 133
stop(), 133

D

data. *See also* stored data values
getting, 146–149
removing, 142
setting, 146–149
storing, 142
data elements, explained, 37
date picker widget, using, 171–173
dbconnect.php file, 192
delay() custom animation method, 133
dequeue() custom animation method, 133
detach() DOM insertion manipulator, 56
Developer Tools, described, xx
dialog widget, adding, 174–177
:disabled CSS pseudo-class selectors, 7
document events, calling, 17
documenting code, 22
Dojo versus jQuery, 187
DOM (Document Object Model)
explained, 35–36
modifying markup in, 48
DOM CSS manipulators
css() method, 57
getting measurements, 69–71
height() method, 57, 62–65
innerHeight() method, 57
innerWidth() method, 57
offset() method, 57, 69–71
outerHeight() method, 57
outerWidth() method, 57
position() method, 57
remove() method, 57, 66
removeAttr() method, 57
replaceAll() method, 57
replaceWith() method, 57, 76–79
scrollLeft() method, 57
scrollTop() method, 57, 69–71

DOM CSS manipulators *(continued)*
setting measurements, 69–71
unwrap() method, 57
width() method, 57, 62–65
wrap() method, 57
wrapAll() method, 57
wrapInner() method, 57
DOM elements
checking, 142
sorting array of, 142
storing data on, 146–149
DOM insertion manipulators
after(), 56
append() method, 56, 58–60
appendTo() method, 56, 60–61
before(), 56, 67–68
clone(), 56
detach(), 56
empty(), 56
insertAfter(), 56
insertBefore(), 56
prepend(), 56
prependTo(), 56
rearranging order, 67–68
remove(), 56
removeAfter(), 56
replaceAll(), 56
replaceWith(), 56
unwrap(), 56
wrap(), 56
wrapAll(), 56
wrapInner(), 56
DOM nodes
as XML documents, 142
in XML documents, 142
DOM tree event bubbling, 117–118
DOM tree traversal filters
eq(), 102–106
filter(), 102
first(), 102
has(), 102
is(), 102
last(), 102
map(), 102–103
slice(), 102, 106–107
DOM tree traversal methods
add(), 103
andSelf(), 103
children(), 96
closest(), 96, 100–101
contents(), 103
end(), 103
find(), 96, 99–100
next(), 96–98
nextAll(), 96
nextUntil(), 96
not(), 103
offsetparent(), 96
parent(), 96, 98
parents(), 96
parentsUntil(), 96
prev(), 96–97
prevAll(), 96
prevUntil(), 96
siblings(), 96

E

easing animations, 130
Eclipse IDE, described, xx, 3
element selector, 2
elements, looping through, 143–145
:empty CSS pseudo-class selectors, 7

empty() DOM insertion manipulator, 56
empty function, indicating, 142
:enabled CSS pseudo-class selectors, 7
end() DOM tree traversal method, 103
eq() DOM tree traversal filter, 102–106
event bubbling. *See also* troubleshooting
  explained, 117–118
  using with Ajax-loaded content, 183
event handlers. *See also* form event handlers; keyboard events; mouse events
  attaching, 16
  ready(), 16
ExtJS versus jQuery, 188

F

fadeIn() visibility method, 128
fadeOut() visibility method, 128
fadeTo() visibility method, 128
FadeToggle() visibility method, 128
filter() DOM tree traversal filter, 102
filter() method, using, 8
filters
  :animated, 6
  applying to selectors, 6–7
  content type, 9
  :eq(), 6
  :even, 6, 11–12
  extensions, 6, 10
  :first, 6
  :gt(), 6
  :has(), 6
  :header, 6
  :hidden, 6
  :lt(), 6
  :[name!="value"], 6
  :odd, 6, 11–12
filters *(continued)*
  :parent, 6
  :selected, 6
  using, 8
  :visible, 6
find() DOM tree traversal method, 96, 99–100
Firebug, described, xx
first() DOM tree traversal filter, 102
:first-child CSS pseudo-class selectors, 7
:first-letter CSS pseudo-class selectors, 7
:first-line CSS pseudo-class selectors, 7
floating menu, creating, 69–71
:focus CSS pseudo-class selectors, 7
focus() form event handler, 30
focusout() method, using, 28–29
for condition, using, xiii
form elements
  disabling with **prop()** function, 40–41
  enabling with **prop()** function, 40–41
form event handlers. *See also* event handlers
  blur(), 30–32
  change(), 30, 32–33
  focus(), 30
  select(), 30
  submit(), 30
  test() method, 32
forms, changing input elements, 76–79
functions. *See also* methods
  empty, 142
  executing on queues, 142
  explained, xiii
  versus methods, xiv
  naming, xiii
  showing queue for execution, 142
  using, xiv

G

GitHub site, accessing for Tooltipsy, 156

Google Closure Compiler, minifying code with, xviii–xix

H

has() DOM tree traversal filter, 102

hasClass() function, 36, 52

height() method, 57, 62–65

hide() visibility method, 128–129

:hover CSS pseudo-class selectors, 7

hover() method

- functions, 23
- **hover()**, 22
- navigation elements, 23
- usage, 19
- using, 22

HTML5 boilerplate, 189–190

I

ID attribute, explained, 2

id selector, 2

innerHeight() method, 57

innerWidth() method, 57

input

- getting value with **val()**, 43–44
- setting value with **val()**, 43

input elements, changing, 76–79

input type, changing via **prop()**, 38–39

insertAfter() DOM insertion manipulator, 56

insertBefore() DOM insertion manipulator, 56

installing

- jCarousel Lite, 152–153
- jQuery UI library, 164
- Tooltipsy, 156

is() DOM tree traversal filter, 102

iterating over objects and arrays, 142

J

JavaScript code, executing globally, 142

JavaScript Object Literals, 49

jCarousel Lite. *See also* plugins

- installing, 152–153
- using, 154–156

jQuery

- arguments, xii–xiv
- described, xii
- versus Dojo library, 187
- downloading, xvi–xvii, 191
- versus ExtJS library, 188
- functions, xiii
- learning, xii
- versus MooTools library, 188
- versus Prototype library, 188
- using in projects, xvii
- variables, xii–xiv
- versus YUI library, 187

jQuery library, installing version of, 174–175

jQuery methods

- caching selectors, 75
- chaining, 75

jQuery UI

- configuring, 162–163
- ThemeRoller, 162
- website, 162

jQuery UI files, including in HTML, 165

jQuery UI library, installing, 164

jQuery UI widgets

- accordion, 169–171
- date picker, 171–173
- dialog, 174–177
- implementing tabs, 166–168
- order of, 171

jQuery.*() utility methods, 142

jquery.custom.js file, saving, 21

jQuery.data() method, using, 146–149
jQuery.each() utility
versus **$(element).each()**, 145, 147
using, 143–145
JSON (JavaScript Object Notation), 122. *See also* AJAX (Asynchronous JavaScript and XML)

K

keyboard events. *See also* event handlers
capturing, 25–29
focusin(), 25
focusout(), 25, 28–29
keydown(), 25
keypress(), 25
keyup(), 25
keydown() event handler, problem with, 27
keyup() event handler, using, 25–27

L

last() DOM tree traversal filter, 102
:last-child CSS pseudo-class selectors, 7
libraries, loading, 185–186
lightbox effect
closing picture with **remove()**, 66
creating, 57
displaying translucent shade, 58–60
element size and position, 62–65
image preloader, 60–61
on() method, 66
:link CSS pseudo-class selectors, 7
list, rearranging, 68
loading errors
fixing, 180
tracking down, 180
looping through elements, 143–145

M

MAMP, described, xx
map() DOM tree traversal filter, 102–103
menu, floating, 69–71
methods. *See also* functions
chaining, 75
versus functions, xiv
minifying code, xviii–xix
MooTools versus jQuery, 188
motion methods
slideDown(), 131
slideToggle(), 131–132
slideUp(), 131
mouse events. *See also* event handlers
click(), 19–22
dblclick(), 19
differences in, 24
focusin(), 19
focusout(), 19
hover(), 19
mousedown(), 19
mouseenter(), 19
mouseleave(), 19
mousemove(), 19
mouseout(), 19
mouseover(), 19
mouseup(), 19
trapping, 19–24
MySQL and PHP
dbconnect.php file, 192
registration.php file, 192
using, 120

N

[name*="value"] selector, finding substrings with, 83–85

[name~=value] selector, using, 85–88

[name!=value] selector, using, 89–90

next() DOM tree traversal method, 96–98

nextAll() DOM tree traversal method, 96

nextUntil() DOM tree traversal method, 96

noConflict() method, using with libraries, 185–186

not() DOM tree traversal method, 103

not equal selector

- described, 82
- using, 88–90

:not(selector) CSS pseudo-class selectors, 7

:nth-child(n) CSS pseudo-class selectors, 7, 12

:nth-last-child(n) CSS pseudo-class selectors, 7

O

objects. *See also* array-like objects

- determining empty status of, 142
- iterating over, 142
- merging contents of, 142
- plain type, 142
- returning from JSON string, 142

offset() method, 57, 69–71

- DOM CSS manipulators, 69–71
- usage, 57

offsetparent() DOM tree traversal method, 96

on() method

- using with delegation, 183–184
- using with lightbox effect, 66

:only-child CSS pseudo-class selectors, 7

outerHeight() method, 57

outerHTML() helper function custom animation method, 134

outerWidth() method, 57

P

parent() DOM tree traversal method, 96, 98

parents() DOM tree traversal method, 96

parentsUntil() DOM tree traversal method, 96

password

- error message, 44
- form field for, 38
- visibility, 39

password field, enabling, 41

photo gallery

- setting titles for pictures, 47
- starting point for, 46

PHP and MySQL

- **dbconnect.php** file, 192
- **registration.php** file, 192
- using, 120

plugin code, concatenating, 158

plugins. *See also* jCarousel Lite

- creating, 155
- download options, 152
- looking for, 152
- Tooltipsy, 156

position() method, 57

prepend() DOM insertion manipulator, 56

prependTo() DOM insertion manipulator, 56

prev() DOM tree traversal method, 96–97

prevAll() DOM tree traversal method, 96

prevUntil() DOM tree traversal method, 96

problems. *See* troubleshooting

Products tables, 144–145

progressive enhancement, performing, xix

prop() function, 36
versus **attr()** method, 39
disabling form elements, 40–41
enabling form elements, 40–41
using to change input type, 38–39
properties
changing, 37–42
removeProp() function, 41–42
Prototype versus jQuery, 188

Q

queue, removing items from, 142
queue() custom animation method, 133
quotes ("), using with attribute selectors, 82

R

ready() handler
execution of, 16
relationship to DOM elements, 18
using in separate files, 17
using in web pages, 18
rearranging order DOM insertion manipulator, 67–68
recipe form
changing ingredients, 100
creating, 73
registration.php file, 192
regular expressions, 32
remove()
DOM insertion manipulator, 56
method, 57, 66
removeAfter() DOM insertion manipulator, 56
removeAttr() function, 36, 57
removeClass() function, 36, 51–52
usage, 36
using, 51–52
removeProp() function, 36, 41–42
replaceAll()
DOM insertion manipulator, 56
method, 57
replaceWith()
DOM insertion manipulator, 56
method, 57, 76–79
:root CSS pseudo-class selectors, 7

S

scrollLeft() method, 57
scrollTop() method, 57, 69–71
select() form event handler, 30
selectors. *See also* attribute selectors; CSS pseudo-class selectors
all, 2, 4
applying filters to, 6–7
attribute type, 12–13
caching, 4, 75
class, 2
combining, 5, 91–93
element, 2
id, 2
improving performance, 4
not equal, 82, 88–90
using, 3–4
server-side scripts, testing, 121
show() visibility method, 128–129
siblings() DOM tree traversal method, 96
slice()
DOM tree traversal filter, 102, 106–107
traversal method, 10
stop()method, 23, 133
stored data values, getting, 148–149. *See also* data
storing data, 142
strings, parsing into XML documents, 142

Sublime Text 2, described, xx
submit() form event handler, 30
substrings, finding, 83–85

T

tabs, implementing with jQuery UI, 166–168
test() method form event handler, 32
ThemeRoller
  described, 162
  learning, 163
  use of styles, 163
this versus **$(this)**, 14
time, returning number for, 142
toggle() method, 23, 128–130
toggleClass() function, 36, 53
  usage, 36
  using, 53
tools
  Developer Tools, xx
  Eclipse IDE, xx
  Firebug, xx
  MAMP, xx
  Sublime Text 2, xx
  XAMPP, xx
Tooltipsy
  installing, 156
  using in web pages, 157–159
troubleshooting. *See also* event bubbling
  Ajax requests, 181–182
  Ajax-loaded content, 183–184
  loading errors, 180
  loading libraries, 185–186
  using **noConflict()** method, 185–186
  using **on()** method with delegation, 183–184
tweets, hiding and showing, 129

U

UI (User Interface). *See* jQuery UI
unwrap()method,56–57
utilities website, 141
utility methods
  getting data, 146–149
  jQuery.data(), 146–149
  jQuery.each(), 143–145, 147
  letter case, 145
  setting data, 146–149
  table of, 142

V

val() function
  usage, 36
  using, 43–44
values, managing, 43–44
variables
  explained, xii–xiv
  naming, xiii
visibility methods
  easing plugin, 130
  fadeIn(), 128
  fadeOut(), 128
  fadeTo(), 128
  FadeToggle(), 128
  hide(), 128–129
  show(), 128–129
  toggle(), 128–130
:visited CSS pseudo-class selectors, 7

W

websites
  building, 16
  code for chapters, xii, 191

websites *(continued)*
directory structure, 20
Dojo JavaScript library, 187
ExtJS library, 188
Google Closure Compiler, xviii
HTML5 boilerplate, 189–190
jCarousel Lite, 152
jQuery, xvi–xvii , 191
jQuery UI, 162
link element in form, 20
MooTools library, 188
planning, xix
Prototype class-drive JavaScript library, 188
regular expressions, 32
utilities, 141
YUI JavaScript and CSS library, 187
`while` condition, using, xiii
whitespace, removing around strings, 142
widgets. *See* jQuery UI widgets
`width()` method, 57, 62–65
`wrap()` method, 56–57
`wrapAll()`method, 56–57
`wrapInner()` method,56–57

X

XAMPP, described, xx
XML documents
DOM nodes as, 142
DOM nodes in, 142
parsing strings into, 142

Y

YUI versus jQuery, 187

Unlimited online access to all Peachpit, Adobe Press, Apple Training and New Riders videos and books, as well as content from other leading publishers including: O'Reilly Media, Focal Press, Sams, Que, Total Training, John Wiley & Sons, Course Technology PTR, Class on Demand, VTC and more.

No time commitment or contract required!
Sign up for one month or a year.
All for $19.99 a month

SIGN UP TODAY

peachpit.com/creativeedge

creative edge