Beginning JavaScript®, Fifth Edition
By Jeremy McPeak and Paul Wilton

16
jQuery

WHAT YOU WILL LEARN IN THIS CHAPTER:

- Using jQuery can simplify common tasks
- Creating, modifying, and removing elements with jQuery is easier than with traditional DOM methods
- jQuery makes style modifications, both with individual CSS properties and CSS classes, a breeze
- Handling HTTP requests and responses is much easier than writing pure XMLHttpRequest code
- Deferred objects are useful, especially when used with Ajax requests

WROX.COM CODE DOWNLOADS FOR THIS CHAPTER

You can find the wrox.com code downloads for this chapter at `http://www.wiley.com/go/BeginningJavaScript5E` on the Download Code tab. You can also view all of the examples and related files at `http://beginningjs.com`.

JavaScript is essential to web development. And even though JavaScript development is relatively straightforward today, it was extremely challenging until early 2011 when Microsoft released the ninth version of Internet Explorer.

Let's rewind the clock to the year 2001. The first browser wars were coming to a close, and Microsoft sealed its overwhelming victory with IE6's release. A few months later the software giant released Windows XP, the longest-supported operating system in Microsoft's history, with IE6 as its default browser.

At first, Microsoft enjoyed its 85 percent market share, but as the years passed, growing pressure from Mozilla's Firefox spurred Microsoft to resume development on IE. In 2006, Microsoft

released IE7. This new version included many bug fixes, as well as the implementation of new (and often nonstandard) features.

This was the beginning of a very challenging time for JavaScript developers. The problem with client-side development was the many different web browsers developers needed to support. Not only did developers have to support IE and Firefox, but developers had to support three major versions of IE (6, 7, and 8). Be it writing event-driven code or an Ajax application, somewhere down the line developers ran into the many incompatibilities between different browsers and versions.

Many professional developers found cross-browser development to be too time-consuming and cumbersome to deal with on a daily basis, so they set out to develop frameworks or libraries to aid in their cross-browser development. Some developers released their frameworks to the public, and a few of them gained quite a following. And much like the browser wars of old, eventually a victor emerged. Frameworks such as MooTools and Prototype were quite popular, but jQuery became the de facto standard.

jQuery, like most other frameworks, originated as an Ajax library to simplify client/server communication. Today, however, jQuery simplifies just about every common aspect of JavaScript development; DOM manipulation, Ajax, animation, and component development are much easier with jQuery. In this chapter, you look at jQuery and learn how to use it to simplify your JavaScript development.

Before beginning, a word of note: There is no doubt that jQuery adds benefit to your development time and process. But it is not a substitute for a solid understanding of the JavaScript language and the intricacies of the different browsers for which you have to develop. Frameworks and libraries come and go, but knowledge (and pure JavaScript) is forever.

GETTING jQUERY

Installing jQuery (or any framework for that matter) is very different from installing an application on your computer; there is no setup program, and the installation doesn't change any portion of your system. Basically, all you do is reference the jQuery JavaScript file in your web page.

Open your browser and go to `http://jquery.com/download/`. On this page, you'll find several links to different jQuery-related files. First, you'll see two versions of jQuery: 1.x and 2.x. The two versions are almost identical except that v1.x supports IE 6, 7, and 8 and v2.x does not.

Second, you'll need to choose the compressed or uncompressed version:

- **Compressed version:** This is *minified* (all comments and unnecessary white space are removed from the code files) to make their size as small as possible; doing so makes them faster to download when someone visits your web page. Unfortunately, the minification process makes the JavaScript code difficult to read if you open it in a text editor, but that's a reasonable trade-off in a production environment.
- **Uncompressed version:** This is not minified; it is simply normal JavaScript code files with their white space and comments intact. It's perfectly OK to use uncompressed JavaScript files. Because they are easier to read than compressed files, you can learn much from the gurus who design and develop these frameworks. However, if you plan to roll out a web page using

a framework, be sure to download and use the compressed version, because its file sizes are smaller and download faster.

> ***NOTE*** *The production version of jQuery 2.1.1 is provided in the code download from Wrox.*

You can obtain jQuery in one of two ways. First, simply download whichever version you prefer by right-clicking (or control-clicking on a Mac) the link, and save it in a location you can easily get to.

Alternatively, you can use jQuery's Content Delivery Network (CDN) to add jQuery to your web pages. This prevents you from having to download your own copy of jQuery, and it can also slightly increase the performance of your web pages.

Regardless of how you obtain jQuery, you add it to your pages just like any other external JavaScript file with the `<script/>` element:

```
<script src="jquery-2.1.1.min.js"></script>
<script src="//code.jquery.com/jquery-2.1.1.min.js"></script>
```

The only difference is the value of the `src` attribute. The first `<script/>` element in this example uses a local copy of jQuery 2.1.1, whereas the second uses jQuery's CDN. The examples in this chapter use a local copy of jQuery.

jQUERY'S API

jQuery is JavaScript, but it changes the way that you interact with the browser and document. Whether you're creating HTML elements and appending them to the page or making Ajax calls to the server, jQuery lets you do it in an easy fashion.

At the heart of jQuery is the `jQuery()` function, but in most cases, you won't write your code with `jQuery()`. Instead, you'll use an alias: the dollar function, `$()`. This can seem very weird at first, but it will become completely natural the more you use it.

You'll use the `$()` function for just about everything, including:

- Finding and selecting elements
- Creating, appending, and removing elements
- Wrapping normal DOM objects with `jQuery` objects

Selecting Elements

jQuery revolutionized the way developers find elements in the DOM: with CSS selectors. In fact, the `querySelector()` and `querySelectorAll()` methods discussed in Chapter 9 exist because of jQuery. To retrieve elements with jQuery, you use `$()` and pass it your CSS selector, like this:

```
var elements = $("a");
```

This code assigns a special object, called a `jQuery` object, that represents an array of all `<a/>` elements in the page to the `elements` variable.

jQuery was designed to make DOM manipulation easy, and because of this design philosophy, you can make changes to several elements at the same time. For example, imagine you built a web page with more than 100 links in the document, and one day you decide you want them to open in a new window by setting the `target` attribute to `_blank`. That's a tall task to take on, but it is something you can easily achieve with jQuery. Because you can retrieve all `<a/>` elements in the document by calling `$("a")`, you can call the `attr()` method, which gets or sets the value of an attribute, to set the `target` attribute. The following code does this:

```
elements.attr("target", "_blank");
```

Calling `$("a")` results in a `jQuery` object, but this object also doubles as an array. Any method you call on this particular `jQuery` object will perform the same operation on all elements in the array. By executing this line of code, you set the `target` attribute to `_blank` on every `<a/>` element in the page, and you didn't even have to use a loop!

Because `jQuery` objects are an array, you can use the `length` property to find out how many elements were selected with a CSS query:

```
var length = elements.length;
```

This information can be useful, but you usually won't need to know the length of a `jQuery` object. The most common use of an array's `length` property is for looping, and jQuery is designed to work with multiple elements at the same time. The methods you execute on a `jQuery` object have built-in loops; so, the `length` property is rarely used.

jQuery has a built-in CSS selector engine, and you can use just about any valid CSS selector to retrieve your desired elements—even if the browser doesn't support it. For example, IE6 does not support the `parent > child` CSS selector. If you have the unfortunate need to support that browser, jQuery can still select the appropriate elements with that selector. Consider the following HTML as an example:

```
<p>
    <div>Div 1</div>
    <div>Div 2</div>
    <span>Span 1</div>
</p>

<span>Span 2</span>
```

This HTML code defines a `<p/>` element that contains two `<div/>` elements and a `<span/>` element. Outside the `<p/>` element is another `<span/>` element. Let's say that you need the `<span/>` element inside the paragraph. You can easily select that element with the following:

```
var innerSpan = $("p > span");
```

This line of code uses the `parent > child` CSS selector syntax, and because jQuery has its own CSS selector engine, this code will work in every browser.

jQuery also lets you use multiple selectors in one function call. Simply delimit each selector with a comma as shown in the following code:

```
$("a, #myDiv, .myCssClass, p > span")
```

This code retrieves all <a/> elements, an element with an id of myDiv, elements with the CSS class myCssClass, and all children of <p/> elements. If you wanted to change the text's color of these elements to red, you could simply use the following code:

```
$("a, #myDiv, .myCssClass, p > span").attr("style", "color:red;");
```

This isn't the best way to change an element's style. In fact, jQuery provides you with many methods to alter an element's style.

> ***NOTE*** *For a complete list of supported selectors, see* `http://docs.jquery.com/Selectors`.

Changing Style

Changing an element's style requires you to either modify individual CSS properties or manipulate its CSS classes. jQuery makes it easy to do both. To change individual CSS properties, the `jQuery` object has a method called `css()`, and you can use this method in two ways.

First, you can pass two arguments to the `css()` method: the CSS property's name and its value. For example:

```
$("#myDiv").css("color", "red");
```

This code sets the `color` property to red, thus changing the element's text color to red. The property names you pass to the `css()` method can be in either style sheet format or in script format. That means if you want to change an element's background color, you can pass `background-color` or `backgroundColor` to the method, like this:

```
var allParagraphs = $("p");

allParagraphs.css("background-color", "yellow"); // correct!
allParagraphs.css("backgroundColor", "blue"); // correct, too!
```

This code changes the background color of every <p/> element in the page to yellow and then to blue.

> ***NOTE*** *It's important to remember that jQuery's methods work with one or multiple elements. It doesn't matter how many elements are referenced by a* `jQuery` *object, a method like* `css()` *will change the style of every element in the object.*

Many times, however, you need to change more than one CSS property. Although you can easily accomplish this by calling `css()` multiple times like this:

```
// don't do this
allParagraphs.css("color", "blue");
allParagraphs.css("background-color", "yellow");
```

a better solution would be to pass an object that contains the CSS properties and their values to the `css()` method. The following code calls `css()` once and achieves the same results:

```
allParagraphs.css({
    color: "blue",
    backgroundColor: "yellow"
});
```

Here, you pass an object that has `color` and `backgroundColor` properties to the `css()` method, and jQuery changes the element's or elements' text color to blue and background color to yellow.

Typically, though, if you want to change an element's style, it's better to change the element's CSS class instead of individual style properties.

Adding and Removing CSS Classes

The `jQuery` object exposes several methods to manipulate an element's `className` property; you can add, remove, and even toggle the classes that are applied to an element.

> ***NOTE*** *Did you know you can assign multiple CSS classes to an element? Simply separate each class name with a space!*

Let's assume that the following HTML is in one of your web pages:

```
<div id="content" class="class-one class-two">
    My div with two CSS classes!
</div>
```

This HTML defines a `<div/>` element with two CSS classes, `class-one` and `class-two`. You need to apply two more classes (`class-three` and `class-four`) to this element, and jQuery makes that very easy to do with the `addClass()` method. For example:

```
var content = $("#content");

content.addClass("class-three");
content.addClass("class-four");
```

This code first retrieves the `<div/>` element and then calls the `addClass()` method to add the desired classes. But you can simplify this code by using a technique called *method chaining*. Most jQuery methods return a `jQuery` object, so it's possible to call a method immediately after calling another method—essentially chaining the method calls as demonstrated with this code:

```
content.addClass("class-three").addClass("class-four");
```

This code achieves the same results as before but with fewer keystrokes.

> ***NOTE*** *Most jQuery methods return a* `jQuery` *object, allowing you to immediately call methods one after another.*

But you can simplify this code even more because you can pass both CSS class names to `addClass()`, like this:

```
content.addClass("class-three class-four");
```

You just have to separate the class names with a space.

The `removeClass()` method removes one or multiple classes:

```
content.removeClass("class-one");
```

This code uses the `removeClass()` method to remove the `class-one` class from the element. If you need to remove multiple classes, simply separate them with a space, like this:

```
content.removeClass("class-two class-four");
```

As you can see, the same concepts that let you add classes to an element apply to removing classes. But there's one very important difference: The arguments you pass to `removeClass()` are optional. If you do not pass any arguments to `removeClass()`, it will remove all classes from the element:

```
content.removeClass();
```

This code, therefore, removes all CSS classes from the element(s) represented by the `content` object.

Toggling Classes

Although the `addClass()` and `removeClass()` methods are certainly useful, sometimes you need to just toggle a class. In other words, you remove a class if it's present or add it if it's not. jQuery makes this easy with the aptly named `toggleClass()` method:

```
content.toggleClass("class-one");
```

This code first toggles the `class-one` class. If it is already applied to the element, jQuery removes `class-one`. Otherwise, it adds `class-one` to the element's class list.

This behavior is useful when you need to add or remove a specific class from the element. For example, the following code is vanilla JavaScript and DOM coding to add and remove a specific CSS class depending on the type of event:

```
var target = e.target;

if (e.type == "mouseover") {
    target.className = "class-one";
} else if (e.type == "mouseout") {
```

```
        eSrc.className = "";
    }
```

You can greatly simplify this code by using the `toggleClass()` method, like this:

```
var target = $(e.target);

if (e.type == "mouseover" || e.type == "mouseout") {
    target.toggleClass("class-one");
}
```

Notice how the `$()` function is used in this code: It passes `e.target`, a DOM object, to `$()`. This can at first seem like a strange thing to do, but remember what we said earlier: `$()` is used for many things. One of those things is to *wrap* a normal DOM object with a `jQuery` object.

In technical terms, we call the resulting `jQuery` object a *wrapper object*. Wrapper objects are typically used to enhance the functionality of another object. With jQuery, you're wrapping a `jQuery` object around an element object, enabling you to use jQuery's API to manipulate the element. In the case of this code, you're wrapping a `jQuery` object around an element object so that you can use `toggleClass()` to toggle the `class-one` class.

Checking if a Class Exists

The last CSS class method is the `hasClass()` method, and it returns `true` or `false` depending on if an element has the specified CSS class. For example:

```
var hasClassOne = content.hasClass("class-one");
```

This code uses `hasClass()` to determine if the `class-one` is applied to `content`. If it is, `hasClassOne` is `true`. Otherwise, it's `false`.

Creating, Appending, and Removing Elements

Think back to Chapter 9 and how you create and append elements to the page. The following code will refresh your memory:

```
var a = document.createElement("a");

a.id = "myLink";
a.setAttribute("href", "http://jquery.com");
a.setAttribute("title", "jQuery's Website");

var text = document.createTextNode("Click to go to jQuery's website");

a.appendChild(text);
document.body.appendChild(a);
```

This code creates an `<a/>` element, assigns it an `id`, and sets the `href` and `title` attributes. It then creates a text node and assigns the object to the `text` variable. Finally, it appends the text node to the `<a/>` element and appends the `<a/>` element to the document's `<body/>` element. It goes without saying that creating elements with the DOM methods requires a lot of code.

Creating Elements

jQuery simplifies how you create elements with JavaScript. The following code shows you one way:

```
var a = $("<a/>").attr({
    id: "myLink",
    href: "http://jquery.com",
    title: "jQuery's Website"
}).text("Click here to go to jQuery's website");

$(document.body).append(a);
```

Let's break down this code to get a better understanding of what's taking place. First, this code calls `$()` and passes it the HTML to create. In this case, it's an `<a/>` element:

```
var a = $("<a/>")
```

Next, it chains the `attr()` method to set the `<a/>` element's attributes:

```
.attr({
    id: "myLink",
    href: "http://jquery.com",
    title: "jQuery's Website"
})
```

The `attr()` method is a lot like the `css()` method in that you can set an individual attribute by passing it the attribute's name and value, or you can set multiple attributes by passing an object that contains the attributes and their values.

The code then chains the `text()` method, setting the element's text:

```
.text("Click here to go to jQuery's website");
```

You can also create the same element by passing the entire HTML to `$()`, like this:

```
var a = $('<a href="http://jquery.com" title="jQuery\'s Website">' +
          "Click here to go to jQuery's website</a>");
```

This approach, however, can become less of a benefit and more of a hassle. Not only do you have to keep track of which type of quote you use where, but you might also have to escape quotes.

Appending Elements

The `append()` method is similar to the DOM `appendChild()` method in that it appends child nodes to the parent object. The similarities end there because jQuery's `append()` method is much more flexible than its DOM counterpart. The `append()` method can accept a DOM object, a `jQuery` object, or a string containing HTML or text content. Regardless of what you pass as the parameter to `append()`, it will append the content to the DOM object.

The previous code appends the jQuery-created `<a/>` element to the document's body, like this:

```
$(document.body).append(a);
```

Once again, the `$()` function wraps a `jQuery` object around the native `document.body` object to take advantage of jQuery's simple API.

Removing Elements

Removing elements from the DOM is also much easier with jQuery than with traditional DOM methods. With DOM methods, you need at least two element objects: the element you want to remove and its parent element.

As with everything thus far, jQuery simplifies this process. The only thing you need is the element that you want to remove. Simply call jQuery's `remove()` method, and it removes the element. For example:

```
$(".class-one").remove();
```

This code finds all elements that have the `class-one` CSS class and removes them from the document.

You can also completely empty, or remove all children of, an element with the aptly named `empty()` method. If you wanted to remove every element within the `<body/>`, you could use the following code:

```
$(document.body).empty();
```

Most DOM changes you'll make are in response to something the user did, whether moving the mouse over a particular element or clicking somewhere on the page. So naturally, you'll have to handle events at some point.

Handling Events

When jQuery was created, JavaScript developers had to contend with both the W3C standard and legacy IE event models. Although many developers wrote and used their own event utilities, the vast majority looked to third-party tools to make cross-browser code easier to write and maintain. jQuery was one such tool, and while standard support has gotten substantially better in all browsers, jQuery's event API, specifically the methods used to register event listeners, is still easier to use.

All `jQuery` objects expose a method called `on()` that you use to register event listeners for one or more events on the selected elements. Its most basic usage is very simple, as demonstrated here:

```
function elementClick(e) {
    alert("You clicked me!");
}

$(".class-one").on("click", elementClick);
```

This code registers a `click` event listener on all elements that have a `class-one` CSS class. Therefore, the `elementClick()` function executes when the user clicks any of these elements.

You can also register multiple event listeners with the same event handler function by passing multiple event names in the first argument. Simply separate each event name with a space, like this:

```
function eventHandler(e) {
    if (e.type == "click") {
        alert("You clicked me!");
```

```
    } else {
        alert("You double-clicked me!");
    }
}

$(".class-two").on("click dblclick", eventHandler);
```

This code registers event listeners for the `click` and `dblclick` events on all elements with a `class-two` CSS class. The code inside `eventHandler()` determines what event caused the function to execute and responds appropriately.

Although it can be useful to use a single function for handling multiple events, jQuery also lets you define multiple event listeners with different functions. Instead of passing two arguments to `on()` as shown in the previous examples, you pass an ordinary JavaScript object in which the properties are the event names, and their values are the functions that handle the events. For example:

```
function clickHandler(e) {
    alert("You clicked me!");
}

function dblclickHandler(e) {
    alert("You double-clicked me!");
}

$(".class-three").on({
    click: clickHandler,
    dblclick: dblclickHandler
});
```

This code registers `click` and `dblclick` event listeners for every element with the `class-three` CSS class; different functions handle the `click` and `dblclick` events.

Removing event listeners is equally simple with the `off()` method. Simply supply the same information you passed to `on()`. The following code removes the event listeners registered in the previous examples:

```
$(".class-one").off("click", elementClick);
$(".class-two").off("click dblclick", eventHandler);
$(".class-three").off({
    click: clickHandler,
    dblclick: dblclickHandler
});
```

You can also use the `off()` method to remove all event listeners for the selected elements by not passing any arguments to the method:

```
$(".class-four").off();
```

The jQuery Event Object

As you learned in Chapter 10, some big differences exist between the standard and legacy IE event models. Remember that jQuery was created to make cross-browser JavaScript easier to write and

maintain. So when it came to events, John Resig, the creator of jQuery, decided to create his own `Event` object and base it on the standard DOM `Event` object. This means that you do not have to worry about supporting multiple event models; it's already done for you. All you have to do is write standard code inside your event handlers.

To demonstrate, you can write something like the following code, and it'll work in every browser:

```
function clickHandler(e) {
    e.preventDefault();

    alert(e.target.tagName + " clicked was clicked.");
}

$(".class-two").on("click", clickHandler);
```

> ***NOTE*** *For a complete list of supported events, see jQuery's website at* `http://docs.jquery.com/Events`.

Rewriting the Tab Strip with jQuery

You have learned how to retrieve elements in the DOM, change an element's style by adding and removing classes, add and remove elements from the page, and use events with jQuery.

Now you'll put this newfound knowledge to work by refactoring the toolbar from Chapter 10.

TRY IT OUT Revisiting the Tab Strip with jQuery

Open your text editor and type the following code:

```
<!DOCTYPE html>

<html lang="en">
<head>
    <title>Chapter 16: Example 1</title>
    <style>
        .tabStrip {
            background-color: #E4E2D5;
            padding: 3px;
            height: 22px;
        }

        .tabStrip div {
            float: left;
            font: 14px arial;
            cursor: pointer;
        }

        .tabStrip-tab {
            padding: 3px;
```

```
        }

        .tabStrip-tab-hover {
            border: 1px solid #316AC5;
            background-color: #C1D2EE;
            padding: 2px;
        }

        .tabStrip-tab-click {
            border: 1px solid #facc5a;
            background-color: #f9e391;
            padding: 2px;
        }
    </style>
</head>
<body>
    <div class="tabStrip">
        <div data-tab-number="1" class="tabStrip-tab">Tab 1</div>
        <div data-tab-number="2" class="tabStrip-tab">Tab 2</div>
        <div data-tab-number="3" class="tabStrip-tab">Tab 3</div>
    </div>
    <div id="descContainer"></div>

    <script src="jquery-2.1.1.min.js"></script>
    <script>
        function handleEvent(e) {
            var target = $(e.target);
            var type = e.type;

            if (type == "mouseover" || type == "mouseout") {
                target.toggleClass("tabStrip-tab-hover");
            } else if (type == "click") {
                target.addClass("tabStrip-tab-click");

                var num = target.attr("data-tab-number");
                showDescription(num);
            }
        }

        function showDescription(num) {
            var text = "Description for Tab " + num;

            $("#descContainer").text(text);
        }

        $(".tabStrip > div").on("mouseover mouseout click", handleEvent);
    </script>
</body>
</html>
```

Save this as `ch16_example1.html` and open it in your browser and notice that its behavior is identical to that of `ch10_example17.html`.

If you compare this example with `ch10_example17.html`, you'll notice quite a few differences in the code's structure. Let's start with the very last line of code where you register the event listeners. Your

tab strip needs to respond to the `mouseover`, `mouseout`, and `click` events on the `<div/>` elements inside of the tab strip. jQuery makes it extremely easy to register event listeners on these `<div/>` elements for the desired events:

```
$(".tabStrip > div").on("mouseover mouseout click", handleEvent);
```

This code selects all "tab elements" in the document with jQuery's `$()` function and uses the `on()` method to register the `mouseover`, `mouseout`, and `click` event listeners on those elements.

jQuery code aside, this approach is different from the original. Here, the event listeners are on the `<div/>` elements themselves as opposed to `document`. This will simplify the `handleEvent()` function. Let's look at that now.

The first two lines of `handleEvent()` do two things. The first line wraps a `jQuery` object around the event target, and the second gets the type of event that occurred:

```
function handleEvent(e) {
    var target = $(e.target);
    var type = e.type;
```

In the original version of this code, you used both the event type and the target's CSS class to determine which new CSS class you assigned to the element's `className` property. In this new version, you only need to know the event type. For `mouseover` and `mouseout` events, you simply toggle the `tabStrip-tab-hover` class:

```
    if (type == "mouseover" || type == "mouseout") {
        target.toggleClass("tabStrip-tab-hover");
    }
```

But for `click` events, your new code closely resembles the original's. First, you add the `tabStrip-tab-click` class to the element using jQuery's `addClass()` method:

```
    else if (type == "click") {
        target.addClass("tabStrip-tab-click");

        var num = target.attr("data-tab-number");
        showDescription(num);
    }
```

Then you get the value of the `data-tab-number` attribute. You could optionally use jQuery's `data()` method to do the same thing by passing it the attribute name without `data-`, like this: `data("tab-number")`.

Once you have the tab's number, you pass it on to `showDescription()`. This function did not change much; it simply uses jQuery's API to accomplish its task:

```
function showDescription(num) {
    var text = "Description for Tab " + num;

    $("#descContainer").text(text);
}
```

After you build the description text, you select the element serving as the description container and set its text using jQuery's `text()` method.

As you can see from this example, jQuery simplifies DOM manipulation and event handling. In this particular example, you wrote less JavaScript to attain the same results. That's well worth the time of learning jQuery, isn't it?

But that's not all; jQuery can do the same thing for your Ajax code.

Using jQuery for Ajax

In Chapter 14, you learned about Ajax and how asynchronous requests require you to write a lot of extra code. You wrote a simple utility to help alleviate the complexity of Ajax code, but jQuery can simplify Ajax even more.

Understanding the jQuery Function

The jQuery function (`$()`) is the doorway into all things jQuery, and you've used it quite a bit throughout this chapter. However, this function has other uses.

Functions are objects and they have a property called `prototype`. Like all other objects, you access a `Function` object's properties and methods using the `object.property` or `object.method()` syntax. Well, jQuery's `$` function has many methods, and some of them are for making Ajax requests. One of them is the `get()` method, which is for making GET requests. The following code shows an example:

```
$.get("textFile.txt");
```

This code makes a GET request to the server for the `textFile.txt` file. But this code isn't very useful because it doesn't do anything with the server's response. So like the `HttpRequest` module you built in Chapter 14, the `$.get()` method lets you define a callback function that handles the response from the server:

```
function handleResponse(data) {
    alert(data);
}

$.get("textFile.txt", handleResponse);
```

This code defines a function called `handleResponse()` and passes it to `$.get()`. jQuery calls this function on a successful request and passes it the requested data (represented by the `data` parameter).

Remember the examples from Chapter 14? You created a form that checked if usernames and e-mail addresses were available using Ajax, and you sent those values to the server as parameters in the URL. For example, when you wanted to test a username, you used the `username` parameter, like this:

```
phpformvalidator.php?username=jmcpeak
```

With the `$.get()` method, you can do the same thing by passing an object containing the key/value pairs to the method. For example:

```
var parms = {
    username = "jmcpeak"
};

function handleResponse(json) {
    var obj = JSON.parse(json);

    // do something with obj
}

$.get("phpformvalidator.php", parms, handleResponse);
```

This code creates a new object called `parms`, and it has a `username` property with the value of `jmcpeak`. This object is passed to the `$.get()` method as the second argument, with the `handleResponse()` callback function passed as the third.

You can send as many parameters as you need; simply add them as properties to the parameter object.

Automatically Parsing JSON Data

In Chapter 14, the form validator PHP file returns the requested data in JSON format, and notice that the previous sample code expects JSON data and parses it with the `JSON.parse()` method. jQuery can eliminate this step and parse the response for you. Simply use the `$.getJSON()` method instead of `$.get()`. For example:

```
var parms = {
    username = "jmcpeak"
};

function handleResponse(obj) {
    // obj is already an object
}

$.getJSON("phpformvalidator.php", parms, handleResponse);
```

This code is almost identical to the previous example except for two things. First, this code uses `$.getJSON()` to issue a request to the PHP file. By doing so, you are expecting JSON-formatted data in the response, and jQuery will automatically parse it into a JavaScript object.

The second difference is inside the `handleResponse()` function. Because the response is automatically parsed, you don't have to call `JSON.parse()` in `handleResponse()`.

The jqXHR Object

As you've seen in the previous sections, jQuery's `get()` and `getJSON()` methods do not actually return the data you requested; they rely upon a callback function that you provide and pass the requested data to it. But these methods do, in fact, return something useful: a special `jqXHR` object.

The `jqXHR` object is called a *deferred* object; it represents a task that hasn't yet completed. When you think about it, an asynchronous Ajax request is a deferred task because it doesn't immediately complete. After you make the initial request, you're left waiting for a response from the server.

jQuery's `jqXHR` object has many methods that represent different stages of a deferred task, but for the sake of this discussion, you'll look at only three. They are:

METHOD NAME	DESCRIPTION
`done()`	Executes when the deferred task successfully completes
`fail()`	Executes when the task fails
`always()`	Always executes, regardless if the task completed or failed

These methods let you add functions to what are called *callback queues*—collections of functions that serve as callbacks for a specified purpose. For example, the `done()` method lets you add functions to the "done" callback queue, and when the deferred action successfully completes, all of the functions in the "done" queue execute.

With this in mind, you can rewrite the previous code like this:

```
var parms = {
    username = "jmcpeak"
};

function handleResponse(obj) {
    // obj is already an object
}

var xhr = $.getJSON("phpformvalidator.php", parms);

xhr.done(handleResponse);
```

Notice that in this code, the `handleResponse()` function isn't passed to the `getJSON()` method; instead, it's added to the "done" queue by passing it to the `jqXHR` object's `done()` method. In most cases, you'd see this code written as follows:

```
$.getJSON("phpformvalidator.php", parms).done(handleResponse);
```

You can also chain these method calls to easily add multiple functions to the callback queues:

```
$.getJSON("phpformvalidator.php", parms)
          .done(handleResponse)
          .done(displaySuccessMessage)
          .fail(displayErrorMessage);
```

In this example, two functions, `handleResponse()` and `displaySuccessMessage()`, are added to the "done" queue; when the Ajax call successfully completes, both of these functions will execute. Additionally, this code adds the `displayErrorMessage()` function to the "fail" queue, and it executes if the Ajax request fails.

Using these callback queue methods does require you to write slightly more code, but they make your code's intentions absolutely clear. Plus, using them is generally accepted as a best practice, and you'll find them used in most modern jQuery-based code that you read.

TRY IT OUT Revisiting the Form Validator

Apply what you've learned and modify the form validator from `ch14_example1.html`. Open your text editor and type the following code:

```
<!DOCTYPE html>

<html lang="en">
<head>
    <title>Chapter 16: Example 2</title>
    <style>
        .fieldname {
            text-align: right;
        }

        .submit {
            text-align: right;
        }
    </style>
</head>
<body>
    <form>
        <table>
            <tr>
                <td class="fieldname">
                    Username:
                </td>
                <td>
                    <input type="text" id="username" />
                </td>
                <td>
                    <a id="usernameAvailability" href="#">Check Availability</a>
                </td>
            </tr>
            <tr>
                <td class="fieldname">
                    Email:
                </td>
                <td>
                    <input type="text" id="email" />
                </td>
                <td>
                    <a id="emailAvailability" href="#">Check Availability</a>
                </td>
            </tr>
            <tr>
                <td class="fieldname">
                    Password:
                </td>
                <td>
```

```
                <input type="text" id="password" />
            </td>
            <td />
        </tr>
        <tr>
            <td class="fieldname">
                Verify Password:
            </td>
            <td>
                <input type="text" id="password2" />
            </td>
            <td />
        </tr>
        <tr>
            <td colspan="2" class="submit">
                <input type="submit" value="Submit" />
            </td>
            <td />
        </tr>
    </table>
</form>
<script src="jquery-2.1.1.min.js"></script>
<script>
    function checkUsername(e) {
        e.preventDefault();

        var userValue = $("#username").val();

        if (!userValue) {
            alert("Please enter a user name to check!");
            return;
        }

        var parms = {
            username: userValue
        };

        $.getJSON("ch14_formvalidator.php", parms).done(handleResponse);
    }

    function checkEmail(e) {
        e.preventDefault();

        var emailValue = $("#email").val();

        if (!emailValue) {
            alert("Please enter an email address to check!");
            return;
        }

        var parms = {
            email: emailValue
        };

        $.getJSON("ch14_formvalidator.php", parms).done(handleResponse);
```

```
        }

        function handleResponse(response) {
            if (response.available) {
                alert(response.searchTerm + " is available!");
            } else {
                alert("We're sorry, but " + response.searchTerm +
                      " is not available.");
            }
        }

        $("#usernameAvailability").on("click", checkUsername);
        $("#emailAvailability").on("click", checkEmail);
    </script>
</body>

</html>
```

Save this as `ch16_example2.html` in your web server's root directory. Like the examples in Chapter 14, this file must be hosted on a web server in order to work correctly. Open your web browser to `http://yourserver/ch16_example2.html`. Type **jmcpeak** into the Username field and click the Check Availability link next to it. You'll see an `alert` box telling you the username is taken.

Now type **someone@xyz.com** in the Email field and click the Check Availability link next to it. Again, you'll be greeted with an `alert` box stating that the e-mail is already in use. Now input your own username and e-mail into these fields and click the appropriate links. Chances are an `alert` box will tell you that your username and/or e-mail is available (the usernames `jmcpeak` and `pwilton` and the e-mails `someone@xyz.com` and `someone@zyx.com` are the only ones used by the application).

The HTML and CSS in this example are identical to `ch14_example1.html`. So, let's dig into the JavaScript starting with the final two lines of code that set up the `click` event listeners on the Check Availability links. You could easily reuse the code from the original, but jQuery makes it a little easier to set up events:

```
$("#usernameAvailability").on("click", checkUsername);
$("#emailAvailability").on("click", checkEmail);
```

You select the elements by their ID and use jQuery's `on()` method to register the `click` event on those elements. Once again, checking the username value is the job of `checkUsername()`, and `checkEmail()` is responsible for checking the e-mail value.

The new `checkUsername()` function is somewhat similar to the original. You start by preventing the default behavior of the event by calling `e.preventDefault()`:

```
function checkUsername(e) {
    e.preventDefault();
```

Next, you need to get the value of the appropriate `<input/>` element. You haven't learned how to retrieve the value of a form control with jQuery, but don't worry—it's very simple:

```
    var userValue = $("#username").val();

    if (!userValue) {
```

```
            alert("Please enter a user name to check!");
            return;
        }
```

You use `$()` to select the appropriate `<input/>` element and call the `val()` method. This retrieves the value of the form control and assigns it to the `userValue` variable.

After you validate the user's input, you're ready to start issuing a GET request to the server. First, you create an object to contain the information you want to send to the server:

```
        var parms = {
            username: userValue
        };
```

You call this object `parms` and populate it with the username property. As you learned earlier in this chapter, jQuery will add this property and its value to the query string.

Now, you can send the request using jQuery's `getJSON()` method:

```
        $.getJSON("ch14_formvalidator.php", parms).done(handleResponse);
    }
```

You add the `handleResponse()` function to the "done" queue, so that when the request successfully completes, an `alert` box will display the search results.

The new `checkEmail()` function is very similar to `checkUsername()`. The two main differences, of course, are the data you retrieve from the form and the data you send to the server:

```
    function checkEmail(e) {
        e.preventDefault();

        var emailValue = $("#email").val();

        if (!emailValue) {
            alert("Please enter an email address to check!");
            return;
        }

        var parms = {
            email: emailValue
        };

        $.getJSON("ch14_formvalidator.php", parms).done(handleResponse);
    }
```

The final function, `handleResponse()`, is mostly unchanged from the original version. Because jQuery's `getJSON()` method automatically parses the response into a JavaScript object, the new `handleResponse()` function simply uses the passed data as-is:

```
    function handleResponse(response) {
        if (response.available) {
            alert(response.searchTerm + " is available!");
        } else {
```

```
            alert("We're sorry, but " + response.searchTerm + " is not available.");
        }
    }
```

jQuery is an extensive framework, and adequately covering the topic in depth requires much more than this chapter can provide. Entire books are devoted to jQuery! However, the jQuery documentation is quite good, and you can view it at `http://docs.jquery.com`. jQuery's website also lists a variety of tutorials, so don't forget to check them out at `http://docs.jquery.com/Tutorials`.

SUMMARY

This chapter introduced you to jQuery, the most popular JavaScript library.

- You learned where and how to obtain jQuery and reference it in your pages.
- You also learned about jQuery's `$()` function, and how it is central to jQuery's functionality.
- jQuery popularized using CSS selectors to find elements within the DOM, and you learned how find elements with the `$()` function.
- You can change element styles with either the `css()` method, or by modifying the CSS classes with the `addClass()`, `removeClass()`, and `toggleClass()` methods.
- Cross-browser events can be a drag when dealing with older browser versions, but jQuery make registering event listeners and working with event data easy (and mostly standards compliant).
- jQuery also simplifies Ajax with its `get()` and `getJSON()` methods, and you learned that `getJSON()` automatically parses the response into a JavaScript object.
- You learned about deferred objects. You also learned about the "done," "fail," and "always" queues, and how you can chain them together to assign multiple handlers to the different queues.

EXERCISES

You can find suggested solutions for these questions in Appendix A.

1. Example 1 is based on Chapter 10's Example 17, and as you probably remember, you modified that example in response to one of Chapter 10's exercise questions. Modify this chapter's Example 1 so that only one tab is active at a time.
2. There is some repetitive code in Example 2. Refactor the code to reduce the duplication. Additionally, add a function to handle any errors that may occur with the request.