

## Introducing jQuery Part 1

This introductory article is aimed at those of you that are completely new to the jQuery JavaScript library. During the course of the article we will look at what the library is, where it came from and who made it, as well as how it can be used in your own web pages. It is the first part in a series, with successive articles exposing each of the different core components that make up the library. Over the course of the series we'll be looking at the advanced events created by the library, managing CSS effectively, and looking in detail at the AJAX aspect of the library.

### *What is jQuery?*

The jQuery library, referred to from here on simply as jQuery, is an extremely popular and ludicrously easy to use JavaScript library originally written by John Resig. The initial release came on 14 January 2006 and since this time it has continued to be modified and refined by its creator and a growing community of contributors. The current release is version 1.2.1.

The library is open-source and completely free of charge. It is dual licensed under the MIT and GPL licensing schemes making you free to use, modify and redistribute the library in its existing, or any derivative form. More information about the MIT license can be found on the [OSI site](#). An excellent definition of the GPL License can be found on [Wikipedia](#).

jQuery is primarily an element selection and DOM manipulation library. It supports the latest CSS3 and XPath selectors which enable you to get practically any element from a page. Once you have an element, there are many additional methods defined by the library that enable you to manipulate the element in any way you can imagine.

Incredible features of the library enable:

- Easy DOM traversal; you can get a <p> element and then walk the DOM to its last-child, parent or next-sibling for example.
- Effortless element manipulation; you can get or set the HTML or plain text content of any element for example, or remove, replace, copy or insert any number of elements.
- Complete mastery of the CSS styles of any element; jQuery lets you get or set any CSS value on any element or series of elements.
- The detection or firing of standard or custom events; detect the click event or any element, for example, or trigger the click event on an element, even when the visitor hasn't actually clicked the element in question.
- A rich effect portfolio; show or hide, fade-in or fade-out, slide, or animate any element on the page, quickly and easily.
- A unified AJAX request and response system; no more `if (browser == ie) { //do something } else { //do the same thing differently }`. jQuery manages the request and response via an easy to use and practical interface with built in success and failure handlers.
- A set of basic utilities for determining the browser and version in use, and for working with arrays more effectively among other things.

Since September 2007, an additional component of the library has also been available - the UI suite; this is a set of pre-packaged widgets that can be dropped on a page and put to work with very little additional configuration. These include visual controls such as a DatePicker or Accordion pane, and behaviour-based widgets like draggables and sortables.

This UI suite forms an additional layer, built on top of the existing library framework. Whereas the standard jQuery library is compacted into one minified JavaScript file, the UI components each have their own separate JavaScript files and core dependencies, allowing you to easily pick and choose which ones to use.

## Using jQuery

Of all of the many JavaScript libraries available, jQuery is without doubt one of the easiest. It has an easy and familiar syntax because a lot of the element selection techniques are used in the same way as CSS selectors.

Let's look at a basic example of how easy the fundamental aspects of jQuery are to master. In a text editor, begin with the following HTML page:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html lang="en">
  <head>
    <title>jQuery Element Selection Example</title>
    <script type="text/javascript" src="jquery/jquery-1.2.1.min.js"></script>
  </head>
  <body>
    <div id="div1" class="standardDiv">This is div 1</div>
    <div id="div2" class="boringDiv">This is div 2</div>
    <div id="div3" class="specialDiv">This is div 3</div>
  </body>
</html>
```

So we have a fairly innocuous page containing three standard `<div>` elements. Each has a couple of attributes and some plain text content. Nothing is styled, even though the divs each have a class. Let's now see what we can do with a little jQuery magic. In the `<head>` section of the page, just after the existing `<script>` tags add the following code:

```
<script type="text/javascript">
  $(document).ready(function() {
    var theText = $("#div1").text();
    alert(theText);
  });
</script>
```

There are two components to the simple script above; the first line of code `$(document).ready` allows us to specify an anonymous function that is executed when the DOM is ready to be manipulated (i.e. has finished loading) and is analogous (but much neater) than an `onload` event handler. `$(document)` allows us to create a special jQuery object that targets the entire document as opposed to individual elements.

The code within the curly braces of the anonymous function, which is executed once when the page has loaded, allows us to select our target element. Using jQuery revolves around this style of syntax and anonymous functions abound in any jQuery specific code.

We use `#div1` to obtain the element that has the `id` attribute `div1` in exactly the same way that we would write a CSS selector to target the same element. The element is returned not as a basic `HTMLElement`, but as a jQuery object. Once we have a jQuery object, we can call other jQuery methods on it, like the `.text()` method, which allows us to obtain the text within our div. We can then do whatever we need to do with the text, which in this case is to simply alert it.

The world of jQuery is composed of special jQuery objects created by the library. jQuery methods can only be called on jQuery objects, so instead of using `var someEl = document.getElementById("someid")` to create a variable based element reference, we use `$("#someid")` which creates and returns a jQuery object.

In the same way that we targeted the `id` of the element we wanted to select in the above example, we can also target other aspects of elements, the following code would allow us to select an element add it to a jQuery object based on the element's class:

```
var theText2 = $(".boringDiv").text();  
alert(theText2);
```

This time, we've used a period to specify the class attribute in order to select the desired element. Each jQuery object we've created so far has only matched, and therefore only consisted of one element. It is just as easy to populate our jQuery object with a collection of objects as opposed to a single element:

```
alert($(".div").size());
```

This single line of code alerts the size of the jQuery `$(".div")` object; there are three `div` elements on our page so the alert should simply read 3. The `.size()` method of jQuery is exactly the same as calling `.length()` on a standard JavaScript array. As well as the simple selection methods looked at so far, specificity can also be used; so instead of getting all elements with a given class, we can narrow this down to all `<p>` elements with a given class, for example, by using `$(".p.givenClass")`.

Another important, and highly useful, feature of the library is known as chainability, which allows you to chain commands one after the other, without having to reselect elements. In our example so far, we've only called one method on each jQuery object, but as nearly every method returns a modified (by the actions of whichever method was called) object, we can continue to interact with it and manipulate for as long as we want or need. Add the following new code:

```
$(".specialDiv").text("I am the special div!").css({backgroundColor:"lightBlue",  
border:"1px solid black"});
```

We first get a jQuery object that contains the element with the class 'specialDiv'. Then, instead of getting the textual contents, we set it by supplying a new string to be used as an argument of the `.text()` method. This method will apply our new string, and then return the modified jQuery object.

We then apply the `.css()` method to the object, which allows us to set some CSS properties of the element, in this case we can add a background colour and a border. The CSS rules are specified as a literal object within the `.css()` method. Each rule consists of a key:value pair with the style property as a camel-cased key and the value enclosed in quotes. Notice how we can even use the short-hand method of specifying border properties.

The `.css()` method has been chained on to the end of the `.text()` method, which is a highly useful feature of the library. We needn't stop there either; we could go on to chain almost any number of methods on to our jQuery object and alter or adjust it until it met our requirements.

One final thing that we can look at in this part of the tutorial is the library's element creation process. In the same way that most methods can either get or set a property, depending on whether or not data is supplied as an argument, elements can either be obtained or created from scratch by the library. So far our jQuery objects have all been populated with elements from the page, let's now create a brand new `<div>` element and insert it into the page:

```
$("#<div>").attr("id", "div4").addClass("freshDiv").text("I'm a freshly baked div, just  
out of the jQuery oven!").insertAfter("#div3");
```

Again, we can make the most of jQuery's chainability to cut down drastically on the amount of code that we need to write. Just to create manually an element the old fashioned way, you'd need to call several JavaScript methods, and then to set the class and id attributes, you'd need a few more, etc, etc. With jQuery it takes just a simple `$( "<div>" )` to create the jQuery object.

Because we are supplying an actual element instead of an id, class or element name, the library will create the div instead of looking for it in the existing DOM. After we've added the attributes and set the text content of the new element, the highly useful `.insertAfter( )` method allows us to specify an element that the new element should be added after. If you view the page in a browser now, the new div element should be proudly displayed, with the text content and attributes that we have set.

## **Summary**

Using jQuery can empower you with an effective set of web design tools that means that you can do everything that you need to do, but with much less code. jQuery is renowned for how much it can achieve with so little additional coding from you, the developer.

Over the course of this part of the series, we've looked at some of the most basic and fundamental concepts of the library including the most important aspect of all which is the creation of the jQuery object. We've seen how this jQuery object can be populated with elements from the existing DOM, or with newly created elements of our own design.

We've also seen how methods can be chained onto each other in succession so that a range of properties can be obtained about the selected elements, or how these same properties can be set with arbitrary values. These are some of the most fundamental aspects of the library and once you have mastered these, it is simply a case of learning how to use each of the available methods for maximum effect.

The library is compatible with most modern, capable, standards-supporting browsers, as well as Internet Explorer, which means even less code that you need to write. Most methods perform equally as well on any browser, and for the odd occasion when there is something that has to be coded differently to cater for different browsers, you can make use of the library's advanced browser detection to help you. Join me in part two, where we will be looking at the rich event portfolio created by the library.