

## Introducing jQuery Part 3

Welcome to Part 3 of the jQuery introductory series; during the course of this part we'll be looking in detail at the range of CSS tools that come with jQuery. While other components provide essential behaviour for your pages, CSS provides the decoration that adds personality to your pages.

The CSS component allows you to get or set almost any CSS property of any element on the page, and provides special methods for working with the heights and widths of specified elements. It also allows you to determine the specific location, relative to the viewport, of any element.

### Getting the CSS

Due to the differences in support and implementation of CSS between the major browser vendors, it can sometimes be a pain to work with CSS in a coherent and compatible way; more often than not, what will work in one browser simply won't in another, and you end up with a script file stuffed full of hacks, work-arounds and user-agent detection routines.

Thankfully, jQuery puts an end to most of the different browser inconsistencies, and allows you to do more with CSS whilst doing less with JavaScript. Let's take a look at a brief example so that you can see exactly how easy it is to get and set various CSS values. In a new page in your text editor, create the following basic page:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html lang="en">
  <head>
    <title>jQuery CSS Example</title>
    <script type="text/javascript" src="jquery/jquery-1.2.1.min.js"></script>
  </head>
  <body>
    <div id="div1" class="div"><p>This is div 1</p></div>
    <button id="button1" class="button">Get the CSS of div 1</button>
  </body>
</html>
```

Our test page has nothing but a simple `<div>` element with some inner text on it, but it should be enough to highlight how easy the `.get()` method is to work with. Save this page as **example6.html** and make sure you have a copy of the library in the same folder. In order to get the CSS properties of an element, that element must of course have some CSS properties. We can give our test element a few basic style properties so that we have something to work with.

Normally all CSS would go into its own file (as would JavaScript), but to keep this example as compact (and as relevant) as possible, we can just put everything onto the same page. Add the following `<style>` tag to the `<head>` of the page:

```
<style type="text/css">
  .div {
    width:200px;
    height:50px;
    border:2px solid black;
    background-color:lightblue;
    color:#ffffff;
    text-align:center;
    margin-bottom:10px;
```

```
font-weight:bold;
}
.button {
width:205px;
margin-bottom:5px;
}
</style>
```

Now that we have some CSS we can use jQuery to find out what these properties are (even though we know what they are, having just written them). In the **<head>** of the page, add the following **<script>** tag:

```
<script type="text/javascript">
$(function() {
$("#button1").click(function() {
var border = $("#div1").css("border");
var bgColor = $("#div1").css("backgroundColor");
var color = $("#div1").css("color");
var tAlign = $("#div1").css("textAlign");
var mBottom = $("#div1").css("margin-bottom");
var fWeight = $("#div1").css("font-weight");
var styles = " border: " + border + "\n background-color: " + bgColor + "\n color: " +
color + "\n text-align : " + tAlign + "\n margin-bottom: " + mBottom + "\n font-weight:
" + fWeight;
alert(styles);
});
});
</script>
```

The script is very simple; it uses the click event of the button to initiate the gathering of our CSS properties. We create a new variable, then use the **\$("#div1")** jQuery object and the **.css()** method to obtain each property. The property that we want to get is then specified as an argument of the method. When you run the page in a browser and click the button, you should see something like in Figure 1 below:

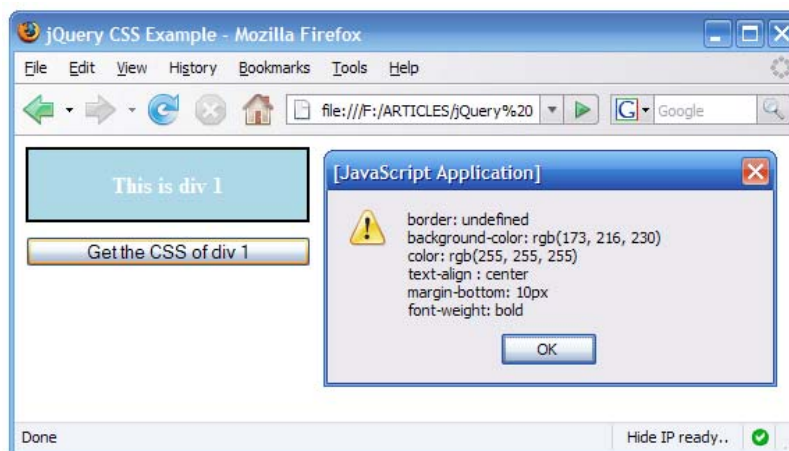


Figure 1 – Styles Alerted

You'll notice straight away that the **border** property that we tried to obtain has returned **undefined**. This is because we used the short-hand way of addressing the different border properties in our JavaScript, and is an important lesson in working with CSS and jQuery. Change the script so that the full properties are used:

```
$(function() {
$("#button1").click(function() {
```

```

var borderWidth = $("#div1").css("borderBottomWidth");
var borderCol = $("#div1").css("borderBottomColor");
var borderStyle = $("#div1").css("borderBottomStyle");
var bgColor = $("#div1").css("backgroundColor");
var color = $("#div1").css("color");
var tAlign = $("#div1").css("textAlign");
var mBottom = $("#div1").css("marginBottom");
var fWeight = $("#div1").css("fontWeight");
var styles = " border-width: " + borderWidth + "\n border-color: " + borderCol + "\n
border-style: " + borderStyle + "\n background-color: " + bgColor + "\n color: " +
color + "\n text-align : " + tAlign + "\n margin-bottom: " + mBottom + "\n font-weight:
" + fWeight;
alert(styles);
});
});

```

Now when the alert is produced, it should contain the correct values, as in Figure 2 below:

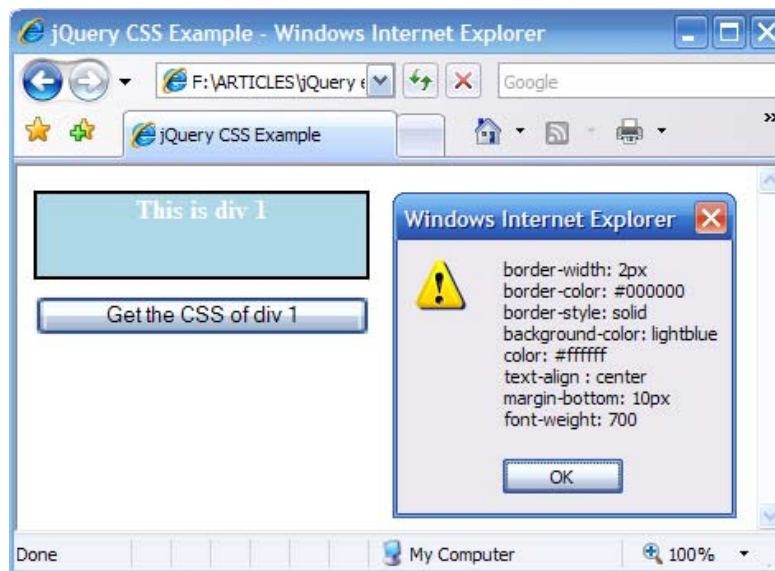


Figure 2 – the Correct Border Properties

You may also notice another difference between the two browsers; FireFox uses the RGB system for translating the colours, whereas IE uses hexadecimal. It's not a major setback, but it is something to be aware of.

## Setting the CSS

We already had a quick look at using the `.css()` method of jQuery to set the value of CSS properties back in Part 2. We only set one property at a time using this method before, so we'll adapt the example from just a minute ago in order to set multiple styles simultaneously.

Let's add some input fields and another button to the page; you can then enter the styles that you want to set into the text fields and then use a button to apply them. Change the HTML for the `<body>` of the page so that it appears as follows:

```

<body>
<div id="div1" class="div"><p>This is div 1</p></div>
<button id="button1" class="button">Get the CSS of div 1</button><br>
<input type="text" id="borderWidth"> New border-width<br>

```

```
<input type="text" id="borderColor"> New border-color<br>
<input type="text" id="borderStyle"> New border-style<br>
<input type="text" id="backgroundColor"> New background-color<br>
<input type="text" id="color"> New color<br>
<input type="text" id="textAlign"> New text-align<br>
<input type="text" id="marginBottom"> New margin-bottom<br>
<input type="text" id="fontWeight"> New font-weight<br>
<button id="button2" class="button">Set the CSS of div 1</button>
</body>
```

Now let's add the additional JavaScript required for making everything work right. Directly after the `.click()` function for `button1`, add the following code:

```
$("#button2").click(function() {
  var newBW = $("#borderWidth").attr("value");
  var newBC = $("#borderColor").attr("value");
  var newBS = $("#borderStlye").attr("value");
  var newBackC = $("#backgroundColor").attr("value");
  var newc = $("#color").attr("value");
  var newTA = $("#textAlign").attr("value");
  var newMB = $("#marginBottom").attr("value");
  var newFW = $("#fontWeight").attr("value");
  $("#div1").css({
    borderWidth:newBW,
    borderColor:newBC,
    borderStyle:newBS,
    backgroundColor:newBackC,
    color:newc,
    textAlign:newTA,
    marginBottom:newMB,
    fontWeight:newFW
  });
});
```

The second button has its own anonymous function that is executed on its click event; it simply obtains the value of each text field using the `.attr()` method, and then uses a literal object of values to pass into the `.css()` method. Each member of the object relates to one style rule and is a **key:value** pair of the CSS property and value.

FireFox and IE have subtle differences when setting CSS as well as getting them; you'll notice that in FireFox when specifying widths or other length properties, you need to enter 'px' as well as a numerical value into the text fields but in IE, you can just specify the new value. When using colours, you can use the colour name rather than having to worry about entering the hexadecimal or RGB value.

Setting the border properties, unlike getting them, can be achieved by targeting all four sides of the element at once, instead of having to target each side specifically. You'll notice that it's not possible to set the border style using this method. Each of the other properties should work fine, as shown in Figure 3 on the next page:

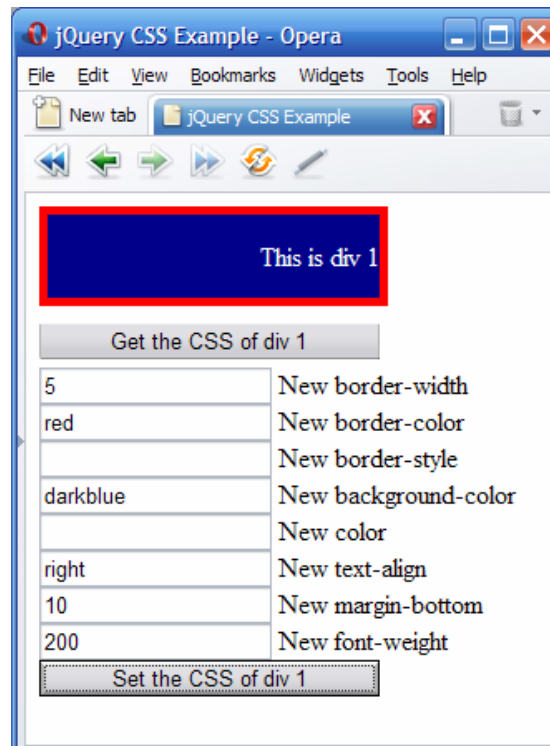


Figure 3 – Setting Styles

## Heights & Widths

We haven't looked at heights and widths yet, jQuery provides separate methods for getting and setting the dimensions of on-page elements. All four of the dimensional methods are extremely easy to use and follow a similar format as the CSS methods. Let's extend our example page so that we can set the height and width of our `<div>`. Add two more text fields and two more buttons to the page:

```
<button id="button3" class="button">Get the dimensions</button><br>
<input type="text" id="width"> New width<br>
<input type="text" id="height"> New height<br>
<button id="button3" class="button">Set the dimensions</button>
```

Now let's add the first of our new anonymous functions to the JavaScript section in the `<head>` of the page:

```
$("#button3").click(function() {
    alert("The width is: " + $("#div1").width() + "\n the height is: " +
    $("#div1").height());
});
```

When button three is clicked, you should get an alert revealing the width and height of the element, just like in Figure 4 on the next page:

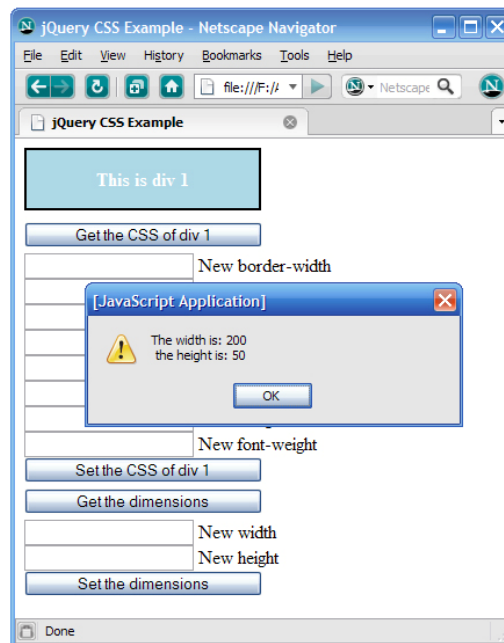


Figure 4 – Getting the Width and Height

If no value is specified in the `.width()` and `.height()` methods, the properties are gotten from the specified element. We can set these two properties by specifying values in exactly the same methods:

```
$("#button4").click(function() {
  var newW = $("#width").attr("value");
  var newH = $("#height").attr("value");
  $("#div1").width(newW).height(newH);
});
```

The height and width methods both return the jQuery object so it is possible to chain the two methods one after the other. When getting the properties, the method returns the property it has obtained, so the methods cannot be chained in these situations. When using the set dimensions button in gecko-based browsers, you must still specify 'px' in the text fields. Figure 5 below shows the new code in action:

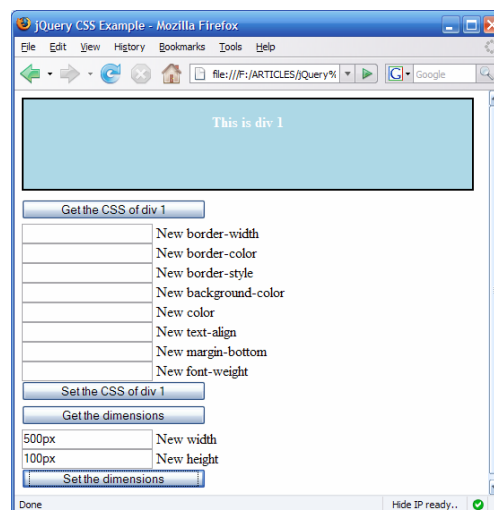


Figure 5 – Setting the Width and Height

## Working with the offset

The last remaining aspect of the CSS suite that we haven't looked at yet is the offset; the offset refers to the location of a specified element relative to the viewport. The viewport, for those of you that haven't come across this term before, is simply the visible portion of the current page in the browser.

The `.offset()` method can only be used to retrieve the offset; it cannot, as yet, be used to move elements around the page by specifying new offset properties (although you could always manipulate the `top` and `left` properties using the `.css()` method of course).

Like when retrieving properties using the `.css()` method, the `.offset()` method doesn't return a jQuery object, so no further methods can be chained upon it. It still returns an object however, and its members correspond to the `top` and `left` properties of the specified element.

We'll create a new page for this next example; in a new file in your text editor, add the following basic 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 CSS Example</title>
    <script type="text/javascript" src="jquery/jquery-1.2.1.min.js"></script>
    <style type="text/css">
      #div1 {
        position:absolute;
        left:100px;
        top:100px;
        width:50px;
        height:50px;
        background-color:red;
      }
    </style>
  </head>
  <body>
    <button id="button1">Get the offset</button>
    <div id="div1"></div>
  </body>
</html>
```

Save this page as `example7.html` in the same folder as the previous example. Now we can wire up the button so that it alerts the current offset of the `<div>` whenever it is clicked. Add the following `<script>` tag to the `<head>` of the page:

```
<script type="text/javascript">
$(function() {
  $("#button1").click(function() {
    o = $("#div1").offset();
    alert("The offset of div1 is: " + "\n left: " + o.left + "\n top: " + o.top);
  });
});
</script>
```

When the button is clicked, the `o` variable is created and the object returned by the `.offset()` method is stored in it. We can then examine this object's members, `left` and `top`, to find the values we are looking for. When the button is clicked, you should get the alert shown in Figure 6 below:

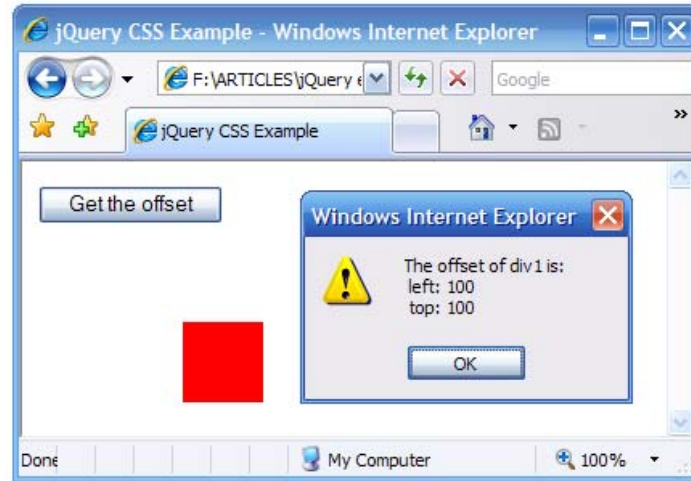


Figure 6 – The Offset at Work

## Summary

The CSS section of jQuery provides an interface through which the properties of objects on the page can be obtained or changed in a fairly consistent and mostly cross-browser manner. Although each browser typically displays some kind of quirkiness when working with CSS the jQuery way, this is a massive improvement on the often 'not working at all' way of working with CSS which often occurs when using standard JavaScript.

In the next part of this introductory series, we'll take a look at another important section that can add visual appeal to your site; this is of course the effects interface which can be used to create a range of stunning transition effects to apply to elements on the page.