

## Introducing jQuery Part 4

Welcome to Part 4 of the introductory jQuery series; in the last part we looked at the CSS manipulating tools available to us via the library. In this part we're going to look at the Effects section; this part of the library can really add visual impact and appeal to your pages.

There are four categories of effects within the Effects section of the library; the first category deals with the simple showing and hiding of elements, the next two sections are devoted to sliding and fading transitions, while the final category enables you to create custom animations yourself, and also manages the effects queue.

Throughout this tutorial we'll be looking at what can be achieved using each of the effect methods exposed by this section of the library. The code we'll be playing with has been tested extensively in the most popular browsers available, but only on the Windows platform.

### Showing & Hiding Elements

The basic effects of the first category are extremely easy to use. Like other methods defined by the library, the `.show()` and `.hide()` methods can be used with or without arguments. In their most basic invocation they simply show or hide the specified element. Let's look at these methods in action. In your text editor, begin with 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 Effect Example</title>
    <script type="text/javascript" src="jquery/jquery-1.2.1.min.js"></script>
  </head>
  <body>
    <button id="show">Show</button><button id="hide">Hide</button>
    <div id="div1">I am div 1!</div>
  </body>
</html>
```

Next, add the following `<style>` tag to the `<head>` of the page:

```
<style type="text/css">
div {
border:2px solid black;
margin-top:10px;
background-color:lightblue;
color:white;
padding:40px 10px 0px 20px;
width:80px;
height:60px;
}
</style>
```

Normally CSS would always go into its own file but for the purpose of this example it can just sit in the head of the page. Now let's wire up those buttons with some jQuery JavaScript. Also in the head of the page, either before or after the CSS, add the code shown on the next page:

```
<script type="text/javascript">
$(function() {
$("#hide").click(function() {
$("#div1").hide();
});
$("#show").click(function() {
$("#div1").show();
});
});
});
</script>
```

The script is very simple, thanks to jQuery; the `.show()` and `.hide()` do exactly what they say which is simply to display or hide the specified element. Like many other methods of the library, these two can be passed arguments in order to alter what it is they do. These methods can take up to two optional arguments; the first can be either an integer or string that represents the speed at which the showing or hiding of the element is carried out, and the second argument is a reference to a callback function that can be called during the animation. Using these additional arguments is incredibly easy; change your JavaScript so that it appears as follows (new code is shown in bold):

```
<script type="text/javascript">
$(function() {
var myCallback = function(type) {
  alert("The div will be " + type);
}
$("#hide").click(function() {
$("#div1").hide("slow", myCallback("hidden"));
});
$("#show").click(function() {
$("#div1").show("slow", myCallback("shown"));
});
});
});
</script>
```

This time, when the animation occurs it should take longer; instead of simply disappearing, the `<div>` should shrink until it exists no more. You'll notice that the callback function `myCallback()` is executed *before* the animation takes place; this is because of how the function is constructed. We're passing an argument to the function, a simple string, like a traditional function instead of a callback.

The `.toggle()` method is like a combination of the `.show()` and `.hide()` methods; the specified element will either be shown or hidden, depending on whether it is currently visible or not:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html lang="en">
<head>
<title>jQuery Effect Example</title>
<script type="text/javascript" src="jquery/jquery-1.2.1.min.js"></script>
<script type="text/javascript">
$(function() {
$("#showHide").click(function() {
$("#div1").toggle();
});
});
</script>
<style type="text/css">
div {
```

```
border:2px solid black;
margin-top:10px;
background-color:lightblue;
color:white;
padding:40px 10px 0px 20px;
width:80px;
height:60px;
}
</style>
</head>
<body>
<button id="showHide">Show/Hide</button>
<div id="div1">I am div 1!</div>
</body>
</html>
```

## Fading and Sliding

The fading effect of jQuery is a natural progression from simple showing and hiding. The `.fadeIn()` and `.fadeOut()` methods are very similar to the methods for showing and hiding, but the opacity of the specified element is adjusted rather than the display property. To illustrate the fade methods, we can use 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 Effect Example</title>
<script type="text/javascript" src="jquery/jquery-1.2.1.min.js"></script>
<script type="text/javascript">
$(function() {
$("#show").click(function() {
$("#div1").fadeIn("slow");
});
$("#hide").click(function() {
$("#div1").fadeOut("slow");
});
});
</script>
<style type="text/css">
div {
border:2px solid black;
margin-top:10px;
background-color:lightblue;
color:white;
padding:40px 10px 0px 20px;
width:80px;
height:60px;
}
</style>
</head>
<body>
<button id="show">Show</button><button id="hide">Hide</button>
<div id="div1">I am div 1!</div>
</body>
</html>
```

It's a neat effect and in some situations will be much better than simply making an element appear or disappear. Like the `.show()` and `.hide()` methods, an optional callback function can also be passed into the methods as an argument. As well as `.fadeIn()` and `.fadeOut()`, you can also make use of the `.fadeTo()` method, which fades the specified element to the specified opacity. Change the function associated with the `.hide()` method in the previous example so that it appears as follows:

```
<script type="text/javascript">
$(function() {
$("#show").click(function() {
$("#div1").fadeIn("slow");
});
$("#hide").click(function() {
$("#div1").fadeTo("slow", 0.25);
});
});
</script>
```

This time, instead of being completely hidden, the `<div>` element's opacity is adjusted so that it ends up a quarter of its starting opacity. Now let's move on to element sliding with the `.slideUp()` and `.slideDown()` methods. When using these methods, the element's opacity stays the same, it is its height that changes; the element either grows into or shrinks out of view, which is great effect for floating panels that function like dialog boxes. We can illuminate the workings of these methods with the following simple page:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html lang="en">
<head>
<title>jQuery Effect Example</title>
<script type="text/javascript" src="jquery/jquery-1.2.1.min.js"></script>
<script type="text/javascript">
$(function() {
$("#show").click(function() {
$("#div1").slideDown("slow");
});
$("#hide").click(function() {
$("#div1").slideUp("slow");
});
});
</script>
<style type="text/css">
div {
border:2px solid black;
margin-top:10px;
background-color:lightblue;
width:100px;
height:100px;
display:none;
}
</style>
</head>
<body>
<button id="show">Show</button><button id="hide">Hide</button>
<div id="div1"></div>
</body>
</html>
```

Like the other effects, we can specify the speed at which the animation is to occur, as well as an optional callback function. Note that padding has an effect on the height of the element that the slide methods are called on; if an element has 20 pixels of padding on the top and bottom and the `.slideUp()` method is called on it, the element will slide up to until it is 40 pixels high and then disappear.

Here is also another method related to sliding; the `.slideToggle()` method works exactly in the same way as the `.toggle()` method that we looked at earlier. This method will either slide the element up, or it will slide it down, depending on the element's initial condition. It supports the same arguments as the standard slide methods.

## Custom Animations

Perhaps the most powerful set of methods in the effects section of the library are those dedicated to allowing you to create your very own custom animations. There are four methods for us to make use of in this section; two of them are devoted to creating and controlling your custom animations, while the remaining two enable you to effectively manage a queue of animations.

The `.animate()` method is used in the construction of custom animations and can accept a range of different arguments. The first technique involves four arguments, three of which are optional. The first argument, which is required, is a literal object containing members which define what the animation does to the matched element(s). The other arguments are all optional and consist of the duration of the animation, an easing effect (although a plug-in is required for this to work) and a callback function to execute.

The `.animate()` method can also be used in conjunction with two arguments; the first argument is again the required parameters object which specifies what should be animated. The second, also optional argument to be used is another object literal. The second object consists of members relating to a series of options, such as the duration, easing, callback and queue. Let's take a look at the `.animate()` method in action:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html lang="en">
  <head>
    <title>jQuery Effect Example</title>
    <script type="text/javascript" src="jquery/jquery-1.2.1.min.js"></script>
    <script type="text/javascript">
      var myCallBack = function() {
        alert("Animation Complete");
      }
      $(function() {
        $("#anim").click(function() {
          $("#div1").animate({
            left:"200px",
            top:"200px"
          }, "fast", myCallBack)
        });
      });
    </script>
    <style type="text/css">
      #div1 {
        border:2px solid red;
        position:absolute;
        height:70px;
        width:70px;
        left:10px;
        top:50px;
      }
    </style>
  </head>
  <body>
    <div id="anim">
      <div id="div1">
      </div>
    </div>
  </body>
</html>
```

```
</style>
</head>
<body>
<button id="anim">Animate</button>
<div id="div1"></div>
</body>
</html>
```

In this example, our callback function is called without the parenthesis, making it behave like the callback it is supposed to be. This time, the alert will appear after the animation has completed. All we do is move the `<div>` by 200 pixels along both the x and y axis, which is specified using an object literal with two members; left and top. We make use of the duration argument to specify that the animation should occur quickly, and the callback argument to specify our callback function. Let's rework this example so that we make use of the second object literal instead of the extra arguments. Change the `<script>` section of the last example so that it appears as follows instead:

```
<script type="text/javascript">
var myCallBack = function() {
alert("Animation Complete");
}
$(function() {
$("#anim").click(function() {
$("#div1").animate({
left:"200px",
top:"200px"
}, {
duration:"fast",
complete:myCallBack
});
});
});
</script>
```

The script will work in exactly the same way, but this time, each of the arguments that we used are defined in an object literal format. In this basic example it doesn't add much value, but when it comes to defining multiple animations that occur on the same, or a number of elements, this format can improve performance and tidy up your code considerably.

The `.stop()` method is a simple way of stopping animations that are no longer required. No arguments are used with this method; it is simply called on an element that you wish to stop any currently running animations on. Add a new button to the most recent page and then update the `<script>` block so that it appears as follows instead:

```
<script type="text/javascript">
var myCallBack = function() {
alert("Animation Complete");
}
$(function() {
$("#anim").click(function() {
$("#div1").animate({
left:"200px",
top:"200px"
}, {
duration:"slow",
complete:myCallBack
});
});
});
```

```
});  
$("#stop").click(function() {  
  $("#div1").stop();  
})  
});  
</script>
```

You might want to set the custom animation so that it runs slowly (giving you enough time to end the animation using the stop button). When the animation is stopped using the button, the callback function does not get executed. Our final part of call in this article will be a quick look at the queue management methods exposed by the library.

## Form an Orderly Queue

There are two methods related to the animation queue, these are the **.queue()** and **.dequeue()** methods. The **.dequeue()** method is simple and analogous (in use) to the **.stop()** function; no arguments are passed to this method and it performs just one function which is to remove the specified function from the queue and execute it once.

The **.queue()** method is more complex; this method can use a variety of arguments in order to subtly alter its behaviour. When used without specifying an argument, this method simply returns a reference to the queue in the form of an array. As arguments, you can either specify a callback function, which is added as the last item in the queue array, or you can supply a new array which will replace the existing queue array. The queue is formed when multiple animations (defined using chained individual **.animate()** methods) are created for a single element.

## Summary

Using the effects section of the jQuery library is just as easy as using any of the other sections and can quickly and easily add flair and style to any page. You have a fine degree of control over the built-in animations and can even extend the library to create your own custom animations. In the next and final part of this introductory series on using jQuery, we will look at the section of the library that is dedicated to AJAX.