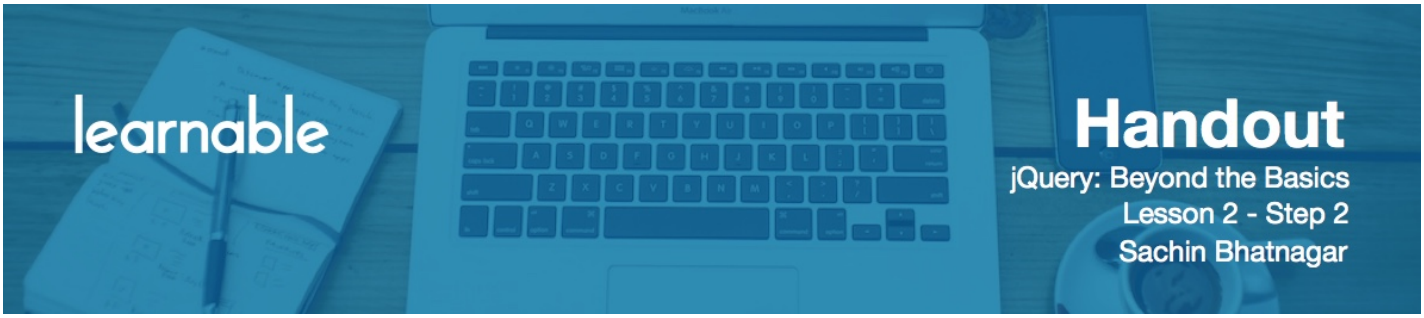

Introduction to element visibility

When working with dynamic user interfaces, in many cases we want to control the visibility of elements such as buttons, images, and other status indicators and widgets. Changes made to the visibility of an element can have a major effect on the user's perception about the status of the feature that it represents. Good design practices, however, dictate that a sudden change in an elements' visibility, such as as a button that is instantly made invisible, might either go unnoticed or produce a jarring uncomfortable effect on the users perception. Therefore, modern web apps and sites employ subtle animation effects when changing element visibility states.

While all of this can be achieved with CSS3 rather efficiently, there are many scenarios when you want to change visibility with JavaScript based on some internal processing. You might also want more control over the animated effect, such as knowing when the animation has ended.

Techniques of animating element visibility with jQuery

Fortunately for us, jQuery comes fully equipped with handy functions to animate the visibility of elements. Also jQuery, especially the 1.X versions, enable older browsers (even Internet Explorer 6) to have animated effects.

Understanding fade in and fade out of elements

Let's now try and examine visibility animation techniques one by one.

Fading in an element essentially means bringing it from a hidden state, which can be done by setting its CSS `display` property to `none`. During the animation phase the `opacity` property of the element is animated from zero to one. At the same time, its `display` property is changed from `none` to typically either `block` or `list-item`, depending on the type of the element. Note that for an element to be fitted into view, its `display` property needs to be set to `none` before jQuery can fill it in.

So we have the `fadeIn` shorthand method, which can be simply attached to an element that needs to be faded in, and given a duration in milliseconds over which the fade will take place. An optional **callback function** can also be provided - this function will fire when the animation ends.

```
$(<selector>).fadeIn(2000, function() {});
```

When **fading out** an element, its `opacity` is animated from one to zero, and its `display` property is set to `none` at the end of the animation. jQuery sets the `display` property to `none` because this way the element stops occupying space in the layout. If we just animate the `opacity` to zero, the element still continues to block the space where it sits in the visible layout.

Just like the `fadeIn` method, the `fadeOut` method does precisely what you would expect. It fades the element out from the layout.

```
$(<selector>).fadeOut(2000, function() {});
```

Working with fade in/out demo:

Download the included zip archive and extract it into a folder within your web server's root. I am using an Apache web server, but you're free to use whatever you want. By serving these files from a web server, you'll ensure that jQuery loads up properly since we are using protocol relative URLs.

Now, open the *fadeBasics_Begin.htm* file. We have a small user interface with three anchor tags that resemble buttons. These buttons should either fade in an element or fade out, or just toggle the element's fade status. The element that we will be fading in and out is a `div` which contains

a background image. What we also want to do is change the color of the small status indicator to green when the element fades into view, and red when the element is faded out.

The first thing I should point out is that I'm using the 2.X branch of jQuery, more specifically, version 2.1.3, which is the most recent one at the time of recording the screencast. Do note that the 2.X branch of the jQuery does not support all the versions of Internet Explorer. So if you want support for those browser versions, you might want to consider using the 1.X branch instead.

The `div` with the class name of `bigBox` is what we will play with. Additionally, we have a `div` that acts like a status indicator. We'll add a class, which will set its background color to green when we fade in the `bigBox`. And we'll remove the class to revert it to red when the `div` is faded out. Let's fade in the `bigBox` first, since its `display` is set to `none` by default.

```
$('.clickToFadeIn').on('click', function() {
    $('.bigBox').fadeIn(2000, function() {
        $('.status').addClass('green');
    });
});
```

Refresh the page and check it out.

Next, let's write our `fadeOut` method:

```
$('.clickToFadeOut').on('click', function() {
    $('.bigBox').fadeOut(2000, function() {
        $('.status').removeClass('green');
    });
});
```

Finally, let's use the `fadeToggle` method to toggle the fade status of the `bigBox`:

```
$('.toggleFade').on('click', function() {
    $('.bigBox').fadeToggle(2000, function() {
        $('.status').toggleClass('green');
    });
});
```

Fade with options

Both `fadeIn` and `fadeOut` provide a number of additional options. To use them, you have to provide the JavaScript object. For instance, instead of directly typing in the duration as an argument you may provide the corresponding option:

```
$(<selector>).fadeIn({
    duration: 2000
});
```

Another very cool option is the `step` handler. It allows you to define a function that will be executed at every step during the fade animation. As the value of opacity and emits from zero to one or one to zero at every step, its value can be extracted within the `step` function by using the `now` variable:

```
$(<selector>).fadeIn({
    step: function(now) {}
});
```

You can also set the `easing` property which controls the speed ramp of the fade animation. The default `swing` method gently begins the animation from zero and then accelerates to normal speed before decelerating down towards the end. You may choose to use `linear` instead, which basically runs the animation at a constant speed:

```
$(<selector>).fadeIn({
    easing: 'linear'
});
```

While jQuery natively provides only `linear` and `swing`, additional easing functions can be brought in by using third party plugins as we'll see later

in this course.

The `complete` handler runs when the animation is complete:

```
$(<selector>).fadeIn({
    complete: function() {}
});
```

Working with fade options demo

Open the *fadeOptions_Begin.htm* file. The interface on this demo page looks similar to the one we had earlier, however, there is an HTML5 progress bar element as well. What we want to do is animate the progress bar as the element fades in or out. For this, we'll use the `step` handler.

Let's write the `fadeIn` method, but this time, instead of directly passing in the duration as an argument, we'll pass in an object which contains options as key-value pairs:

```
$('.clickToFadeIn').on('click', function() {
// FadeIn Code Goes Here
  $('.bigBox').fadeIn({
    duration:2000,
    step:function(now) {
      $('#progressBar').val(now * 100);
    },
    complete:function() {
      alert('FadeIn Completed!');
    }
  });
});
```

As the `fadeIn` animates, the `step` function would be executed at every iteration, and this opens up a lot of possibilities for us such as being able to animate the progress bar element. The argument variable `now` is used to grab the current value of the fade on a scale of zero to one, so we have to multiply it by 100 and set as value for progress bar.

The `complete` handler runs the function once the animation ends.

So refresh the page and observe the result.

Fade to a given value

Now jQuery also let's you animate the fade to a given value of opacity instead of simply going from zero to one or one to zero. This is achieved by the `fadeTo` method:

```
$(<selector>).fadeTo(2000, 0.25, function() {});
```

You provide the duration, and a target opacity value. As earlier, the optional callback function is fired when the animation is complete.

Let's build a page with a set of `div`s with base opacity's set to `0.2`. We'll add a simple mouse over effect which animates the `div`'s opacity to `1` when the mouse hovers over it, and fades it back to `0.2` when the mouse moves away.

Working with fade to demo

Open *fadeTo_Begin.htm* file. There are three `div`s with a class name of `fadeBox`. We're using a `hover` method on the `fadeBox` `div`s, which provides us with two event handlers. The first one executes with the mouse pointer hovers above the `div`, while the second one executes when the mouse moves away.

Write the mouseover function first:

```
$('.fadeBox').hover(function() {
    // Code for Mouseover
  $(this).stop(true).fadeTo(200,1.0);
}, function() {
```

```
    // Code for Mouseout
});
```

Since we have more than one instances of the `div` with the class name of `fadeBox`, using the `this` keyword gives us access to that particular instance on which the mouse currently is.

In same way, we'll write the `fadeTo` method for the mouseout event:

```
$('.fadeBox').hover(function() {
    // Code for Mouseover
  $(this).fadeTo(200,1.0);
}, function() {
    // Code for Mouseout
  $(this).fadeTo(400,0.2);
});
```

There is one tiny improvement that we can still do. If a user rapidly hovers the mouse back and forth between these `div`s and then moves away, you'll notice the animation keeps going for a while before settling back to normal. This happens because all jQuery animation builds up as a queue, so the queue keeps playing until it ends. Fortunately, we do have a simple way to overcome this problem by inserting a `stop` function with a `true` argument:

```
$('.fadeBox').hover(function() {
    // Code for Mouseover
  $(this).stop(true).fadeTo(200,1.0);
}, function() {
    // Code for Mouseout
  $(this).stop(true).fadeTo(400,0.2);
});
```

This ensures that any animation in the queue on the element stops playing first before the `fadeTo` animation is executed.