CHAPTER

6

Animations and Effects

In the last two chapters, you learned the basics of using jQuery: how to load the jQuery library, select page elements, and respond to events like a visitor clicking on a button or mousing over a link. Most jQuery programs involve three steps: selecting an element on the page, attaching an event handler to that element, and then responding to that event by doing something. In this chapter, you'll start learning about the "doing something" part with jQuery's built-in effect and animation functions. You'll also get a little CSS refresher covering a few important CSS properties related to creating visual effects. In addition, you'll also learn how to combine CSS3 animations with jQuery's tools for super-smooth (and easy) animation effects.

jQuery Effects

Making elements on a web page appear and disappear is a common JavaScript task. Drop-down navigation menus, pop-up tooltips, and automated slideshows all rely on the ability to show and hide elements when you want to. jQuery supplies a handful of functions that achieve the goal of hiding and showing elements.

To use each of these effects, you apply them to a jQuery selection, like any other jQuery function. For example, to hide all tags with a class of `submenu`, you can write this:

```
$('.submenu').hide();
```

Each effect function also can take an optional speed setting and a *callback* function. The speed represents the amount of time the effect takes to complete, while a callback is a function that runs when the effect is finished. (See page 194 for details on callbacks.)

To assign a speed to an effect, you supply one of three string values—`'fast'`, `'normal'`, or `'slow'`—or a number representing the number of milliseconds the effect takes (1,000 is 1 second, 500 is half of a second, and so on). For example, the code to make an element fade out of view slowly would look like this:

```
$('element').fadeOut('slow');
```

Or if you want the element to fade out *really* slowly, over the course of 10 seconds:

```
$('element').fadeOut(10000);
```

When you use an effect to make an element disappear, the element isn't actually removed from the web browser's understanding of the page. The element still exists in the DOM, or Document Object Model (page 117). The element's code is still in the browser's memory, but its `display` setting (same as the CSS `display` setting) is set to `none`. Because of that setting, the element no longer takes up any visual space, so other content on the page may move into the position previously filled by the hidden element. You can see all of the jQuery effects in action on the *effects.html* file included in the *testbed* tutorial folder, as shown in Figure 6-1.

NOTE The keywords used for setting the speed of an effect–`'fast'`, `'normal'`, and `'slow'`–are the same as 200 milliseconds, 400 milliseconds, and 600 milliseconds. So

```
$('element').fadeOut('slow');
```

is the same as

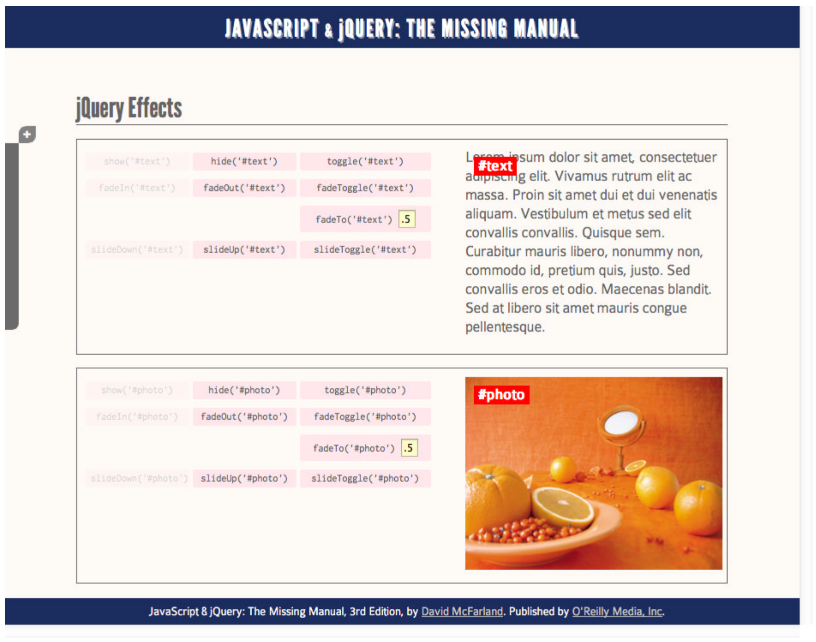
```
$('element').fadeOut(600);
```

Basic Showing and Hiding

jQuery provides three functions for basic hiding and showing of elements:

- **`show()`** makes a hidden element visible. It doesn't have any effect if the element is already visible on the page. If you don't supply a speed value, the element appears immediately. However, if you supply a speed value—`show(1000)`, for example—the element animates from the top-left corner down to the bottom-left corner.
- **`hide()`** hides a visible element. It doesn't have any effect if the element is already hidden, and as with the `show()` function, if you don't supply a speed value, the element immediately disappears. However, with a speed value the element animates out of view in a kind of shrinking motion.
- **`toggle()`** switches an element's current display value. If the element is currently visible, `toggle()` hides the element; if the element is hidden, then `toggle()` makes the element appear. This function is ideal when you want to have a single control (like a button) alternately show and hide an element.

In the tutorial on page 176 in Chapter 5, you saw the `hide()` function in action. That script uses `hide()` to make all of the answers on an FAQ page disappear when the page's HTML loads.



Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Vivamus rutrum elit ac massa. Proin sit amet dui et dui venenatis aliquam. Vestibulum et metus sed elit convallis convallis. Quisque sem. Curabitur mauris libero, nonummy non, commodo id, pretium quis, justo. Sed convallis eros et odio. Maecenas blandit. Sed at libero sit amet mauris congue pellentesque.

JavaScript & jQuery: The Missing Manual, 3rd Edition, by David McFarland. Published by O'Reilly Media, Inc.

FIGURE 6-1

You can test out jQuery's visual effects on the effects.html *file located in the* testbed *folder. Click the function text—*`fadeOut('#photo')`*, for example—to see how text and images look when they fade out, slide up, or appear. Some of the buttons will be faded out to indicate that they don't apply to the element. For example, it doesn't make much sense for the code to make a photo appear if it's already visible on the page.*

Fading Elements In and Out

For a more dramatic effect, you can fade an element out or fade it in—in either case, you're just changing the opacity of the element over time. jQuery provides four fade-related functions:

- **`fadeIn()`** makes a hidden element fade into view. First, the space for the element appears on the page (this may mean other elements on the page move out of the way); then the element gradually becomes visible. This function doesn't have any effect if the element is already visible on the page. If you don't supply a speed value, the element fades in using the *'normal'* setting (400 milliseconds).
- **`fadeOut()`** hides a visible element by making it fade out of view like a ghost. It doesn't have any effect if the element is already hidden, and like the `fadeIn()` function, if you don't supply a speed value, the element fades out over the course of 400 milliseconds. The FAQ tutorial on page 176 used this function to make the answers disappear.

- **`fadeToggle()`** combines both fade in and fade out effects. If the element is currently hidden, it fades into view; if it's currently visible, the element fades out of view. You could use this function to make an instruction box appear or disappear from a page. For example, say you have a button with the word "instructions" on it. When a visitor clicks the button, a div with instructions fades into view; clicking the button a second time fades the instructions out of view. To make the box fade in or out over the course of half a second (500 milliseconds), you could write this code:

```
$('#button').click(function() {
    $('#instructions').fadeToggle(500);
}); // end click
```

- **`fadeTo()`** works slightly differently than other effect functions. It fades an image to a specific opacity. For example, you can make an image fade so that it's semitransparent. Unlike other effects, you must supply a speed value. In addition, you supply a second value from 0 to 1 that indicates the opacity of the element. For example, to fade all paragraphs to 75% opacity, you'd write this:

```
$('p').fadeTo('normal',.75);
```

  This function changes an element's opacity regardless of whether the element is visible or invisible. For example, say you fade a currently hidden element to 50% opacity, the element fades into view at 50% opacity. If you hide a semitransparent element and then make it reappear, its opacity setting is recalled.

  If you fade an element to 0 opacity, the element is no longer visible, but the space it occupied on the page remains. In other words, unlike the other disappearing effects, fading to 0 will leave an empty spot on the page where the element is.

  In addition, if you fade an element and then hide it, that element disappears from the page. If you then show the element, it remembers its opacity setting, so the browser makes the element visible on the page again, but only at 50% opacity.

Sliding Elements

For a little more visual action, you can also slide an element in and out of view. The functions are similar to the fading elements in that they make page elements appear and disappear from view, and may have a speed value:

- **`slideDown()`** makes a hidden element slide into view. First, the top of the element appears and anything below the element is pushed down as the rest of the element appears. It doesn't have any effect if the element is already visible on the page. If you don't supply a speed value, the element slides in using the `'normal'` setting (400 milliseconds). The tutorial on page 179 used this function to make an answer appear on the FAQ page.

- **`slideUp()`** removes the element from view by hiding the bottom of the element and moving anything below the element up until the element disappears. It doesn't have any effect if the element is already hidden, and as with the `slideDown()` function, if you don't supply a speed value, the element slides out over the course of 400 milliseconds.
- **`slideToggle()`** applies the `slideDown()` function if the element is currently hidden, and the `slideUp()` function if the element is visible. This function lets you have a single control (like a button) both show and hide an element.

UP TO SPEED

Absolute Positioning with CSS

Normally, when you hide an element on a web page, other elements move to fill the space. For example, if you hide an image on a page, the image disappears, and content below that image moves up the page. Likewise, making an element appear forces other content to move to make room for the newly displayed element. You may not want content on your page to jump around like that. In that case, you can turn to CSS and *absolute positioning* to place an element outside the flow of normal page content. In other words, you can have a div, image, or paragraph appear on top of the page, as if sitting on its own separate layer, using the CSS position property.

To make an element appear above the page, give it a position value of absolute. You can then specify the placement for that element on the page using the `left`, `right`, `top`, and/or `bottom` properties. For example, say you have a `<div>` tag containing a login form. The login form won't normally be visible, but when a visitor clicks a link, that form slides into place, sitting above other content on the page. You could position that div like this:

```
#login {
    position: absolute;
    left: 536px;
    top: 0;
    width: 400px;
}
```

This style places the div at the top of the browser window and 536px from the left edge. You can also place an element from the right edge of the browser window using the right property, or in relationship to the bottom edge of the browser window using the bottom property.

Of course, you may want to place an element in relation to something other than the browser window. For example, pop-up tooltips are usually positioned in relation to some other element: A word on the page might have a ? next to it, that when clicked, opens a small box with a definition for that word. In this case, the tooltip needs to be positioned not in relationship to the top, left, right, or bottom of the browser window, but next to the word. To achieve this, you need to supply a position that's relative to an element that surrounds the absolutely positioned item. For example, look at this HTML:

```
<span class="word">Heffalump
<span class="definition">An  imaginary,
elephant-like creature from Winnie the
Pooh</span>
</span>
```

To make the definition span appear below the word, you first need to position the word span relatively, and then position the definition absolutely like this:

```
.word { position: relative; }
.definition {
    position: absolute;
    bottom: -30px;
    left: 0;
    width: 200px;
}
```

For more information on absolute positioning, visit *www.elated.com/articles/css-positioning/* or pick up a copy of *CSS3: The Missing Manual*.

Tutorial: Login Slider

In this tutorial, you'll get a little practice with using jQuery effects by creating a common user interface element: a panel that slides into and out of view with a click of the mouse (see Figure 6-2).

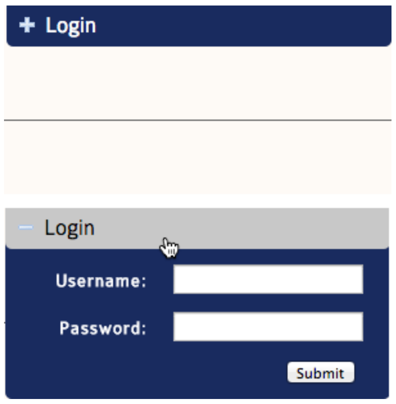


FIGURE 6-2

Now you don't see it, now you do. The login form is normally hidden from view (top), but a simple mouse click reveals the form, ready for the visitor to fill out and log on.

The basic task is rather simple:

1. **Select the paragraph with the "Login" message on it.**

   Remember that a lot of jQuery programming first begins with selecting an element on the page. In this case, the "Login" paragraph will receive clicks from a visitor.

2. **Attach a `click` event handler to that paragraph.**

   JavaScript isn't interactive without events: The visitor needs to interact with the selection (the Login paragraph) to make something happen.

3. **Toggle the display of the form on and off.**

   The previous two steps are just review (but necessary for so much of jQuery programming). This last step is where you'll use the effects you've learned about. You can make the form instantly appear (the `show()` function), slide into view (the `slideDown()` function), or fade into view (the `fadeIn()` function.)

NOTE See the note on page 12 for information on how to download the tutorial files.

The Programming

1. **In a text editor, open the file *login.html* in the *chapter06* folder.**

   This file already contains a link to the jQuery file, and the `$(document).ready()` function (page 160) is in place. First, you'll select the paragraph with the "Login" text.

2. **Click in the empty line after the `$(document).ready()` function, and then type `$('#open')`.**

   The "Login" text is inside a paragraph tag that's surrounded by a link: `<a href="form.html"><p id="open">Login</p></a>`. The link will direct users to another page that has the login form in it. The link is there so if a visitor doesn't have JavaScript turned on in her browser, she won't be able to see the hidden login form. By adding a link, you assure that even people without JavaScript turned on have an alternative way to reach a login form.

   The `$('#open')` you just typed selects the paragraph. Now, it's time to add an event handler.

NOTE The `<p>` tag mentioned in step 2 is wrapped in an `<a>` tag. If you've been building websites for a while, you may think that's invalid HTML. And it was—in HTML 4 and earlier. When you use the HTML5 doctype (page xvii), it's valid to wrap block-level elements like `<p>`, `<h1>`, and even `<div>` tags with links. Doing so is an easy way to create a large, clickable link target.

3. **Add the bolded code below, so the script looks like this:**

   ```
   $(document).ready(function() {
     $('#open').click(function(evt) {

     }); // end click
   }); // end ready
   ```

   This code adds a click handler, so each time a visitor clicks on the paragraph, something happens. In this case, the form should appear when clicked once and then disappear when clicked again, appear on the next click, and so on. In other words, the form toggles between visible and invisible. jQuery offers three functions that will serve this purpose: `toggle()`, `fadeToggle()`, and `slideToggle()`. The difference is merely in how the effect looks.

   You're also adding an argument to the anonymous function inside the `click()` method. As discussed on page 164, every event automatically gets passed an event object, which contains different properties and methods. You'll need this object to tell jQuery to stop the browser from following the link to the form page.

4. **Click in the empty line inside the `click()` function and type:**

   ```
   evt.preventDefault();
   ```

   The `preventDefault()` method stops the browser from following the link that surrounds the paragraph. Remember, the link is there to give visitors whose browsers don't have JavaScript turned on a way to reach a login form. But for browsers with JavaScript, you must tell the browser to stay on the page and run some more JavaScript.

5. **After the last line you added, type:**

   ```
   $('#login form').slideToggle(300);
   ```

   This code selects the login form, slides it into view if it currently isn't visible, and then slides it back out of view. Finally, you'll change the class of the paragraph, so that the "Login" paragraph can change appearance using a CSS class style.

6. **Add the code in bold below, so the finished script looks like this:**

   ```
   $(document).ready(function() {
     $('#open').click(function(evt) {
       evt.preventDefault();
       $('#login form').slideToggle(300);
       $(this).toggleClass('close');
     }); // end click
   }); // end ready
   ```

   As you'll recall from page 159, when you're inside of an event handler, you can use `$(this)` to refer to the element that responds to the event. In this case, `$(this)` refers to the paragraph the visitor clicks on—the `$('#open')` in line 2 above. The `toggleClass()` function simply adds or removes a class from the element. Like the other toggle functions, `toggleClass()` adds the specified class if it's missing or removes the class if it's present. In this example, there's a class style—`.close`—in a style sheet on the page. This style simply adds a different background image to indicate that the visitor can close the login slider. (Look in the <head> and you can see the style and what it does.)

7. **Save the page and preview it in a web browser.**

   Make sure you click the "Login" paragraph several times to see how it works. You'll find a finished version of the tutorial—*complete_login.html*—in the *chapter06* folder. Try out the other toggle effects as well, by replacing `slideToggle()` with `toggle()` or `fadeToggle()`.

But what if you want two different effects? One for making the form appear—slide the form down into view, for example—and a different effect to make it disappear—fade out of view, for example. The code in step 5 won't work because the `click()` function doesn't let you choose between two different actions. You'll need to use the same trick you used in the FAQ tutorial in the previous chapter (page 174): when the visitor clicks the "Login" link, you need to check if the form is hidden. If it is, then show it; if it's visible then hide it.

To make the form slide into view then fade out of view on alternating clicks, you use this code:

```
$(document).ready(function() {
  $('#open').click(function(evt) {
    evt.preventDefault();
    if ($('#login form').is(':hidden')) {
      $('#login form').fadeIn(300);
```

```
        $(this).addClass('close');
      } else {
        $('#login form').slideUp(600);
        $(this).removeClass('close');
      }
    }); // end click
  }); // end ready
```

NOTE You'll find the above code in the *completed_login2.html* file in this chapter's tutorial folder.

Animations

You aren't limited to just the built-in effects jQuery supplies. Using the `animate()` function, you can animate any CSS property that accepts numeric values such as pixel, em, or percentage values. For example, you can animate the size of text, the position of an element on a page, the opacity of an object, or the width of a border.

NOTE jQuery, by itself, can't animate color—for example, the color of text, background color, or border color. However, jQuery UI has many additional animation effects including the ability to animate color. You'll learn about jQuery UI's animation tools in Chapter 12.

To use this function, you must pass an object (page 136) containing a list of CSS properties you wish to change and the values you wish to animate to. For example, say you want to animate an element by moving it 650 pixels from the left edge of the page, changing its opacity to 50%, and enlarging its font size to 24 pixels. The following code creates an object with those properties and values:

```
{
  left: '650px',
  opacity: .5,
  fontSize: '24px'
}
```

Note that you only have to put the value in quotes if it includes a measurement like px, em, or %. In this example, you need quotes around `'650px'` because it contains px, but not around the opacity value because .5 is simply a number and doesn't contain any letters or other characters. Likewise, putting quotes around the property (`left`, `opacity`, and `fontSize`) is optional.

NOTE JavaScript doesn't accept hyphens for CSS properties. For example, `font-size` is a valid CSS property, but you can't use it in the name of key in an object literal. The hyphen has a special meaning in JavaScript (it's the minus operator). When using CSS properties as a key in an object literal, remove the hyphen and capitalize the first letter of the word following the hyphen. For example, `font-size` becomes `fontSize`, and `border-left-width` becomes `borderLeftWidth`.

NOTE However, if you want to stick with the true CSS property names (to prevent confusion), put the property name in quotes like this:

```
{
   'font-size': '24px',
   'border-left-width': '2%'
}
```

Suppose you want to animate an element with an ID of `message` using these settings. You can use the `animate()` function like this:

```
$('#message').animate(
  {
    left: '650px',
    opacity: .5,
    fontSize: '24px'
  },
  1500
);
```

The `animate()` function can take several arguments. The first is an object literal containing the CSS properties you wish to animate. The second is the duration (in milliseconds) of the animation. In the above example, the animation lasts 1,500 milliseconds, or 1.5 seconds.

NOTE In order to animate a position of an element using the CSS `left`, `right`, `top`, or `bottom` properties, you must set that element's CSS `position` property to either `absolute` or `relative`. Those are the only two positioning properties that let you assign positioning values to them (see the box on page 187).

You can also set a property relative to its current value using += or -= as part of the animation options. For example, say you want to animate an element by moving it 50 pixels to the right each time you click on it. Here's how:

```
$('#moveIt').click(function() {
  $(this).animate(
    {
      left:'+=50px'
    },
   1000); // end animate
}); // end click
```

Easing

The jQuery effects functions (`slideUp()`, `fadeIn()`, an so on) and the `animation()` function accept another argument that controls the speed during the animation: *easing*, which refers to the speed during different points of the animation. For example, while moving an element across the page, you could have the element's movement start slowly, then get really fast, and finally slow down as the animation completes. Adding easing to an animation can make it more visually interesting and dynamic.

jQuery includes only two easing methods: `swing` and `linear`. The linear method provides a steady animation so each step of the animation is the same (for example, if you're animating an element across the screen, each step will be the same distance as the previous one). Swing is a bit more dynamic, as the animation starts off a bit more quickly, then slows down. Swing is the normal setting, so if you don't specify any easing, jQuery uses the swing method.

The easing method is the second argument for any jQuery effect, so to make an element slide up using the linear method, you'd write code like this:

```
$('#element').slideUp(1000,'linear');
```

When using the `animate()` function, the easing method is the third argument after the object containing the CSS properties you wish to animate, and the overall speed of the animation. For example, to use the linear easing method with the animation code from page 192, you'd write:

```
$('#message').animate(
{
  left: '650px',
  opacity: .5,
  fontSize: '24px'
},
1500,
'linear'
);
```

You're not limited to the two easing methods jQuery supplies, however. Thanks to the industrious work of other programmers, you can add a whole bunch of other easing methods—some quite dramatic and fun to watch. In fact, the jQueryUI library includes many additional easing methods. You'll learn more about jQuery UI in Part Three, but there's no reason not to start using it now to add a little fun to your animations.

To use the jQueryUI (which is an external JavaScript file), you attach the file to your page after the code that links to the jQuery library. Once you've linked to the jQueryUI file, you can use any of the easing methods available (see *http://api.jqueryui.com/easings/* for a complete list). For example, say you want to make a div tag grow in size when a visitor clicks on it, and you want to make the animation more interesting by using the `easeInBounce` method. Assuming the div has an ID of `animate`, your code may look like this:

```
<script src="_js/jquery.min.js"></script>
<script src="_js/jquery-ui.min.js"></script>
<script>
$(document).ready(function() {
  $('#animate').click(function() {
    $(this).animate(
 7     {
 8       width: '400px',
 9       height: '400px'
10     },
11     1000,
12     'easeInBounce'); // end animate
13  }); // end click
14 }); // end ready
15 </script>
```

Lines 1 and 2 load jQuery and jQueryUI. Line 4 is the ever-present `ready()` function (page 160), and line 5 adds a click handler to the div. The heart of the action is in lines 6–12. As you'll recall from page 159, when you're inside of an event, `$(this)` refers to the element that's responding to the event—in this case, the `<div>` tag. In other words, by clicking the div, you also animate that div by changing its width and height (lines 8 and 9). Line 11 makes the animation occur over 1 second (1,000 milliseconds), and line 12 sets the easing method to `easeInBounce` (you can substitute any easing method, like `easeInOutSine`, `easeInCubic`, and so on).

NOTE You can find an example of this code in action in the *chapter06* folder of the tutorial files. Open the file *easing_example1.html* in a web browser. The file *easing_example2.html* shows how to toggle between clicks: animate a `<div>` one way on the first click, and another way on the next.

Performing an Action After an Effect Is Completed

Sometimes you want to do something once an effect is complete. For example, suppose when a particular photo fades into view, you want a caption to appear. In other words, the caption must pop onto the page after the photo finishes fading into view. Normally, effects aren't performed one after the other; they all happen at the same time they're called. So if your script has one line of code to fade the photo into view, and another line of code to make the caption appear, the caption will appear while the photo is still fading in.

To do something after the effect is finished you can pass a *callback function* to any effect. That's a function that runs only after the effect is completed. The callback function is passed as the second argument to most effects (the third argument for the `fadeTo()` function).

For example, say you have an image on a page with an ID of `photo`, and a paragraph below it with an ID of `caption`. To fade the photo into view and then make the caption fade into view, you can use a callback function like this:

```
$('#photo').fadeIn(1000, function() {
  $('#caption').fadeIn(1000);
});
```

Of course, if you want to run the function when the page loads, you'd want to hide the photo and caption first, and then do the `fadeIn` effect:

```
$('#photo, #caption').hide();

$('#photo').fadeIn(1000, function() {

      $('#caption').fadeIn(1000);

});
```

If you use the `animate()` function, then the callback function appears after any other arguments—the object containing the CSS properties to animate, the animation duration, and the easing setting. The easing setting is optional, however, so you can also just pass the `animate()` function, property list, duration, and callback function. For instance, say you want to not only fade the photo into view but also increase its width and height from zero to full size (a kind of zooming effect). You can use the `animate()` function to do that, and then display the caption like this:

```
 1 $('#photo').width(0).height(0).css('opacity',0);
 2 $('#caption').hide();
 3 $('#photo').animate(
 4    {
 5       width: '200px',
 6       height: '100px',
 7       opacity: 1
 8    },
 9    1000,
10    function() {
11       $('#caption').fadeIn(1000);
12    }
13 ); // end animate
```

Line 1 of the code above sets the width, height, and opacity of the photo to 0. (This hides the photo and gets it ready to be animated.) Line 2 hides the caption. Lines 3–13 are the animation function in action and the callback occurs on lines 10–12. This probably looks a little scary, but, unfortunately, the callback function is the only way to run an action (including an effect on a different page element) at the completion of an effect.

NOTE The file *callback.html* in the *chapter06* folder shows the above code in action.

Callback functions can get tricky when you want to animate several elements in a row: for example, to make an image move into the center of the screen, followed by a caption fading into view, and then having both the image and caption fade out. To make that happen, you need to pass a callback function to a callback function like this:

```
$('#photo').animate(
   {
      left: '+=400px'
   },
   1000,
   function() { // first callback function
      $('#caption').fadeIn(1000,
        function() { // second callback function
          $('#photo, #caption').fadeOut(1000);
        } // end second callback
      ); // end fadeIn
   } // end first callback function
); // end animate
```

NOTE The file *multiple-callbacks.html* in the *chapter06* folder shows this code in action.

However, you don't need to use a callback function if you want to add additional animations to the same page element. For example, say you want to move a photo onto the screen, then make it fade out of view. In this case, you simply use the `animate()` function to move the photo and then fade the image out of view. You can do that like this:

```
$('#photo').animate(
   {
      left: '+=400px',
   },
   1000
); // end animate
$('#photo').fadeOut(3000);
```

In this case, although the browser executes the code immediately, jQuery places each effect into a queue, so that first the animation runs and then the `fadeOut()` function runs. Using jQuery chaining (page 127), you could rewrite the code above like this:

```
$('#photo').animate(
   {
      left: '+=400px',
   },
   1000).fadeOut(3000);
```

If you want a photo to fade in, fade out, and then fade in again, you can use chaining like this:

```
$('#photo').fadeIn(1000).fadeOut(2000).fadeIn(250);
```

NOTE For more information on how the effects queue works visit the jQuery website: *http://api.jquery.com/jQuery.queue/.*

One additional jQuery function that can come in handy when queuing up effects on an element is the `delay()`. This function simply waits the specified number of milliseconds before beginning the next effect in the queue. For example, say you want to fade an image into view, wait 10 seconds, and then fade it out of view. You can use the `delay()` function like this:

```
$('#photo').fadeIn(1000).delay(10000).fadeOut(250);
```

Tutorial: Animated Dashboard

In this tutorial, you'll use the `animate()` function to move a `<div>` tag right from off the left edge of the page into view. The div is absolutely positioned (see the box on page 187 for more on absolute positioning) so that most of the box hangs off the left edge of the page outside the boundaries of the browser window (Figure 6-3, left). When a visitor mouses over the visible edge of the div, that div moves completely into view (Figure 6-3, right). To make this effect more fun, use the jQueryUI library to animate the background color of the div and to use a couple of different easing methods.

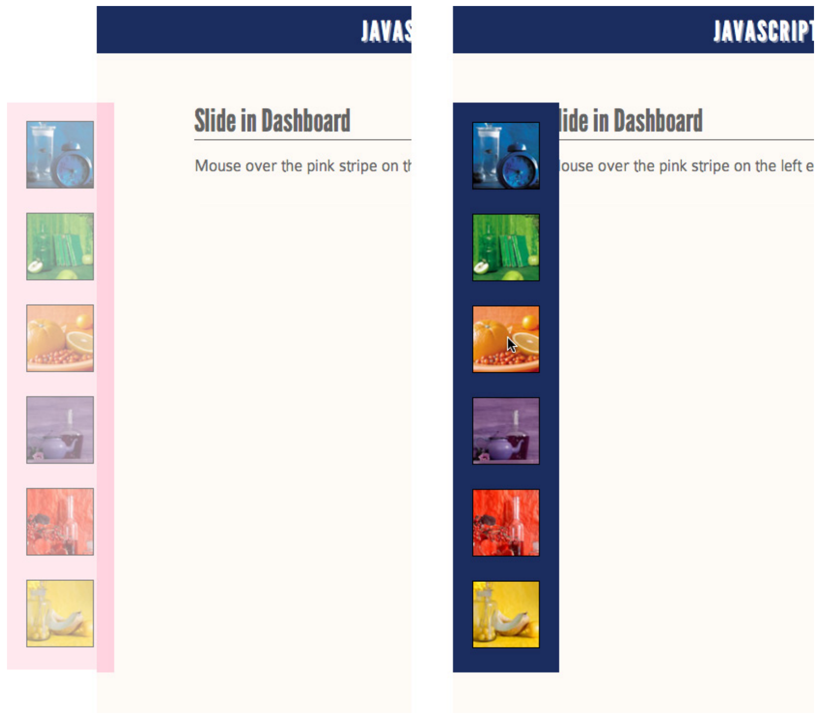

JAVAS
Slide in Dashboard
Mouse over the pink stripe on th
JAVASCRIPT
lide in Dashboard
ouse over the pink stripe on the left e

FIGURE 6-3

You can have a lot of fun by hiding page elements off one of the edges of the browser window (like the div in the left image, which is mostly out of view). Using the `animate()` *function, you can then bring that page element fully into view (right).*

NOTE See the note on page 12 for information on how to download the tutorial files.

The basic task is rather simple:

1. **Select the `<div>` tag.**

   Remember that a lot of jQuery programming begins with selecting an element on the page—in this case, the `<div>` tag that a visitor mouses over.

2. **Attach a `hover` event.**

   The `hover` event (described on page 162) is a special jQuery function, not a real JavaScript event, that lets you perform one set of actions when a visitor mouses over an element, then a second set of actions when the visitor mouses off the element (the `hover` event is really just a combination of the `mouseEnter` and `mouseLeave` events).

3. **Add the animate function for the `mouseEnter` event.**

   When a visitor mouses over the div, you'll animate the left position of the div, moving it from out of view on the left edge of the browser window. In addition, you'll animate the background color of the div.

4. **Add another animate function for the `mouseLeave` event.**

   When a visitor moves the mouse off the div, you'll animate the div back to its original position and with its original background color.

The Programming

1. **In a text editor, open the file *animate.html* in the *chapter06* folder.**

   This file already contains a link to the jQuery file, and the `$(document).ready()` function (page 160) is in place. However, because you'll be animating the background color of the div and using a couple of interesting easing methods, you need to attach the jQueryUI file.

2. **Click in the empty line after the first `<script>` tag and add the code in bold below:**

   ```
   <script src="../_js/jquery-1.7.2.min.js"></script>
   <script src="../_js/jquery-ui.min.js"></script>
   ```

   jQueryUI is a jQuery *plug-in*. In the jQuery world, a plug-in is simply an external JavaScript file that lets you add complex effects and features to your site without a lot of programming on your part. Next you'll select the div and add the `hover()` function to it.

3. **Click in the empty line inside the `$(document).ready()` function and type `$('#dashboard').hover(); // end hover` so your code looks like this:**

   ```
   $(document).ready(function() {
     $('#dashboard').hover(); // end hover
   }); // end ready
   ```

`$('#dashboard')` selects the `<div>` tag (which has the ID dashboard applied to it). The `hover()` function takes two arguments—two functions (page 162)—that describe what to do when a visitor moves his mouse over the div, and then moves his mouse off the div. Instead of typing all of the code at once, you'll build it up piece by piece, first adding the `hover()` function, then adding the "shell" for two anonymous functions. This approach is helpful because the nest of parentheses, braces, commas, and semicolons can overwhelm you if you're not careful.

4. **Click between the parentheses of the `hover()` function and add two empty, anonymous functions:**

```
$(document).ready(function() {
  $('#dashboard').hover(
   function() {

   },
   function() {

   }
  ); // end hover
}); // end ready
```

jQuery's `.hover()` function is a bit wild looking when you've finished adding all of its programming (see step 12). However, it's really just a function that accepts two functions as arguments. It's a good idea to add the "shells" of the anonymous functions first—so you can make sure you've set it up correctly—before adding all of the programming inside the functions.

TIP Test your code frequently to make sure you haven't made any typos. In step 4, you can type `console.log('mouseEnter')` inside the first anonymous function and `console.log('mouseLeave')` inside the second anonymous function, and then preview the page in a web browser. You'll need to open the browser's console to see the results: F12 for Internet Explorer; Ctrl+Shift+J (Win) or ⌘-Option-J (Mac) for Chrome; Ctrl+Shift+K (Win) or ⌘-Option-K (Mac) for Firefox; and, ⌘-Option-C for Safari. The console should spit out the message "mouseEnter" when you mouse over the div, and "mouseLeave" when you mouse out. If no message appears in the console, you've made a typo. Double-check your code against these steps, or use the steps on page 18 to find errors using the browser's error console.

5. **Inside the first anonymous function, type `$(this).animate(); // end animate`.**

As discussed on page 159, inside an event, `$(this)` refers to the page element to which you've attached the event. In this case, `$(this)` refers to the `<div>` tag with the ID dashboard. In other words, mousing over this div will also animate this div.

6. **Add an object literal with the CSS properties you wish to animate:**

```
$(document).ready(function() {
  $('#dashboard').hover(
   function() {
      $(this).animate(
    {
      left: '0',
      backgroundColor: 'rgb(27,45,94)'
    }
      ); // end animate
    },
    function() {

    }
   ); // end hover
}); // end ready
```

The first argument to the `animate()` function is an object (page 136) containing CSS properties. In this case, the div currently has a left value of -92px, so that most of the div is hidden, hanging off the left edge of the browser window. By animating its left value to 0, you're essentially moving it completely into view on the left edge. Likewise, thanks to jQueryUI, you're changing its background color from pink to dark blue. Next, you'll set a duration for the animation.

7. **Type a comma after the closing `}` of the object literal, press Return and then type `500`.**

   The comma marks the end of the first argument passed to the `animate()` function, while the 500 sets the length of the animation to half a second or 500 milliseconds. Lastly, you'll set an easing method.

8. **Type a comma after the 500, hit Return, and type `'easeInSine'` so your code looks like this:**

```
$(document).ready(function() {
  $('#dashboard').hover(
    function() {
      $(this).animate(
        {
          left: '0',
          backgroundColor: 'rgb(27,45,94)'
        },
        500,
     'easeInSine'
      ); // end animate
    },
    function() {

    }
   ); // end hover
}); // end ready
```

The last argument to the `animate()` function here—`'easeInSine'`—tells the function to use an easing method that starts off somewhat slowly and then speeds up.

9. **Save the file. Preview it in a browser and mouse over the div.**

   The div should scoot into view. If it doesn't, troubleshoot using the techniques described on page 18. Of course, when you mouse off the div, nothing happens. You have to add the animate function to the second anonymous function.

10. **Add the code below to the second anonymous function:**

```
$(this).animate(
    {
       left: '-92px',
       backgroundColor: 'rgb(255,211,224)'
    },
    1500,
    'easeOutBounce'
); // end animate
```

   This code reverses the process of the first animation, moving the div back off the left edge of the window and reverting the background color to pink. The timing is a bit different—1.5 seconds instead of half a second—and the easing method is different.

11. **Save the file. Preview it in a browser and move your mouse over and off of the div.**

   You'll see the div move into view and then out of view. However, if you move the mouse over and off the div repeatedly and quickly, you'll notice some strange behavior: The div will keep sliding in and out of view long after you've finished moving the mouse. This problem is caused by how jQuery queues up animations on an element. As described on page 194, any animation you add to an element gets put into a sort of queue for that element. For example, if you have an element fade into view, fade out of view, and then fade back into view, jQuery performs each effect in order, one after the other.

   What's happening in the code for this tutorial is that each time you mouse onto and off of the div, an animation is added to the queue; so, rapidly mousing over the div creates a long list of effects for jQuery to perform: Animate the div into view, animate the div out of view, animate the div into view, animate the div out of view, and so on. The solution to this problem is to stop all animations on the div before performing a new animation. In other words, when you mouse over the div, and that div is in the process of being animated, then jQuery should stop the current animation, and proceed with the animation required by the `mouseEnter` event. Fortunately, jQuery supplies a function—the `stop()` function—for just such a problem.

12. **Add `.stop()` between `$(this)` and `.animate` in the two anonymous functions. The finished code should look like this (additions are in bold):**

```
$(document).ready(function() {
  $('#dashboard').hover(
   function() {
     $(this).stop().animate(
       {
         left: '0',
         backgroundColor: 'rgb(27,45,94)'
       },
       500,
       'easeInSine'
     ); // end animate
   },
   function() {
     $(this).stop().animate(
       {
         left: '-92px',
         backgroundColor: 'rgb(255,211,224)'
       },
       1500,
       'easeOutBounce'
     ); // end animate
   }
  ); // end hover
}); // end ready
```

The `.stop()` function here simply ends any animations on the div before starting a new one, and prevents multiple animations from building up in the queue.

Save the page and try it out in a web browser. You can find a finished version of this tutorial—*complete_animate.html*—in the *chapter06* folder.

jQuery and CSS3 Transitions and Animations

If you're keeping up with the latest and greatest in Cascading Style Sheets techniques, you may be wondering why even bother with jQuery? After all, CSS transitions let you use CSS alone to produce animations between two different CSS styles, and CSS animations let you produce complex animated effects (see Figure 6-4).

The new CSS animation effects are awesome, but not all browsers can enjoy them. Two still widely used versions of Internet Explorer—8 and 9—don't support either CSS transitions or animations. So, if you need to include those browsers in your website building plans, you'll need another way to add animation. In this instance, the jQuery animations discussed earlier in this chapter are your best bet for cross-browser animations.

NOTE Browsers can't animate all CSS properties. For example, you can't animate the `font-family` property by morphing text from one font to another. But you can use many CSS properties in transitions and animations. For a complete list, visit *https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_animated_properties.*

FIGURE 6-4

Cartoonist, JavaScript expert, and animator Rachel Nabors brings together CSS3 animations and jQuery to create a complete, animated adventure story at http://codepen.io/rachelnabors/full/lqswg. *She uses jQuery to trigger animations created with CSS. Unfortunately, this cutting-edge example doesn't work in all browsers.*

However, even if you want to use CSS transitions and animations, jQuery can still be a big asset. Unlike JavaScript, CSS doesn't really have any events. There's the `:hover` pseudo-class that lets you apply one style on `mouseEnter` and return to the previous style on `mouseLeave`. And the `:active` pseudo-class can sort of simulate a click. But for many events like double-clicks, scrolling, keyboard taps, there are no CSS equivalents. This means you can't use CSS alone to start an animation when a visitor types in a text field or double-clicks a button. In addition, there's no way, with just CSS, to trigger an animation on one element, when a visitor interacts with another element elsewhere on the page: for example, clicking a "Show settings" button at the top of the page to reveal a `<div>` elsewhere on the page.

NOTE This book doesn't cover CSS transitions and animations in much detail. If you want to learn more, pick up a copy of *CSS3: The Missing Manual*. For a quick primer on CSS transitions, visit *www.css3files.com/transition/*. For a primer on CSS animations, check out *www.css3files.com/animation/.*

jQuery and CSS Transitions

A CSS transition animates changes in CSS properties. An easy way to do this is just by applying a new style to an element and then animating that change. For example, you might create a class style for a button—`.button`, for example—that has a blue background. With a `:hover` pseudo-class—`.button:hover`—you can change that button to yellow. By adding a transition property to the `.button` style, you're telling the browser to animate the change from blue to yellow when the visitor mouses over the button, and animate the change from yellow to blue when the visitor mouses off of button.

You can add a transition to animate the change from one CSS property to another. For example, say you want to make all images on a page fade out of view using CSS transitions. You'd start by giving the images a style, like this:

```
img {
  opacity: 1;
}
```

The CSS `opacity` property controls the transparency of an element. A value of `1` means fully opaque and a value of `0` means completely transparent. You could then create a CSS class that sets the opacity to `0`, like this:

```
img.faded {
  opacity: 0;
}
```

To add an animation between these two styles you use the CSS `transition` property. You should add this property to the original style (the one applied first to the element on the page). In this case, that's the `img` style. In addition, to make sure this works with all browsers that support CSS transitions, you'll need to use vendor prefixes, like this:

```
img {
  opacity: 1;
  -webkit-transition: opacity 1s;
  -moz-transition: opacity 1s;
  -o-transition: opacity 1s;
  transition: opacity 1s;
}
```

In this example, you're specifying that you want to animate any changes in `opacity`. You also want the animation to last for 1 second—that's the `1s` in the code.

NOTE When browser makers add new and innovative CSS properties to their browsers, they usually begin by adding a vendor prefix before the property name, like `-webkit-transition`. This prefix lets the browser maker test out a feature safely before all the kinks are worked out of the new property. Once everyone agrees on how a new CSS property should work, newer versions of the browsers drop the prefix and just use the standard CSS property name—`transition` in place of `-webkit-transition`, for example.

Now that the transition is added, if the `faded` class is added to an image, it will animate from 100% opacity to 0% opacity over the course of 1 second. In other words, it's like using jQuery's `fadeOut()` function with a 1 second duration. The key is getting that new class—`faded`—onto an image. Here's where jQuery can help you out. If you want to apply the style when the visitor clicks an image, use the `click()` event function like this:

```
$('img').click(function() {
  $(this).addClass('faded');
}
```

This way, when the visitor clicks the image, jQuery simply adds a class to it; the web browser then does all the heavy lifting by animating the opacity change (see Figure 6-5). If you want to fade the image back when it's clicked on again, you can use the `toggleClass()` function:

```
$('img').click(function() {
  $(this).toggleClass('faded');
}
```

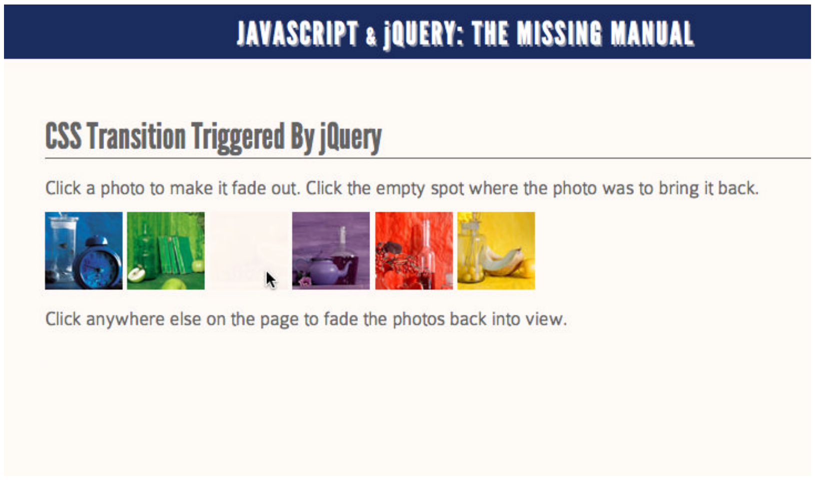
The `toggleClass()` function adds a class if it's not already applied to the element, and removes it if it is. Because CSS doesn't have a selector for when a visitor clicks something, jQuery offers a simple way to trigger CSS transitions.

NOTE Setting an element's opacity doesn't remove that element from the page. It's still there, taking up space on the page (and in the DOM), but it can't be seen. That's why the `toggleClass()` example works. You can still click on the invisible image to remove the faded class. However, if the element is actually hidden—for example, `display: hidden`—it's removed from the page and can't be clicked on a second time.

jQuery and CSS Animations

CSS animations provide much more control than simple transitions. With a CSS animation, you define *keyframes* that define CSS properties at certain steps in the animation. For example, you can make a button change color many times—from blue to red to orange to green—or move a `<div>` tag to the top of the screen, bottom of the screen to the left, and then to the right. In other words, while CSS transitions let you specify the beginning and ending styles of an animation, CSS animations let you specify any number of intermediate styles from the start of the animation to the end.

CSS animations can get pretty complex, pretty fast, and you can find resources beyond this book where you can learn more about how to create them. To get you started, this section will step you through a quick example of using jQuery to trigger an animation. Say you want to make a `<div>` change color and grow wider when a visitor presses the play button. (You may want to do this to highlight and reveal hidden text inside that box.)



JAVASCRIPT & jQUERY: THE MISSING MANUAL

CSS Transition Triggered By jQuery

Click a photo to make it fade out. Click the empty spot where the photo was to bring it back.

Click anywhere else on the page to fade the photos back into view.

FIGURE 6-5

jQuery and CSS transitions play well together. Using jQuery's event handlers, you can trigger CSS transition animations. In this example, clicking an image fades it out of view; clicking the empty space left behind fades the image back. This code won't work on Internet Explorer 9 or earlier, however—those browsers don't understand CSS transitions. You'll find the finished version of this page, jquery-trigger-css-animation.html*, in the* ch06 tutorial *folder.*

The first step is creating the animation. You do that with the `@keyframes` directive:

```
@keyframes growProgressBar{
  0% {
    width: 0%;
    background-color: red;
  }
  50% {
    background-color: yellow;
  }
  100% {
    width:88%;
    background-color: green;
  }
}
```

This kind of code may look unfamiliar to you, but it simply sets up an animation with a name—`growProgressBar`, in this case—and a series of keyframes. Each keyframe identifies one or more CSS properties that will change as the animation progresses. For example, the first keyframe above—0%—is the beginning of the animation and indicates that the element should be 0% wide and have a red background color.

NOTE Chrome and Safari require vendor prefixes—`@webkit-keyframes`—for this code to work; and, of course, it won't work at all in IE 9 or earlier.

The element will change color between keyframes from red to yellow to green. The percentage value used for each keyframe indicates when the specified CSS value should appear. Now imagine this animation will last 10 seconds (you'll specify the animation time separately, as you'll see in a moment). So at 0% of 10 seconds, the background color is red and the element is 0% wide. At 50% of 10 seconds, or half way through the animation at 5 seconds, the color will be yellow. Finally, when the animation is over at 100% of the time or 10 seconds, the element will be green and 88% wide.

Because in the example above the width is specified only in the first and last keyframes, that width will change from 0% to 88% over the entire duration of the animation.

Once you create the keyframes, you can add that animation to any number of elements. For example, say you had the following HTML: `<div class="progressBar">`. You could add this animation to that div like this:

```
.progressBar {
  animation-name: growProgessBar;
  animation-duration 10s;
  animation-fill-mode: forwards;
}
```

This code applies the animation to this element and makes it run for 10 seconds. The last CSS line—`animation-fill-mode: forwards;`—just makes sure that when the animation is finished, the element keeps the end keyframe properties. That is, the div should be 88% wide with green background. (Without this setting, the element will revert back to the styles it had before the animation ever happened.)

However, the animation in this code would begin immediately, as soon as the web page loads. What you want to do is start the animation when the visitor clicks a button. You can "pause" an animation (that is, stop it before it ever starts), using another animation property: `animation-play-state`. To prevent the animation from starting immediately, add that property to the style, with a value of `paused`:

```
 .progressBar {
  animation-name: growProgessBar;
  animation-duration 10s;
  animation-fill-mode: forwards;
  animation-play-state: paused;
}
```

Adding the jQuery to trigger this animation is the easy part. All you need to do is change the `animation-play-state` property to `running` to start the animation. You can do that easily with the `css()` function. For example, you could create a button with an ID of `start` that, when clicked, would begin the animation. Here's how the jQuery code would look:

```
$('#start').click(function() {
  $('.progressBar').css('animation-play-state', 'running');
});
```

If you had a pause button with an ID of `pause`, you could use that to pause the animation as well:

```
$('#pause').click(function() {
  $('.progressBar').css('animation-play-state', 'paused');
});
```

Fortunately, jQuery is savvy about vendor prefixes. When you set the value of a CSS property that requires vendor prefixes, jQuery will set each of the vendor-prefixed versions of the CSS properties as well. Thank you, jQuery!

Another approach is to create the keyframes and a separate class style that has all of the animation properties defined in it. It looks something like this:

```
.animateDiv {
  animation-name: growProgessBar;
  animation-duration 10s;
  animation-fill-mode: forwards;
}
```

The element you want to animate won't have this `.animateDiv` class style applied to it when the page loads; as a result, that element won't start animating yet, which is exactly what you want. Then, you use jQuery to add this class to the element. As soon as the element receives this new class, the animation will begin. This approach lets you skip the `animation-play-state` property:

```
$('#start').click(function() {
  $('.progressBar').addClass('animateDiv');
});
```

You'll find examples of both approaches in this chapter's tutorial folder: *jquery-trigger-css-animation1.html* and *jquery-trigger-css-animation2.html*.

At this point, there are some drawbacks to using CSS animations. As mentioned earlier, IE 9 and earlier don't understand them at all. In addition, it's not as easy to keep track of the progress of a CSS animation as it is in jQuery.

NOTE Eventually, developers will probably all end up using CSS animations in combination with JavaScript. The W3C and browser vendors are working on lots of ways to control CSS animations with JavaScript by adding new events that detect the progress of CSS animations. If you do want to run jQuery code when a CSS animation is finished running, check out this article: *http://blog.teamtreehouse.com/using-jquery-to-detect-when-css3-animations-and-transitions-end.*