CHAPTER 4

CREATING ANIMATION AND VISUAL EFFECTS

Today, more and more businesses want their websites to include formatting and images that can be updated without requiring the users to reload a web page from the server. They also want to use animation and interactive web pages in innovative ways to attract and retain visitors and to make their websites effective and easy to navigate. You cannot create these kinds of effects with standard HTML; instead, you need to use DHTML or jQuery.

ANIMATING WEB PAGE ELEMENTS

As you have probably realized by now, web pages are much more useful when they are dynamic. In Internet terminology, the word *dynamic* means several things. Primarily, it refers to web pages that respond to user requests through buttons or other kinds of controls. Among other things, a dynamic web page can allow users to change the document background color, submit a form, process a query, and participate in an online game or quiz. The term *dynamic* also refers to various effects, such as animation, that appear automatically in a web browser. In this section, you learn about basic techniques for animating web page elements.

Working with Timeouts and Intervals

As you develop web pages, you may need to have some JavaScript code execute repeatedly, without user intervention. Alternately, you might want to create animation or allow for some kind of repetitive task that executes automatically. For example, you might want to include an advertising image that changes automatically every few seconds. Or, you might want to use animation to change the ticking hands of an online analog clock (in which case each position of the clock hands would require a separate image).

You use the `Window` object's timeout and interval methods to create code that executes automatically. The *`setTimeout()` method* is used in JavaScript to execute code after a specific amount of time has elapsed. Code executed with the `setTimeout()` method executes only once. The syntax for the `setTimeout()` method is `var variable = setTimeout("code", milliseconds);`. This statement declares that the variable will refer to the `setTimeout()` method. The code argument must be enclosed in double or single quotation marks and can be a single JavaScript statement, a series of JavaScript statements, or a function call. The amount of time the web browser should wait before executing the code argument of the `setTimeout()` method is expressed in milliseconds.

Note

A millisecond is one-thousandth of a second; there are 1,000 milliseconds in a second. For example, five seconds is equal to 5,000 milliseconds.

The *`clearTimeout()` method* is used to cancel a `setTimeout()` method before its code executes. The `clearTimeout()` method receives a single argument, which is the variable that represents a `setTimeout()` method call. The variable that represents a `setTimeout()` method call must be declared as a global variable.

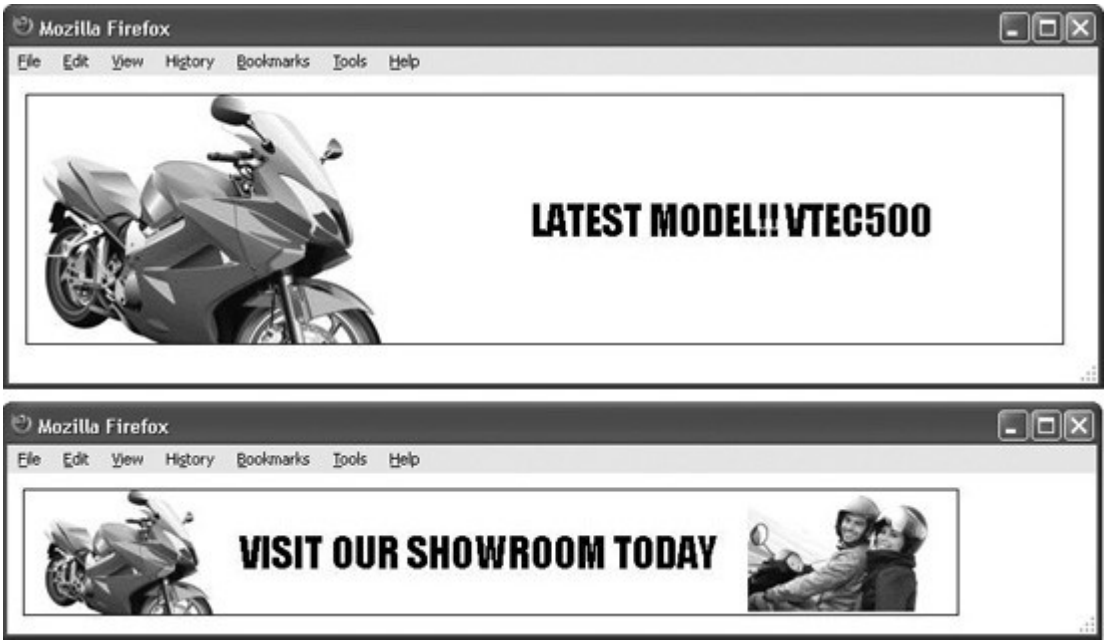
The script section in the following code contains a `setTimeout()` method and a `clearTimeout()` method call. The `setTimeout()` method is set to execute after 10,000 milliseconds (10 seconds) have elapsed. If a user clicks the OK button, the `buttonPressed()` function calls the `clearTimeout()` method.

```
<head>
<script type="text/javascript">
/* <![CDATA[ */
var buttonNotPressed = setTimeout(
    "window.alert ('You must press the OK button to continue!')", 10000);
function buttonPressed() {
    clearTimeout(buttonNotPressed);
    window.alert("The setTimeout() method was cancelled!");
}
/* ]]> */
</script>
</head>
<body>
<form action="">
<input type="button" value=" OK "
   onclick="buttonPressed();" />
</form>
</body>
</html>
```

Two other JavaScript methods that create code and execute automatically are `setInterval()` and `clearInterval()`. The *`setInterval()` method* is similar to the `setTimeout()` method, except that it repeatedly executes the same code after being called only once. The *`clearInterval()` method* is used to clear a `setInterval()` method call in the same fashion that the `clearTimeout()` method clears a `setTimeout()` method call. The `setInterval()` and `clearInterval()` methods are most often used for starting animation code that executes repeatedly. The syntax for the `setInterval()` method is the same as the syntax for the `setTimeout()` method: `var variable = setInterval("code", milliseconds);`. As with the `clearTimeout()` method, the `clearInterval()` method receives a single argument, which is the global variable that represents a `setInterval()` method call.

By combining the `src` attribute of the `Image` object with the `setTimeout()` or `setInterval()` methods, you can create simple animation on a web page. The following code uses the `setInterval()` method to automatically swap the motorcycle images shown in Figure 4.1 every couple of seconds.

Figure 4.1
Banner images animated with the setInterval() method



Ministr-84/Shutterstock.com, auremar/Shutterstock.com

```
<head>
<script type="text/javascript">
/* <![CDATA[ */
var curBanner="cycle1";
function changeBanner() {
   if (curBanner == "cycle2") {
         document.images[0].src = "v500tec.gif";
         curBanner = "cycle1";
   }
   else   {
         document.images[0].src = "showroom.gif";
         curBanner = "cycle2";
   }
}
/* ]]> */
</script>
</head>
<body onload="var begin=setInterval('changeBanner()', 2000);">
<p><img src="v500tec.gif" height="90px" width="700px"
alt="Banner images" /></p>
</body>
```

The simple stationary animations you have seen so far were created by swapping the image files assigned to an `<img>` element's `src` attribute. With DHTML, you can use dynamic positioning to create traveling animation—that is, images that appear to travel across the screen, as described in the following section.

Using Simple Animation with jQuery

One of the most basic of the jQuery visual effect methods is the `.animate()` method. How this method works is by incrementing or decrementing CSS properties that accept numeric values according to a specified duration. You can use the `.animate()` method to animate the following CSS properties:

```
backgroundPosition
borderBottomWidth
borderLeftWidth
borderRightWidth
borderSpacing
borderTopWidth
borderWidth
bottom
font
fontSize
height
Left
letterSpacing
lineHeight
margin
marginBottom
marginLeft
marginRight
marginTop
maxHeight
maxWidth
minHeight
minWidth
outlineWidth
padding
paddingBottom
paddingLeft
paddingRight
paddingTop
right
textIndent
top
width
wordSpacing
```

Caution

Keep in mind that you can use the `.animate()` method with the preceding CSS properties only because they accept numeric values. The `.animate()` method does not work with CSS properties that accept text values, such as `fontFamily`.

Tip

Notice that the CSS properties listed in the preceding table use JavaScript syntax instead of CSS syntax, such as `fontSize` instead of `font-size`. Unlike the jQuery `.css()` property, which allows you to use either JavaScript or jQuery CSS property syntax, you must use JavaScript syntax with the `.animate()` method.

The basic syntax for the `.animate()` method is as follows:

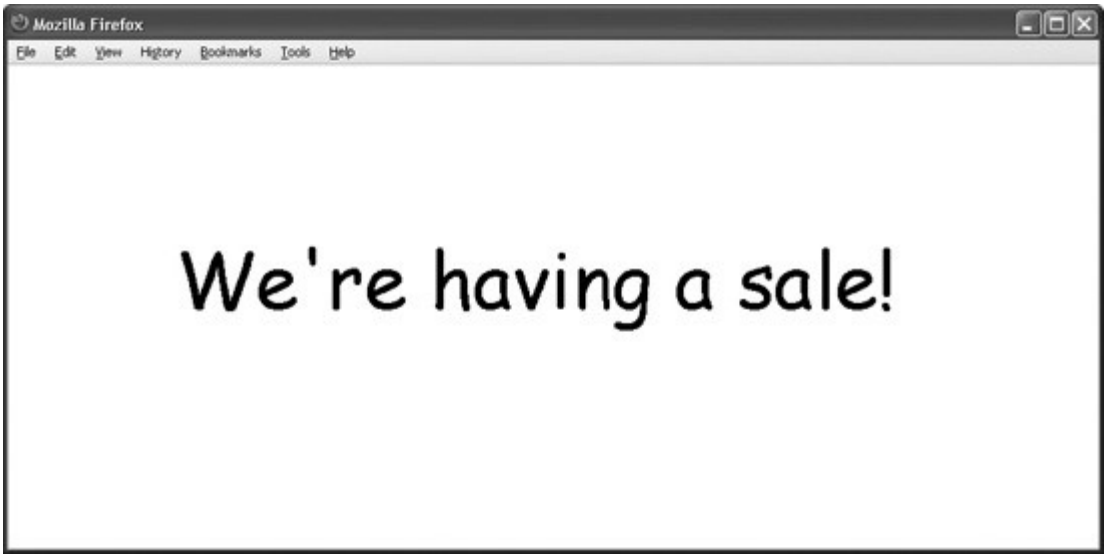
```
(selector).animate({properties}, duration, easing, complete)
```

The *properties* portion of the method contains the list of styles, separated by commas, that you want to change for the selected element. Be sure to notice that the styles are contained within a pair of braces `{ }`. The following code demonstrates how to update multiple styles for a text string. Figure 4.2 shows the starting page.

Figure 4.2
Starting size and position of page animated with .animate()

jQuery Foundation

Figure 4.3
Ending size and position of page animated with .animate()



jQuery Foundation

If you run the code in a browser, the text string will appear to grow and move to the size and shape shown in Figure 4.3.

```
<head>
<script type="text/javascript" src="jquery-1.7.2.js"></script>
<script type="text/javascript">
/* <![CDATA[ */
$(document).ready(function(){
     $("#sale").animate({fontSize:"48pt", marginLeft: "100pt",
     marginTop: "100pt"});
});
/* ]]> */
</script>
</head>
<body>
<p id="sale" style="font-family: Comic Sans MS; font-size: 10pt">
We're having a sale!</p>
</body>
```

The *duration* portion of the `.animate()` method determines the speed in which the style properties change from their starting to their final values. This is essentially the same as the second argument you pass to the JavaScript `setInterval()` method to control the speed of an animation. You can apply a specified number of milliseconds to the `speed` portion of the `.animate()` method or one of three values: `slow`, `normal`, and `fast`. A value of `slow` represents 200 milliseconds, a value of `normal` represents 400 milliseconds, and a value of `fast` represents 600 milliseconds. The default value is `normal`, or 400 milliseconds.

The *easing* portion of the `.animate()` method is not so easy to understand. There are two built-in easing values: “linear” and “swing”. The linear easing value essentially applies the same change intervals to the affected CSS properties, depending on the value assigned to the `speed` portion of the method. For example, if you use the `.animate()` method to change the

`marginLeft` property of an element from 0 to 100, with a speed value of 500 and linear easing, jQuery will change the element's left margin every 1/2 second. You can compare linear easing to a train traveling between two stations at the exact same speed without stopping at either station, with each click of a mile representing an update to the element values. In comparison, the swing easing value starts the animation slowly, speeds up, and then slows down to "ease" the elements into their final position. The swing easing value is more comparable to an airplane, which starts out slowly on the runway, takes off and achieves fast speed, and then slows down when it lands and "eases" into the terminal. In other words, the swing easing value is similar to a child's swing set, which slows down at each end of the swing arc, and significantly speeds up during the progression between each end of the arc. Swing is the default easing value and is usually sufficient for most animations.

Tip

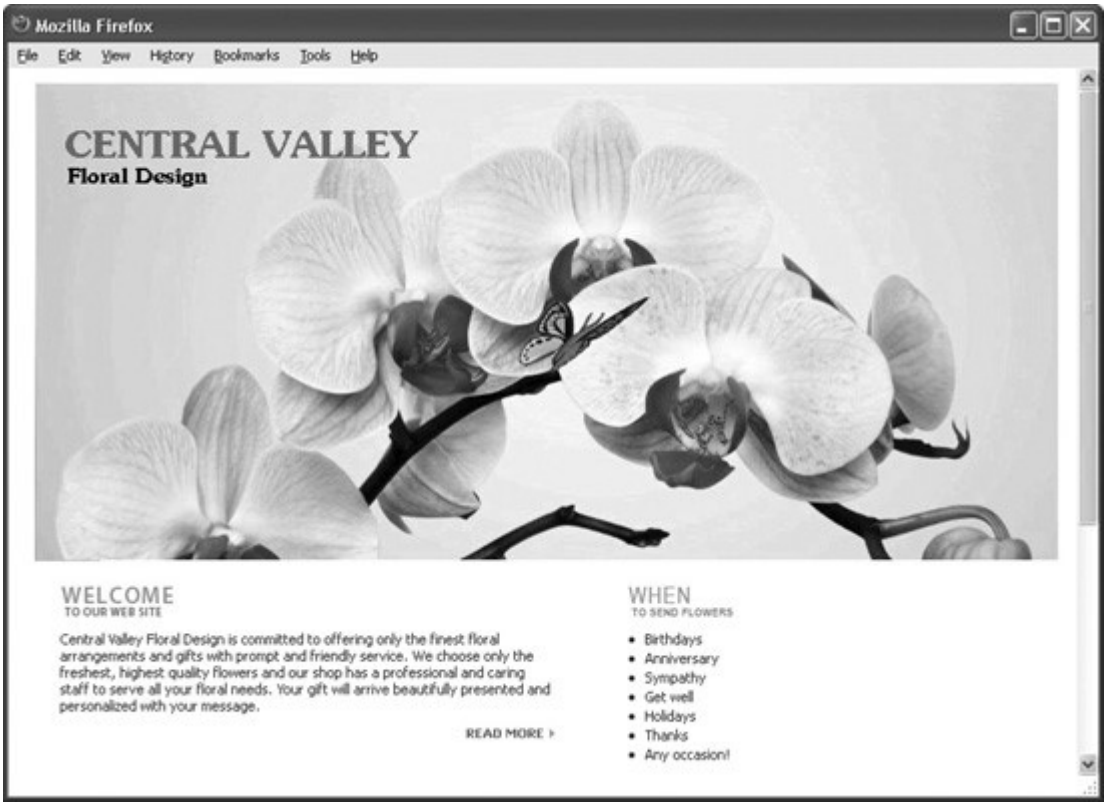
Many more types of easing functions are available as jQuery plug-ins. Refer to Chapter 9, "Extending jQuery with Plug-Ins," for more information on using and developing jQuery plug-ins.

The last argument passed to the `.animate()` method is *`complete`*, which refers to a callback function that is called when the animation completes. Callback functions are further described in the next section.

The following code demonstrates how to create traveling animation with an animated GIF image of a butterfly. The butterfly travels from the lower-left side of the screen, over the paragraph, to the upper-right corner. The global `topPosition` and `leftPosition` variables define the initial starting position of the image. The `$(document).ready()` method calls a JavaScript function named `flyButterfly()`. The first statement in the `flyButterfly()` function uses the `.css()` method to display the butterfly image, which is initially hidden. The second and third statements modify the values assigned to the `topPosition` and `leftPosition` variables, which the function uses to dynamically position the butterfly image.

The `if` statement then calculates the current location of the butterfly. If it's within 200 pixels of the right edge of the window, the image's position is reset to its starting values and then hidden with the `.css()` method. Hiding the image is necessary in this case because if you reset the image's position and it is still visible, it will appear to "zip" across the screen to the starting point. The `.animate()` method then changes the `left` and `top` CSS properties, and then calls the `flyButterfly` function to repeat the method and continue the animation. Figure 4.4 shows how the web page appears in a browser.

Figure 4.4
Butterfly animation web page



mama_mia/Shutterstock.com

```
<head>
<link href="css.css" type="text/css" rel="stylesheet" />
<script type="text/javascript" src="jquery-1.7.2.js"></script>
<script type="text/javascript">
/* <![CDATA[ */
var topPosition = 250;
var leftPosition = -100;
$(document).ready(function() { flyButterfly(); });
  function flyButterfly() {
     $("#butterfly").attr("display", "block");
     topPosition -= 2;
     leftPosition += 10;
     if (leftPosition >= window.innerWidth - 200) {
         topPosition = 250;
         leftPosition = -100;
     $("#butterfly").attr("display", "hide");
     }
     $("#butterfly").animate( {
        left: leftPosition, top: topPosition
     }, 100, "linear", flyButterfly);
   }
/* ]]> */
</script>
</head>
<body>
   <p>
     <img src="butterfly.gif" id="butterfly"
```

```
        style="position: absolute; left: -100px; top: 250px;
        display: "none" alt="Image of a butterfly"
        height="120" width="150" />
    </p>
…
```

Understanding Callback Functions

In the previous section, you saw how to use a callback function with the `.animate()` method to call the `flyButterfly` function to continue the animation. Another important aspect of callback functions is that they ensure an animation completes before executing subsequent lines of code. Like most programming languages, JavaScript executes code statements line-by-line, in the order that they appear in a script section or function. However, animations such as those created with the `.animate()` method usually execute according to the *`duration`* portion of the `.animate()` method. By default, JavaScript will not wait for the animation (or any functionality) to complete before executing the next statement in the queue. This can cause problems if you need to execute JavaScript code after an animation sequence completes.

For example, the following code uses an `.animate()` statement with the CSS `fontsize` property to grow the text `"Memorial Day Sale!"`, contained within a `<p>` element with an ID of `ad`, from 1em to 3em. The second statement uses the `setInterval()` method with the `.toggleClass()` method and CSS visibility property to show and hide the text, simulating a blinking effect.

```
<style type="text/css">
.blink { visibility: hidden }
</style>
<script type="text/javascript">
/* <![CDATA[ */
$(document).ready(function(){
  $("#ad").animate({fontSize: "3em"}, 2000, "linear");
  setInterval("$('#ad').toggleClass('blink')", 500);
/* ]]> */
</script>
</head>
<body>
<p id="ad" style="text-align: center; margin-top: 175px;
font-family: Arial, sans-serif;font-weight: bold;font-size: 1em">
Memorial Day Sale!</p>
</body>
```

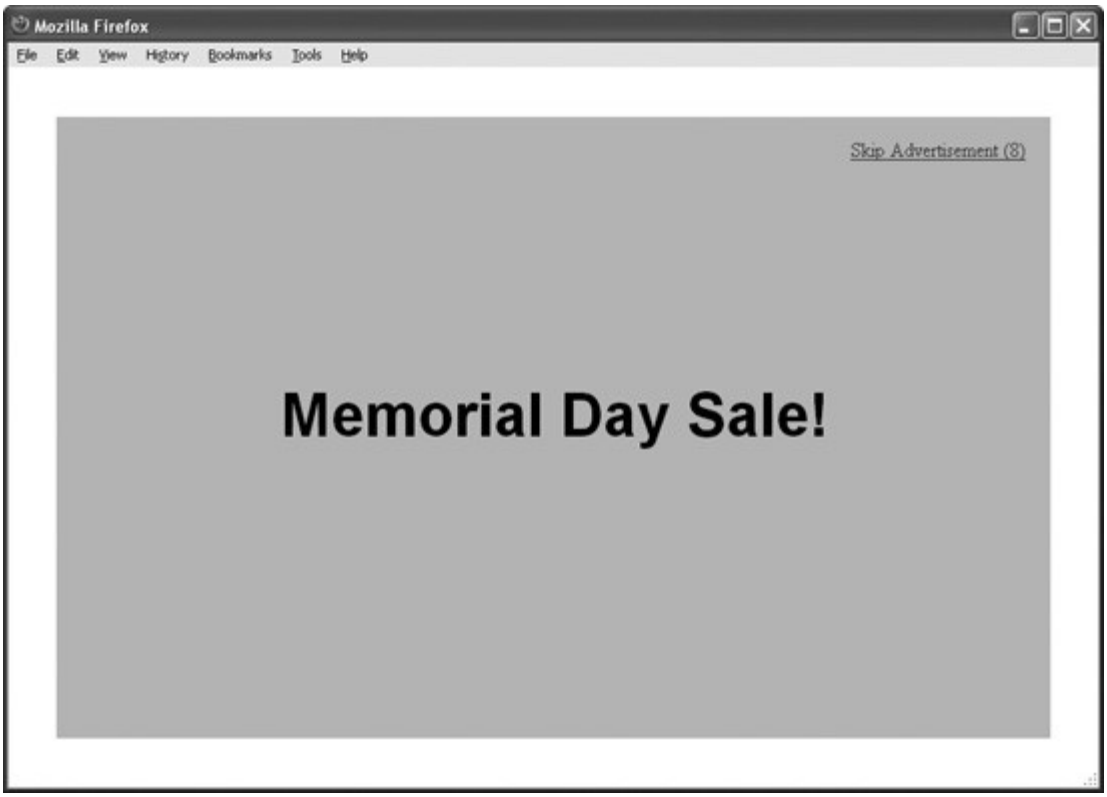
The problem with the preceding code is that the `setInterval()` method will begin executing before the `.animate()` method finishes, causing the text to both grow in size and blink at the same time. Although this result may be desirable in some cases, if you want the blinking effect to run after the `.animate()` method grows the text, you need to call the `setInterval()` method from a callback function, as follows:

```
$(document).ready(function(){
  $("#ad").animate({fontSize: "3em"}, 2000, "linear", function() {
    blinkText = setInterval("$('#ad').toggleClass('blink')", 500);
  });
});
```

The following example presents a simple splash advertisement that displays the animated `"Memorial Day Sale!"` text for 15 seconds before hiding the `<div>` element, represented by an ID of `splash`, that contains it. Visitors can either view the page for 15 seconds or click the Skip Advertisement link, represented by an ID of `a1`, in the upper-right corner.

The first four statements in the `document.ready()` method dynamically position the `<div>` element in the center of the document, according to the dimensions of the user's browser. The next two statements use a `setInterval()` method to dynamically change the value next to the Skip Advertisement link, which represents the number of seconds remaining for the advertisement and is contained within a `<span>` element with an ID of `sp1`. Notice that instead of calling a separate function, the new `setInterval()` method uses an anonymous function to count down and display the number of remaining seconds.

Figure 4.5
Splash advertisement



jQuery Foundation

The `if` statement within the anonymous function contains a conditional expression that determines when a counter variable named `secondsLeft` is equal to 1, at which point the `if` statement calls the `.click()` method for the `anchor` element. The statements within the `.click()` method clear the two `setInterval()` methods, hide the `<div>` element, and return a value of `false` to prevent the link from attempting to jump to another page. Figure 4.5 shows how the page appears in a web browser.

```
<head>
<script type="text/javascript" src="jquery-1.7.2.js"></script>
<style type="text/css">
.blink { visibility: hidden }
</style>
<script type="text/javascript">
/* <![CDATA[ */
var countSeconds;
var blinkText;
$(document).ready(function(){
  var splashElement = $("#splash");
  splashElement.css("position","absolute");
  splashElement.css("top", (($(window).height()
     - splashElement.outerHeight()) / 2)
     + $(window).scrollTop() + "px");
  splashElement.css("left", (($(window).width()
     - splashElement.outerWidth()) / 2)
     + $(window).scrollLeft() + "px");
  var secondsLeft = 15;
  countSeconds = setInterval(function(){
     if (secondsLeft == 1)
        $("#a1").click();
     $("#s1").html(--secondsLeft);
  }, 1000);
  $("#ad").animate({fontSize: "3em"}, 2000, "linear", function() {
    blinkText = setInterval("$('#ad').toggleClass('blink')", 500);
  });
  $("#a1").click(function(){
    clearInterval(blinkText);
    $("#splash").css("visibility", "hidden");
    return false;
  });
});
/* ]]> */
</script>
</head>
<body>
<div id="splash" style="background-color: #99CCCC; height:500; width:800">
<p style="text-align: right; padding-right: 20px">
<a id="a1" href="">Skip Advertisement (<span id="s1">15</span>)</a></p>
<p id="ad" style="text-align: center; margin-top: 175px;
font-family: Arial, sans-serif; font-weight: bold; font-size: 1em">
Memorial Day Sale!</p>
</div>
</body>
```

CREATING DHTML MENUS

Creating menus is one of the more popular uses of DHTML. DHTML menus are most often used for organizing navigational links to other web pages, although they are also useful for displaying and hiding information. This section discusses basic expandable and navigation menus and the next section explains how to create sliding menus.

Building an Expandable Menu

The CSS `display` property specifies whether to display an element on a web page.

You can use the `display` property to simulate expandable and collapsible menus on a web page. You typically use the `display` property with a block-level element, which gives a web page its structure. Most web browsers render block-level elements so that they appear on their own line. Block-level elements can contain other block-level elements or inline elements. The `<div>`, `<p>`, and heading elements (`<h1>`, `<h2>`, and so on) are examples of common block-level elements. Inline elements, or text-level elements, describe the text that appears on a web page. Unlike block-level elements, inline elements do not appear on their own lines; instead, they appear within the line of the block-level element that contains them. Examples of inline elements include the `<b>` (bold) and `<br />` (line break) elements. By placing elements and text within a `<div>` element, you can use the `display` property to simulate expandable and collapsible menus.

If you assign a block-level element's `display` property a value of `none`, the associated element is not displayed. In fact, the web page does not even allocate space for the element on the page. However, if you assign a value of `block` to a block-level element's `display` property, the web page is reformatted to allocate sufficient space for the element and its contents, which are then displayed.

The following code shows a web page that displays Hall of Fame players for the National Football League. The style section defines a class selector named `collapsed` for the `<div>` element. The `collapsed` class selector includes the `display` property, which turns off the display of each `<div>` element when the web page is first rendered. This example uses two functions in the jQuery section for the `mouseover` and `mouseout` event handlers. Notice how the single statements within each function identify the `<div>` element to expand or collapse. Each statement uses `$("#" + $(this).attr("id(")` to identify the calling paragraph, and then appends `" + div"` to create a `Next Adjacent ("prev + next")` hierarchical selector. The `.css()` methods appended to each hierarchical selector then use the `display` property to either display or hide the `<div>` element. Figure 4.6 shows the document in a web browser when the mouse pointer passes over the Cleveland Browns link.

```
<head>
<style type="text/css">
  div.collapsed
  {
     display: none;
  }
</style>
<script type="text/javascript" src="jquery-1.7.2.js"></script>
<script type="text/javascript">
/* <![CDATA[ */
$(document).ready(function(){
  $("p").mouseover(function() {
    $("#" + $(this).attr("id") + " + div").css("display", "block");
  });
  $("p").mouseout(function() {
    $("#" + $(this).attr("id") + " + div").css("display", "none");
  });
});
```

```
/* ]]> */
</script>
</head>
<body>
   <h1>
     National Football League</h1>
   <h2>
     Hall of Fame Players</h2>
   <p id="billsMenu"><a href="">
     Buffalo Bills</a></p>
   <div id="bills" class="collapsed">
     <p>Joe DeLamielleure '03, Jim Kelly '02, Marv Levy ' 01 (coach),
     James Lofton '03, Billy Shaw '99, Thurman Thomas ' 07</p>
   </div>
   <p id="brownsMenu"><a href="">
     Cleveland Browns</a></p>
   <div id="browns" class="collapsed">
     <p>Doug Atkins '82, Jim Brown '71, Paul Brown '67 (coach/owner),
     Willie Davis '81, Len Dawson '87, Joe DeLamielleure '03, Len Ford '76,
     Frank Gatski '85, Otto Graham '65, Lou Groza '74, Gene Hickerson '07,
     Henry Jordan '95, Leroy Kelly '94, Dante Lavelli '75,
     Mike McCormack '84, Tommy McDonald '98, Bobby Mitchell '83,
     Marion Motley '68, Ozzie Newsome '99, Paul Warfield ' 83,
     Bill Willis '77</p>
  </div>
</body>
```

jQuery includes two methods, `.show()` and `.hide()`, that simplify displaying and hiding elements. To use either method, append it to the selector that you want to show or hide, as the following jQuery code for the Hall of Fame players demonstrates:

Figure 4.6
Web page with expandable menus

jQuery Foundation

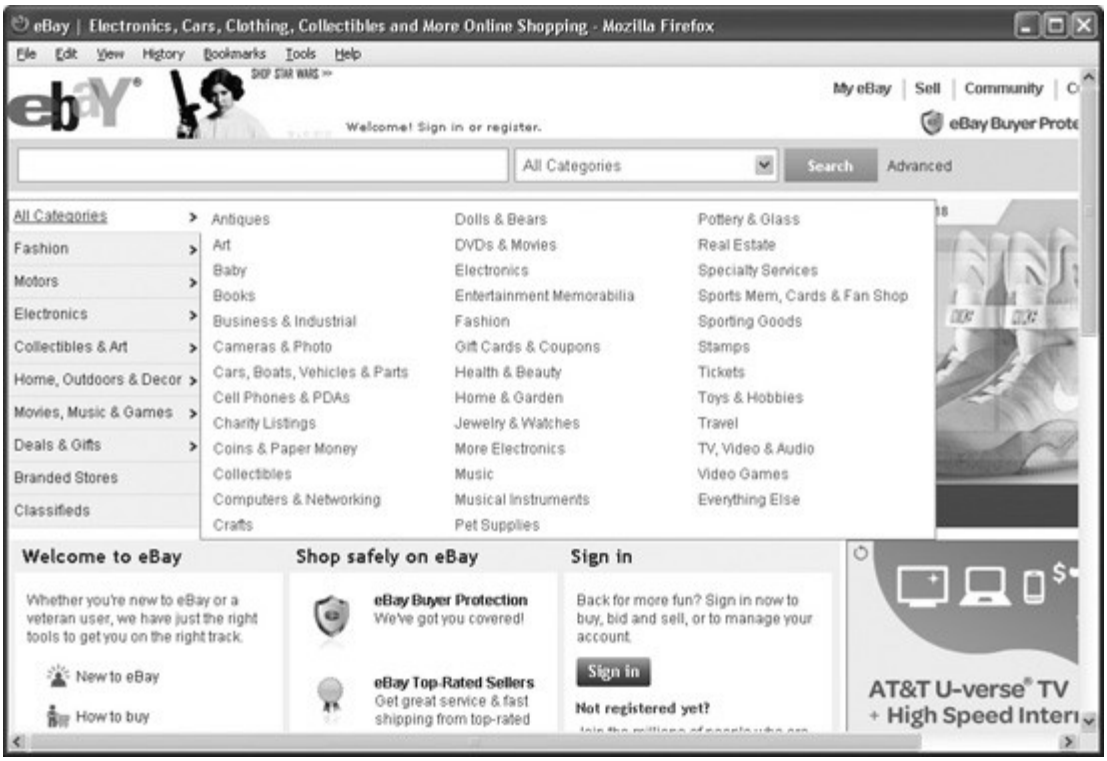
```
$(document).ready(function(){
  $("p").mouseover(function() {
    $("#" + $(this).attr('id') + " + div").show();
  });
  $("p").mouseout(function() {
    $("#" + $(this).attr('id') + " + div").hide();
  });
});
/* ]]> */
```

Constructing Navigation Menus

You are probably already familiar with drop-down menus, or pull-down menus, which are similar to the menus you find in a Windows application. Menus can greatly improve the design of your web page, and they help visitors navigate through your website. Figure 4.7 shows a navigation menu that helps visitors to eBay's website locate specific shopping categories.

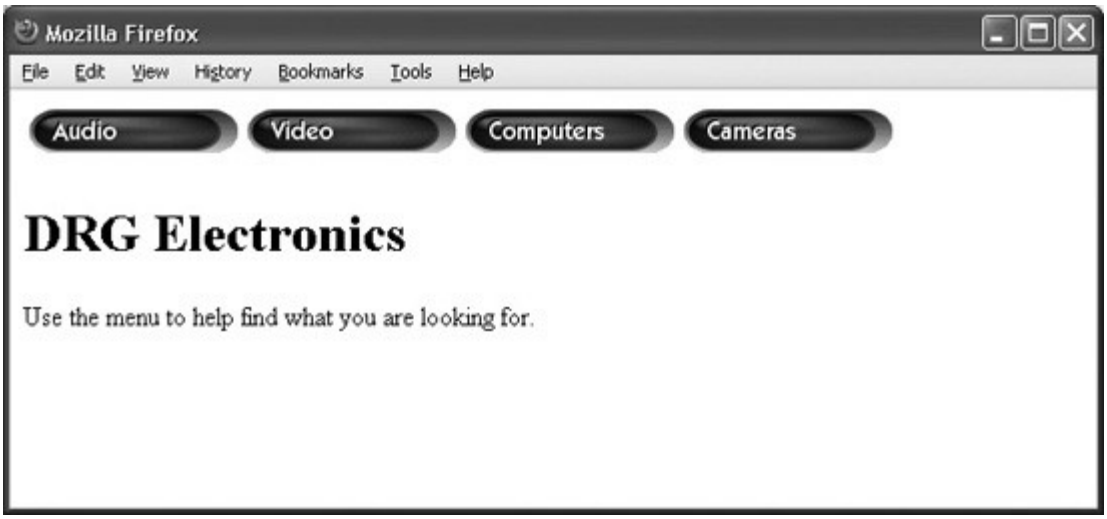
Although you can create a navigation menu in several ways, the easiest way is to use a table to contain your menu items. First, you create a master table whose purpose is to contain nested tables for each individual menu. The following code shows the beginnings of a table that will create a navigation menu for an electronics store. Figure 4.8 shows the document in a web browser.

Figure 4.7
Navigation menu on eBay



jQuery Foundation

Figure 4.8
Document with a top navigation menu



jQuery Foundation

```
<table>
   <tr align="left">
     <td>
       <a href="">
```

```
      <img src="button1up.png" id="b1"
          border="0" vspace="1"
          hspace="1" />
      </a></td>
    <td>
       <a href="">
       <img src="button2up.png" id="b2"
           border="0" vspace="1"
           hspace="1" /></a>
    </td>
    <td>
      <a href="">
      <img src="button3up.png" id="b3"
           border="0" vspace="1"
           hspace="1" /></a>
    </td>
    <td>
      <a href="">
      <img src="button4up.png" id="b4"
           border="0" vspace="1"
           hspace="1" /></a>
    </td>
  </tr>
</table>
<h1>
  DRG Electronics</h1>
<p>
  Use the menu to help find what you
  are looking for.</p>
```

You nest the contents of a navigation menu within the same cell as the top navigation menu heading. The following code shows a `<div>` element in which the Audio menu items are nested within the same cell as the Audio menu:

```
<td
  onmouseover="document.getElementById(
  ('b1').src='button1over.png'"
  onmouseout="document.getElementById(
  ('b1').src='button1up.png'"><a href=""><br />
  <img src="button1up.png" id="b1" border="0"
    vspace="1" hspace="1" /></a>
  <div class="dropmenu">
     <ul>
       <li><a href="audiosys.html">
          Audio Systems</a></li>
       <li><a href="audioplayers.html">
          iPods and MP3 Players</a></li>
       <li><a href="headphones.html">
          Headphones</a></li>
     </ul>
  </div>
</td>
```

To show and hide each menu, you use the `display` property, as shown in the following version of the table elements for the Audio menu. This time, the `<div>` element containing the menu

includes a `style` property that hides the element and sets its position to absolute. In the following code, the `mouseover` and `mouseout` event handlers in the jQuery section use the `.show()` and `.hide()` methods to handle the display of the menu. Note that the `mouseover` and `mouseout` event handlers also include statements that modify the menu buttons. Therefore, when the mouse passes over each image, it is replaced with a more vivid one and then changes back to the original image when the mouse pointer is removed. Figure 4.9 shows the web page in a browser with the mouse pointer over the Audio menu.

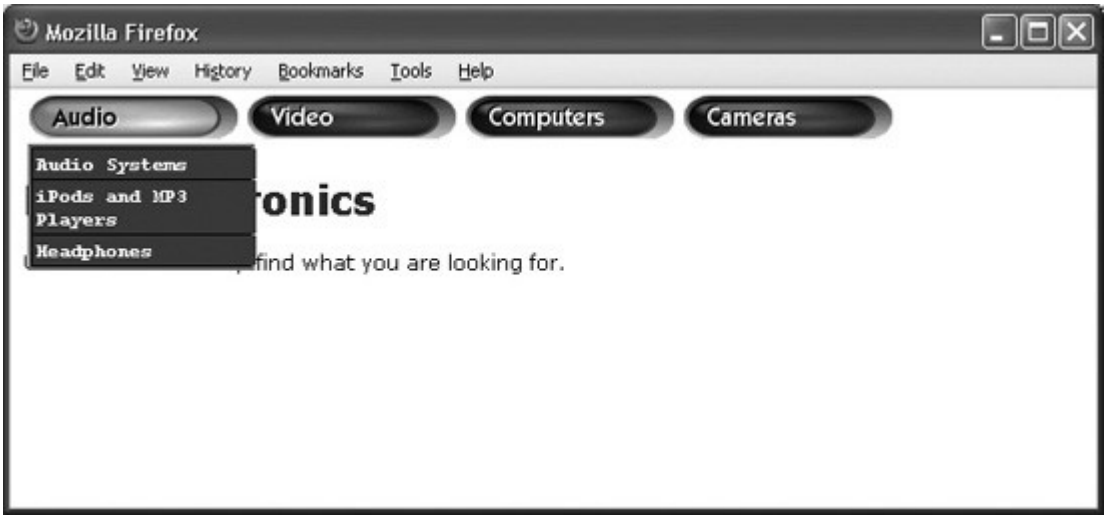
```
<head>
  <link rel="stylesheet" type="text/css" href="menustyle.css" />
<script type="text/javascript" src="jquery-1.7.2.js"></script>
<script type="text/javascript">
/* <![CDATA[ */
$(document).ready(function(){
  $("tr").mouseover(function(){
    $("#audio").show();
    $("#b1").attr("src", "button1over.png");
  });
  $("tr").mouseout(function(){
    $("#audio").hide();
    $("#b1").attr("src", "button1up.png");
  });
});
/* ]]> */
</script>
</head>
<body>
<table>
  <tr align="left">
    <td>
       <a href="">
    <img src="button1up.png" id="b1"
      border="0" vspace="1"
      hspace="1" />
    </a>
    <div id="audio" class="dropmenu"
    style="display:none; position:absolute">
    <ul>
       <li><a href="audiosys.html">Audio Systems</a></li>
       <li><a href="audioplayers.html">iPods and MP3 Players</a></li>
       <li><a href="headphones.html">Headphones</a></li>
    </ul>
    </div>
 </td>
```

In addition to the `.show()` and `.hide()` methods, which simplify displaying and hiding elements, jQuery includes a related method named `.toggle()`. When used without any arguments, the `.toggle()` method simply toggles between displaying and hiding the specified elements, as shown with the following version of the jQuery code for the Audio menu:

```
$(document).ready(function(){
  $("#b1").mouseover(function(){
     $("#audio").toggle();
```

```
      $(this).attr("src", "button1over.png");
   });
   $("#b1").mouseout(function(){
      $("#audio").toggle();
      $(this).attr("src", "button1up.png");
   });
});
```

Figure 4.9
Audio navigation menu



jQuery Foundation

USING SLIDING FUNCTIONALITY

Sliding refers to an animation technique that causes elements to appear to slide open or closed. As their name implies, *sliding menus* are menus that appear to slide open and closed. To create a simple sliding menu, you can use the `.show()`, `.hide()`, and `.toggle()` methods to perform simple animation by passing to them the same *`speed`*, *`duration`*, and *`complete`* arguments that the `.animate()` method accepts. By default, these methods show and hide selected elements immediately. However, you can assign to each method the same speed values that you assign to the `.animation()` method: "`slow`" (200 milliseconds), "`normal`" (400 milliseconds), "`fast`" (600 milliseconds), or any number representing milliseconds. For example, the `.toggle()` methods from the Audio menu you saw in the preceding code show and hide the menu immediately. By adding a `speed` argument to the `.toggle()` method, you can turn the menu into a sliding menu, which appears to slide open and closed when the mouse passes over it. The following version of the jQuery code creates a sliding menu by passing a speed argument of `"slow"` to the `.toggle()` methods:

```
<head>
  <link rel="stylesheet" type="text/css" href="menustyle.css" />
<script type="text/javascript" src="jquery-1.7.2.js"></script>
<script type="text/javascript">
```

```
/* <![CDATA[ */
$(document).ready(function(){
  $("tr").mouseover(function(){
    $("#audio").show();
    $("#b1").attr("src", "button1over.png");
  });
  $("tr").mouseout(function(){
    $("#audio").hide();
    $("#b1").attr("src", "button1up.png");
  });
});
/* ]]> */
</script>
</head>
<body>
<table>
   <tr align="left">
     <td>
       <a href="">
       <img src="button1up.png" id="b1"
          border="0" vspace="1"
          hspace="1" />
       </a>
       <div id="audio" class="dropmenu"
       style="display:none; position:absolute">
       <ul>
          <li><a href="audiosys.html">Audio Systems</a></li>
          <li><a href="audioplayers.html">iPods and MP3 Players</a></li>
          <li><a href="headphones.html">Headphones</a></li>
       </ul>
       </div>

$(document).ready(function(){
  $("#b1").mouseover(function(){
    $(this).attr("src", "button1over.png");
    $("#audio").toggle("slow");
  });
  $("#b1").mouseout(function(){
    $(this).attr("src", "button1up.png");
    $("#audio").toggle("slow");
  });
});
```

jQuery includes three additional methods for creating sliding functionality: `.slideDown()`, `.slideUp()`, and `.slideToggle()`. These methods are almost identical to the `.show()`, `.hide()`, and `.toggle()` methods. The difference between these two sets of methods is that the `.show()`, `.hide()`, and `.toggle()` methods show and hide elements immediately by default, whereas the `.slideDown()`, `.slideUp()`, and `.slideToggle()` methods show and hide selected elements using a speed of 400 milliseconds by default. The following shows an example of the Audio menu code using the `.slideToggle()` method:

```
$(document).ready(function(){
  $("#b1").mouseover(function(){
    $(this).attr("src", "button1over.png");
```

```
    $("#audio").slideToggle();
  });
  $("#b1").mouseout(function(){
    $(this).attr("src", "button1up.png");
    $("#audio").slideToggle();
  });
});
```

USING FADING TECHNIQUES

Fading refers to an animation technique that causes the opacity of elements to fade in or out. jQuery includes four basic methods to handle fading: `.fadeIn(), .fadeOut(), .fadeTo()`, and `.fadeToggle()`. The `.fadeIn(), .fadeOut()`, and `.fadeToggle()` methods are very similar in form to the `.slideDown(), .slideUp()`, and `.slideToggle()` methods, except that they change the opacity instead of the position of selected elements. With the exception of the `.fadeTo()` method, all of the fading methods accept the same *`duration, easing`*, and *`complete`* arguments that the `.animate()` and sliding methods accept.

At the beginning of this chapter, you saw how to use the basic JavaScript `setTimeout()` and `setInterval()` methods in order to help you understand the challenges involved with using basic JavaScript to handle animation. Consider the following version of the motorcycles banner website you saw at the beginning of this chapter. Instead of simply swapping the images out for each iteration of the function, each image gradually fades in and out before changing. In the `changeBanner()` function, the `.fadeIn()` method first fades the currently displayed image and the final statement uses the `.fadeOut()` method to show the new version.

```
<head>
<script type="text/javascript" src="jquery-1.7.2.js"></script>
<script type="text/javascript">
/* <![CDATA[ */
$(document).ready(function() { changeBanner(); });
  function changeBanner() {
    var bannerVar = $("#banner");
    bannerVar.fadeIn(2000);
    if (bannerVar.attr("src") == "showroom.jpg")
        bannerVar.attr("src", "v500tec.jpg");
    else
        bannerVar.attr("src", "showroom.jpg");
    bannerVar.fadeOut(2000, changeBanner);
}
/* ]]> */
</script>
</head>
<body>
<p><img id="banner" src="showroom.jpg" height="90px" width="700px"
alt="Banner images" /></p>
</body>
```

The final jQuery fading method is `.fadeTo()`, which changes the opacity of selected elements to a specified value. The `.fadeTo()` method requires two arguments—*`duration`* and *`opacity`*—and also supports an optional callback argument. The *`duration`* argument specifies how long the

animation should last, in milliseconds. The *opacity* argument determines the final opacity of the selected elements. You can assign a value from 0.0 to 1.0 to the `opacity` argument; the lower the value the more transparent the element becomes. A value of 0.0 makes the element completely transparent, whereas a value of 1.0 makes the element completely opaque.

The following page demonstrates how to use the `.fadeTo()` method. The page includes an image of the Golden Gate bridge. When you place your mouse over the image, a text box fades in and displays some information about the bridge, and then fades out when you move your mouse off of the image. The `<div>` element containing the text is initially assigned an opacity value of 0.0, making the element completely transparent. A `.hover()` method in the `document.ready()` method uses the `.fadeTo()` method to change the opacity of the `<div>` element to a value of 0.7 when the mouse is placed over it. The `.fadeTo()` method's callback argument then calls another `.fadeTo()` method to reset the element's opacity back to 0.0 when the mouse moves off the element. Figure 4.10 shows the image with the mouse off of it and Figure 4.11 shows the image with the mouse placed over it.

Figure 4.10
Web page with a element's opacity set to 0.0

Lien/Shutterstock.com

Figure 4.11
Web page with a element's opacity set to 0.7

Henry Lien/Shutterstock.com

```
<head>
<script type="text/javascript" src="jquery-1.7.2.js"></script>
<script type="text/javascript">
$(document).ready(function(){
  $("#imageDiv").hover(function(){
    $("#messageBox").fadeTo(500, 0.7);
  },function(){
    $("#messageBox").fadeTo(500, 0);
  });
});
</script>
<style>
div.imageDivStyle {
   position: relative;
   width: 423px;
   height: auto
}
div.messageBoxStyle{
   opacity: 0.0;
   width: 403px;
   position: absolute;
   bottom: 0px;
   left: 0px;
   background-color: black;
   font-family: tahoma;
   font-size: 14px;
   color: white;
```

```
   padding: 10px
}
</style>
</head>
<body>
  <div id="imageDiv" class="imageDivStyle">
    <img src="ggbridge.jpg" width="423px" height="270px"
      alt="Photo of the Golden Gate Bridge" />
    <div id="messageBox" class="messageBoxStyle">
      The Golden Gate Bridge spans the opening of San Francisco Bay
      into the Pacific Ocean. Construction began in January, 1933
      and the bridge was finished in April, 1937, at a cost of $35
      million. The bridge celebrated its 75th anniversary in 2012.</div>
  </div>
</body>
```

UNDERSTANDING JQUERY QUEUE FUNCTIONS

Earlier in this chapter, you learned how to use callback functions to schedule an animation to run when the current animation finishes. You can also use the queue methods listed in Table 4.1 to control when animations execute in jQuery.

jQuery has a built-in effects queue named `fx` and effect methods such as `.animate()`, `.fadeIn()`, and `.fadeOut()` are automatically added to this queue and executed in consecutive order. For example, with the splash page, the following methods are executed on the text in the order in which they are encountered. In other words, each method will wait its turn before running. After the font size is increased with the `.animate()` method, the text fades out and then fades back in with the `.fadeOut()` and `.fadeIn()` methods.

Table 4.1 jQuery Queue Methods

Method	Description
`.clearQueue()`	Clears all remaining functions in the effects queue.
`.delay()`	Delays the execution of queued effect function by a specified number of milliseconds.
`.dequeue()`	Executes a function and removes it from the effects queue.
`.queue()`	Displays functions in or adds functions to the effects queue.
`.stop()`	Stops the currently running function in the animation queue.

```
$("#ad").animate({fontSize: "3em"}, 2000, "linear");
$("#ad").fadeOut(2000);
$("#ad").fadeIn(2000);
```

Note

The `fx` queue is sufficient for most web pages. For greater control over your animations, you can create your own queue. By default, each of the methods in Table 4.1 operates on the `fx` queue. However, you can pass to each method a parameter with the name of a custom queue in

quotations. For example, to specify a custom queue named `customEffects` with the `.queue()` method, you use `.queue("customEffects")`. Remember that unless you specify a custom queue name, each of the queue methods will operate on the `fx` queue by default.

If you want to run a custom animation, such as dynamically changing a CSS property, the animation will start before any running built-in effect methods are finished executing. The last statement in following code executes the blinking functionality you saw earlier. However, it will begin executing while the font size of the text is still being increased with the `.animate()` method.

```
$("#ad").animate({fontSize: "3em"}, 2000, "linear");
$("#ad").fadeOut(2000);
$("#ad").fadeIn(2000);
blinkText = setInterval("$('#ad').toggleClass('blink')", 500);
```

To resolve this issue, you need to manually add a function containing the `setInterval()` statement to the `fx` queue with the `.queue()` method. Then, you need to call the `.dequeue()` method to execute the function and remove it from the `fx` queue. The following demonstrates how to use the queue methods with the `setInterval()` statement:

```
$("#ad").animate({fontSize: "3em"}, 2000, "linear");
$("#ad").fadeOut(2000);
$("#ad").fadeIn(2000);
$("#ad").queue(function() {
  blinkText = setInterval("$('#ad').toggleClass('blink')", 500);
  $(this).dequeue
});
```

The `.delay()` method is used for delaying the execution of queued effect function. To use the `.delay()` method, you chain it to a selector, followed by an effect method. You pass to the `.delay()` method the number of milliseconds you want to delay function execution, or the values `fast` for 200 milliseconds or `slow` for 600 milliseconds. For example, the following statements use the `.delay()` method to delay execution of the `.fadeOut()` and `.fadeIn()` methods on the splash page by 1,000 milliseconds (one second).

```
$("#ad").delay(1000).fadeOut(2000);
$("#ad").delay(1000).fadeIn(2000);
```

The `.queue()` method gives you more flexibility than using callback functions because it allows you to schedule multiple functions to execute for an element. The following code adds a function that changes the text to `"Big Savings!"` and restarts the blinking functionality, but with a faster interval. Notice that the new function uses the `.delay()` method to delay execution for 3,000 milliseconds, which gives the preceding function three seconds to execute. Then the function clears the `blinkText` interval and reuses the variable to restart the blinking animation with a faster interval of 200.

```
$("#ad").animate({fontSize: "3em"}, 2000, "linear");
$("#ad").delay(1000).fadeOut(2000);
$("#ad").delay(1000).fadeIn(2000);
```

```
$("#ad").queue(function() {
  blinkText = setInterval(function(){
    $("#ad").toggleClass("blink");
  }, 500);
  $(this).dequeue();
});
$("#ad").delay(3000).queue(function() {
  clearInterval(blinkText);
  $(this).text("Big Savings!");
  blinkText = setInterval(function(){
    $("#ad").toggleClass("blink");
  }, 200);
  $(this).dequeue();
});
```

The final two jQuery queue methods are `.stop()` and `.clearQueue()`. The `.stop()` method stops the currently running animation in the effects queue, whereas the `.clearQueue()` method clears the effects queue of any remaining animation functions. On the splash page, you need to include both `.stop()` and `.clearQueue()` in the `.click()` method for the element that displays the text, as follows. This ensures that all animations stop and are cleared from the queue when the user clicks the Skip Advertisement link.

```
$("#a1").click(function(){
  $("#ad").stop();
  $("#ad").clearQueue();
  clearInterval(blinkText);
  $("#splash").css("visibility", "hidden");
  return false;
});
```

Tip

The `.stop()` method includes two additional parameters—`clearQueue` and `jumpToEnd`. The `clearQueue` parameter clears the effects queue, the same as the `.clearQueue()` method, and the `jumpToEnd` parameter jumps to the end of the currently running animation instead of simply stopping it. Both parameters default to `false`. To enable these parameters, pass values of `true` to the `.stop()` method, separated by commas. For example, `.stop(true, true)` enables both parameters. If you are using a custom queue, be sure to pass these parameters after the name of the queue parameter as follows: `.stop("customEffects", true, true)`.

Here is a completed version of the splash page:

```
<head>
<script type="text/javascript" src="jquery-1.7.2.js">
</script>
<style type="text/css">
.blink { visibility: hidden }
</style>
<script type="text/javascript">
/* <![CDATA[ */
var countSeconds;
var blinkText;
```

```
$(document).ready(function(){
  var splashElement = $("#splash");
  splashElement.css("position","absolute");
  splashElement.css("top", (($(window).height()
    - splashElement.outerHeight()) / 2)
    + $(window).scrollTop() + "px");
  splashElement.css("left", (($(window).width()
    - splashElement.outerWidth()) / 2)
    + $(window).scrollLeft() + "px");
  var secondsLeft = 15;
  countSeconds = setInterval(function(){
    if (secondsLeft == 1)
        $("#a1").click();
    $("#s1").html(-secondsLeft);
  }, 1000);
  $("#ad").animate({fontSize: "3em"}, 2000, "linear");
  $("#ad").delay(1000).fadeOut(2000);
  $("#ad").delay(1000).fadeIn(2000);
  $("#ad").queue(function() {
    blinkText = setInterval(function(){
      $("#ad").toggleClass("blink");
    }, 500);
    $(this).dequeue();
  });
  $("#ad").delay(3000).queue(function() {
    clearInterval(blinkText);
    $(this).text("Big Savings!");
    blinkText = setInterval(function(){
      $("#ad").toggleClass("blink");
    }, 200);
    $(this).dequeue();
  });
  $("#a1").click(function(){
    $("#ad").stop();
    $("#ad").clearQueue();
    clearInterval(blinkText);
    $("#splash").css("visibility", "hidden");
    return false;
  });
});
/* ]]> */
</script>
</head>
<body>
<div id="splash" style="background-color: #99CCCC; height:500; width:800">
<p style="text-align: right; padding-right: 20px">
<a id="a1" href="">Skip Advertisement (<span id="s1">15</span>)</a></p>
<p id="ad" style="text-align: center; margin-top: 175px;
font-family: Arial, sans-serif; font-weight: bold; font-size: 1em">
Memorial Day Sale!</p>
</div>
</body>
```

SUMMARY

In this chapter, you learned about the various animation techniques available with jQuery, including how to use timeouts and intervals, create simple animation, and how to use callbacks. You also learned how to create DHTML menus, use sliding and fading techniques, and use queue functions. You will continue to apply these techniques in upcoming chapters.