CHAPTER 11

Manipulating the Browser Object Model

In this chapter, you will:

- Study the browser object model
- Work with the `Window` object
- Study the `History`, `Location`, `Navigator`, and `Screen` objects

In some situations, you may need to use JavaScript to control the Web browser. For example, you might want to change the Web page being displayed or write information to the Web browser's status bar. Or, you may want to control elements of the Web page itself. To control the Web browser window or the Web page, you use the browser object model. This chapter discusses the components of the browser object model.

Understanding the Browser Object Model

The **browser object model (BOM)**, or **client-side object model**, is a hierarchy of objects, each of which provides program access to a different aspect of the Web browser window or the Web page. You can use the methods and properties of objects in the browser object model to manipulate the window and elements displayed in a Web browser. The most basic objects in the browser object model are illustrated in Figure 11-1.

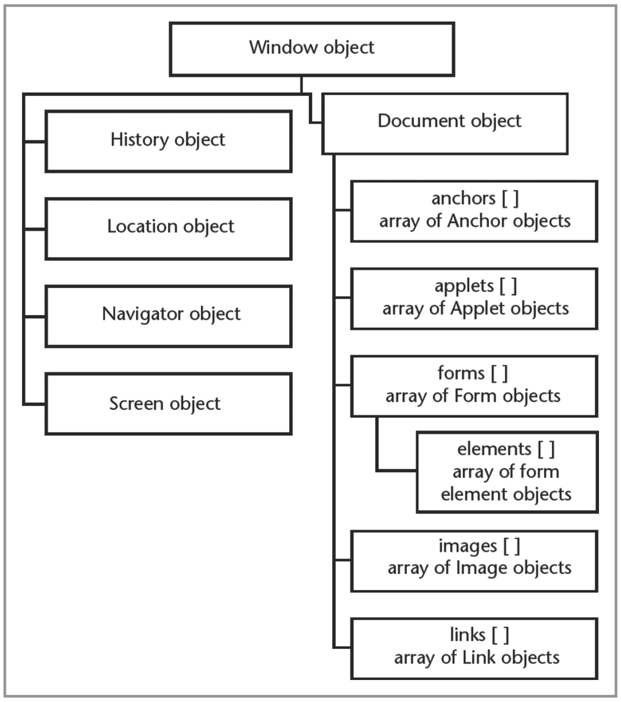


Figure 11-1 Browser object model

The browser object model is also called the JavaScript object model or the `Navigator` object model. However, other scripting technologies, such as VBScript, can also control aspects of the Web browser window or Web page. Therefore, the term *browser object model* or *client-side object model* is more accurate.

The concept of object models is fairly complex. You do not need to understand the details of working with object models to work with the browser object model in JavaScript. Instead, you should simply understand that object models define groups of interrelated objects.

You do not have to create any objects or arrays explicitly in the browser object model; they are created automatically when a Web browser opens a Web page. The top-level object in the browser object model is the **`Window` object**, which represents a Web browser window. The Web browser automatically creates the `Window` object for you. The `Window` object is called the **global object** because all other objects in the browser object model are contained within it. For example, the `Window` object contains the `Document` object, just as a Web browser window contains a Web page document. You use the methods and properties of the `Window` object to control the Web browser window, and you use the methods and properties of the `Document` object to control the Web page. Figure 11-2 illustrates the concepts of the `Window` object and the `Document` object.

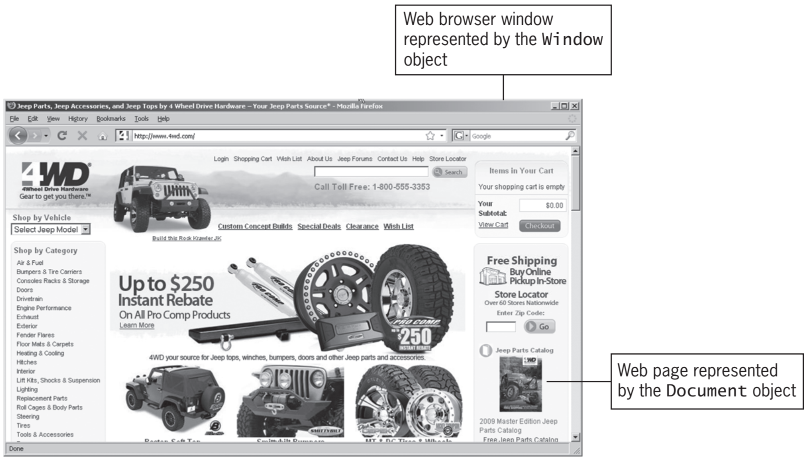

Web browser window represented by the Window object
Web page represented by the Document object
Jeep Parts, Jeep Accessories, and Jeep Tops by 4 Wheel Drive Hardware – Your Jeep Parts Source* - Mozilla Firefox
http://www.4wd.com/
Call Toll Free: 1-800-555-3353
Up to $250 Instant Rebate On All Pro Comp Products
Free Shipping
Store Locator
Done

Figure 11-2 `Window` object and `Document` object

Using the `Document` Object

The `Document` object is arguably the most important object in the browser object model because it represents the Web page displayed in a browser. You are already familiar with the `write()` and `writeln()` methods, which refer to the `Document` object. The statement `document.write("Go Patriots!");` adds the text "Go Patriots!" to a Web page when it is rendered by a Web browser. All elements on a Web page are contained within the `Document` object, and each element is represented in JavaScript by its own object. This means that the `Document` object contains all of the elements you create on a Web page. For example, the `Form` object, which is used by JavaScript to represent forms created with the `<form>` element, is contained within the `Document` object, which is contained within the `Window` object. The `Radio` object, which is used by JavaScript to represent a radio button created with an `<input>` element, is contained within the `Form` object, which is contained within the `Document` object, which is contained within the `Window` object.

In this book, objects in the browser object model are referred to with an initial uppercase letter (`Document` object). However, when you use the object name in code, you must use a lowercase letter. For example, the following statement refers to the `Document` object: `document.write("Go Patriots!");`. Note the use of the lowercase *d* in `document`.

The `Document` object branch of the browser object model is represented by its own object model called the Document Object Model, or DOM. You will learn more about the DOM in Chapter 13.

Referencing JavaScript Objects

Some of the objects in the browser object model represent arrays. In Figure 11-1, objects that are arrays are followed by brackets, such as `forms[]` or `images[]`. The arrays contain objects created from the corresponding elements on a Web page. For example, the `images[]` array contains `Image` objects that represent all the `<img>` elements on a Web page. `Image` objects for each `<img>` element are assigned to the elements of the `images[]` array in the order that they appear on the Web page. The first `Image` object is represented by `images[0]`, the second `Image` object is represented by `images[1]`, and so on.

As you learned in Chapter 8, you can use JavaScript to reference any element on a Web page by using periods to append the element's name to the name of any elements in which it is nested, starting with the `Document` object. For elements that are represented by arrays, you can reference the object through the array instead of with the element name. Consider an `Image` object, which contains a `src` property that contains the URL assigned to an `<img>` element's `src` attribute. Assuming that the image is assigned a name of `companyLogo`, use the following code to display the image's URL in an alert dialog box:

```
<img src="company_logo.gif" name="companyLogo"
    height="100" width="200" onclick="window.alert(↵
    'This image is located at the following URL: '
    + document.companyLogo.src);"
    alt="Image of a company logo." />
```

Instead of referencing the image by name, you can access it through the `images[]` array. The following `<img>` element includes an `onclick` event handler that uses the `Document` object to display the image's URL in an alert dialog box. The code assumes that the image is the first one on the page by referencing the first element (0) in the `images[]` array.

```
<img src="company_logo.gif" height="100" width="200"
     onclick="window.alert(↵
    'This image is located at the following URL:'↵
     + document.images[0].src);"
     alt="Image of a company logo." />
```

Next, you start working on a simple Web site for an online bicycle retailer named DRG Cycles. You will find four prewritten Web pages in the DRGCycles folder within your Chapter folder for Chapter 11: index.html, cannondale.html, intense.html, and pinarello.html. The index.html document is the home page; the other pages display photos and information about different bicycles. You will modify these Web pages throughout the chapter.

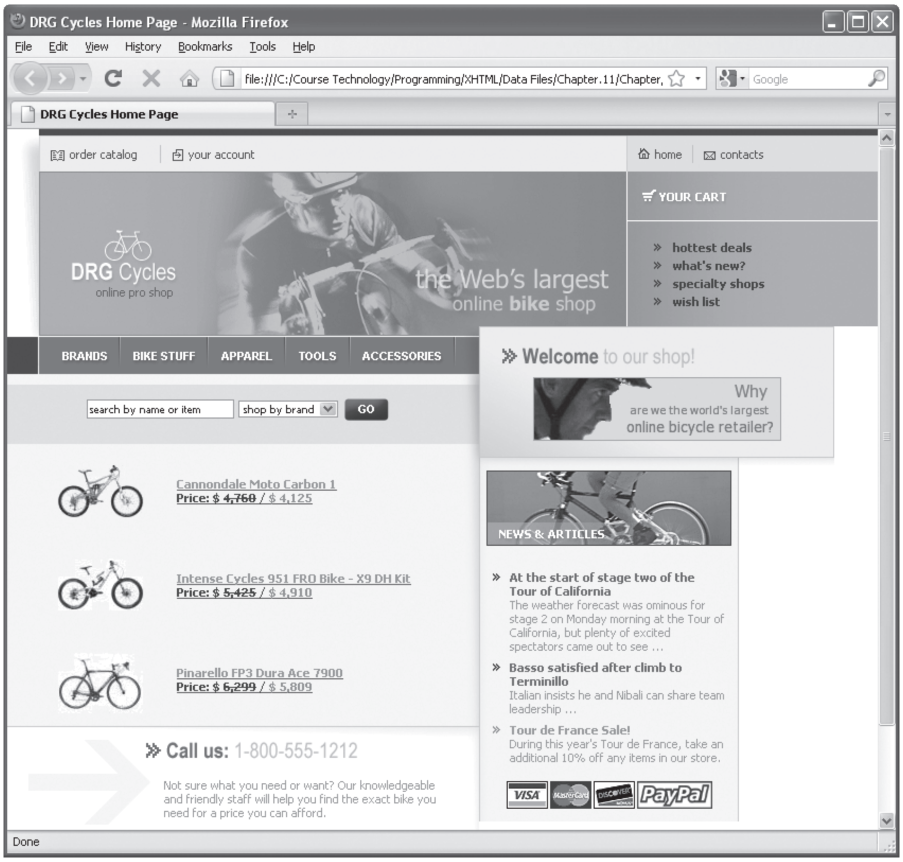
In this exercise, you add an advertisement to the DRG Cycles home page. The ad changes when users click the image. You will change the image using the `images[]` array.

To add an advertisement that changes when users click the image:

1. Open your text editor, then open the **index.html** document from the DRGCycles folder in your Chapter folder for Chapter 11.
2. Locate the `<img>` element that displays the banner1.png image and add an `onclick` event handler, as follows. When the user clicks the image, an `onclick` event handler changes the image to another image named banner2.png. Note that the banner image is the 27th image on the page, so the `images[]` array references element 26.

```
<img src="images/banner1.png" width="234"
    height="60" alt="Banner ads"
    onclick="document.images[26].src↵
    ='images/banner2.png';" />
```

3. Save the **index.html** document and open it in your Web browser. Figure 11-3 shows how the Web page appears. Click the image to make sure that it changes to banner2.png.



Welcome to our shop!

Why are we the world's largest online bicycle retailer?

Cannondale Moto Carbon 1
Price: $ ~~4,760~~ / $ 4,125

Intense Cycles 951 FRO Bike - X9 DH Kit
Price: $ ~~5,425~~ / $ 4,910

Pinarello FP3 Dura Ace 7900
Price: $ ~~6,299~~ / $ 5,809

NEWS & ARTICLES

» At the start of stage two of the Tour of California
The weather forecast was ominous for stage 2 on Monday morning at the Tour of California, but plenty of excited spectators came out to see ...

» Basso satisfied after climb to Terminillo
Italian insists he and Nibali can share team leadership ...

» Tour de France Sale!
During this year's Tour de France, take an additional 10% off any items in our store.

Call us: 1-800-555-1212

Not sure what you need or want? Our knowledgeable and friendly staff will help you find the exact bike you need for a price you can afford.

Figure 11-3 DRG Cycles Web page with an advertisement

4. Close your Web browser window.

The code you entered in the preceding exercise refers to the 27th element (26) in the `images[]` array. If other images are added to the Web page before the preceding statement, then referring to the 27th element in the `images[]` array would result in the wrong URL being displayed. When referring to the current object (in this case, the `Image` object for the preceding statement), you can simply use the `this` keyword instead of including the `Document` object and `images[]` array. The `this` keyword refers to the current object. The following code shows the example you saw before the last exercise, but this time it is written with the `this` keyword:

```
<img src="company_logo.gif" height="100" width="200"
    onclick="window.alert('This image is located↵
    at the following URL: ' + this.src);"
    alt="Image of a company logo." />
```

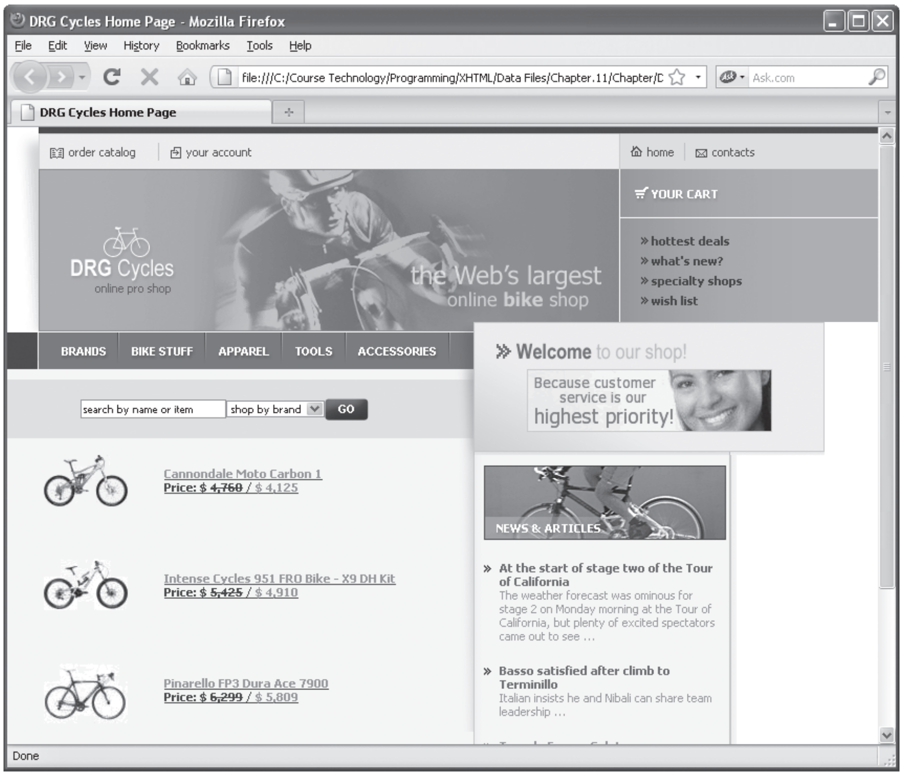
Next, you will modify the `onclick` event handler in the index.html document so that it uses `this` references instead of referring to the `Document` object and `images[]` array.

To modify the event handler in the index.html document so that it uses `this` references instead of referring to the `Document` object and `images[]` array:

1. Return to the **index.html** file in your text editor.
2. Modify the `onclick` event handler in the banner image as follows:

```
<img src="images/banner1.png" width="234"
    height="60" alt="Banner ads"
    onclick="this.src='images/banner2.png';" />
```

3. Save the **index.html** document and open it in your Web browser. Figure 11-4 shows how the Web page appears after clicking the banner image.



Cannondale Moto Carbon 1
Price: $ ~~4,760~~ / $ 4,125

Intense Cycles 951 FRO Bike - X9 DH Kit
Price: $ ~~5,425~~ / $ 4,910

Pinarello FP3 Dura Ace 7900
Price: $ ~~6,299~~ / $ 5,809

» At the start of stage two of the Tour of California
The weather forecast was ominous for stage 2 on Monday morning at the Tour of California, but plenty of excited spectators came out to see ...

» Basso satisfied after climb to Terminillo
Italian insists he and Nibali can share team leadership ...

Figure 11-4 DRG Cycles Web page after adding a `this` reference

4. Close your Web browser window.

Short Quiz 1

1. Explain what the browser object model is and why it's important to JavaScript.
2. What is the top-level object in the browser object model?
3. Explain how to reference arrays that are part of the browser object model.

Manipulating the Browser with the `Window` Object

The `Window` object includes several properties that contain information about the Web browser window. For instance, the `status` property contains information displayed in a Web browser's status bar. Also contained in the `Window` object are various methods that allow you to manipulate the Web browser window itself. You have already used some methods of the `Window` object, including the `window.alert()`, `window.confirm()`, and `window.prompt()` methods, which all display dialog boxes. Table 11-1 lists the `Window` object properties, and Table 11-2 lists the `Window` object methods.

Property	Description
`closed`	Returns a Boolean value that indicates whether a window has been closed
`defaultStatus`	Sets the default text that is written to the status bar
`document`	Returns a reference to the `Document` object
`history`	Returns a reference to the `History` object
`location`	Returns a reference to the `Location` object
`name`	Returns the name of the window
`opener`	Refers to the window that opened the current window
`self`	Returns a self-reference to the `Window` object; identical to the `window` property
`status`	Specifies temporary text that is written to the status bar
`window`	Returns a self-reference to the `Window` object; identical to the `self` property

Table 11-1 `Window` object properties

Method	Description
`alert()`	Displays a simple message dialog box with an OK button
`blur()`	Removes focus from a window
`clearInterval()`	Cancels an interval that was set with `setInterval()`
`clearTimeout()`	Cancels a timeout that was set with `setTimeout()`
`close()`	Closes a Web browser window
`confirm()`	Displays a confirmation dialog box with OK and Cancel buttons
`focus()`	Makes a `Window` object the active window
`moveBy()`	Moves the window relative to the current position
`moveTo()`	Moves the window to an absolute position
`open()`	Opens a new Web browser window
`print()`	Prints the document displayed in the current window
`prompt()`	Displays a dialog box prompting a user to enter information
`resizeBy()`	Resizes a window by a specified amount
`resizeTo()`	Resizes a window to a specified size
`scrollBy()`	Scrolls the window by a specified amount
`scrollTo()`	Scrolls the window to a specified position
`setInterval()`	Repeatedly executes a function after a specified number of milliseconds have elapsed
`setTimeout()`	Executes a function once after a specified number of milliseconds have elapsed

Table 11-2 `Window` object methods

Some Web browsers, including Internet Explorer, have custom properties and methods for the `Window` object. This book describes only properties and methods that are common to browser objects in all current Web browsers.

Another way of referring to the `Window` object is by using the **`self` property**, which refers to the current `Window` object. Using the `self` property is identical to using the `window` property to refer to the `Window` object. For example, the following lines are identical:

```
window.alert("Your order has been received.");
self.alert("Your order has been received.");
```

Some JavaScript programmers prefer to use the `window` property; others prefer to use the `self` property. The choice is yours. However, when attempting to decipher JavaScript code created by other programmers, be aware that both properties refer to the current `Window` object.

Because a Web browser assumes that you are referring to the global object, you do not need to refer explicitly to the `Window` object when using one of its properties or methods. For example, the `alert()` method is a method of the `Window` object. Throughout this text, you have used the full syntax of `window.alert(`*`text`*`);`, although the syntax `alert(`*`text`*`);` without the `Window` object works equally well. However, it's good practice to use the `window` or `self` references when referring to a property or method of the `Window` object to clearly identify them as belonging to the `Window` object. If you do not use the `window` or `self` reference, then you or another programmer might confuse a property or method of the `Window` object with JavaScript variables or functions.

Understanding Windows and Events

In Chapter 8, you learned how to use events with your Web pages. Events are particularly important when it comes to working with the browser object model because they allow you to execute the methods and change the properties of objects in the browser object model. In this section, you learn more about mouse events.

The `click` and `dblclick` Events

You have already extensively used the `click` event with form controls, such as radio buttons, to execute JavaScript code. However, keep in mind that the `click` event can be used with other types of elements. Earlier in this chapter, you used the `click` event to change the image displayed on the DRG Cycles Web page. The `click` event is often used for the anchor element. In fact, the primary event associated with the anchor element is the `click` event. When a user clicks a link, the Web browser handles execution of the `onclick` event handler automatically, so you do not need to add an `onclick` event handler to your anchor elements.

Sometimes, however, you might want to override an anchor element's automatic `onclick` event handler with your own code. For instance, you may want to warn the user about the content of a Web page that a particular link will open. To override the automatic `click` event with your own code, you add an `onclick` event handler that executes custom code to the <a> element. When you override an internal event handler with your own code, your code must return a value of true or false using the return statement. With the <a> element, a value of true indicates that you want the Web browser to perform its default event handling operation of opening the URL referenced in the link. A value of false indicates that you do not want the <a> element to perform its default event handling operation. For example, the <a> element in the following code includes an `onclick` event handler. The `warnUser()` function that is called by the `onclick` event handler returns a value generated by the `window.confirm()` method. Recall that when a user clicks the OK button in a confirm dialog box, a value of true is returned. When a user clicks the Cancel button, a value of false is returned. Notice the two return statements in the following code. The return statement in the `warnUser()` function returns a value to the `onclick` event handler. The return statement in the `onclick` event handler returns the same value to the Web browser.

```
...
<script type="text/javascript">
/* <![CDATA[ */
function warnUser() {
    return window.confirm("This link is only for ↵
        Red Sox fans. Are you sure you want to ↵
        continue?");
}
/* ]]> */
</script>
</head>
<body>
<p><a href="redsox.html"
onclick="return warnUser();">
Red Sox Fan Club</a></p>
</body>
</html>
```

The `dblclick` event works the same as the `click` event, except that users need to double-click the mouse instead of single-clicking it. The `dblclick` event is rarely used, and it is not generally used with links because they are driven by single mouse clicks. From the user's point of view, single clicks are much easier than double-clicks.

The `mouseover` and `mouseout` Events

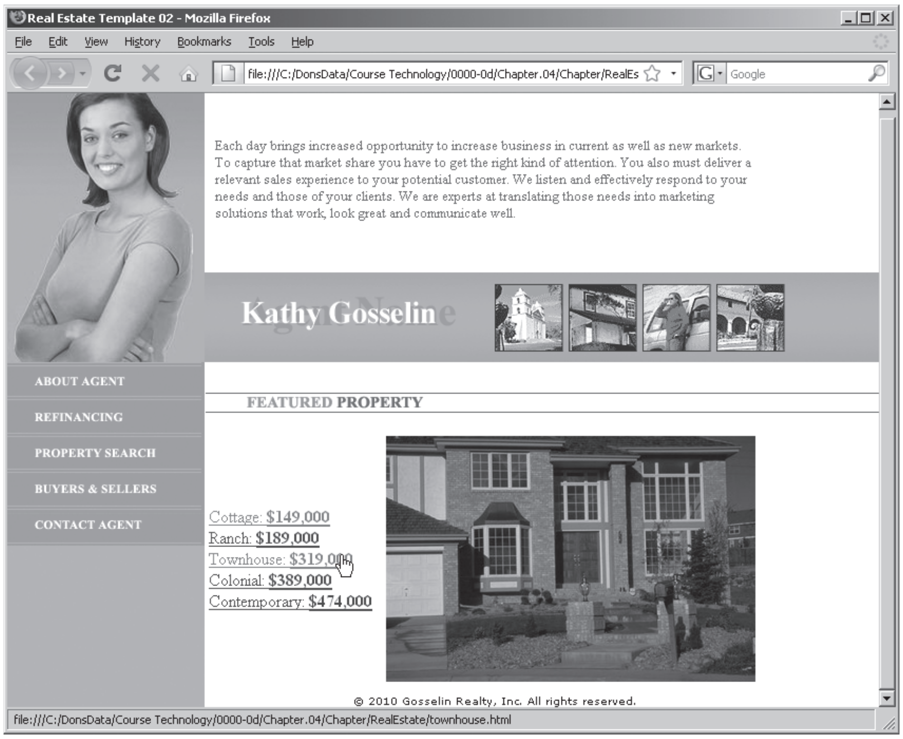
You use the `mouseover` and `mouseout` events to create **rollover** effects, which occur when your mouse moves over an element. The `mouseover` event occurs when the mouse passes over an element and the `mouseout` event occurs when the mouse moves off an element. These events are also commonly used to change an element's style, such as the formatting of a link when the mouse passes over it. To refer to a CSS style in JavaScript, you use the `this` reference and the `style` property in an event handler within the element itself. You use the **style property** to modify an element's CSS properties with JavaScript. To refer to a style with the `this` reference, you use a period to append the `style` property to it, followed by another period and a CSS property. CSS properties without hyphens are referred to in JavaScript with all lowercase letters. However, when you refer to a CSS property that contains a hyphen in JavaScript code, you remove the hyphen, convert the first word to lowercase, and convert the first letter of subsequent words to uppercase. For example, the `text-decoration` property is referred to as `textDecoration`, font-family is referred to as fontFamily, font-size is referred to as fontSize, and so on. In the following code, the onmouseover event handler underlines the link when the mouse passes over it, and the onmouseout event handler removes the link when the mouse passes off it:

```
<a href="redsox.html"
onmouseover="this.style.textDecoration='underline';"
onmouseout="this.style.textDecoration='none';">
Red Sox Fan Club</a>
```

The mouseover and mouseout events are also commonly used to display an alternate image or explanatory text when the mouse passes over an element. The following table cell shows a more complex example of the mouseover and mouseout events. The cell contains five links representing different types of homes that a real estate agent is selling. When the user passes the mouse over a link, the link changes from blue to red and an image of the house is displayed. Moving the mouse off the link changes the link back to blue and displays an empty image. Figure 11-5 shows the page with the mouse over the Townhouse link.

```
<td>
    <p>
        <a href="cottage.html"
            onmouseover="document.images[9].src↵
            ='cottage.jpg';this.style.color='Red'"
            onmouseout="document.images[9].src↵
            ='noselection.jpg';this.style.color↵
            ='Blue'">Cottage:<strong>
            $149,000</strong></a><br />
        <a href="ranch.html"
            onmouseover="document.images[9].src↵
            ='ranch.jpg';this.style.color='Red'"
            onmouseout="document.images[9].src↵
            ='noselection.jpg';this.style.color↵
            ='Blue'">Ranch:<strong>
            $189,000</strong></a><br />
        <a href="townhouse.html"
            onmouseover="document.images[9].src↵
            ='townhouse.jpg';this.style.color='Red'"
            onmouseout="document.images[9].src↵
            ='noselection.jpg';this.style.color↵
            ='Blue'">Townhouse:<strong>
            $319,000</strong></a><br />
        <a href="colonial.html"
            onmouseover="document.images[9].src↵
            ='colonial.jpg';this.style.color='Red'"
            onmouseout="document.images[9].src↵
            ='noselection.jpg';this.style.color↵
            ='Blue'">Colonial:<strong>
            $389,000</strong></a><br />
```

```
                    <a href="contemporary.html"
                        onmouseover="document.images[9].src ↵
                        ='contemporary.jpg';this.style.color='Red'"
                        onmouseout="document.images[9].src ↵
                        ='noselection.jpg';this.style.color ↵
                        ='Blue'">Contemporary:<strong>
                        $474,000</strong></a></p>
            </td>
```


Real Estate Template 02 - Mozilla Firefox

file:///C:/DonsData/Course Technology/0000-0d/Chapter.04/Chapter/RealEs

Each day brings increased opportunity to increase business in current as well as new markets. To capture that market share you have to get the right kind of attention. You also must deliver a relevant sales experience to your potential customer. We listen and effectively respond to your needs and those of your clients. We are experts at translating those needs into marketing solutions that work, look great and communicate well.

Kathy Gosselin

ABOUT AGENT
REFINANCING
PROPERTY SEARCH
BUYERS & SELLERS
CONTACT AGENT

FEATURED PROPERTY

Cottage: $149,000
Ranch: $189,000
Townhouse: $319,000
Colonial: $389,000
Contemporary: $474,000

© 2010 Gosselin Realty, Inc. All rights reserved.

file:///C:/DonsData/Course Technology/0000-0d/Chapter.04/Chapter/RealEstate/townhouse.html

Figure 11-5 Real estate page with the mouse over the Townhouse link

You can find a working copy of the real estate page in a folder named RealEstate in your Chapter folder for Chapter 11.

By default, Firefox does not allow scripts to change status bar text. To allow scripts to change status bar text on Windows systems, select the Tools menu, select Options, and then select Content in the Options dialog box. Click the Advanced button next to the Enable JavaScript button, and then click the Change status bar text box in the Advanced JavaScript Settings dialog box. Click OK twice to close each dialog box. To make the same setting on Linux systems, select Preferences from the Edit menu; on Macintosh systems, select Preferences from the Firefox menu.

The `defaultStatus` property specifies the default text that appears in the status bar whenever the mouse is not positioned over a link. The syntax for the `defaultStatus` property is `window.defaultStatus = "`*`status bar text here`*`";`. You will now add the `defaultStatus` property to the DRG Cycles Web page so the text "Welcome to DRG Cycles!" is displayed in the status bar by default. You will also add `onmouseover` event handlers to each of the bike model links; these event handlers display messages in an `<input>` box about clicking the link for more information. Finally, you will add `onmouseout` event handlers that remove the value assigned to the `<input>` box by changing its value to an empty string.

To add the `defaultStatus` property and `onmouseover` and `onmouseout` event handlers to the DRG Cycles Web page:

1. Return to the **index.html** document in your text editor.
2. Add the following script section immediately above the closing `</head>` tag. The script contains a single statement that sets the Web page's default status bar text to "Welcome to DRG Cycles!"

```
<script type="text/javascript">
/* <![CDATA[ */
window.defaultStatus = "Welcome to DRG Cycles!";
/* ]]> */
</script>
```

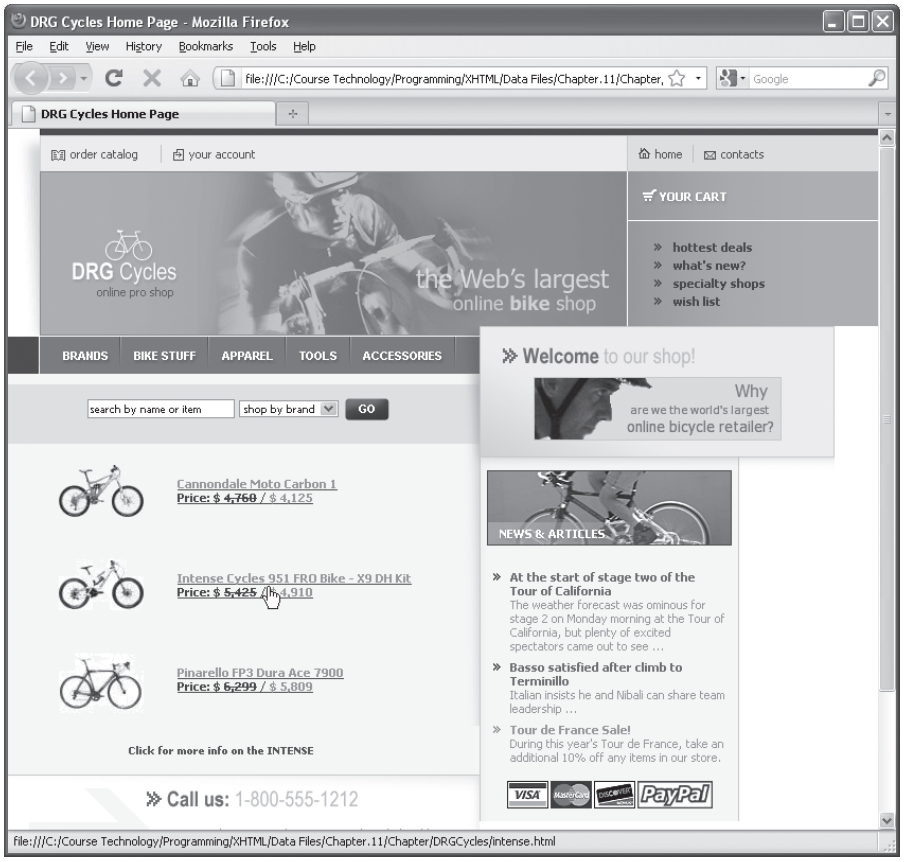
3. Locate `<!--[Add form here]-->` in the document body and replace it with the following form, which contains a single text box that will display messages when the mouse passes over a bike model link:

```
<form action="" name="messageForm">
  <p><input type="text" name="bikeLink" size="40"
      style="color:Blue; font-weight:bold; ↵
      border-style:none; border-color: inherit; ↵
      border-width:medium; background-color: ↵
      Transparent" /></p>
</form>
```

4. Add `onmouseover` event handlers to the `<tr>` element containing the bicycle model links to modify the value assigned to the text box when the mouse pointer passes over the link. Also, add `onmouseout` event handlers that reset the text box to an empty string. For example, the `<tr>` element for the Cannondale bike should appear as follows:

```
<tr onmouseover ="document.messageForm.bikeLink. ↵
    value ='Click for more info on the CANNONDALE'"
    onmouseout ="document.messageForm.bikeLink. ↵
    value =''">
```

5. Save the **index.html** document and open it in your Web browser. Figure 11-6 shows how the Web page appears when you hold your mouse pointer over Intense Cycles 951 FRO Bike - X9 DH Kit.



Cannondale Moto Carbon 1
Price: $ ~~4,760~~ / $ 4,125

Intense Cycles 951 FRO Bike - X9 DH Kit
Price: $ ~~5,425~~ / $ 4,910

Pinarello FP3 Dura Ace 7900
Price: $ ~~6,299~~ / $ 5,809

Click for more info on the INTENSE

Call us: 1-800-555-1212

At the start of stage two of the Tour of California
The weather forecast was ominous for stage 2 on Monday morning at the Tour of California, but plenty of excited spectators came out to see ...

Basso satisfied after climb to Terminillo
Italian insists he and Nibali can share team leadership ...

Tour de France Sale!
During this year's Tour de France, take an additional 10% off any items in our store.

file:///C:/Course Technology/Programming/XHTML/Data Files/Chapter.11/Chapter/DRGCycles/intense.html

Figure 11-6 DRG Cycles Web page after adding the `defaultStatus` property and `onmouseover` and `onmouseout` event handlers

6. Close your Web browser window.

One of the more common uses of rollovers is to replace (or swap) an image on a Web page with another image. Consider the following code. By default, the v500tec.gif file is displayed. The `onmouseover` event handler changes the image to showroom.gif, and the `onmouseout` event handler changes the image back to the v500tec.gif file. Figure 11-7 shows the Web page before the mouse is placed on the image. Once the mouse moves over the image, the image shown in Figure 11-8 is displayed.

```
<p><img src="v500tec.gif" height="90px" width="700px"
    alt="Banner images"
    onmouseover="this.src='showroom.gif'"
    onmouseout="this.src='v500tec.gif'" /></p>
```

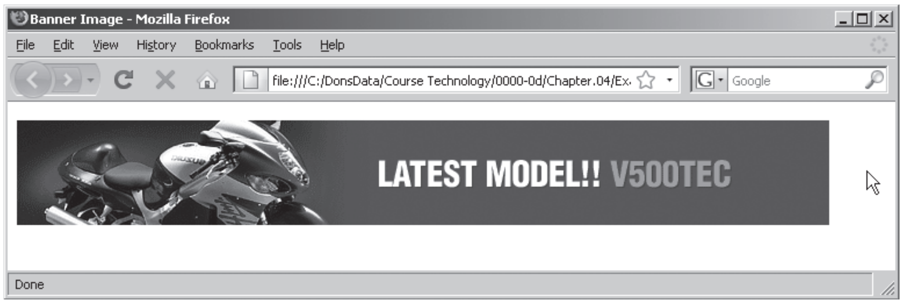

Banner Image - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/DonsData/Course Technology/0000-0d/Chapter.04/Ex.
Google
LATEST MODEL!! V500TEC
Done

Figure 11-7 Web page before the mouse passes over the image

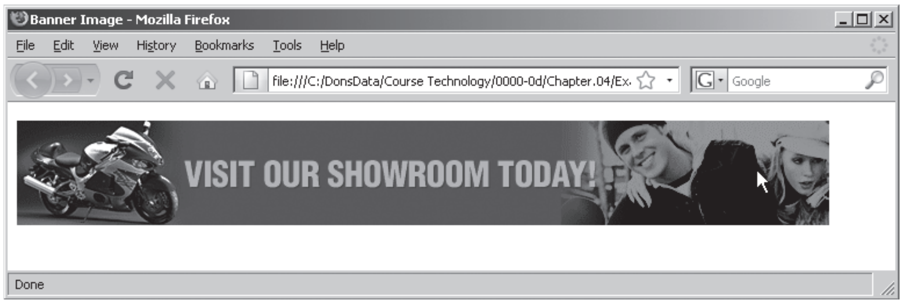

Banner Image - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/DonsData/Course Technology/0000-0d/Chapter.04/Ex.
Google
VISIT OUR SHOWROOM TODAY!
Done

Figure 11-8 Web page with the mouse placed over the image

The *mousedown* and *mouseup* Events

The mousedown event occurs when you point to an element and hold the mouse button down; the mouseup event occurs when you release the mouse button. The following code shows the <img> element that displays the motorcycle and showroom images, this time using mousedown and mouseup events:

```
<p><img src="v500tec.gif" height="90px" width="700px"
    alt="Banner images"
    onmousedown="this.src='showroom.gif'"
    onmouseup="this.src='v500tec.gif'" /></p>
```

Next, you will modify the <img> element that displays the banner ads in the index.html document so the second image in the banner is displayed when you hold the mouse button down over the image.

To modify the `<img>` element that displays the banner ads in the index.html document so that the second image in the banner is displayed when you hold the mouse over it:

1. Return to the **index.html** document in your text editor.
2. Replace the `onclick` event handler in the banner-ad `<img>` element with `onmousedown` and `onmouseup` event handlers that swap the images.

   ```
   <img src="images/banner1.png" width="234" height="60"
       alt="Banner ads"
       onmousedown="this.src='images/banner2.png';"
       onmouseup="this.src='images/banner1.png'" />
   ```
3. Save the **index.html** document and open it in your Web browser. Press and hold the mouse button over the banner image, then release it. You should see the images change when you press and release the mouse button.
4. Close your Web browser window.

Opening and Closing Windows

Most Web browsers allow you to open new browser windows in addition to the browser window(s) that may already be open. You may need to open a new browser window for several reasons. For example, you may want to launch a new Web page in a separate window, allowing users to continue viewing the current page in the current window. Or, you may want to use an additional window to display information such as a picture or an order form.

Whenever a new Web browser window is opened, a new `Window` object is created to represent the new window. You can have as many browser windows open as your system will support, each displaying a different Web page. For example, you can have one browser window display Microsoft's Web site, another browser window display Firefox's Web site, and so on.

You may be familiar with how to open a link in a new window by using the `<a>` element's `target` attribute. For example, the following link opens the Wikipedia home page in a new window named `wikiWindow`:

```
<p><a href="http://www.wikipedia.org/"
    target="wikiWindow">
    Wikipedia home page</a></p>
```

Whenever the user clicks the preceding link, the Web browser looks for another browser window named `wikiWindow`. If the window exists, the link is opened in it. If the window does not exist, a new window named `wikiWindow` is created where the link opens.

Some Web browsers, including Firefox and Internet Explorer, can be configured to open new pages in either a new window or a tab in the current window. To configure Firefox to open pages in a new window, select the Tools menu, select Options, and then select Tabs in the Options dialog box. Select the check box that opens pages in a new window and then click OK. To configure Internet Explorer to open pages in a new window, select the Tools menu, select Internet Options, and then select the General tab in the Internet Options dialog box. In the Tabs section, click the Settings button. In the Tabbed Browsing Settings dialog box, select the radio button that opens links in a new window and then click OK twice to close each dialog box.

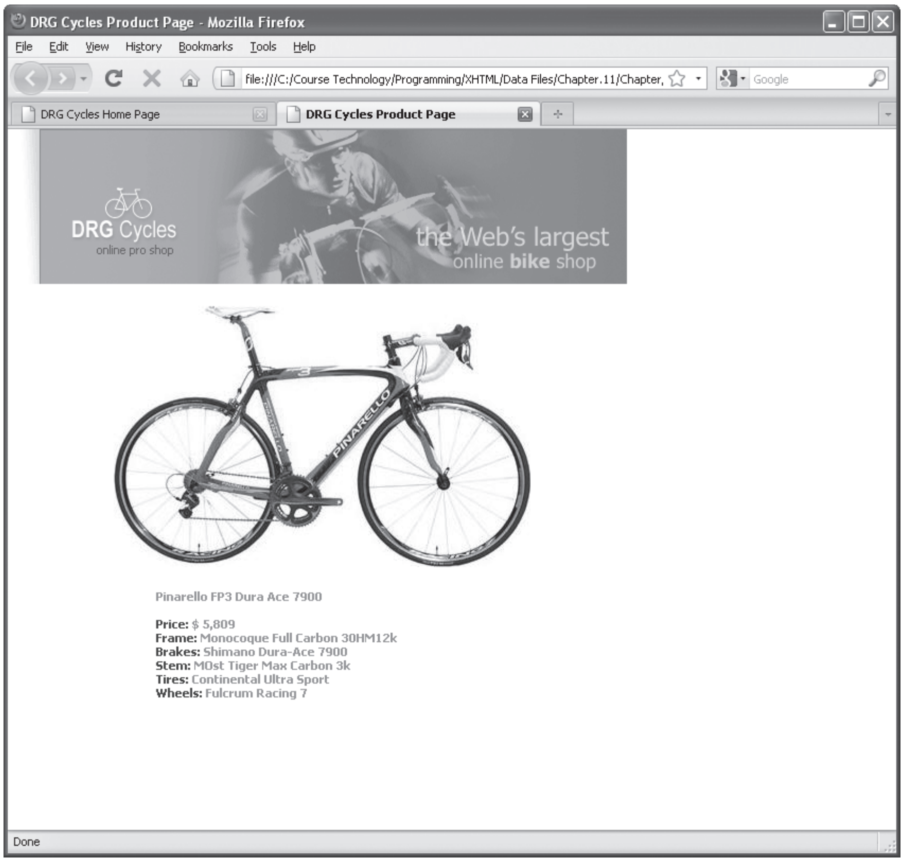
The links in the DRG Cycles Web page now open in the current window; they do not open in a new window. Next, you will modify the links so they use the `<a>` element's `target` attribute to open each URL in a separate window.

To modify the links in the DRG Cycles Web page so that they use the `<a>` element's `target` attribute to open each URL in a separate window:

1. Return to the **index.html** document in your text editor.
2. Add the following attribute before the closing bracket for each of the `<a>` elements that open the bike model pages. Note that there are two `<a>` elements for each bike: one for the bike's picture and another for its description.

   ```
   target="bikeInfo"
   ```

3. Save the **index.html** document and open it in your Web browser. Click one of the links to see if the Web page opens in a new browser window. If you click other links on the DRG Cycles Web page, you should notice that each Web page opens in the `bikeInfo` window (if it is currently open) instead of opening in a separate window. Figure 11-9 shows the `bikeInfo` window opened to the Pinarello Web page.


DRG Cycles Product Page - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/Course Technology/Programming/XHTML/Data Files/Chapter.11/Chapter,
Google
DRG Cycles Home Page
DRG Cycles Product Page
DRG Cycles
online pro shop
the Web's largest
online bike shop
Done

Pinarello FP3 Dura Ace 7900

Price: $ 5,809
Frame: Monocoque Full Carbon 30HM12k
Brakes: Shimano Dura-Ace 7900
Stem: MOst Tiger Max Carbon 3k
Tires: Continental Ultra Sport
Wheels: Fulcrum Racing 7

Figure 11-9 Pinarello Web page opened in the `bikeInfo` window

4. Close your Web browser window.

Opening a Window

The problem with using the `target` attribute is that it is deprecated in HTML 5 and XHTML. To open new windows in HTML 5 and the strict DTD, you must use the **`open()` method** of the `Window` object. The syntax for the `open()` method is as follows:

```
window.open(url, name, options, replace);
```

Table 11-3 lists the arguments of the `window.open()` method.

Argument	Description
`URL`	Represents the Web address or filename to be opened
`name`	Assigns a value to the `name` property of the new `Window` object
`options`	Represents a string that allows you to customize the new Web browser window's appearance
`replace`	A Boolean value that determines whether the URL should create a new entry in the Web browser's history list or replace the entry

Table 11-3 Arguments of the `Window` object's `open()` method

You can include all or none of the `window.open()` method arguments. The statement `window.open("http://www.wikipedia.org");` opens the Wikipedia home page in a new Web browser window, as shown in Figure 11-10. If you exclude the `URL` argument, a blank Web page opens. For example, the statement `window.open();` opens the browser window displayed in Figure 11-11.

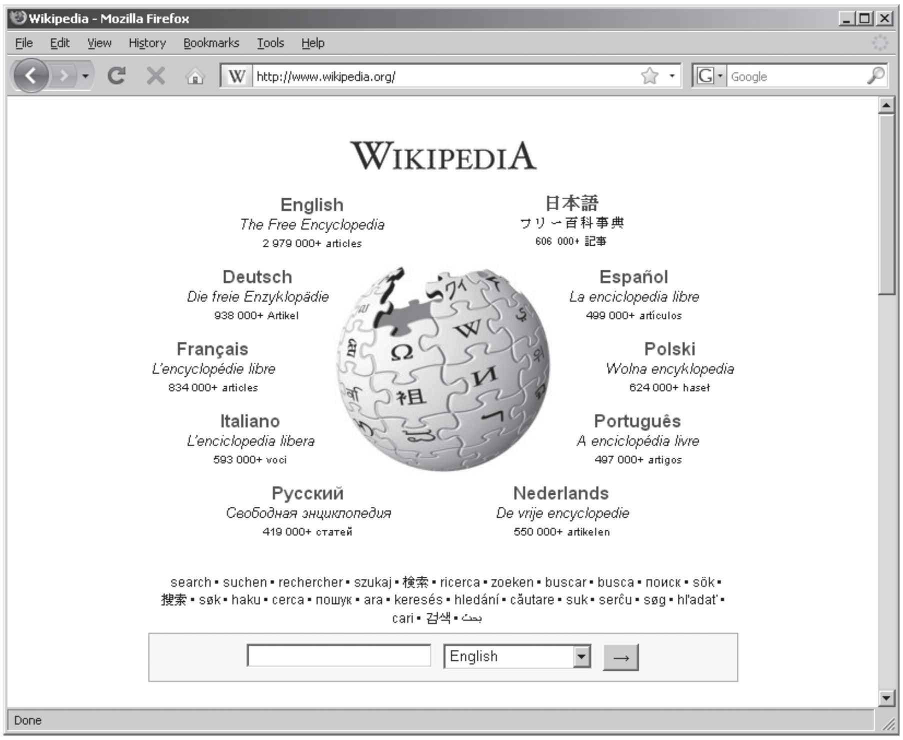


Figure 11-10 Web browser window opened with the `URL` argument of the `open()` method

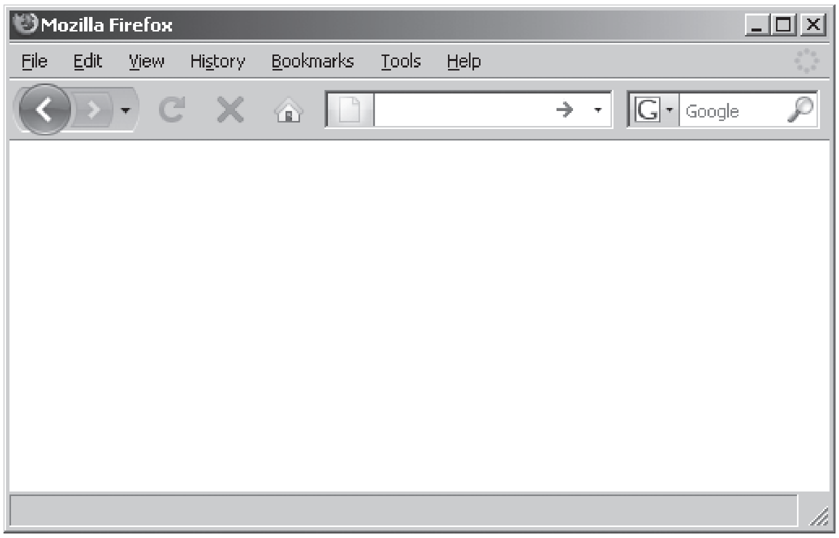


Figure 11-11 Blank Web browser window opened with the `window.open()` statement

If you are writing code that requires a user to click a link or a button, then you can use an event handler to call the `window.open()` method, and the window will open successfully. However, if you include JavaScript code that opens a new window without a request from the user, then the pop-up blocker feature that is available in most current Web browsers will prevent the window from opening.

When you open a new Web browser window, you can customize its appearance by using the `options` argument of the `window.open()` method. Table 11-4 lists some common options that you can use with the `window.open()` method.

Name	Description
`height`	Sets the window's height
`left`	Sets the horizontal coordinate of the left side of the window, in pixels
`location`	Includes the URL Location text box
`menubar`	Includes the menu bar
`resizable`	Determines if the new window can be resized
`scrollbars`	Includes scroll bars
`status`	Includes the status bar
`toolbar`	Includes the Standard toolbar
`top`	Sets the vertical coordinate of the top of the window, in pixels
`width`	Sets the window's width

Table 11-4 Common options of the `Window` object's `open()` method

All the options listed in Table 11-4, with the exception of the `width` and `height` options, are set using values of "yes" or "no", or 1 for yes and 0 for no. To include the status bar, for example, the options string should read "status=yes". You set the `width` and `height` options using integers representing pixels. For example, to create a new window that is 200 pixels high by 300 pixels wide, the string should read "height=200,width=300". When including multiple items in the options string, you must separate the items by commas. If you exclude the options string of the `window.open()` method, then all the standard options are included in the new Web browser window. However, if you include the options string, you must include all the components you want to create for the new window; that is, the new window is created with only the components you specify.

Figure 11-12 shows the Photo Gallery Web page from the Woodland Park Zoo in Seattle, Washington. If you select a link from one of the menus on the page, such as the Fennec fox link that is highlighted in Figure 11-12, the Photo Gallery Slideshow Web page shown in Figure 11-13 opens.

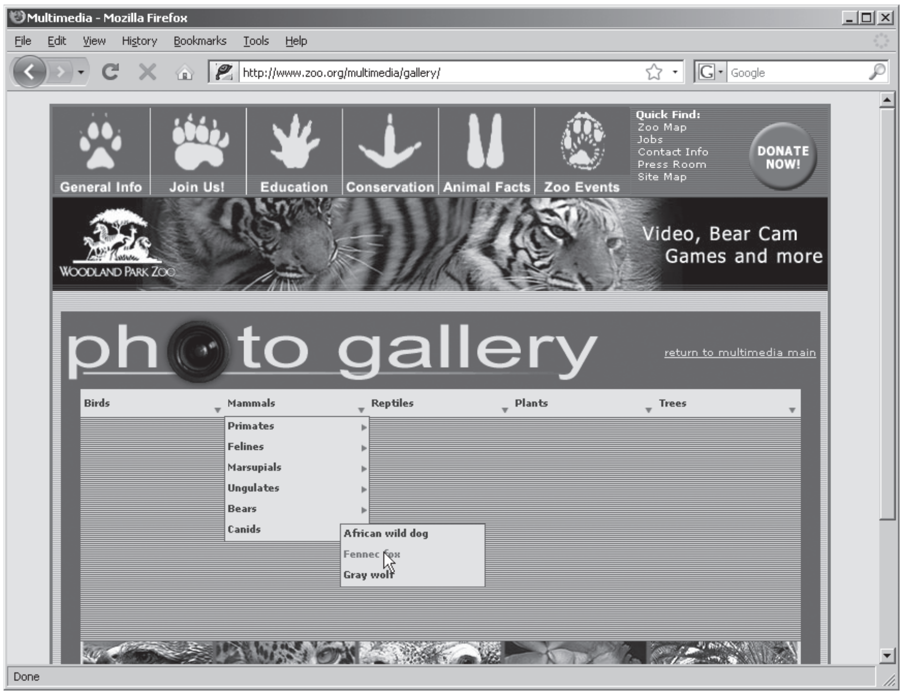


Figure 11-12 Woodland Park Zoo Photo Gallery Web page

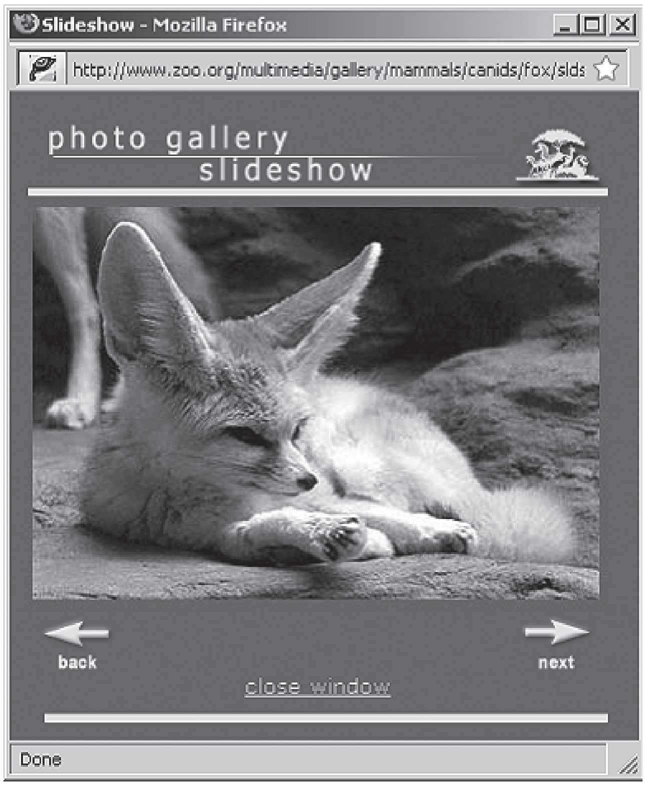

Slideshow - Mozilla Firefox
http://www.zoo.org/multimedia/gallery/mammals/canids/fox/slds
photo gallery
slideshow
back
next
close window
Done

Figure 11-13 Woodland Park Zoo Photo Gallery Slideshow Web page displaying a fox

Notice that the Photo Gallery Slideshow Web page does not display toolbars, the menu, the URL Location box, or the scroll bars. Also, keep in mind that it is sized to specific dimensions. If you tried to resize the window, you would find that it couldn't be resized. The Photo Gallery Web page uses a JavaScript statement similar to the following to open the Photo Gallery Slideshow Web page when a user clicks the name of an animal:

```
var OpenWin = window.open(page, "CtrlWindow",
"toolbar=no,menubar=no,location=no,scrollbars=no,↵
resizable=no,width=380,height=405");
```

The `name` argument of the `window.open()` method is essentially the same as the value assigned to the deprecated `target` attribute in that it specifies the name of the window where the URL should open. If the `name` argument is already in use by another Web browser window, then JavaScript changes focus to the existing Web browser window instead of creating a new window. For instance, the Photo Gallery Web page opens the Photo Gallery Slideshow Web page and assigns it a name of "CtrlWindow". If the `CtrlWindow` Web page already exists when you select another menu item from the Photo Gallery Web page, then the `CtrlWindow` Web page is reused; another window does not open. This is especially important with a Web page such as the Photo Gallery Web page, which allows you to view dozens of different Web pages for each of the animals listed in the menu. Imagine how crowded a user's screen would be if the program kept opening a new Photo Gallery Slideshow Web page window for each selected animal.

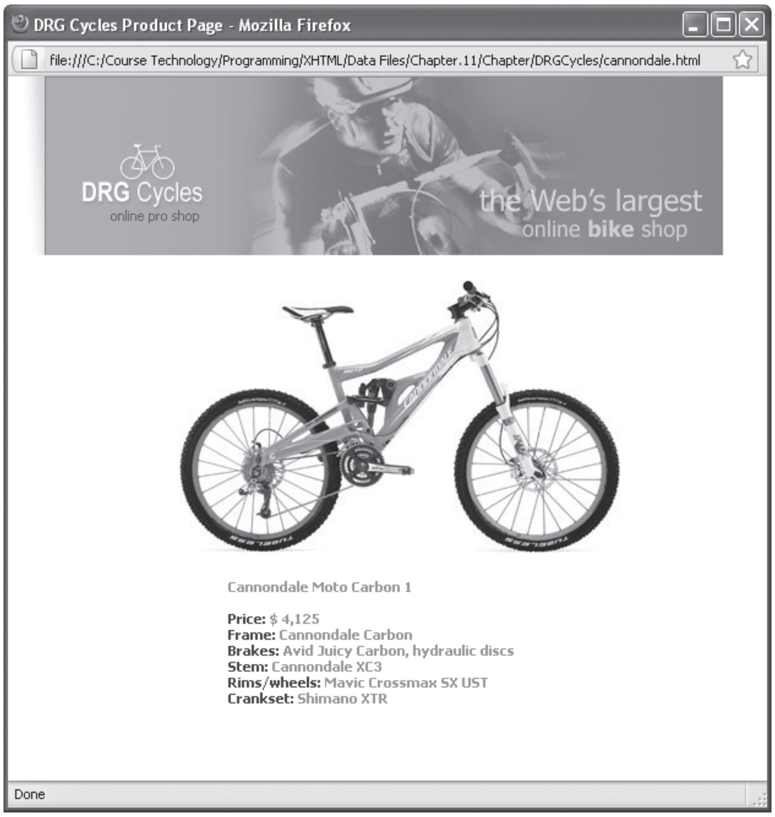
Next, you will modify the DRG Cycles Web page so the links use the `window.open()` method instead of the `target` attribute to open the URLs in a separate page.

To modify the DRG Cycles Web page so the links use the `window.open()` method instead of the `target` attribute to open the URLs in a separate page:

1. Return to the **index.html** document in your text editor.
2. Add the following global variable declaration and function to the end of the script section. The function will be called by `onclick` event handlers in each of the links.

```
var bikeWindow;
function showBike(linkTarget) {
  bikeWindow = window.open(linkTarget, "bikeInfo",
    "toolbar=no,menubar=no,location=no, ↵
    scrollbars=no,resizable=no,width=620, ↵
    height=575");
}
```

3. Next, replace the `target` attribute in the six <a> elements with an `onclick` event handler that calls the `showBike()` function, passing to it the URL of the target Web page. The `onclick` event handler should also return a value of "false" to prevent the index.html Web page from being replaced with the target Web page that you are opening in a separate window.
4. Save the **index.html** document and open it in your Web browser. Click one of the links to see if the Web page opens in a new browser window. Figure 11-14 shows how the window appears with the cannondale.html Web page displayed.


DRG Cycles Product Page - Mozilla Firefox
file:///C:/Course Technology/Programming/XHTML/Data Files/Chapter.11/Chapter/DRGCycles/cannondale.html
DRG Cycles
online pro shop
the Web's largest
online bike shop
Done

Cannondale Moto Carbon 1

Price: $ 4,125
Frame: Cannondale Carbon
Brakes: Avid Juicy Carbon, hydraulic discs
Stem: Cannondale XC3
Rims/wheels: Mavic Crossmax SX UST
Crankset: Shimano XTR

Figure 11-14 Window opened with the `open()` method

5. Close your Web browser windows.

A `Window` object's `name` property can be used only to specify a target window with a link and cannot be used in JavaScript code. If you want to control the new window by using JavaScript code located within the Web browser in which it was *created*, then you must assign the new `Window` object created with the `window.open()` method to a variable. The statement that opens the Photo Gallery Slideshow Web page assigns an object representing the new Web browser window to a variable named `OpenWin`. You can use any of the properties and methods of the `Window` object with a variable that represents a `Window` object.

One problem with Web pages such as the DRG Cycles Web page is that windows that open in response to the user clicking a link can get hidden or "lost" behind other windows on the user's screen. For example, suppose that the user clicks the Cannondale Moto Carbon 1 link on the DRG Cycles Web page, thereby opening a new window. Then suppose that the user returns to the DRG Cycles Web page (without closing the Cannondale Moto Carbon 1 window) and clicks a different link. The window that displays the bicycle pages is not automatically displayed as the active window on the screen. That is, it does not necessarily appear as the top window; it could instead be hidden behind other windows. The user may continually click links, thinking that nothing is happening in response to his or her clicks, when in fact the code is working. The problem might be that the windows are open but not visible. In order to make a window the active window, you use the **`focus()` method** of the `Window` object. You append the `focus()` method to the variable that represents the window, not to the `name` argument of the `window.open()` method. For example, to make the Photo Gallery Slideshow window the active window, you use the following statement:

```
OpenWin.focus();
```

Next, you add a `focus()` method to the `showBike()` function in the DRG Cycles Web page.

It is not necessary to include the `Window` object or `self` property when using the `open()` and `close()` methods of the `Window` object. However, the `Document` object also contains methods named `open()` and `close()`, which are used for opening and closing Web pages. Therefore, the `Window` object is usually included with the `open()` and `close()` methods to distinguish between the `Window` object and the `Document` object.

To add a `focus()` method to the `showBike()` function in the DRG Cycles Web page:

1. Return to the **index.html** document in your text editor.
2. Add the following statement to the end of the `showBike()` function:

   ```
   bikeWindow.focus();
   ```

3. Save the **index.html** document and open it in your Web browser. Click one of the links to open the window that displays the bicycle pages. Leave the Web page open, navigate back to the DRG Cycles Web page, and click a different link. The window that displays the bicycle pages should become the active window and display the URL for the Web page link you clicked.
4. Close your Web browser windows.

Closing a Window

The **`close()` method**, which closes a Web browser window, is the method you will probably use the most with variables representing other `Window` objects. To close the Web browser window represented by the `OpenWin` variable, you use the statement `OpenWin.close();`. To close the current window, you use the statement `window.close()` or `self.close()`.

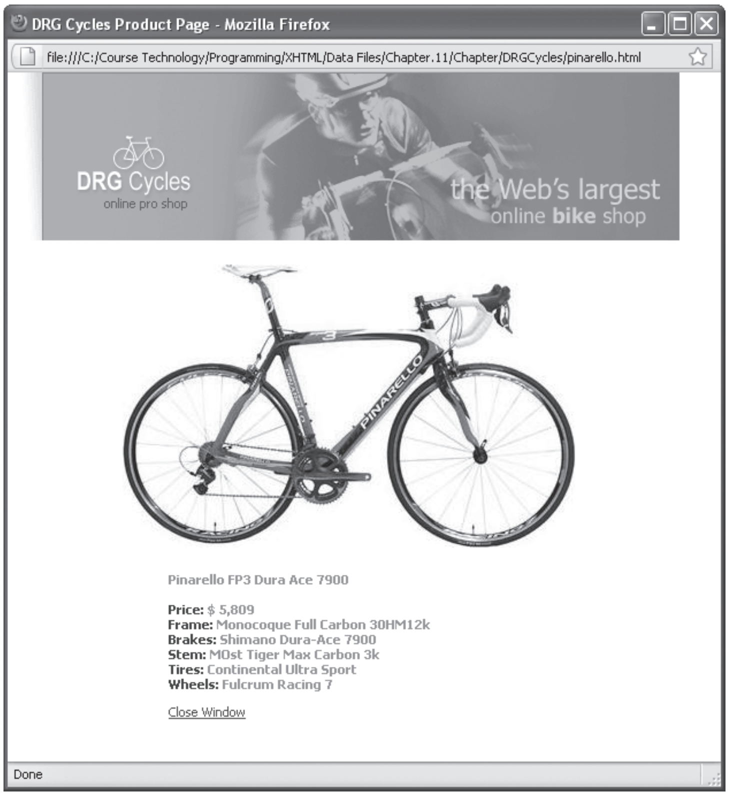
Next, you add links to each of the bicycle Web pages that call the `close()` method, which will close the window.

To add links to each of the bicycle Web pages that call the `close()` method:

1. Return to your text editor and open the **cannondale.html** document from the DRGCycles folder in your Chapter folder for Chapter 11.
2. Locate the table cell that displays the image of the Cannondale bike along with its specifications. Add the following paragraph and anchor elements to the end of the table cell. The `onclick` event handler in the anchor element calls the `close()` method, which will close the window.

```
<p><a href="" onclick="self.close();">
    Close Window</a></p>
```

3. Save and close the **cannondale.html** document.
4. Repeat Steps 2 and 3 for the **intense.html** and **pinarello.html** documents.
5. Open the **index.html** document in your Web browser and click one of the links. Figure 11-15 shows the new link (which closes the window) in the Pinarello FP3 Dura Ace 7900 Web page.


DRG Cycles Product Page - Mozilla Firefox
file:///C:/Course Technology/Programming/XHTML/Data Files/Chapter.11/Chapter/DRGCycles/pinarello.html
DRG Cycles
online pro shop
the Web's largest
online bike shop
Done

Pinarello FP3 Dura Ace 7900

Price: $ 5,809
Frame: Monocoque Full Carbon 30HM12k
Brakes: Shimano Dura-Ace 7900
Stem: MOst Tiger Max Carbon 3k
Tires: Continental Ultra Sport
Wheels: Fulcrum Racing 7

Close Window

Figure 11-15 Pinarello Web page after adding a link with a `close()` method

6. Click the **Close Window** link to close the window you opened.
7. Close the Web browser window containing the DRG Cycles Web page.

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 may want to create animation or allow for some kind of repetitive task that executes automatically. For example, you may want to include an advertising image that changes automatically every few seconds. Or, you may 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.

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. (Recall from Chapter 9 that a global variable is declared outside of a function and is available to all parts of a JavaScript program.)

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.

```
...
<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. In this context, "animation" does not necessarily mean a complex cartoon character, but any situation in which a sequence of images changes automatically. However, Web animation can include the traditional type, involving cartoons and movement (like advertising with changing images or the ticking hands of the clock mentioned earlier). The following code uses the `setInterval()` method to automatically swap the motorcycle images you saw in Figures 11-7 and 11-8 every couple of seconds.

```
...
<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>
</html>
```

Next, you will modify the DRG Cycles Web page so that it uses the `setInterval()` method to change the banner image automatically.

To modify the DRG Cycles Web page so that it uses the `setInterval()` method to change the banner image automatically:

1. Return to the **index.html** document in your text editor.
2. To the end of the script section, add the following global variable and `bannerAd()` function. The `bannerAd()` function will be called by a `setInterval()` method. As a result, the images will change automatically.

```
var curImage="banner1";
function bannerAd() {
    if (curImage == "banner2") {
       document.images[26].src = "images/banner1.png";
       curImage = "banner1";
    }
    else {
      document.images[26].src = "images/banner2.png";
      curImage = "banner2";
    }
}
```

3. Modify the opening `<body>` tag so it includes an `onload` event handler that calls the `setInterval()` method and `bannerAd()` function, as follows:

```
<body onload="var changeImages=setInterval ↵
    ('bannerAd()',2000);">
```

4. Finally, remove the `onmousedown` and `onmouseup` event handlers from the `<img>` banner element.
5. Save the **index.html** document and then open it in your Web browser. The image should begin alternating automatically.
6. Close your Web browser window.

Short Quiz 2

1. What are the different ways that you can refer to the `Window` object?

2. Explain how to override an internal event handler with your own code.

3. How do you open and close a window? How do you customize its appearance?

4. Explain how to use timeouts and intervals to execute JavaScript code repeatedly.

Working with the `History`, `Location`, `Navigator`, and `Screen` Objects

In this section, you will learn how to work with the `History`, `Location`, `Navigator`, and `Screen` objects.

Using the `History` Object

The **`History` object** maintains an internal list (known as a history list) of all the documents that have been opened during the current Web browser session. Each browser window contains its own internal `History` object. You cannot view the URLs contained in the history list, but you can write a script that uses the history list to navigate to Web pages that have been opened during a Web browser session.

Two important security features are associated with the `History` object. First, the `History` object will not actually display the URLs contained in the history list. This is important because individual user information in a Web browser, such as the types of Web sites a user likes to visit, is private information. Preventing others from viewing the URLs in a history list is an essential security feature because it keeps people's online interests confidential. This security feature is available in both Firefox and Internet Explorer.

A second important security feature of the `History` object is specific to Internet Explorer and the domain in which a Web page exists. As mentioned earlier, you can write a script that uses the history list to navigate to Web pages that have been opened during a Web browser session. In Internet Explorer, you can use JavaScript code to navigate through a history list. However, this is only possible if the currently displayed Web page exists within the same domain as the Web page containing the JavaScript code that is attempting to move through the list. For example, a user may open the home page for a company that sells office supplies. Suppose that the user then clicks a link on the company's home page that opens another Web page in the company's domain, such as an online ordering page. In this case, the office supply company's home page is added to the user's history list. JavaScript code on the online ordering page can use the `History` object to navigate back to the company's home page. If JavaScript code attempts to access the `History` object of a Web browser that contains a URL located in a different domain, the Web browser ignores the JavaScript code. This security feature helps prevent malicious programmers and unscrupulous Web sites from seizing control of your browser or even your computer. As a general rule, you should only use the `History` object to help visitors navigate through your particular Web site.

The `History` object includes the three methods listed in Table 11-5.

Method	Description
`back()`	Produces the same result as clicking a Web browser's Back button
`forward()`	Produces the same result as clicking a Web browser's Forward button
`go()`	Opens a specific document in the history list

Table 11-5 Methods of the `History` object

When you use a method or property of the `History` object, you must include a reference to the `History` object itself. For example, the `back()` and `forward()` methods allow a script to move backward or forward in a Web browser's history. To use the `back()` method, you must use the syntax `history.back()`.

The `go()` method is used for navigating to a specific Web page that has been previously visited. The argument of the `go()` method is an integer that indicates how many pages you want to navigate forward or backward in the history list. For example, `history.go(-2);` opens the Web page that is two pages back in the history list; the statement `history.go(3);` opens the Web page that is three pages forward in the history list. The statement `history.go(-1);` is equivalent to using the `back()` method, and the statement `history.go(1);` is equivalent to using the `forward()` method.

The `History` object contains a single property, the `length` property, which contains the specific number of documents that have been opened during the current browser session. To use the `length` property, you use the syntax `history.length;`. The `length` property does not contain the URLs of the documents themselves, but only an integer representing how many documents have been opened. The following code uses an alert dialog box to display the number of Web pages that have been visited during a Web browser session:

```
window.alert("You have visited " + history.length
    + " Web pages.");
```

The `History` object is included in this chapter to introduce you to all of the major objects in the browser object model. However, you should avoid using the `History` object to navigate to Web pages that have been opened during a Web browser session. Instead, you should use the full URL with the `href` property of the `Location` object, as explained in the next section.

Using the `Location` Object

When you want to allow users to open one Web page from within another Web page, you usually create a hypertext link with the `<a>` element. You can also use JavaScript code and the `Location` object to open Web pages. The **`Location` object** allows you to change to a new Web page from within JavaScript code. One reason you may want to change Web pages with JavaScript code is to redirect your Web site visitors to a different or updated URL. The `Location` object contains several properties and methods for working with the URL of the document that is currently open in a Web browser window. When you use a method or property of the `Location` object, you must include a reference to the `Location` object itself. For example, to use the `href` property, you must write `location.href = `*`URL`*`;`. Table 11-6 lists the `Location` object's properties, and Table 11-7 lists the `Location` object's methods.

Properties	Description
`hash`	A URL's anchor
`host`	The host and domain name (or IP address) of a network host
`hostname`	A combination of the URL's host name and port sections
`href`	The full URL address
`pathname`	The URL's path
`port`	The URL's port
`protocol`	The URL's protocol
`search`	A URL's search or query portion

Table 11-6 Properties of the `Location` object

Method	Description
`assign()`	Loads a new Web page
`reload()`	Causes the page that currently appears in the Web browser to open again
`replace()`	Replaces the currently loaded URL with a different one

Table 11-7 Methods of the `Location` object

The properties of the `Location` object allow you to modify individual portions of a URL. When you modify any properties of the `Location` object, you generate a new URL, and the Web browser automatically attempts to open that new URL. Instead of modifying individual portions of a URL, it is usually easier to change the `href` property, which represents the entire URL. For example, the statement `location.href = "http://www.google.com";` opens the Google home page.

The `assign()` method of the `Location` object performs the same action as changing the `href` property: It loads a new Web page. The statement `location.assign ("http://www.google.com");` is equivalent to the statement `location.href = "http://www.google.com";`.

The `reload()` method of the `Location` object is equivalent to the Reload button in Firefox or the Refresh button in Internet Explorer. It causes the page that currently appears in the Web browser to open again. You can use the `reload()` method without any arguments, as in `location.reload();`, or you can include a Boolean argument of true or false. Including an argument of true forces the current Web page to reload from the server where it is located, even if no changes have been made to it. For example, the statement `location.reload(true);` forces the current page to reload. If you include an argument of false, or do not include any argument at all, then the Web page reloads only if it has changed.

The `replace()` method of the `Location` object is used to replace the currently loaded URL with a different one. This method works somewhat differently from loading a new document by changing the `href` property. The `replace()` method actually overwrites one document with another and replaces the old URL entry in the Web browser's history list. In contrast, the `href` property opens a different document and adds it to the history list.

You can use `this.location` to retrieve the URL of the current Web page.

Using the `Navigator` Object

The **`Navigator` object** is used to obtain information about the current Web browser. It gets its name from Netscape Navigator, but it is also supported by Firefox, Internet Explorer, and other current browsers.

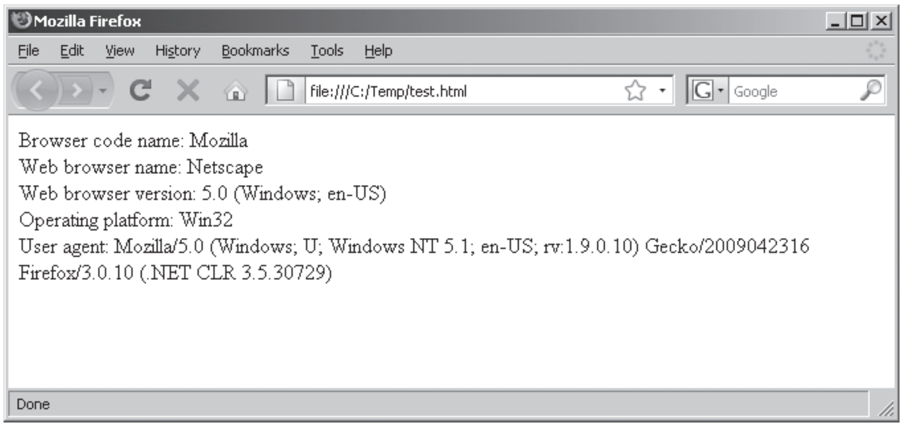
Some Web browsers, including Internet Explorer, contain unique methods and properties of the `Navigator` object that cannot be used with other browsers. Table 11-8 lists properties of the `Navigator` object that are supported by most current Web browsers, including Firefox and Internet Explorer.

Properties	Description
`appCodeName`	The Web browser code name
`appName`	The Web browser name
`appVersion`	The Web browser version
`platform`	The operating system in use on the client computer
`userAgent`	The string stored in the HTTP user-agent request header, which contains information about the browser, the platform name, and compatibility

Table 11-8 Properties of the `Navigator` object

The `Navigator` object is most commonly used to determine which type of Web browser is running. The statement `browserType = navigator.appName;` returns the name of the Web browser in which the code is running to the `browserType` variable. You can then use the `browserType` variable to determine which code to run for the specific type of browser. The **`with` statement** eliminates the need to retype the name of an object when properties of the same object are being referenced in a series. To use the `with` statement, you create a structure similar to an `if` statement and pass the name of the object as a conditional expression. You can then refer to all of the object properties without referring to the object itself. The following `with` statement prints the five properties of the `Navigator` object for Firefox 3.0. Figure 11-16 shows the output.

```
with (navigator) {
    document.write("<p>Browser code name: "
        + appCodeName + "<br />");
    document.write("Web browser name: "
        + appName + "<br />");
    document.write("Web browser version: "
        + appVersion + "<br />");
    document.write("Operating platform: "
        + platform + "<br />");
    document.write("User agent: " + userAgent
        + "</p>");
}
```


Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/Temp/test.html
Google
Done

```
Browser code name: Mozilla
Web browser name: Netscape
Web browser version: 5.0 (Windows; en-US)
Operating platform: Win32
User agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.0.10) Gecko/2009042316
Firefox/3.0.10 (.NET CLR 3.5.30729)
```

Figure 11-16 `Navigator` object properties in Firefox

Using the `Screen` Object

Computer displays can vary widely, depending on the type and size of the monitor, the type of installed graphics card, and the screen resolution and color depth selected by the user. For example, some notebook computers have small screens with limited resolution, while some desktop systems can have large monitors with very high resolution. The wide range of possible display settings makes it challenging to determine the size and positioning of windows generated by JavaScript. The **`Screen` object** is used to obtain information about the display screen's size, resolution, and color depth. Table 11-9 lists the properties of the `Screen` object that are supported by most current Web browsers, including Firefox and Internet Explorer.

Properties	Description
`availHeight`	Returns the height of the display screen, not including operating system features such as the Windows taskbar
`availWidth`	Returns the width of the display screen, not including operating system features such as the Windows taskbar
`colorDepth`	Returns the display screen's bit depth if a color palette is in use; if a color palette is not in use, returns the value of the `pixelDepth` property
`height`	Returns the height of the display screen
`pixelDepth`	Returns the display screen's color resolution in bits per pixel
`width`	Returns the width of the display screen

Table 11-9 Properties of the `Screen` object

The colorDepth and pixelDepth properties are most useful in determining the color resolution that the display supports. For example, if the colorDepth property returns a value of 32, which indicates high-color resolution, then you can use JavaScript to display a high-color image. However, if the colorDepth property returns a value of 16, which indicates medium-color resolution, then you may want to use JavaScript to display a lower-color image. The following code illustrates how to use the colorDepth property to determine which version of an image to display:

```
if (screen.colorDepth >= 32)
    document.write(
       "<img href='companyLogo_highres.jpg' />");
else if (screen.colorDepth >= 16)
    document.write(
       "<img href='companyLogo_mediumres.jpg' />");
else
    document.write(
       "<img href='companyLogo_lowres.jpg' />");
```

The remaining Screen object properties determine the size of the display area. For example, on a computer with a screen resolution of 1280 by 768, the following statements print "Your screen resolution is 1280 by 768."

```
var screenWidth = screen.width;
var screenHeight = screen.height;
document.write("<p>Your screen resolution is " +
     screenWidth + " by " + screenHeight + ".</p>");
```

One of the more common uses of the Screen object properties is to center a Web browser window in the display area. For windows generated with the window.open() method, you can center a window when it first opens by assigning values to the left and top options of the options argument. To center a window horizontally, subtract the width of the window from the screen width, divide the remainder by two, and assign the result to the left option. Similarly, to center a window vertically, subtract the height of the window from the screen height, divide the remainder by two, and assign the result to the top option. The following code demonstrates how to create a new window and center it in the display area:

Remember that the statements for opening a new window must be called from an event handler; otherwise, a Web browser's pop-up blocker will prevent the window from opening.

```
var winWidth=300;
var winHeight=200;
var leftPosition = (screen.width-winWidth)/2;
var topPosition = (screen.height-winHeight)/2;
var optionString = "width=" + winWidth + ",height="
    + winHeight + ",left=" + leftPosition + ",top="
    + topPosition;
OpenWin = window.open("", "CtrlWindow",
    optionString);
```

Next, you will modify the DRG Cycles Web page so that the bicycle window is centered in the display area.

To modify the DRG Cycles Web page so that the bicycle window is centered in the display area:

1. Return to the **index.html** document in your text editor.
2. Modify the `showBike()` function as follows so that it uses the `Screen` object to calculate the left and top positions of the bicycle window:

```
function showBike(linkTarget) {
    var propertyWidth=620;
    var propertyHeight=575;
    var winLeft = (screen.width-propertyWidth)/2;
    var winTop = (screen.height-propertyHeight)/2;
    var winOptions = "toolbar=no,menubar=no, ↵
        location=no,scrollbars=no,resizable=no";
    winOptions += ",width=" + propertyWidth;
    winOptions += ",height=" + propertyHeight;
    winOptions += ",left=" + winLeft;
    winOptions += ",top=" + winTop;
    bikeWindow = window.open(linkTarget,
        "bikeInfo", winOptions);
    bikeWindow.focus();
}
```

3. Save the **index.html** document, and then validate it with the W3C Markup Validation Service. Once the document is valid, close it in your text editor and open it in your Web browser. Click one of the bike links. The bicycle window should open and be centered on your screen.
4. Close your Web browser windows.

Short Quiz 3

1. Explain the security features of the `History` object.
2. How do you use the `History` object to navigate backward or forward in a Web browser's history?
3. How do you use the `Location` object to change to a new Web page?
4. What is the `Navigator` object and how do you use it?

Summing Up

- The browser object model (BOM) or client-side object model is a hierarchy of objects, each of which provides program access to a different aspect of the Web browser window or the Web page.
- The top-level object in the browser object model is the `Window` object, which represents a Web browser window.
- The `Document` object is arguably the most important object in the browser object model because it represents the Web page displayed in a browser.
- For elements that are represented by arrays, you can reference the object through the array instead of with the element name.
- Because the `Window` object is the global object, you do not have to include it in your statements.
- When you override an internal event handler with your own code, your code must return a value of true or false using the return statement.
- A rollover is an effect that occurs when your mouse moves over an element.
- You use the `style` property to modify an element's CSS properties with JavaScript.
- Whenever a new Web browser window is opened, a new `Window` object is created to represent the new window.
- When you open a new Web browser window, you can customize its appearance by using the `options` argument of the `window.open()` method.
- A `Window` object's `name` property can be used only to specify a target window with a link and cannot be used in JavaScript code.
- To control a new window by using JavaScript code located within the Web browser in which it was created, you must assign the new `Window` object created with the `window.open()` method to a variable.
- The `setTimeout()` method is used in JavaScript to execute code after a specific amount of time has elapsed.

- The `clearTimeout()` method is used to cancel a `setTimeout()` method before its code executes.
- The `setInterval()` method repeatedly executes the same code after being called only once.
- The `clearInterval()` method is used to clear a `setInterval()` method call.
- The `History` object maintains an internal list (known as a history list) of all the documents that have been opened during the current Web browser session.
- The `Location` object allows you to change to a new Web page from within JavaScript code.
- The `Navigator` object is used to obtain information about the current Web browser.
- The `with` statement eliminates the need to retype the name of an object when properties of the same object are being referenced in a series.
- The `Screen` object is used to obtain information about the display screen's size, resolution, and color depth.

Comprehension Check

1. Which of the following objects is also referred to as the global object?
   a. `Document` object
   b. `Window` object
   c. `Browser` object
   d. `Screen` object
2. Which of the following elements in the browser object model are referenced with arrays? (Choose all that apply.)
   a. images
   b. paragraphs
   c. forms
   d. links

3. Which of the following terms does not refer to the browser object model?
   a. Firefox object model
   b. JavaScript object model
   c. client-side object model
   d. Navigator object model
4. You must use the `Window` object or `self` property when referencing a property or method of the `Window` object. True or False?
5. Explain how to override an event with an event handler function.
6. Which of the following events are used to create rollover effects? (Choose all that apply.)
   a. `onclick`
   b. `onload`
   c. `onmouseover`
   d. `onmouseout`
7. Explain how to open a blank window with the `window.open()` method.
8. You use the options string of the `window.open()` method to specify any elements that you do not want created for the new window. True or False?
9. Which of the following arguments of the options string of the `window.open()` method identifies the horizontal coordinate where the window will be positioned?
   a. `left`
   b. `leftPosition`
   c. `x-axis`
   d. `moveTo`
10. Explain why you should include the `Window` object or `self` property when using the `open()` and `close()` methods of the `Window` object.

11. How do you control a new window that you have created with JavaScript code?
    a. by using the appropriate element in the `windows[]` array of the Windows object
    b. by using the `name` argument of the `window.open()` method
    c. by assigning the new `Window` object created with the `window.open()` method to a variable
    d. You cannot control a new window with JavaScript code.
12. Explain the difference between the `setTimeout()` and `setInterval()` methods. Which method is most often used for starting animation code that executes repeatedly?
13. How do you use JavaScript to modify an element's CSS properties?
14. Which of the following arguments do you pass to the `history.go()` method to navigate three pages back in the history list?
    a. `3`
    b. `-3`
    c. `2`
    d. `4`
15. You can use JavaScript code to navigate through a history list, but only if the currently displayed Web page exists within the same domain as the Web page containing the JavaScript code that is attempting to move through the list. True or False?
16. The full URL of a Web page is located in the ________ property of the `Location` object.
    a. `href`
    b. `hash`
    c. `src`
    d. `url`

17. Which property of the `Navigator` object returns the Web browser name?
    a. `browser`
    b. `browserName`
    c. `appName`
    d. `platform`
18. Explain how to use the `with` statement to reference an object's properties.
19. Which of the following properties of the `Screen` object returns the height of the display screen, not including operating system features such as the Windows taskbar?
    a. `displayHeight`
    b. `screenHeight`
    c. `availHeight`
    d. `height`
20. Explain how to center a window when it is created with the `window.open()` method.

Reinforcement Exercises

Exercise 11-1

Most Windows applications include an About dialog box that displays copyright information and other details about the program. In this exercise, you will create a script that opens a new window that is similar to an About dialog box.

1. Create a new HTML 5 document in your text editor and use "About Dialog Box Example" as the content of the `<title>` element.
2. Add a form to the document body that includes a single command button that reads "About this JavaScript Program".
3. Add code to the Web page that opens a new browser window when a user clicks the command button. Make the new window 100 pixels high by 300 pixels wide, and center it in the screen. Do not use any other display options. The new browser window should display a document named About.html (which you will create later in this exercise).

4. Save the document as **AboutExample.html** in the Exercises folder for Chapter 11.

5. Use the W3C Markup Validation Service to validate the **AboutExample.html** document and fix any errors. Once the document is valid, close it in your text editor.

6. Create another Web page that displays a single paragraph with the following text. Be sure to use your name in the paragraph.

```
<p>This program was created by your name.</p>
```

7. Add a button to the document body that closes the current window.

8. Save the document as **About.html** in the Exercises folder for Chapter 11.

9. Use the W3C Markup Validation Service to validate the **About.html** document and fix any errors. Once the document is valid, close it in your text editor.

10. Open **AboutExample.html** in your Web browser and test the script's functionality. The About window should appear centered in your screen.

11. Close your Web browser window.

Exercise 11-2

In this exercise, you create a script that repeatedly flashes advertising messages in a text box for a company named Central Valley Florist.

1. Create a new HTML 5 document in your text editor and use "Central Valley Florist" as the content of the `<title>` element.

2. Add a script section to the document head.

3. Next, add the following heading elements and form to the document body, which will display a message in a text box:

```
<h1>Central Valley Florist</h1>
<h2>Valentine's Day Specials</h2>
<form name="advertising" action="">
<p><input type="text" name="message" size="60"
value="Place your Valentine's Day orders today!" /></p>
</form>
```

4. In the script section, add the following code, which changes the message that is displayed in the text box:

```
var curMessage="message1";
var changeMessage;
function adMessage(){
  if (curMessage == "message2"){
    document.advertising.message.value
      = "Place your Valentine's Day orders today!";
    curMessage = "message1";
  }
  else {
    document.advertising.message.value
      = "All orders must be received by ↵
           February 12th!";
    curMessage = "message2";
  }
}
```

5. Finally, add the following onload event handler to the opening <body> tag:

```
<body onload="var changeQuote=setInterval ↵
    ('adMessage()',2000);">
```

6. Save the document as **ValentinesDayOrders.html** in the Exercises folder for Chapter 11.

7. Use the W3C Markup Validation Service to validate the **ValentinesDayOrders.html** document and fix any errors. Once the document is valid, close it in your text editor and then open it in your Web browser. The message should change every few seconds.

8. Close your Web browser window.

Exercise 11-3

In this exercise, you will create a script that redirects users to a different Web page after 10 seconds, or allows them to click a hyperlink.

1. Create a new HTML 5 document in your text editor and use "New Web Address" as the content of the `<title>` element.

2. Add a script section to the document head.

3. In the script section, add the following global variable declaration and function to handle the task of redirecting the Web page:

```
var killRedirect;
function updatedURL() {
     location.href="UpdatedURL.html";
}
```

4. Add the following `onload` event handler to the opening <body> tag:

```
<body onload="killRedirect = ↵
setTimeout('updatedURL()', 10000);">
```

5. Add the following elements and text to the document body:

```
<h2>The URL for the Web page you are trying to
reach has changed!</h2>
<p><strong>You will be automatically redirected in
ten seconds. Click the link if JavaScript is
disabled in your browser.</strong></p>
<p>Be sure to update your bookmark!</p>
<p><a href="UpdatedURL.html">UpdatedURL.html</a></p>
```

6. Save the document as **Redirect.html** in the Exercises folder for Chapter 11.

7. Use the W3C Markup Validation Service to validate the **Redirect.html** document and fix any errors. Once the document is valid, close it in your text editor.

8. Create another Web page that displays a single paragraph with the following text:

```
<p>You have reached the updated Web page.</p>
```

9. Save the document as **UpdatedURL.html** in the Exercises folder for Chapter 11.

10. Use the W3C Markup Validation Service to validate the **UpdatedURL.html** document and fix any errors. Once the document is valid, close it in your text editor and then open the **Redirect.html** document in your Web browser. In 10 seconds, the UpdatedURL.html document should open automatically.

11. Close your Web browser window.

Exercise 11-4

In addition to specifying the size and position of a window when it first opens, you can also change the size and position of an open window by using methods of the `Window` object. The `resizeTo()` method resizes a window to a specified size, and the `moveTo()` method moves a window to an absolute position. Using these methods with properties of the `Screen` object, you will create a script that resizes and repositions an open window so that it fills the screen.

1. Create a new HTML 5 document in your text editor and use “Maximize Browser Window” as the content of the `<title>` element.

2. Add a form to the document body that includes two command buttons: one that reads “Create New Window” and another that reads “Maximize New Window”.

3. Add a script section to the document head.

4. Add the following function for the Create New Window button. This function opens a document named MaxWindow.html (which you will create shortly) in a new browser window that is centered in the screen when a user clicks the command button.

```
var maxWindow;
function createWindow() {
    var winWidth=300;
    var winHeight=100;
    var winLeft = (screen.width-winWidth)/2;
    var winTop = (screen.height-winHeight)/2;
    var winOptions = ",width=" + winWidth;
    winOptions += ",height=" + winHeight;
    winOptions += ",left=" + winLeft;
    winOptions += ",top=" + winTop;
    maxWindow = window.open("MaxWindow.html",
        "newWindow", winOptions);
    maxWindow.focus();
}
```

5. Add the following function for the Maximize New Window button. The first statement in the function uses the `moveTo()` method of the `Window` object to move the window named maxWindow (which is created by the `createWindow()`

function) to position 0, 0, which represents the upper-left corner of the screen. The second statement uses the `resizeTo()` method of the `Window` object and the `availWidth` and `availHeight` properties of the `Screen` object to maximize the window. The final statement changes focus to the maximized window.

```
function maximizeWindow() {
     maxWindow.moveTo(0,0);
     maxWindow.resizeTo(screen.availWidth,
          screen.availHeight);
     maxWindow.focus();
}
```

6. Save the document as **MaximizeBrowser.html** in the Exercises folder for Chapter 11.

7. Use the W3C Markup Validation Service to validate the **MaximizeBrowser.html** document and fix any errors. Once the document is valid, close it in your text editor.

8. Create a Web page that conforms to the strict DTD, and add the following text and elements to the document body:

```
<p><strong>Resizing and Repositioning
Example</strong></p>
<form action="">
<p><input type="button" value="Close Window"
      onclick="window.close();" /></p>
</form>
```

9. Save the document as **MaxWindow.html** in the Exercises folder for Chapter 11.

10. Use the W3C Markup Validation Service to validate the **MaxWindow.html** document and fix any errors. Once the document is valid, close it in your text editor.

11. Open **MaximizeBrowser.html** in your Web browser, and click the **Create New Window** button. The new window should appear centered in the screen. Return to the **MaximizeBrowser.html** file in your Web browser, and click the Maximize New Window button. The new window should be resized and repositioned to fill the screen.

12. Close your Web browser windows.

Exercise 11-5

In this exercise, you will create a Web page for a greeting card company. The page will contain links that display images of greeting cards in a separate window. Your Exercises folder for Chapter 11 contains the following greeting card images that you can use for this project: birthday.jpg, halloween.jpg, mothersday.jpg, newyear.jpg, and valentine.jpg.

1. Create a new HTML 5 document in your text editor and use “Gosselin Greeting Cards” as the content of the `<title>` element.

2. Add the following text and elements to the document body. The `onclick` events in the links call a function named `showCard()` that handles the process of displaying each greeting card in a separate window. You create the `showCard()` function later in this exercise.

```
<h1>Gosselin Greeting Cards</h1>
<h2>All Occasions</h2>
<hr />
<p><a href="valentine.jpg"
onclick="showCard('valentine.jpg');return false">
Valentine's Day</a><br />
<a href="mothersday.jpg"
onclick="showCard('mothersday.jpg');return false">
Mother's Day</a><br />
<a href="halloween.jpg"
onclick="showCard('halloween.jpg');return false">
Halloween</a><br />
<a href="newyear.jpg"
onclick="showCard('newyear.jpg');return false">
New Year</a><br />
<a href="birthday.jpg"
onclick="showCard('birthday.jpg');return false">
Birthday</a></p>
```

3. Add a script section to the document head.

4. Add the following global variable to the script section. This variable will represent the window that displays the greeting card images.

```
var cardWindow;
```

5. Add the following function to the end of the script section. The function opens a new window, centered in the screen, that displays the selected greeting card image.

```
function showCard(linkTarget) {
   var propertyWidth=400;
   var propertyHeight=350;
   var winLeft = (screen.width-propertyWidth)/2;
   var winTop = (screen.height-propertyHeight)/2;
   var winOptions =
       "toolbar=no,menubar=no,location=no, ↵
       scrollbars=yes,resizable=no";
   winOptions += ",width=" + propertyWidth;
   winOptions += ",height=" + propertyHeight;
   winOptions += ",left=" + winLeft;
   winOptions += ",top=" + winTop;
   cardWindow = window.open(linkTarget,
       "cardInfo", winOptions);
   cardWindow.focus();
}
```

6. Save the document as **GreetingCards.html** in the Exercises folder for Chapter 11.

7. Use the W3C Markup Validation Service to validate the **GreetingCards.html** document and fix any errors. Once the document is valid, close it in your text editor, open it in your Web browser, and test the functionality.

8. Close your Web browser window.

Exercise 11-6

You have probably seen Web sites that invite you to add them to your browser's favorites list. With Internet Explorer, you can create a link that automatically adds the Web page to the favorites list by assigning a value of `javascript:window.external.AddFavorite(`*`url, site name`*`)` to the link's `href` property. Firefox does not contain similar functionality, so you need to use the `Navigator` object to determine the browser type. In this exercise, you will create a script that contains functionality for adding Course Technology's Web site to a browser's favorites list.

1. Create a new HTML 5 document in your text editor and use "Add to Favorites" as the content of the `<title>` element.

2. Add a script section to the document body.

3. Add the following statements to the script section. The first two statements retrieve the browser's name and version from the `Navigator` object. The remaining statements create text variables that will be used to create the bookmark link.

```
var browserName = navigator.appName;
var browserVer = parseInt(navigator.appVersion);
var linkText = "Add Course Technology↵
    to your favorites!";
var url = "http://www.course.com";
var pageName = "Course Technology";
var favLink = "";
```

4. Add the following `if` statement to the end of the script section. The conditional expression determines whether the browser name is equal to "Microsoft Internet Explorer" and whether the browser version is greater than or equal to 4. If so, statements within the `if` statement build a link that automatically adds the Course Technology Web site to the favorites list in Internet Explorer.

```
if (browserName == "Microsoft Internet Explorer"
    && browserVer >= 4) {
    favLink = "<p><a href=\"javascript:window.↵
       external.AddFavorite(url, pageName)\"";
    favLink += " onmouseover=\"window.status='";
    favLink += linkText + "'; return true\"";
    favLink += " onmouseout=\"window.status=";
    favLink += "''" + "; return true\"";
    favLink += ">" + linkText + "</a></p>";
    document.write(favLink);
}
```

5. Add the following `else` clause to the end of the script section to print "Add Course Technology to your favorites! (Ctrl+D)" for all other browsers:

```
else
    document.write("<p>Add Course Technology↵
        to your favorites! (Ctrl+D)</p>");
```

6. Save the document as **AddToFavorites.html** in the Exercises folder for Chapter 11, and validate the document with the W3C Markup Validation Service. Once the document is valid, close it in your text editor and then open it in Internet Explorer and test the functionality.

7. Close your Web browser window.

Discovery Projects

For the following projects, save the files you create in your Projects folder for Chapter 11. Be sure to validate each Web page with the W3C Markup Validation Service.

Discovery Project 11-1

Your Projects folder for Chapter 11 contains five advertising images for a concert series, concert1.gif through concert5.gif. Create a script that cycles through the images and displays each one for five seconds. Save the document as **ConcertAds.html**.

Discovery Project 11-2

Create a Web page with a list of your favorite links. At the top of the page, include a check box with the text "Open link in a new window." If a user clicks the check box, the links on the page should open in a new window. Otherwise, the links should be loaded into the current window. Save the document as **LinkWindow.html**.

Discovery Project 11-3

A common use of the `onmouseover` and `onmouseout` event handlers is to change the button image displayed for a navigational link on a Web page. For example, holding your mouse over an image of a Home button (that jumps to the Web site's home page) could replace the image with one that is more vivid in order to clearly identify the page that is the target of the link. Your Projects folder for Chapter 11 contains eight images: home1.gif, home2.gif, faq1.gif, faq2.gif, guestbook1.gif, guestbook2.gif, join1.gif, and join2.gif. These images represent typical navigational buttons you find on a Web site. The second version of each button is slightly more vivid than the first version. Create a Web page that displays the first version of each button as image links, using the `<a>` element. Holding your mouse over each image should display the more vivid version of the image, while moving your mouse off the image should display the less vivid version. Do not worry about actually creating a Web page as the target of each link; just assign an empty string to each `<a>` element's `href` attribute. Save the document as **Buttons.html**.

Discovery Project 11-4

You have probably come across Web sites that briefly display a sponsor's advertisement before redirecting you to the page you originally requested. Create such an ad for a real estate company named Central Valley Realtors. Start by creating a Web page named **CVR1.html**. In the document body, create a table with two columns. In the left column, display the cvb1.gif image, which is located in your Projects folder for Chapter 11. The cvb1.gif file is an animated GIF file that displays an advertisement for a company named Central Valley Builders. In the right column, display three paragraphs. In the first paragraph, display the word "Advertisement". In the second paragraph, display the text "The Central Valley Realtors home page will be displayed in *n* seconds." Use a text field for the number of seconds, which means you will need to create a form to contain the text field. Set the default value of the text field to 15 seconds. In the third paragraph, include a "Skip advertisement" link that opens a Web page named CVR2.html (which is the "real" home page for Central Valley Realtors). Within the CVR1.html page's opening `<body>` tag, add an `onload` event handler that calls a function named `startAdPage()`. Within the `startAdPage()` function, include two statements: one that uses a `setInterval()` method to call a function named `changeAd()` every five seconds, and another statement that uses a `setInterval()` method to call a function named `startCountdown()` every second. Create the `changeAd()` function so that, every five seconds, it alternates the image in the document body with the three images in your Projects folder for Chapter 11: cvb1.gif, cvb2,gif, and cvb3.gif. Create the `startCountdown()` function so that it changes the value assigned to the text field in the document body to the value of a variable named `count`, which is decreased by a value of 1 (from 15 to 1) each time the `startCountdown()` function executes. When the count reaches zero, clear both of the intervals and redirect the browser to the CVR2.html page. Create the **CVR2.html** page so it contains an `<h1>` element that reads "Central Valley Realtors" and a paragraph element that reads "Welcome to our home page."