CHAPTER 12

Images and Dynamic HTML

In This Chapter

How to precache images

How to swap images for mouse rollovers

What you can do with Dynamic HTML and scripting

The previous eight lessons have been intensive, covering a lot of ground for both programming concepts and JavaScript. Now it's time to apply what you've learned to learn some more. I will cover two areas here. First, I show you how to implement the ever-popular mouse rollover, where images swap when the user rolls the cursor around the screen. Finally, I introduce you to concepts surrounding scripted control of Dynamic HTML in the level 4 browsers, and the Netscape <LAYER> tag in particular.

The Image Object

One of the objects contained by the document is the image object. Unfortunately this object is not available in all scriptable browsers. You can script it in Navigator 3 and later or Internet Explorer 4. Therefore, everything you learn here about the image object won't apply to Navigator 2 or Internet Explorer 3 for Windows.

Because a document can have more than one image, image object references are stored in the object model as an array. You can therefore reference an image by array index or image name:

```
document.images[n]
document.imageName
```

Each of the <IMG> tag's attributes is accessible to JavaScript as a property of the image object. No mouse-related event handlers are affiliated with the image object (although they are in Internet Explorer 4). If you want to make an image a clickable item, surround it with a link (and set the image's border to zero) or attach a client-side image map to it. The combination of a link and image is how you make a clickable image button (the image type of form input element is not a scriptable object).

Interchangeable images

When the image became an object with the release of Navigator 3, it did so with a special behavior: a script could

change the image occupying the rectangular space already occupied by an image. This was one of the first examples of dynamically changing a page's content after a page has loaded.

The script behind this kind of image change is simple enough. All it entails is assigning a new URL to the image object's `src` property. The size of the image on the page is governed by the `HEIGHT` and `WIDTH` attributes set in the `<IMG>` tag as the page loads and cannot be modified. Therefore, you need to select your images carefully so that all images that supply content to a document image object are the same size. Otherwise the image scales to the original dimensions assigned to the object.

Precaching images

Images often take several extra seconds to download from a Web server. If you are designing your page so an image changes in response to user action, you usually want the same fast response that users are accustomed to in multimedia programs. Making the user wait many seconds for an image to change can severely distract from enjoyment of the page.

To the rescue comes the ability to precache images. The tactic that works best is to preload the image into the browser's image cache when the page initially loads. Users will be less impatient for those few extra seconds during the main page loading than they would be while waiting for an image to change.

Precaching an image requires constructing an image object in memory. An image object created in memory is different in some respects from the document image object that you create with the `<IMG>` tag. Memory-only objects are created by script, and you don't see them on the page at all. But their presence in the document forces the browser to load the images as the page loads. JavaScript provides a constructor function for the memory type of image object as follows:

```
var myImage = new Image(width, height)
```

Parameters to the constructor function are the width and height of the image. These dimensions should match the width and height of the `<IMG>` tag attributes where the new image will eventually go. Once the image object exists in memory, you can then assign a filename or URL to the `src` property of that image object:

```
myImage.src = "someArt.gif"
```

When JavaScript encounters a statement assigning a URL to an image object's `src` property, it instructs the browser to go out and load that image into the image cache. All the user sees is some extra loading information in the status bar, as if another image were in the page. By the time the `onLoad=` event handler fires, all images generated in this way are tucked away in the image cache. You can then assign your cached image's `src` property or the actual image URL to the `src` property of the document image created with the `<IMG>` tag:

```
document.images[0].src = myImage.src
document.images[0].src = someArt.gif
```

The change to the image in the document is instantaneous.

Listing 12-1 shows a simple listing for a page that has one `<IMG>` tag and a select list that lets you replace the document image with any of four precached images (including the original image specified for the tag). If you type this listing — as I

strongly recommend — you can obtain copies of the four image files from the companion CD-ROM in the Chapter 18 listings.

As the page loads, it executes several statements immediately. These statements create four new memory image objects and assigns a file name to the `src` property of each one. These images will be loaded into the image cache as the page loads. Down in the Body portion of the document, an `<IMG>` tag stakes its turf on the page and loads one of the images as a starting image.

A select object lists user-friendly names for the pictures while housing the names of the image files (without the extension) already precached. When the user makes a selection from the list, the `loadCached()` function extracts the value of the selected item and assembles the complete name of the desired image. That name is assigned to the `src` property of the document's image object, and the image changes in a snap.

Listing 12-1: **Precaching Images**

```
<HTML>
<HEAD>
<TITLE>Image Object</TITLE>
<SCRIPT LANGUAGE="JavaScript1.1">
// pre-cache four images
image1 = new Image(120,90)
image1.src = "desk1.gif"
image2 = new Image(120,90)
image2.src = "desk2.gif"
image3 = new Image(120,90)
image3.src = "desk3.gif"
image4 = new Image(120,90)
image4.src = "desk4.gif"

// load an image chosen from select list
function loadCached(list) {
        var img = list.options[list.selectedIndex].value
        document.thumbnail.src = img + ".gif"
}
</SCRIPT>
</HEAD>

<BODY >
<H2>Image Object</H2>
<IMG SRC="desk1.gif" NAME="thumbnail" HEIGHT=90 WIDTH=120>
<FORM>
<SELECT NAME="cached" onChange="loadCached(this)">
<OPTION VALUE="desk1">Bands
<OPTION VALUE="desk2">Clips
<OPTION VALUE="desk3">Lamp
<OPTION VALUE="desk4">Erasers
</SELECT>
</FORM>
</BODY>
</HTML>
```

Creating image rollovers

A favorite technique to add some pseudo-excitement to a page is to swap button images as the user rolls the cursor atop them. The degree of change to the image is largely a matter of taste. The effect can be subtle — a slight highlight or glow around the edge of the original image — or drastic — a radical change of color. Whatever your approach, the scripting is the same.

When several of these graphical buttons occur in a group, I tend to organize the memory image objects as arrays and create naming and numbering schemes that facilitate working with the arrays. Listing 12-2 shows such an arrangement for four buttons that control a jukebox. The code in the listing is confined to the image-swapping portion of the application. This is the most complex and lengthiest listing of the tutorial, so it requires a bit of explanation as it goes along.

Listing 12-2: **Image Rollovers**

```
<HTML>
<HEAD>
<TITLE>Jukebox/Image Rollovers</TITLE>
```

Because the image object was not in Navigator until Version 3, I limit access to the code via the `LANGUAGE` attribute of the `<SCRIPT>` tag. Here the language is specified as JavaScript Version 1.1, the version incorporated into Navigator 3. All versions of Navigator beyond that and Version 4 of Internet Explorer will be able to use this script.

```
<SCRIPT LANGUAGE="JavaScript1.1">
```

The first task in the script is to build two arrays of image objects. One array stores information about the images depicting the graphical button's "off" position; the other is for images depicting their "on" position. After creating the array and assigning new blank image objects to the first four elements of the array, I go through the array again, this time assigning file pathnames to the `src` property of each object stored in the array. Since these lines of code are executing as the page loads, the images load into the image cache along the way.

```
// pre-cache all 'off' button images
var offImgArray = new Array()
offImgArray[0] = new Image(75,33)
offImgArray[1] = new Image(75,33)
offImgArray[2] = new Image(75,33)
offImgArray[3] = new Image(86,33)

// off image array -- set 'off' image path for each button
offImgArray[0].src = "images/playoff.jpg"
offImgArray[1].src = "images/stopoff.jpg"
offImgArray[2].src = "images/pauseoff.jpg"
offImgArray[3].src = "images/rewindoff.jpg"

// pre-cache all 'on' button images
var onImgArray = new Array()
onImgArray[0] = new Image(75,33)
onImgArray[1] = new Image(75,33)
```

```
onImgArray[2] = new Image(75,33)
onImgArray[3] = new Image(86,33)

// on image array -- set 'on' image path for each button
onImgArray[0].src = "images/playon.jpg"
onImgArray[1].src = "images/stopon.jpg"
onImgArray[2].src = "images/pauseon.jpg"
onImgArray[3].src = "images/rewindon.jpg"
```

As you will see in the HTML below, when the user rolls the mouse atop any of the document image objects, the `onMouseOver=` event handler (from the link object surrounding the document image object) invokes the `imageOn()` function, passing an index value for the particular image. The index value corresponds to the index values of the image arrays defined above. The `imageOn()` function uses that index value to assign the URL from the `onImgArray` entry to the corresponding document image `src` property.

```
// functions that swap images & status bar
function imageOn(i) {
        document.images[i].src = onImgArray[i].src
}
```

The same goes for the `onMouseOut=` event handler, which needs to turn the image off by invoking the `imageOff()` function with the same index value.

```
function imageOff(i) {
        document.images[i].src = offImgArray[i].src
}
```

Both the `onMouseOver=` and `onMouseOut=` event handlers set the status bar to prevent the ugly `javascript:` URL from appearing there as the user rolls the mouse atop the image. The `onMouseOut=` event handler sets the status bar message to an empty string.

```
function setMsg(msg) {
        window.status = msg
        return true
}
```

For this demonstration, I have disabled the functions that control the jukebox. But I leave the empty function definitions here so they catch the calls made by the clicks of the links associated with the images.

```
// controller functions (disabled)
function playIt() {
}
function stopIt() {
}
function pauseIt(){
}
function rewindIt() {
}
</SCRIPT>
</HEAD>

<BODY>
<CENTER>
<FORM>
Jukebox Controls<BR>
```

(continued)

Listing 12-2 *Continued*

I surround each document image object with a link because the link object has the event handlers needed to respond to the mouse rolling over the area. Each link's `onMouseOver=` event handler calls the `imageOn()` function with the index value of this image's position in both image arrays defined earlier. Because both the `onMouseOver=` and `onMouseOut=` event handlers require a `return true` statement to work, I combined the second function call (to `setMsg()`) with the `return true` requirement. The `setMsg()` function always returns `true`, which is combined with the `return` keyword before the call to the `setMsg()` function. It's just a trick to reduce the amount of code in these event handlers.

```
<A HREF="javascript:playIt()" onMouseOver="imageOn(0); return
setMsg('Play the selected tune')" onMouseOut="imageOff(0); return
setMsg('')">
       <IMG SRC="images/playoff.jpg" NAME="hiliteBtn0" HEIGHT=33
WIDTH=75 BORDER=0></A>
<A HREF="javascript:stopIt()" onMouseOver="imageOn(1); return
setMsg('Stop the playing tune')" onMouseOut="imageOff(1); return
setMsg('')">
       <IMG SRC="images/stopoff.jpg" NAME="hiliteBtn1" HEIGHT=33
WIDTH=75 BORDER=0></A>
<A HREF="javascript:pauseIt()" onMouseOver="imageOn(2); return
setMsg('Pause the playing tune')" onMouseOut="imageOff(2); return
setMsg('')">
       <IMG SRC="images/pauseoff.jpg" NAME="hiliteBtn2" HEIGHT=33
WIDTH=75 BORDER=0></A>
<A HREF="javascript:rewindIt()" onMouseOver="imageOn(3); return
setMsg('Rewind back to the beginning')" onMouseOut="imageOff(3); return
setMsg('')">
       <IMG SRC="images/rewindoff.jpg" NAME="hiliteBtn3" HEIGHT=33
WIDTH=86 BORDER=0></A>
</FORM>
</CENTER>
</BODY>
</HTML>
```

The results of this lengthy script can be seen in Figure 12-1. As the user rolls the mouse atop one of the images, it changes from a light to dark color by swapping the entire image.

More Dynamism in HTML

The image object swapping technique is but a preview of what the newest developments in Dynamic HTML are all about. With each new generation of browser, scripts can change more content on the fly. Navigator's implementation of this involves a new object and tag. The `<LAYER>` tag identifies an object within a document that can contain its own separate document and can be positioned and sized anywhere in the document. It is kind of like a floating frame. But unlike a frame, a layer is contained by a document, not a parent window.

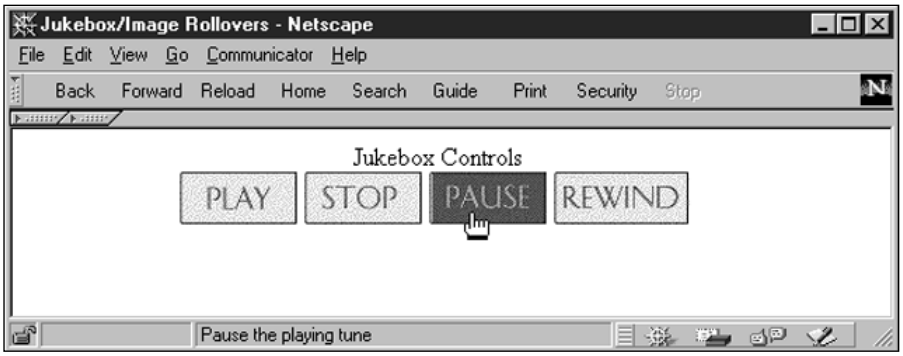

Figure 12-1: Typical mouse rollover image swapping

As described in depth in Chapter 19, the layer object has a large number of properties and methods, most of which are unique to the layer object — new behaviors and facilities for a new kind of object. Most of the methods, for example, concern themselves with the positioning and viewable size of the layer. You can also hide a layer and change the HTML that goes into a layer after the page has loaded.

Internet Explorer 4 treats Dynamic HTML differently, and does not implement a layer object. But as you will discover in Chapter 41, it is possible to create Dynamic HTML pages that work with both systems, thanks to industry standards efforts in this arena.

And so ends the final lesson of the *JavaScript Bible, Third Edition* tutorial. If you have gone through every lesson and tried your hand at the exercises, you are now ready to dive into the rest of the book to learn the fine details and many more features of both the document object model and the JavaScript language.

Exercises

1. Explain the difference between a document image object and the memory type of image object.
2. Write the JavaScript statements needed to precache an image named "jane.jpg" that will later be used to replace the document image defined by the following HTML:

   ```
   <IMG NAME="people" SRC="john.jpg" HEIGHT=120 WIDTH=100>
   ```

3. With the help of the code you wrote for question 2, write the JavaScript statement that replaces the document image with the memory image.
4. Document image objects do not have event handlers for mouse events. How do you trigger scripts needed to swap images for mouse rollovers?

✦ ✦ ✦