

Image and Area Objects

For users of Navigator 3, Internet Explorer 4, and later browsers, images and areas — those items created by the `` and `<AREA>` tags — are first-class objects that can be scripted for enhanced interactivity. The space reserved for an `` tag can be refreshed with other images of the same size, perhaps to show the highlighting of an icon button when the cursor rolls atop it. And with scriptable client-side area maps, pages can be smarter about how users' clicks on image regions respond.

One further benefit afforded scripters is that images can be preloaded into the browser's image cache as the page loads. Therefore, if you intend to swap images in response to user action, no delay occurs in making the first swap: The image is already in the image cache ready to go.

Image Object

<i>Properties</i>	<i>Methods</i>	<i>Event Handlers</i>
<code>border</code>	(None)	<code>onAbort=</code>
<code>complete</code>		<code>onError=</code>
<code>height</code>		<code>onLoad=</code>
<code>hspace</code>		
<code>lowsrc</code>		
<code>name</code>		
<code>src</code>		
<code>vspace</code>		
<code>width</code>		
<code>x</code>		
<code>y</code>		

18

CHAPTER



In This Chapter

How to precache images

Swapping images after a document has loaded

Creating interactive client-side image maps



Syntax

Creating an image:

```
<I MG
    SRC=" I mageURL"
    [ LOWSRC=" LowResI mageURL" ]
    NAME=" I mageName"
    [ HEI GHT=" P i xel Count " | " P ercent ageVal ue%" ]
    [ WI DTH=" P i xel Count " | " P ercent ageVal ue%" ]
    [ HSPACE=" P i xel Count " ]
    [ VSPACE=" P i xel Count " ]
    [ BORDER=" P i xel Count " ]
    [ ALI GN=" l eft" | " r i ght" | " t op" | " a bsmi ddl e" | " a bsbott om" |
      " t ext t op" | " m i ddl e" | " b a s e l i n e" | " b o t t o m"
    ]
    [ I SMAP]
    [ USEMAP=" #AreaMapName" ]
    [ onLoad=" h a n d l e r T e x t O r F u n c t i o n" ]
    [ onAbort=" h a n d l e r T e x t O r F u n c t i o n" ]
    [ onError=" h a n d l e r T e x t O r F u n c t i o n" ] >

</BODY>

i mageName = new I mage([ p i x e l W i d t h, p i x e l H e i g h t ])
```

Accessing image properties or methods:

```
[ w i n d o w. ] document. i mageName. p r o p e r t y | m e t h o d([ p a r a m e t e r s ])

[ w i n d o w. ] document. i m a g e s [ i n d e x ]. p r o p e r t y | m e t h o d([ p a r a m e t e r s ])
```

	Nav2	Nav3	Nav4	IE3/J1	IE3/J2	IE4/J3
Compatibility		✓	✓	(✓)		✓

About this object

Images have been in the HTML vocabulary since the earliest days, but Netscape Navigator 3 was the first to treat them like first-class JavaScript objects. Internet Explorer 3 for the Macintosh includes a partial implementation of the image object (to allow image precaching and swapping), and all flavors of Internet Explorer 4 treat images as true document objects. The primary advantage of this rating is that scripts can read a number of properties from images and, more importantly, change the image that occupies the image object's rectangular space on the page, even after the document has loaded and displayed an initial image. The key to this scriptability is the `src` property of an image.

In a typical scenario, a page loads with an initial image. That image's tags specify any of the extra attributes, such as `HEI GHT` and `WI DTH` (which help speed the rendering of the page), and whether the image uses a client-side image map to make it interactive (see the area object later in this chapter). As the user spends time on the page, the image can then change (perhaps in response to user action or some timed event in the script), replacing the original image with a new one in the same space (the rectangle cannot be modified after the first image loads).

Another benefit of treating images as objects is that a script can create a virtual image to hold a preloaded image (the image gets loaded into the image cache without having to display the image). The hope is that one or more unseen images will load into memory while the user is busy reading the page or waiting for the page to download. Then, in response to user action on the page, an image can change almost instantaneously, rather than forcing the user to wait for the image to load on demand.

To preload an image, begin by generating a new, empty image object as a global variable. You can preload images either in immediate script statements that run as the page loads or in response to the window's `onLoad` event handler. An image that is to take the place of an `` tag picture must be the same size as the `HEIGHT` and `WIDTH` attributes of the tag. Moreover, you help the virtual image object creation if you specify the width and height in the parameters of `new Image()` constructor. Then assign an image file URL to its `src` property:

```
oneImage = new Image(55, 68)
oneImage.src = "neatImage.gif"
```

As this image loads, you see the progress in the status bar, just like any image. Later, assign the `src` property of this stored image to the `src` property of the image object that appears on the page:

```
document.images[0].src = oneImage.src
```

Depending on the type and size of image, you will be amazed at the speedy response of this kind of loading. With small-palette graphics, the image displays instantaneously.

A popular user interface technique is to change the appearance of an image that represents a clickable button when the user rolls the mouse pointer atop that art. If you surround an image with a link in the latest browser versions, you can even change images when the user presses and releases the mouse button (see Chapter 17).

You can accomplish this many ways, depending on the number of images you need to swap. I employ different methods in relevant listings, such as Listing 17-3 and 18-2. But the barest minimum can be accomplished by preloading both versions of an image as the document loads, and then changing the `src` property of the image object in the relevant mouse event handler. For example, in a script in the `<HEAD>` section, you can preload "normal" and "highlighted" versions of some button art in the following manner:

```
var normalButton = new Image(80, 20)
var hilitedButton = new Image(80, 20)
normalButton.src = "homeNormal.gif"
hilitedButton.src = "homeHilited.gif"
```

Then, in the body of the document, you would create a linked `` tag along these lines:

```
<A HREF="default.html"
    onMouseOver="document.ToHome.src = hilitedButton.src; return
true"
    onMouseOut="document.ToHome.src = normalButton.src; return true"
>
```

```
<IMG NAME="ToHome" SRC="homeNormal.gif"
      WIDTH=80 HEIGHT=20 BORDER=0
></A>
```

When a user rolls the mouse over the linked image, the `onMouseOver=` event handler changes the URL of the image to the highlighted version loaded into the cache earlier; when the mouse rolls out of the image, the image changes back.

The speed with which this kind of image swapping takes place may lead you to consider using this method for animation. Though this method may be practical for brief bursts of animation, the many other ways of introducing animation to your Web page (such as via GIF89a-standard images, Java applets, and a variety of plug-ins) produce animation that offers better speed control and the like. In fact, swapping preloaded JavaScript image objects for some cartoon-like animations may actually be too fast. You could build a delay mechanism around the `setInterval()` method, but the precise timing between frames would vary with client processor.

Note

If you place an image inside a table cell, Navigator 3 sometimes generates two copies of the image object in its object model. This can disturb the content of the `document.images[]` array for your scripts. Specifying `HEIGHT` and `WIDTH` attributes for the image can sometimes cure the problem. Otherwise you have to craft scripts so they don't rely on the `document.images[]` array.

Properties

`border`
`height`
`hspace`
`name`
`vspace`
`width`

	Nav2	Nav3	Nav4	IE3/J1	IE3/J2	IE4/J3
Compatibility		✓	✓			✓

Value: Varies **Gettable:** Yes **Settable:** No

This long list of properties for the image object provides read-only access to the corresponding `` tag attributes that affect the visual appearance of the image. The values that these properties return are the same as those used to initially set the attributes. Once the image is defined by its `` tag, you cannot change any of these properties to influence the appearance on the page. More than likely, you will never need to refer to these properties, unless a script that is about to write a new page wants to replicate settings from an existing image object.

If you need to set these attributes with JavaScript writing a page on-the-fly, use the `document.write()` method to write the equivalent of the actual HTML tags

and the values you would use in a regular document. Also see Chapters 41 through 43 regarding the use of Cascading Style Sheets for other ways to impact the appearance of a loaded image.

Related Items: None.

complete

Value: Boolean **Gettable:** Yes **Settable:** No

	Nav2	Nav3	Nav4	IE3/J1	IE3/J2	IE4/J3
Compatibility		✓	✓			✓

There may be times when you want to make sure that an image object is not still in the process of loading before allowing another process to take place. This situation is different from waiting for an image to load before triggering some other process (which you can do via the image object's `onLoad=` event handler). To verify that the image object displays a completed image, check for the Boolean value of the `complete` property. To verify that a particular image file has loaded, first find out whether the `complete` property is true; then compare the `src` property against the desired filename.

An image's `complete` property switches to true even if only the specified `LOWSRC` image has finished loading. Do not rely on this property alone for determining whether the `SRC` image has loaded if both `SRC=` and `LOWSRC=` attributes are set in the `` tag.

Note

This property is not reliable in Navigator 4 and Internet Explorer 4. The value returns true in all instances.

Example

To experiment with the `image.complete` property, quit and relaunch Navigator before loading Listing 18-1 (in case the images are in memory cache). As each image loads, click the "Is it loaded yet?" button to see the status of the `complete` property for the image object. The value is false until the loading has finished, at which time the value becomes true. If you experience difficulty with this property in your scripts, try adding an `onLoad=` event handler (even if it is empty, as in Listing 18-1) to your `` tag.

Listing 18-1: Scripting `image.complete`

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript 1.1">
function loadIt(theImage, form) {
    form.result.value = ""
    document.images[0].src = theImage
}
function checkLoad(form) {
    form.result.value = document.images[0].complete
```

document.imageObject.complete

```

    }
  </SCRIPT>
</HEAD>
<BODY>
  <IMG SRC="cpu2.gif" WIDTH=120 HEIGHT=90 onLoad="" >
  <FORM>
    <INPUT TYPE="button" VALUE="Load keyboard"
    onClick="loadIt('cpu2.gif',this.form)" >
    <INPUT TYPE="button" VALUE="Load arch"
    onClick="loadIt('arch.gif',this.form)" ><P>
    <INPUT TYPE="button" VALUE="Is it loaded yet?"
    onClick="checkLoad(this.form)" >
    <INPUT TYPE="text" NAME="result" >
  </FORM>
</BODY>
</HTML>

```

Related Items: `img.src` property; `img.lowsrc` property; `onLoad=` event handler.

lowsrc

Value: String **Gettable:** Yes **Settable:** No

	Nav2	Nav3	Nav4	IE3/J1	IE3/J2	IE4/J3
Compatibility		✓	✓			✓

For image files that take several seconds to load, recent browsers enable you to specify a lower-resolution image or some other quick-loading placeholder to stand in while the big image crawls to the browser. You assign this alternate image via the `LOWSRC=` attribute in the `` tag. The attribute is reflected in the `lowsrc` property of an image object.

Although I list this property as not being settable, you can make a temporary setting safely, provided that you set it in the same script segment as the statement that loads an image to the `src` property. Be aware that if you specify a `LOWSRC` image file, the `complete` property switches to true and the `onLoad=` event handler fires when the alternate file finishes loading: They do not wait for the main `SRC` file to load.

Example

See the example for the image object's `onLoad=` event handler to see how these elements affect each other.

Related Items: `window.open()` method; `window.focus()` method.

src

Value: String **Gettable:** Yes **Settable:** Yes

	Nav2	Nav3	Nav4	IE3/J1	IE3/J2	IE4/J3
Compatibility		✓	✓	(✓)		✓

The key to best using the image object from JavaScript is the `src` property, which enables you to assign a URL or preloaded image's `src` property as a way of loading a new image into an existing image's rectangular space. Be aware that the height and width of the image object are defined by the attributes in the `` tag. If you simply set the `src` property of an image object to a new image filename, JavaScript scales the image to fit the dimensions defined by the `` tag attributes or (if none are provided) by the dimensions of the first image file loaded into that object. JavaScript ignores any discrepancy that exists between these settings and the parameters passed to a new `Image(x, y)` statement.

Example

In the following example (Listing 18-2), you see a few applications of the image object. Of prime importance is a comparison of how precached and regular images feel to the user. As a bonus, I include an example of how to set a timer to automatically change the images displayed in an image object. This feature seems to be a popular request among sites that display advertising banners.

Listing 18-2: A Scripted Image Object and Rotating Images

```
<HTML>
<HEAD>
<TITLE>Image Object</TITLE>
<SCRIPT LANGUAGE="JavaScript 1.1">

imageDB = new Array(4)
for (var i = 0; i < imageDB.length; i++) {
    imageDB[i] = new Image(120, 90)
    imageDB[i].src = "desk" + (i+1) + ".gif"
}

function loadIndividual(form) {
    var gifName =
form.individual.options[form.individual.selectedIndex].value
    document.thumbnail1.src = gifName
}

function loadCached(form) {
    var gifIndex = form.cached.selectedIndex
    document.thumbnail2.src = imageDB[gifIndex].src
}

function checkTimer() {
    if (document.Timer.timerBox.checked) {
        var newIndex = 0
        var gifName = document.thumbnail2.src
        var gifIndex = (gifName.charAt(gifName.length - 5)) - 1
```

(continued)

Listing 18-2 (continued)

```

        if (gifIndex < imageDB.length - 1) {
            newIndex = gifIndex + 1
        }
        document.thumbnail2.src = imageDB[newIndex].src
        document.selections.cached.selectedIndex = newIndex
        var timeoutID = setTimeout("checkTimer()", 5000)
    }
}
</SCRIPT>
</HEAD>

<BODY onLoad=checkTimer() >
<CENTER>
<H2>Image Object Demonstration</H2>
<TABLE BORDER=3>
<TR><TH></TH><TH>Individually Loaded</TH><TH>Pre-cached</TH></TR>
<TR><TD ALI GN=RI GHT><B>Image: </B></TD>
<TD><IMG SRC="cpu1.gif" NAME="thumbnail1" HEIGHT=90 WIDTH=120></TD>
<TD><IMG SRC="desk1.gif" NAME="thumbnail2" HEIGHT=90 WIDTH=120></TD>
</TR>
<TR><TD ALI GN=RI GHT><B>Select image: </B></TD>
<FORM NAME="selections">
<TD>
<SELECT NAME="individual" onChange="loadIndividually(this.form)">
<OPTION VALUE="cpu1.gif">Wi res
<OPTION VALUE="cpu2.gif">Key board
<OPTION VALUE="cpu3.gif">Di sks
<OPTION VALUE="cpu4.gif">Cabl es
</SELECT>
</TD>
<TD>
<SELECT NAME="cached" onChange="loadCached(this.form)">
<OPTION VALUE="desk1.gif">Bands
<OPTION VALUE="desk2.gif">Cl i ps
<OPTION VALUE="desk3.gif">Lamp
<OPTION VALUE="desk4.gif">Erasers
</SELECT></TD>
</FORM>
</TR></TABLE>
<FORM NAME="Timer">
<INPUT TYPE="checkbox" NAME="timerBox" onClick="checkTimer()">Auto-
cycle through pre-cached images
</FORM>
</CENTER>
</BODY>
</HTML>

```

You should notice that because the images being preloaded are in a related sequence, they are conveniently named to make it easy for an array and repeat

loop to manage their preloading. The same numbering scheme lets you link the images to select object options for a very compact operation.

To observe the maximum effect of the application in Listing 18-2, quit and relaunch the browser. This clears the image cache for the current session. After you open Listing 18-2, wait until all status bar activity finishes, then choose images from the precached list on the right (Figure 18-1). Unless your browser has its memory cache turned off completely, you should see an instantaneous response to selecting an image.

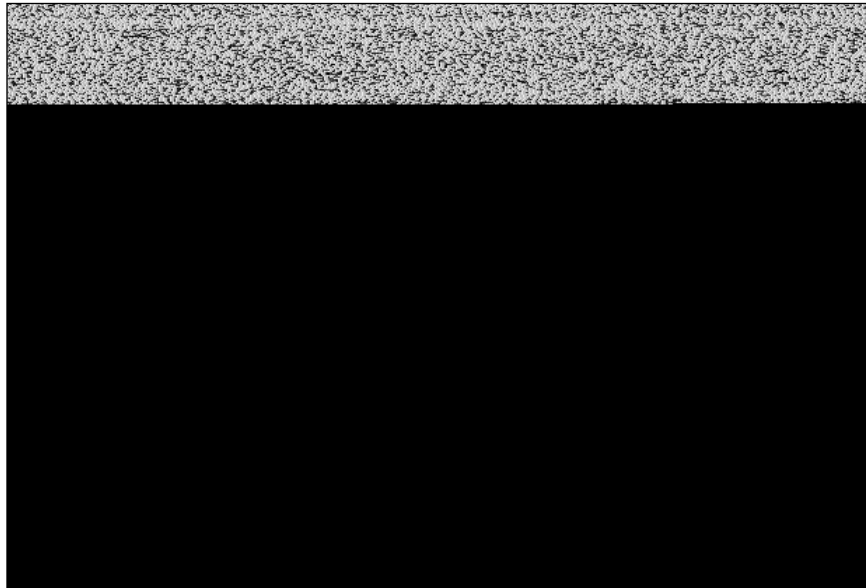


Figure 18-1: The image object demonstration page (*Images © Aris Multimedia Entertainment, Inc. 1994*)

Next, choose an image from the left-hand listing. You should see the image loading individually. The same happens for each new image you load. The images you previously loaded this way are still in the memory cache, so they appear instantaneously thereafter (until you quit again). These images are small images, of course, so the impact on the wait time is not substantial. Imagine the perceived performance improvement for users accessing larger images through the Net.

Finally, check the checkbox to watch the images in the right-hand side of the table cycle through their grouping, changing images every five seconds. This mechanism, too, takes advantage of the array numbering for the images.

Related Items: `image.lowsrc` property.

x
y

Value: Integer Gettable: Yes Settable: No

	Nav2	Nav3	Nav4	IE3/J1	IE3/J2	IE4/J3
Compatibility			✓			

Your Navigator 4 script can retrieve the x and y coordinates of an image object (the top-left corner of the rectangular space occupied by the image) via the `image.x` and `image.y` properties. These properties are read-only, but you can use Dynamic HTML to change the location of a link if you like (see Chapters 41 to 43).

Even without Dynamic HTML, you can use the information from these properties to help scroll a document to a precise position (with the `window.scrollTo()` method) as a navigational aid in your page. Due to the different ways each operating system platform renders pages and the different sizes of browser windows, you can dynamically locate the position of a link given the current client conditions.

Example

Due to the different ways each operating system platform renders pages and the different sizes of browser windows, you can dynamically locate the position of a link given the current client conditions. For example, if you want to scroll the document so that the link is a few pixels below the top of the window, you could use a statement such as this:

```
window.scrollTo(document.images[0].x, (document.images[0].y - 3))
```

Related Items: `window.scrollTo()` method.

Event handlers

`onAbort=`

`onError=`

	Nav2	Nav3	Nav4	IE3/J1	IE3/J2	IE4/J3
Compatibility		✓	✓			✓

Your scripts may need to be proactive when a user clicks on the Stop button while an image is loading or when a network or server problem causes the image transfer to fail. Use the `onAbort=` event handler to activate a function in the event of a user clicking on the Stop button; use the `onError=` event handler for the unexpected transfer snafu.

In practice, these event handlers don't supply all the information you may like to have in a script, such as the filename of the image that was loading at the time. If such information is critical to your scripts, then the scripts need to store the name of a currently loading image to a variable before they set the image's `src` property. You also won't know the nature of the error that triggers an error event.

(*imageObject*) onAbort=

You can treat such problems by forcing a scripted page to reload or by navigating to an entirely different spot in your Web site.

Example

Listing 18-3 includes an `onAbort` = event handler. If the images already exist in the cache, you must quit and relaunch the browser to try to stop the image from loading. In that example, I provide a reload option for the entire page. How you handle the exception depends a great deal on your page design. Do your best to smooth over any difficulties that users may encounter.

onLoad=

	Nav2	Nav3	Nav4	IE3/J1	IE3/J2	IE4/J3
Compatibility		✓	✓			✓

JavaScript sends a load event to an image object when one of three possible conditions occurs: an image's `LOWSRC` image has finished loading; in the absence of a `LOWSRC` image specification, the `SRC` image has finished loading; when each frame of an animated GIF (GIF89a format) appears.

It's important to understand that if you define a `LOWSRC` file inside an `` tag, the image object receives no further word about the `SRC` image having completed its loading. If this information is critical to your script, verify the current image file by checking the `src` property of the image object.

Note

This event handler appears to be broken in Navigator 4.

Example

Quit and restart your browser to get the most from Listing 18-3. As the document first loads, the `LOWSRC` image file (the picture of pencil erasers) loads ahead of the computer keyboard image. When the erasers are loaded, the `onLoad` = event handler writes "done" to the text field, even though the main image has not yet loaded. You can experiment further by loading the arch image. This image takes longer to load, so the `LOWSRC` image (set on the fly, in this case) loads way ahead of it.

Listing 18-3: The Image `onLoad` = Event Handler

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript 1.1">
function loadIt(theImage, form) {
    form.result.value = ""
    document.images[0].lowsrc = "desk1.gif"
    document.images[0].src = theImage
}
function checkLoad(form) {
    form.result.value = document.images[0].complete
}

```

(continued)

(imageObject) `onLoad` =

Listing 18-3 (continued)

```

function signal () {
    if(confirm("You have stopped the image from loading. Do you want
to try again?")) {
        location.reload()
    }
}
</SCRIPT>
</HEAD>
<BODY>
<IMG SRC="cpu2.gif" LOWSRC="desk4.gif" WIDTH=120 HEIGHT=90
onLoad="document.forms[0].result.value='done'" onAbort="signal()">
<FORM>
<INPUT TYPE="button" VALUE="Load keyboard"
onClick="loadIt('cpu2.gif',this.form)">
<INPUT TYPE="button" VALUE="Load arch"
onClick="loadIt('arch.gif',this.form)"><P>
<INPUT TYPE="button" VALUE="Is it loaded yet?"
onClick="checkLoad(this.form)">
<INPUT TYPE="text" NAME="result">
</FORM>
</BODY>
</HTML>

```

Related Items: `image.src` property; `image.lowsrc` property.

Area Object

<i>Properties</i>	<i>Methods</i>	<i>Event Handlers</i>
target	(None)	onClick=
[location object properties]		onMouseOut=
		onMouseOver=

Syntax

Creating an image map area:

```

<MAP NAME="areaMapName">
    <AREA
        COORDS="x1, y1, x2, y2, . . ." | "x-center, y-center, radius"
        HREF="locati onOrURL"
        [NOHREF]
        [SHAPE="rect" | "poly" | "circle" | "default"]
    >

```

```

[ TARGET=" windowName" ]
[ onMouseOver=" handl erText OrFunct i on" ]
[ onMouseOut =" handl erText OrFunct i on" ] >
</MAP>

```

Accessing area object properties:

```
[ window. ] document . l i n k s [ i n d e x ] . p r o p e r t y
```

About this object

JavaScript treats a map area object as one of the link objects in a document (see the link object in Chapter 17). When you think about it, such treatment is not illogical at all, because clicking on a map area generally leads the user to another document or anchor location in the same document — a hyperlinked reference.

Although the HTML definitions of links and map areas are quite different, both kinds of objects have nearly the same properties and event handlers. Therefore, to read about the details for these items, refer to the discussion about the link object in Chapter 17. The one difference is that a map area object does not have the same full array of mouse event handlers.

Client-side image maps are fun to work with, and they have been well-documented in HTML references since the feature was introduced in Netscape Navigator 2. Essentially, you define any number of areas within the image, based on shape and coordinates. Many graphics tools can help you capture the coordinates of images that you need to enter into the COORDS= attribute of the <AREA> tag.

Caution

If one gotcha exists that trips up most HTML authors, it's the link between the and <MAP> tags. The reason is that the link is a little tricky. You must assign a name to the <MAP>; in the tag, the USEMAP= attribute requires a hash symbol (#) and the map name. If you forget the hash symbol, you won't create a connection between image and map.

Note

The onCl i ck= event handler appears in Netscape's area object beginning with Navigator 4. To be backward compatible with Navigator 3, use a j a v a s c r i p t : U R L for the HREF attribute if you want a click on the region to navigate to another page.

Example

To demonstrate how to script some area objects atop an image, Listing 18-4 defines six area objects for a space shuttle's view of the Sinai area. Most of the areas are rectangular, except for a path along a snakey Nile River that runs near the bottom of the image. Run the pointer along this path to see how the complex polygon tracks the twists and turns of the river. Can you find Cairo?

Listing 18-4: A Scripted Image Map

```

<HTML>
<HEAD>
<S C R I P T  L A N G U A G E = " J a v a S c r i p t " >
function show(msg) {
    window.status = msg
    return true
}

```

(continued)

document.areaObject

Listing 18-4 (continued)

```

    }
    function go(wher) {
        alert("We're going to " + wher + "!")
    }
    function clearIt() {
        window.status = ""
        return true
    }
</SCRIPT>
</HEAD>
<BODY>
<H1>Sinai and Vicinity</H1>
<IMG SRC="nile.gif" WIDTH=320 HEIGHT=240 USEMAP="#sinai">
<MAP NAME="sinai">
    <AREA HREF="javascript:go('Cairo')" COORDS="12, 152, 26, 161"
SHAPE="rect" onMouseOver="return show('Cairo')" onMouseOut="return
clearIt()">
    <AREA HREF="javascript:go('the Nile River')"
COORDS="1, 155, 6, 162, 0, 175, 3, 201, 61, 232, 109, 227, 167, 238, 274, 239, 292, 220,
307, 220, 319, 230, 319, 217, 298, 213, 282, 217, 267, 233, 198, 228, 154, 227, 107, 221
, 71, 225, 21, 199, 19, 165, 0, 149" SHAPE="poly" onMouseOver="return
show('Nile River')" onMouseOut="return clearIt()">
    <AREA HREF="javascript:go('Israel')" COORDS="95, 69, 201, 91"
SHAPE="rect" onMouseOver="return show('Israel')" onMouseOut="return
clearIt()">
    <AREA HREF="javascript:go('Saudi Arabia')"
COORDS="256, 57, 319, 121" SHAPE="rect" onMouseOver="return show('Saudi
Arabia')" onMouseOut="return clearIt()">
    <AREA HREF="javascript:go('the Mediterranean Sea')"
COORDS="1, 55, 26, 123" SHAPE="rect" onMouseOver="return
show('Mediterranean Sea')" onMouseOut="return clearIt()">
    <AREA HREF="javascript:go('the Mediterranean Sea')"
COORDS="27, 56, 104, 103" SHAPE="rect" onMouseOver="return
show('Mediterranean Sea')" onMouseOut="return clearIt()">
</MAP>
</BODY>
</HTML>

```

Although most maps have HREF attributes that point to URLs on the Web, I use an internal `javascript: URL` to demonstrate how the `window.status` messages written by `onMouseOver` event handlers override the HREF display in the status bar. For the Mediterranean Sea, I use two adjacent rectangles with identical behaviors to show another way to handle irregular shapes.

