Link and Anchor Objects

CHAPTER 17

✦ ✦ ✦ ✦

In This Chapter

Scripting a link to invoke a script function

Scripting a link to swap an image on mouse rollovers

Triggering actions from double-clicking a link

✦ ✦ ✦ ✦

The Web was based on the notion that the world's information could be strung together by way of the *hyperlink* — the clickable hunk of text or image that navigates an inquisitive reader to a further explanation or related material. Of all the document objects you work with in JavaScript, the link is the one that makes that connection. Anchors also provide guideposts to specific locations within documents.

As scriptable objects, links and anchors are comparatively simple devices. But this simplicity belies their significance in the entire scheme of the Web. And under script control, links can be far more powerful than mere tethers to locations on the Web.

Link Object

Properties	*Methods*	*Event Handlers*
`target`	(None)	`onClick=`
`text`		`onDblClick=`
`x`		`onMouseDown=`
`y`		`onMouseOut=`
[location object properties]		`onMouseOver=`
		`onMouseUp=`

Syntax

Creating a link object:

```
<A HREF="locationOrURL"
       [NAME="anchorName"]
       [TARGET="windowName"]
       [onClick="handlerTextOrFunction"]
       [onDblClick="handlerTextOrFunction"]
       [onMouseDown="handlerTextOrFunction"]
```

```
        [onMouseOut="handlerTextOrFunction"]
        [onMouseOver="handlerTextOrFunction"]
        [onMouseUp="handlerTextOrFunction"]>
        linkDisplayTextOrImage
</A>
```

Accessing link object properties:

```
[window.] document.links[index].property
```

About this object

JavaScript treats an HTML document link as a distinct object type. When a document loads, the browser creates and maintains an internal list (in an array) of all links defined in the document.

When working with a link object in your scripts, the JavaScript object world begins to wrap around itself in a way. Despite all the attributes that define a link, JavaScript regards a link as the same as a location object (Chapter 15). In other words, if you need to refer to a link, you can access the same properties of that link the same way you can for any location object (such as `href`, `host`, `hash`, `pathname`, and so on). This convenience lets your scripts treat all URL-style data the same way.

Defining a link for JavaScript is the same as defining one for straight HTML — with the addition of six possible mouse-related event handlers. In a multiframe or multiwindow environment, it's important to specify the `TARGET=` attribute with the name of the frame or window in which the content at the URL is to appear. If you don't specify another frame, the browser replaces the frame that contains the link with the new page. Speaking of the `TARGET` attribute, don't forget the shortcut window references: `_top`, `_parent`, `_self`, and `_blank`.

As you design your links, consider building `onMouseOver=` and `onMouseOut=` event handlers into your link definitions. The most common application for these event handlers is as a means of adjusting the `window.status` property or swapping images. Thus, as a user rolls the mouse pointer atop a link, a descriptive label (perhaps more detailed or friendly than what the link text or image may indicate) appears in the status line at the bottom of the window. Whether a user notices the change down there is another issue, so don't rely on the status line as a medium for mission-critical communication. Image swaps, however, are more dramatic, letting a user receive visual feedback that the mouse pointer is atop a particular button image. In Navigator 4 and Internet Explorer 4, you can even swap the image when the user presses the mouse button atop the link.

For those times when you want a click of the link (whether the link consists of text or an image) to initiate an action without actually navigating to another URL, you can use a special technique to direct the URL to a JavaScript function. The URL `javascript:functionName()` is a valid location parameter for the `HREF` attribute (and not just in the link object). Using the `javascript:` URL to call a function is also one way you can modify the content of more than one frame at a time. In the function, set the location object of each frame to the desired document.

If you don't want the link to do anything other than change the statusbar in the `onMouseOver=` event handler, define an empty function and set the URL to that empty JavaScript function (such as `HREF="javascript:doNothing()"`). Starting with Navigator 3, you can also add a special `void` operator that guarantees that the function being called does not trigger any true linking action (`HREF="javascript: void someFunction()"`). Specifying an empty string for the `HREF` attribute yields an ftp-like file listing for the client computer — an undesirable artifact. Don't forget, too, that if the URL leads to a type of file that initiates a browser helper application (for example, to play a RealAudio sound file), then the helper app or plug-in loads and plays without changing the page in the browser window.

One additional technique allows a single link tag to operate for both scriptable and nonscriptable browsers (Navigator 3 and up; Internet Explorer 4). For nonscriptable browsers, establish a genuine URL to navigate from the link. Then make sure that the link's `onClick=` event handler evaluates to return false. At click time, a scriptable browser executes the event handler and ignores the `HREF` attribute; a nonscriptable browser ignores the event handler and follows the link. More about this in the event handler discussions for this object later in this chapter.

If you don't specify an `HREF` attribute in a link tag, the definition becomes an anchor object rather than a link object. The optional `NAME` attribute enables the link object to behave like an anchor object, thus enabling other links to navigate directly to the link.

Properties

`target`

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

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

The primary property of the link object is the *target*. This value reflects the window name supplied to the `TARGET` attribute in the link's definition. Because link objects are stored as an array of components in a document, you can reference the `target` property of a particular link only via an indexed link reference.

You can temporarily change the target for a link. But, as with most transient object properties, the setting does not survive soft reloads. Rather than altering the target this way, a safer method is to force the target change by letting the `HREF` attribute call a `javascript:functionName()` URL, where the function assigns a document to the desired `window.location`. If you have done extensive HTML authoring before, you will find it hard to break the habit of relying on the `TARGET` attribute.

Example

```
windowName = document.links[3].target
```

Related Items: `document.links` property; anchor object.

text

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

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

Between the start and end tags for a link goes the text (or image) that is highlighted in the distinguishing link color of the document. Navigator 4 lets you extract that text with the `link.text` property. This property is read-only. Internet Explorer 4 employs a different model and syntax for getting and setting the text and HTML for an object like the link. The two syntaxes are not compatible.

Note

This property was not implemented in releases of Navigator 4 prior to Version 4.02.

Related Items: None.

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 a link object (the top-left corner of the rectangular space occupied by the linked text or image) via the `link.x` and `link.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.

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 the link is a few pixels below the top of the window, you could use a statement such as this:

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

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

Event handlers

onClick=
onDblClick=

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

By and large, the `HREF` attribute determines the action that a link makes when a user clicks it — which is generally a navigational action. But if you need to execute a script before navigating to a specific link (or to change the contents of more than one frame), you can include an `onClick=` and/or `onDblClick=` event handler in that link's definition. Any statements or functions called by either click event handler execute before any navigation takes place. The `onClick=` event handler is in all versions of all browsers; `onDblClick=` is available only in Navigator 4 (but, alas, not the Macintosh version) and Internet Explorer 4.

You can script entirely different actions for the `onClick=` and `onDblClick=` event handlers, but you must be aware of the interaction between the two events. Moreover, interaction between the single- and double-click events is different in Navigator 4 and Internet Explorer 4. In Navigator 4, each of the two clicks of a dblClick event triggers a single click event; in Internet Explorer 4, only the first of the two clicks registers a click event. In neither case is a single click event cancelled automatically or ignored when the dblClick event occurs. Therefore, if you intend to have single- and double-clicks on a link fire entirely different processes, it is up to your script to delay the action of the single-click until it knows that a double-click has occurred.

Before you get carried away with implementing different actions for single- and double-clicks on the same link, take a cue from the big GUI boys who design our desktops. Many users have difficulty accurately performing a double-click action, based on their computer's mouse control panel setting. Moreover, some users may not be aware that double-clicking a link offers anything, since generic links are all single-click beasts. Therefore, if you require a double-click for a special action, make sure that the single-click action (if any) is mild enough such that the user can see that the desired double-click action did not take place and can try again. On the desktop, double-clicking a document icon opens the file and application, but single-clicking simply selects or highlights the object on the desktop — a pair of complementary actions that works pretty well.

Both click event handlers in a link observe special behavior that enables your scripts to prevent the link from completing its navigation job in Navigator 3 or later and Internet Explorer 4. For example, you may want your scripts to perform some data-entry validation before navigating to another page. If some field isn't filled out correctly, you can alert the user about it and abort the link action at the same time. To make this validation work, the last statement of the handler or the handler itself must evaluate to the word `return` and either `true` or `false`. If `true`, then the navigation completes; if `false`, then all action ceases.

Example

In Listing 17-1, I present a click laboratory that lets you see the behavior of `onClick=` and `onDblClick=` event handlers in links. The first link has only an `onClick=` event handler; the second one has only an `onDblClick=` event handler.

Things get more interesting in the third link, which has both. You will see from the readout of events in the lab output fields that a double-click user action results in a total of three events firing: two clicks and one dblClick. If you need to segregate single- from double-click user actions, the functions called by the fourth link's event handlers will interest you. They delay the action of the first single-click slightly and then cancel the first click and ignore the second click if a double-click has occurred. All `HREF` attributes for these links are set to a `javascript:` URL that goes nowhere.

Listing 17-1: **Single- and Double-Clicks**

```
<HTML>
<HEAD>
<TITLE>Clicking on Links</TITLE>
<SCRIPT LANGUAGE="JavaScript">
var timeoutID
var isDouble = false
function fillSingle() {
       document.lab.singleOut.value = document.lab.singleOut.value +
"Single Click  "
}

function fillDouble() {
       document.lab.doubleOut.value = document.lab.doubleOut.value +
"Double Click  "
}

function smartSingle() {
       clearTimeout(timeoutID)
       if (!isDouble) {
          timeoutID = setTimeout("fillSingle()",300)
       }
       isDouble = false
}

function smartDouble() {
       isDouble = true
       fillDouble()
}</SCRIPT>
</HEAD>
<BODY>
Click <A HREF="javascript:void(0)" onClick="fillSingle()">HERE</A> to
trigger an onClick= event handler.<BR>
Click <A HREF="javascript:void(0)" onDblClick="fillDouble()">HERE</A> to
trigger an onDblClick= event handler.<P>
```

```
Click and Double-Click <A HREF="javascript:void(0)"
onClick="fillSingle()" onDblClick="fillDouble()">HERE</A> to trigger a
dumb version of both event handlers.<P>
Click and Double-Click <A HREF="javascript:void(0)"
onClick="smartSingle()" onDblClick="smartDouble()">HERE</A> to trigger
a smart version of both event handlers.<P>
<FORM NAME="lab">
<INPUT TYPE="Reset"><BR>
<HR>
Single Click Signals:<INPUT TYPE="text" NAME="singleOut" SIZE="50"><BR>
Double Click Signals:<INPUT TYPE="text" NAME="doubleOut" SIZE="50">
</FORM>
</BODY>
</HTML>
```

The second `onClick=` event handler is associated with the next link. It has a fixed `HREF` designation, but the `onClick=` handler returns the Boolean value returned by the `verifyMove()` function. If you click the confirm dialog's Cancel button, you won't navigate at all, whereas if you click the Yes button (which supplies a `true` value), you go to the link as if the `onClick=` event handler did not exist.

onMouseDown=
onMouseUp=

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

Additional mouse-related events in Navigator 4 and Internet Explorer 4 are available for links. Events for the mouse being pressed and released supplement the long-standing click event. A click event fires after a matched set of mouseDown and mouseUp events occur on the same link. If the user presses down on the link and slides the mouse pointer off the link to release the mouse button, only the mouseDown event fires on the link.

These events allow authors and designers to add more application-like behavior to images that act as action or icon buttons. If you notice the way most buttons work, the appearance of the button changes while the mouse button is down and reverts to its original style when the mouse button is released (or the cursor is dragged out of the button). These events let you emulate that behavior.

Example

To demonstrate a likely scenario of changing button images in response to rolling atop an image, pressing down on it, releasing the mouse button, and rolling away from the image, Listing 17-2 presents a pair of small navigation buttons (left- and right-arrow buttons). Because the image object is not part of the document object model for Navigator 2 or Internet Explorer 3 (which reports itself as being at

Navigator Version 2), the page is designed to be friendly to all browsers. The duplicate `setImage()` function is required to accommodate the older browsers. For a browser with an image object, images are preloaded into the browser cache as the page loads so that response to the user is instantaneous the first time the new versions of the images are called upon.

Listing 17-2: Image Swapping on Mouse Events

```
<HTML>
<HEAD>
<TITLE>Image Swapping with Links</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function setImage() {}
</SCRIPT>

<SCRIPT LANGUAGE="JavaScript1.1">
if(parseInt(navigator.appVersion) > 2) {
       var RightNormImg = new Image(16,16)
       var RightUpImg = new Image(16,16)
       var RightDownImg = new Image(16,16)
       var LeftNormImg = new Image(16,16)
       var LeftUpImg = new Image(16,16)
       var LeftDownImg = new Image(16,16)

       RightNormImg.src = "RightNorm.gif"
       RightUpImg.src = "RightUp.gif"
       RightDownImg.src = "RightDown.gif"
       LeftNormImg.src = "LeftNorm.gif"
       LeftUpImg.src = "LeftUp.gif"
       LeftDownImg.src = "LeftDown.gif"
}
function setImage(imgName, type) {
       var imgFile = eval(imgName + type + "Img.src")
       document.images[imgName].src = imgFile
       return false
}
</SCRIPT>
</HEAD>
<BODY>
<B>Roll atop and click on the buttons to see how the link event
handlers swap images:</B><P>
<CENTER>
<A HREF="javascript:void(0)"
       onMouseOver="return setImage('Left','Up')"
       onMouseDown="return setImage('Left','Down')"
       onMouseUp="return setImage('Left','Up')"
       onMouseOut="return setImage('Left','Norm')"
>
<IMG NAME="Left" SRC="LeftNorm.gif" HEIGHT=16 WIDTH=16 BORDER=0></A>
&nbsp;&nbsp;
<A HREF="javascript:void(0)"
       onMouseOver="return setImage('Right','Up')"
```

```
        onMouseDown="return setImage('Right','Down')"
        onMouseUp="return setImage('Right','Up')"
        onMouseOut="return setImage('Right','Norm')"
>
<IMG NAME="Right" SRC="RightNorm.gif" HEIGHT=16 WIDTH=16 BORDER=0></A>
</CENTER>
</BODY>
</HTML>
```

Related Items: onMouseOver= event handler; onMouseOut= event handler; image object.

onMouseOver=
onMouseOut=

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

You've seen it a million times: As you drag the mouse pointer atop a link in a document, the status line at the bottom of the window shows the URL defined in the link's HREF attribute. For Net Freaks, URLs are like mother's milk, but for everyday folks, long URLs are more like gibberish. You can override the display of a link's URL by triggering a function with the onMouseOver= event handler assigned to a link.

One potentially tricky aspect of this event handler is that no matter what you ask the handler to do, whether it be a statement within the onMouseOver= attribute or a call to a JavaScript function, the attribute must end with a return true statement. Without this last statement in the onMouseOver= attribute, the status bar will not change.

The converse of the onMouseOver= event handler is onMouseOut=. This handler (not available in Navigator 2 or Internet Explorer 3) fires when the pointer leaves the link's rectangular region. If you use onMouseOver= to set the statusbar, you should return the statusbar to its default setting with an onMouseOut= event handler (as detailed in Chapter 14's discussion about the window.status and window.defaultStatus properties).

No conventions exist for the kind of text you put in the status line, but the text should help the user better understand the result of clicking a link. I like to put instructions or a command-like sentence in the status line. For example, for the iconic images of the outline-style table of contents (see the bonus application in Chapter 50 on the CD-ROM), the message advises the user to click to expand or collapse the nested items, depending on which icon is in place; for the text links in the outline, the message gives information about what the user will see upon clicking the link. For experienced Web surfers, displaying the URL in parentheses after the plain language message may be helpful. Web geeks remember and recognize URLs.

Note

The onMouseOut= event handler sometimes fails to execute in Navigator 3. This happens most often if the link is near a frame border, and the user quickly moves the mouse from the link to outside the frame.

Example

Listing 17-3 uses the Pledge of Allegiance with four links to demonstrate how to use the onMouseOver= and onMouseOut= event handlers. Notice that for each link, the handler runs a general-purpose function that sets the window's status message. The function returns a true value, which the event handler call evaluates to replicate the required return true statement needed for setting the statusbar. In one status message, I supply a URL in parentheses to let you evaluate how helpful you think it may be for users.

Listing 17-3: **Using onMouseOver= and onMouseOut=**

```
<HTML>
<HEAD>
<TITLE>Mousing Over Links</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function setStatus(msg) {
        status = msg
        return true
}
// destination of all link HREFs
function emulate() {
        alert("Not going there in this demo.")
}
</SCRIPT>
</HEAD>
<BODY>
<H1>Pledge of Allegiance</H1>
<HR>
I pledge <A HREF="javascript:emulate()" onMouseOver="return
setStatus('View dictionary definition')" onMouseOut="return
setStatus('')">allegiance</A> to the <A HREF="javascript:emulate()"
onMouseOver="return setStatus('Learn about the U.S. flag
(http://lcweb.loc.gov)')" onMouseOut="return setStatus('')">flag</A> of
the <A HREF="javascript:emulate()" onMouseOver="return setStatus('View
info about the U.S. government')" onMouseOut="return
setStatus('')">United States of America</A>, and to the Republic for
which it stands, one nation <A HREF="javascript:emulate()"
onMouseOver="return setStatus('Read about the history of this phrase in
the Pledge')" onMouseOut="return setStatus('')">under God</A>,
indivisible, with liberty and justice for all.
</BODY>
</HTML>
```

The other technique demonstrated is how to use an internal location (`javascript: emulate()`) as an `HREF` attribute value for each of the links. This technique enables the link to behave like a real link (highlighting the text), but in this case, the link doesn't navigate anywhere. Instead, it invokes the `emulate()` function, defined in the document.

Anchor Object

Properties	***Methods***	***Event Handlers***
`name`	(None)	(None)
`text`		
`x`		
`y`		

Syntax

Creating an anchor object:

```
<A NAME="anchorName">
       anchorDisplayTextOrImage
</A>
```

About this object

As an HTML document loads into a JavaScript-enabled browser, the browser creates and maintains an internal list (as an array) of all the anchors defined in the document. Like link objects, you reference anchor objects according to their indexed value within the `document.anchors[index]` property. New properties in Navigator 4 (one of which is also in Internet Explorer 4) enable your scripts to examine anchors for their names and physical locations in the document.

The preceding syntax shows the simplest way of defining an anchor. You can also turn a link object into an anchor by simply adding a `NAME=` attribute to the link's definition.

Properties

`name`

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

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

The `name` property of an anchor object is the string assigned to the `NAME` attribute of the anchor or link tag. This is a read-only property.

Related Items: None.

`text`

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

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

Between the start and end tags for an anchor goes the text (or image) that is associated with the position in the document. Navigator 4 lets you extract that text with the `anchor.text` property. This property is read-only.

Internet Explorer 4 employs a different model and syntax for getting and setting the text and HTML for an object like the anchor. The two syntaxes are not compatible.

Note

This property was not implemented in releases of Navigator 4 prior to Version 4.02.

Related Items: None.

`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 anchor object (the top-left corner of the rectangular space occupied by the linked text immediately following the opening tag) via the `anchor.x` and `anchor.y` properties. These properties are read-only and can be used for the same purposes as the parallel properties of the link object.

Related Items: `link.x` property; `link.y` property.

✦ ✦ ✦