CHAPTER 15

Location and History Objects

In This Chapter

Loading new pages and other media types via the location object

Security restrictions across frames

Navigating through the browser history under script control

Not all objects in the document object model are "things" you can see in the content area of the browser window. The browser maintains a bunch of other information about the page you are currently visiting and where you have been. The URL of the page you see in the browser is called the *location*, and browsers store this information in the location object. And as you surf the Web, the browser stores the URLs of your past pages in an object called the *history* object. You can manually view what that object contains by looking in the browser menu that lets you jump back to a previously visited page. This chapter is all about these two nearly invisible, but important objects.

Not only are these objects valuable to your browser, but they can also be valuable to snoopers who might want to write scripts to see what URLs you're viewing in another frame or the URLs of other sites you've visited in the last dozen mouse clicks. As a result, there exist security restrictions that limit access to some of these objects' properties unless you use signed scripts in Navigator 4. For older browsers, these properties are simply not available from a script.

Location Object

Properties	*Methods*	*Event Handlers*
`hash`	`assign()`	(None)
`host`	`reload()`	
`hostname`	`replace()`	
`href`		
`pathname`		
`port`		
`protocol`		
`search`		

Syntax

Loading a new document into the current window:

```
[window.]location = "URL"
```

Accessing location properties or methods:

```
[window.]location.property | method([parameters])
```

About this object

In its place one level below window-style objects in the JavaScript object hierarchy, the location object represents information about the URL of any currently open window or of a specific frame. A multiple-frame window displays the parent window's URL in the Location field. Each frame also has a location associated with it, although no overt reference to the frame's URL can be seen in the browser. To get URL information about a document located in another frame, the reference to the location object must include the window frame reference. For example, if you have a window consisting of two frames, Table 15-1 shows the possible references to the location objects for all frames comprising the Web presentation.

Table 15-1
Location Object References in a Two-Frame Browser Window

Reference	*Description*
`location` (or `window.location`)	URL of frame displaying the document that runs the script statement containing this reference
`parent.location`	URL info for parent window that defined the `<FRAMESET>`
`parent.frames[0].location`	URL info for first visible frame
`parent.frames[1].location`	URL info for second visible frame
`parent.otherFrameName.location`	URL info for another named frame in the same frameset

Most properties of a location object deal with network-oriented information. This information includes various data about the physical location of the document on the network, including the host server, the protocol being used, and other components of the URL. Given a complete URL for a typical WWW page, the `window.location` object assigns property names to various segments of the URL, as shown here:

```
http://www.giantco.com:80/promos/newproducts.html#giantGizmo
```

Property	Value
`protocol`	`"http:"`
`hostname`	`"www.giantco.com"`
`port`	`"80"`
`host`	`"www.giantco.com:80"`
`pathname`	`"/promos/newproducts.html"`
`hash`	`"#giantGizmo"`
`href`	`"http://www.giantco.com:80/promos newproducts.html#giantGizmo"`

The `window.location` object can be handy when a script needs to extract information about the URL, perhaps to obtain a base reference on which to build URLs for other documents to be fetched as the result of user action. This object can eliminate a nuisance for Web authors who develop sites on one machine and then upload them to a server (perhaps at an Internet Service Provider) with an entirely different directory structure. By building scripts to construct base references from the directory location of the current document, you can construct the complete URLs for loading documents. You won't have to manually change the base reference data in your documents as you shift the files from computer to computer or from directory to directory. To extract the segment of the URL and place it into the enclosing directory, you can use the following:

```
var baseRef = location.href.substring(0,location.href.lastIndexOf
("/") + 1)
```

Note

Security alert: To allay fears of Internet security breaches and privacy invasions, scriptable browsers prevent your script in one frame from retrieving location object properties from other frames whose domain and server are not your own (unless you are using signed scripts in Navigator 4 — see Chapter 40). This restriction puts a damper on many scripters' well-meaning designs and aids for Web watchers and visitors. If you attempt such property accesses, however, you will receive an "access disallowed" security warning dialog box.

Setting the value of some location properties is the preferred way to control which document gets loaded into a window or frame. Though you may expect to find a method somewhere in JavaScript that contains a plain language "Go" or "Open" word (to simulate what you see in the browser menu bar), the way to "point your browser" to another URL is to set the `window.location` object to that URL, as in

```
window.location.href = "http://www.dannyg.com/"
```

The equals assignment operator (`=`) in this kind of statement becomes a powerful weapon. In fact, setting the location object to a URL of a different MIME type, such as one of the variety of sound and video formats, causes the browser to load those files into the plug-in or helper application designated in your browser's settings. Internet

Explorer's object model includes a `window.navigate()` method that also loads a document into a window, but this method is not part of Netscape — at least through Navigator 4.

Two methods complement the location object's capability to control navigation: One method is the script equivalent of clicking Reload; the other method enables you to replace the current document's entry in the history with that of the next URL of your script's choice.

Properties

hash

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

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

The hash mark (#) is a URL convention that directs the browser to an anchor located in the document. Any name you assign to an anchor (with the `<A NAME="..."> ...</A>` tag pair) becomes part of the URL after the hash mark. A location object's `hash` property is the name of the anchor part of the current URL (which consists of the hash mark and the name).

If you have written HTML documents with anchors and directed links to navigate to those anchors, you have probably noticed that although the destination location shows the anchor as part of the URL (for example, in the Location field), the window's anchor value does not change as the user manually scrolls to positions in the document where other anchors are defined. An anchor appears in the URL only when the window has navigated there as part of a link or in response to a script that adjusts the URL.

Just as you can navigate to any URL by setting the `window.location` property, you can navigate to another hash in the same document by adjusting only the `hash` property of the location, but without the hash mark (as shown in the following example). Such navigation, even within a document, causes Navigator 2 and Internet Explorer 3 to reload the document (and scripted navigation to anchors is incredibly slow in Internet Explorer 3/Windows). No reload occurs in Navigator 3 and up.

Example

When you load the script in Listing 15-1, adjust the size of the browser window so only one section is visible at a time. When you click a button, its script navigates to the next logical section in the progression and eventually takes you back to the top.

Listing 15-1: **A Document with Anchors**

```
<HTML>
<HEAD>
<TITLE>location.hash Property</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function goNextAnchor(where) {
        window.location.hash = where
}
</SCRIPT>
</HEAD>

<BODY>

<A NAME="start"><H1>Top</H1></A>
<FORM>
<INPUT TYPE="button" NAME="next" VALUE="NEXT"
onClick="goNextAnchor('sec1')">
</FORM>
<HR>
<A NAME="sec1"><H1>Section 1</H1></A>
<FORM>
<INPUT TYPE="button" NAME="next" VALUE="NEXT"
onClick="goNextAnchor('sec2')">
</FORM>
<HR>
<A NAME="sec2"><H1>Section 2</H1></A>
<FORM>
<INPUT TYPE="button" NAME="next" VALUE="NEXT"
onClick="goNextAnchor('sec3')">
</FORM>
<HR>

<A NAME="sec3"><H1>Section 3</H1></A>
<FORM>
<INPUT TYPE="button" NAME="next" VALUE="BACK TO TOP"
onClick="goNextAnchor('start')">
</FORM>

</BODY>
</HTML>
```

Anchor names are passed as parameters with each button's `onClick=` event handler. Instead of going through the work of assembling a `window.location` value in the function by appending a literal hash mark and the value for the anchor, here I simply modify the `hash` property of the current window's location. This is the preferred, cleaner method.

If you attempt to read back the `window.location.hash` property in an added line of script, however, the window's actual URL will probably not have been updated yet, and the browser will appear to be giving your script false information.

To prevent this problem in subsequent statements of the same function, construct the URLs of those statements from the same variable values you used to set the `window.location.hash` property — don't rely on the browser to give you the values you expect.

Related Items: `location.href` property.

host

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

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

The `location.host` property describes both the hostname and port of a URL. The port is included in the value only when the port is an explicit part of the URL. If you navigate to a URL that does not display the port number in the Location field of the browser, the `location.host` property returns the same value as the `location.hostname` property.

Use the `location.host` property to extract the `hostname:port` part of the URL of any document loaded in the browser. This capability may be helpful for building a URL to a specific document that you want your script to access on the fly.

Example

Use the documents in Listings 15-2 through 15-4 as tools to help you learn the values that the various `window.location` properties return. In the browser, open the file for Listing 15-2. This file creates a two-frame window. The left frame contains a temporary placeholder (Listing 15-4) that displays some instructions. The right frame has a document (Listing 15-3) that lets you load URLs into the left frame and get readings on three different windows available: the parent window (which creates the multiframe window), the left frame, and the right frame.

Listing 15-2: **Frameset for the Property Picker**

```
<HTML>
<HEAD>
<TITLE>window.location Properties</TITLE>
</HEAD>
<FRAMESET COLS="50%,50%" BORDER=1 BORDERCOLOR="black">
       <FRAME NAME="Frame1" SRC="lst15-04.htm">
       <FRAME NAME="Frame2" SRC="lst15-03.htm">
</FRAMESET>
</HTML>
```

Listing 15-3: **Property Picker**

```
<HTML>
<HEAD>
<TITLE>Property Picker</TITLE>
<SCRIPT LANGUAGE="JavaScript">
var isNav4 = (navigator.appName == "Netscape" &&
navigator.appVersion.charAt(0) == 4) ? true : false

function fillLeftFrame() {
        newURL = prompt("Enter the URL of a document to show in the
left frame:","")
        if (newURL != null && newURL != "") {
        parent.frames[0].location = newURL
        }
}

function showLocationData(form) {
        for (var i = 0; i <3; i++) {
           if (form.whichFrame[i].checked) {
              var windName = form.whichFrame[i].value
              break
           }
        }
        var theWind = "" + windName + ".location"
        if (isNav4) {

netscape.security.PrivilegeManager.enablePrivilege("UniversalBrowserRea
d")
        }
        var theObj = eval(theWind)
        form.windName.value = windName
        form.windHash.value = theObj.hash
        form.windHost.value = theObj.host
        form.windHostname.value = theObj.hostname
        form.windHref.value = theObj.href
        form.windPath.value = theObj.pathname
        form.windPort.value = theObj.port
        form.windProtocol.value = theObj.protocol
        form.windSearch.value = theObj.search
        if (isNav4) {

netscape.security.PrivilegeManager.disablePrivilege("UniversalBrowserRe
ad")
        }
}
</SCRIPT>
</HEAD>
<BODY>
Click the "Open URL" button to enter the location of an HTML document
to display in the left frame of this window.
<FORM>
```

(continued)

Listing 15-3 *(continued)*

```
<INPUT TYPE="button" NAME="opener" VALUE="Open URL..."
onClick="fillLeftFrame()">
<HR>
<CENTER>
Select a window/frame. Then click the "Show Location Properties" button
to view each window.location property value for the desired window.<P>
<INPUT TYPE="radio" NAME="whichFrame" VALUE="parent" CHECKED>Parent
window
<INPUT TYPE="radio" NAME="whichFrame" VALUE="parent.frames[0]">Left
frame
<INPUT TYPE="radio" NAME="whichFrame" VALUE="parent.frames[1]">This
frame
<P>
<INPUT TYPE="button" NAME="getProperties" VALUE="Show Location
Properties" onClick="showLocationData(this.form)">
<INPUT TYPE="reset" VALUE="Clear"><P>
<TABLE BORDER=2>
<TR><TD ALIGN=right>Window:</TD><TD><INPUT TYPE="text" NAME="windName"
SIZE=30></TD></TR>
<TR><TD ALIGN=right>hash:</TD>
<TD><INPUT TYPE="text" NAME="windHash" SIZE=30></TD></TR>

<TR><TD ALIGN=right>host:</TD>
<TD><INPUT TYPE="text" NAME="windHost" SIZE=30></TD></TR>

<TR><TD ALIGN=right>hostname:</TD>
<TD><INPUT TYPE="text" NAME="windHostname" SIZE=30></TD></TR>

<TR><TD ALIGN=right>href:</TD>
<TD><TEXTAREA NAME="windHref" ROWS=3 COLS=30 WRAP="soft">
</TEXTAREA></TD></TR>

<TR><TD ALIGN=right>pathname:</TD>
<TD><TEXTAREA NAME="windPath" ROWS=3 COLS=30 WRAP="soft">
</TEXTAREA></TD></TR>

<TR><TD ALIGN=right>port:</TD>
<TD><INPUT TYPE="text" NAME="windPort" SIZE=30></TD></TR>

<TR><TD ALIGN=right>protocol:</TD>
<TD><INPUT TYPE="text" NAME="windProtocol" SIZE=30></TD></TR>

<TR><TD ALIGN=right>search:</TD>
<TD><TEXTAREA NAME="windSearch" ROWS=3 COLS=30 WRAP="soft">
</TEXTAREA></TD></TR>
</TABLE>
</CENTER>
</FORM>
</BODY>
</HTML>
```

Listing 15-4: Placeholder Document for Listing 15-2

```
<HTML>
<HEAD>
<TITLE>Opening Placeholder</TITLE>
</HEAD>
<BODY>
Initial place holder. Experiment with other URLs for this frame (see
right).
</BODY>
</HTML>
```

Figure 15-1 shows the dual-frame browser window with the left frame loaded with a page from my Web site.

For the best results, open a URL to a Web document on the network from the same domain and server from which you load the listings (this could be your local hard disk). If possible, load a document that includes anchor points to navigate through a long document. Click the Left frame radio button, and then click the button that shows all properties. This action fills the table in the right frame with all the available location properties for the selected window. Figure 15-2 shows the complete results for a page from my Web site that is set to an anchor point.

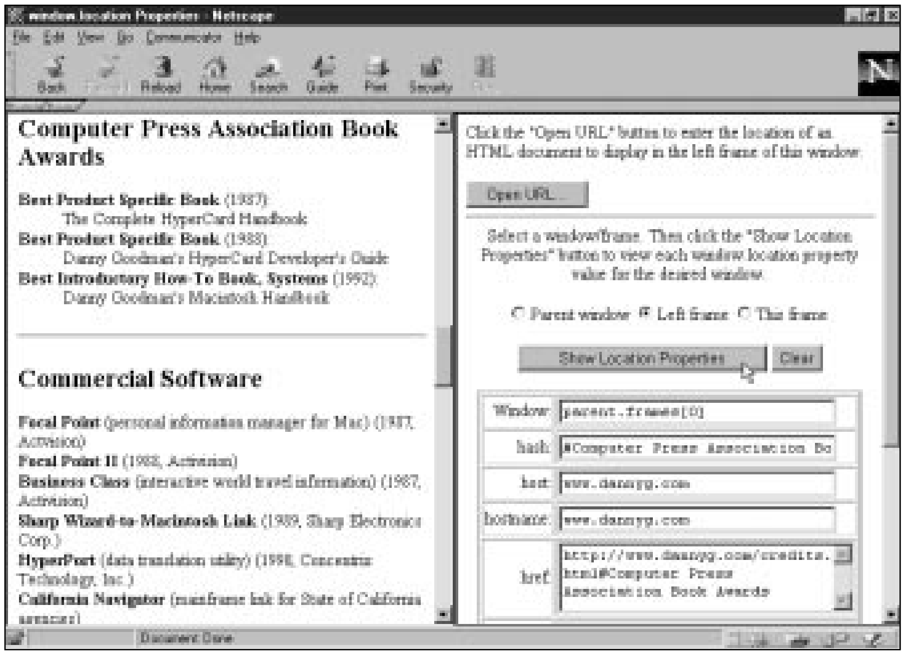
Attempts to retrieve these properties from URLs outside of your domain and server will result in a variety of responses based on your browser and browser version. Navigator 2 returns null values for all properties. Navigator 3 presents an “access disallowed” security alert. With codebase principals turned on in Navigator 4 (see Chapter 40), the proper values will appear in their fields. Internet Explorer 3 does not have the same security restrictions that Navigator does, so all values appear in their fields. In Internet Explorer 4, you get a “permission denied” error alert. See the following discussion for the meanings of the other listed properties and instructions on viewing their values.

Related Items: `location.port` property; `location.hostname` property.

hostname

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

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



window.location Properties - Netscape

Computer Press Association Book Awards

Best Product Specific Book (1987):
The Complete HyperCard Handbook
Best Product Specific Book (1988):
Danny Goodman's HyperCard Developer's Guide
Best Introductory How-To Book, Systems (1992):
Danny Goodman's Macintosh Handbook

Commercial Software

Focal Point (personal information manager for Mac) (1987, Activision)
Focal Point II (1988, Activision)
Business Class (interactive world travel information) (1987, Activision)
Sharp Wizard-to-Macintosh Link (1989, Sharp Electronics Corp.)
HyperPort (data translation utility) (1998, Concentric Technology, Inc.)
California Navigator (mainframe link for State of California agencies)

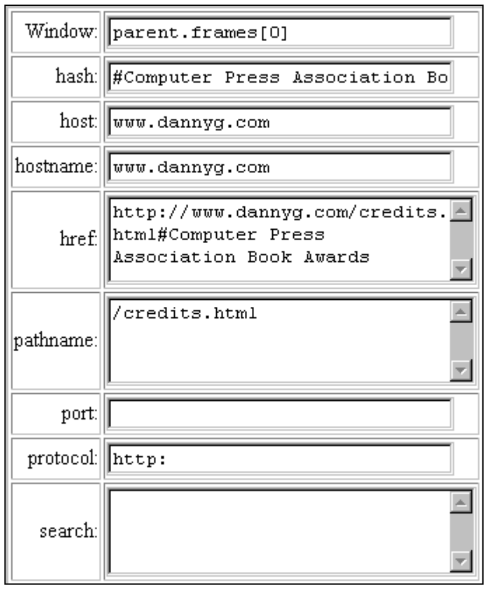
Click the "Open URL" button to enter the location of an HTML document to display in the left frame of this window.

Select a window/frame. Then click the "Show Location Properties" button to view each window location property value for the desired window.

Parent window [✓ Left frame] This frame

Window:	parent.frames[0]
hash:	#Computer Press Association Bo
host:	www.dannyg.com
hostname:	www.dannyg.com
href:	http://www.dannyg.com/credits.html#Computer Press Association Book Awards

Figure 15-1: Browser window loaded to investigate window.location properties



Window:	parent.frames[0]
hash:	#Computer Press Association Bo
host:	www.dannyg.com
hostname:	www.dannyg.com
href:	http://www.dannyg.com/credits.html#Computer Press Association Book Awards
pathname:	/credits.html
port:	
protocol:	http:
search:	

Figure 15-2: Readout of all window.location properties for the left frame

The hostname of a typical URL is the name of the server on the network that stores the document you're viewing in the browser. For most Web sites, the server name includes not only the domain name, but the `www.` prefix as well. The hostname does not, however, include the port number if such a number is specified in the URL.

Example

See Listings 15-2 through 15-4 for a set of related pages to help you view the hostname data for a variety of other pages.

Related Items: `location.host property`; `location.port` property.

href

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

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

Of all location object properties, the `href` (hypertext reference) is probably the one most often called upon in scripting. The `location.href` property supplies a string of the entire URL of the specified window object.

Using this property (or just the `window.location` object reference) on the left side of an assignment statement is the JavaScript method of opening a URL for display in a window. Any of the following statements can load my Web site's index page into a single-frame browser window:

```
window.location="http://www.dannyg.com"
window.location.href="http://www.dannyg.com"
```

At times, you may encounter difficulty by omitting a reference to a window. JavaScript may get confused and reference the `document.location` property. To prevent this confusion, the `document.location` property has been deprecated (put on the no-no list) by Netscape, and will eventually be removed from JavaScript. In the meantime, you won't go wrong by always specifying a window in the reference.

Sometimes you must extract the name of the current directory in a script so another statement can append a known document to the URL before loading it into the window. Although the other location object properties yield an assortment of a URL's segments, none of them provides the full URL to the current URL's directory. But you can use JavaScript string manipulation techniques to accomplish this task. Listing 15-5 shows such a possibility.

Depending on your browser, the values for the `location.href` property may be encoded with ASCII equivalents of nonalphanumeric characters. Such an ASCII value includes the `%` symbol and the ASCII numeric value. The most common encoded character in a URL is the space, `%20`. If you need to extract a URL and display that value as a string in your documents, the safest way is to pass all such potentially encoded strings through the JavaScript internal `unescape()` function. For example, if a URL to one of Giantco's pages is `http://www.giantco.com/product%20list`, you can convert it by passing it through the function, as in the following example:

```
plainURL = unescape(window.location.href)
       // result = "http://www.giantco.com/product list"
```

The inverse function, `escape ()`, is available for sending encoded strings to CGI programs on servers. See Chapter 27 for more details on these functions.

Note

If assigning a URL to `location.href` is causing difficulty for you in Internet Explorer 3, omit the property and assign the URL to the location object. This shorter statement works fine on all browser versions and platforms. You can also try the complete URL (including protocol).

Example

Listing 15-5 includes the `unescape()` function in front of the part of the script that captures the URL. This function serves cosmetic purposes by displaying the pathname in alert dialog boxes for browsers that normally display the ASCII-encoded version.

Listing 15-5: Extracting the Directory of the Current Document

```
<HTML>
<HEAD>
<TITLE>Extract pathname</TITLE>
<SCRIPT LANGUAGE="JavaScript">
// general purpose function to extract URL of current directory
function getDirPath(URL) {
       var result = unescape(URL.substring(0,(URL.lastIndexOf("/")) +
1))
       return result
}
// handle button event, passing work onto general purpose function
function showDirPath(URL) {
       alert(getDirPath(URL))
}
</SCRIPT>
</HEAD>

<BODY>
<FORM>
<INPUT TYPE="button" VALUE="View directory URL"
onClick="showDirPath(window.location.href)">
</FORM>
</BODY>
</HTML>
```

Related Items: `location.pathname` property; `document.location` property; string object (Chapter 27).

pathname

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

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

The pathname component of a URL consists of the directory structure relative to the server's *root* volume. In other words, the root (the server name in an `http:` connection) is not part of the pathname. If the URL's path is to a file in the root directory, then the `location.pathname` property is a single slash (/) character. Any other pathname starts with a slash character, indicating a directory nested within the root. The value of the `location.pathname` property also includes the document name.

Example

See Listings 15-2 through 15-4 for a multiple-frame example you can use to view the `location.pathname` property for a variety of URLs of your choice.

Related Items: `location.href` property.

port

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

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

These days, few consumer-friendly Web sites need to include the port number as part of their URLs. Port numbers are visible mostly in the less-popular protocols or in URLs to sites used for private development purposes or that have no assigned domain names. You can retrieve the value with the `location.port` property. If you extract the value from one URL and intend to build another URL with that component, be sure to include the colon delimiter between the server's IP address and port number.

Example

If you have access to URLs containing port numbers, use the documents in Listings 15-2 through 15-4 to experiment with the output of the `location.port` property.

Related Items: `location.host` property.

protocol

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

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

The first component of any URL is the protocol being used for the particular type of communication. For World Wide Web pages, the Hypertext Transfer Protocol (`http`) is the standard. Other common protocols you will see in your browser include File Transfer Protocol (`ftp`), File (`file`), and Mail (`mailto`). Values for the `location.protocol` property include not only the name of the protocol, but the trailing colon delimiter as well. Thus, for a typical Web page URL, the `location.protocol` property is

```
http:
```

Notice that the usual slashes after the protocol in the URL are not part of the `location.protocol` value. Of all the location object properties, only the full URL (`location.href`) reveals the slash delimiters between the protocol and other components.

Example

See Listings 15-2 through 15-4 for a multiple-frame example you can use to view the `location.protocol` property for a variety of URLs. Also try loading an ftp site to see the `location.protocol` value for that type of URL.

Related Items: `location.href` property.

search

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

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

Perhaps you've noticed the long, cryptic URL that appears in the Location field of your browser whenever you ask one of the WWW search services to look up matches for items you've entered into the keyword field. The URL starts the regular way — with protocol, host, and pathname values. But following the more traditional URL are search commands that are being submitted to the search engine (a CGI program running on the server). That trailing search query can be retrieved or set by using the `location.search` property.

Each search engine has its own formula for query submissions based on the designs of the HTML forms that obtain details from users. These search queries come in an encoded format that appears in anything but plain language. If you plan to script a search query, be sure you fully understand the search engine's format before you start assembling a string to assign to the `location.search` property of a window.

The `location.search` property also applies to any part of a URL after the filename, including parameters being sent to CGI programs on the server.

Example

The same security restrictions apply to retrieving the search property as to most of the location object properties. Because the following example accesses a domain different from yours, you need Navigator 4 with codebase principals turned on (see Chapter 40) to get the desired results. If you don't have Navigator 4 or that feature turned on, just study the code and figures to understand how the property works.

Load Listing 15-6 to view a two-frame window. The upper frame should contain a Yahoo! search engine page that lets you enter search keywords and other specifications (Figure 15-3). The bottom frame (Listing 15-7) contains two buttons.

Listing 15-6: **A Search Frameset**

```
<HTML>
<HEAD>
<TITLE>window.search Property</TITLE>
</HEAD>
<FRAMESET ROWS="50%,50%">
        <FRAME NAME="Frame1" SRC="http://www.yahoo.com/search.html">
        <FRAME NAME="Frame2" SRC="lst15-07.htm">
</FRAMESET>
</HTML>
```

Listing 15-7: **The Search Controller**

```
<HTML>
<HEAD>
<TITLE>Search Viewer/Changer</TITLE>
</HEAD>
<SCRIPT LANGUAGE="JavaScript">
var isNav4 = (navigator.appName == "Netscape" &&
navigator.appVersion.charAt(0) == 4) ? true : false
function scriptedSearch() {
        if (isNav4) {

netscape.security.PrivilegeManager.enablePrivilege("UniversalBrowserRea
d")
        }
        newSearch=prompt("Enter a new search
string:",top.frames[0].location.search)
        if (isNav4) {
netscape.security.PrivilegeManager.revertPrivilege("UniversalBrowserRea
d")
```

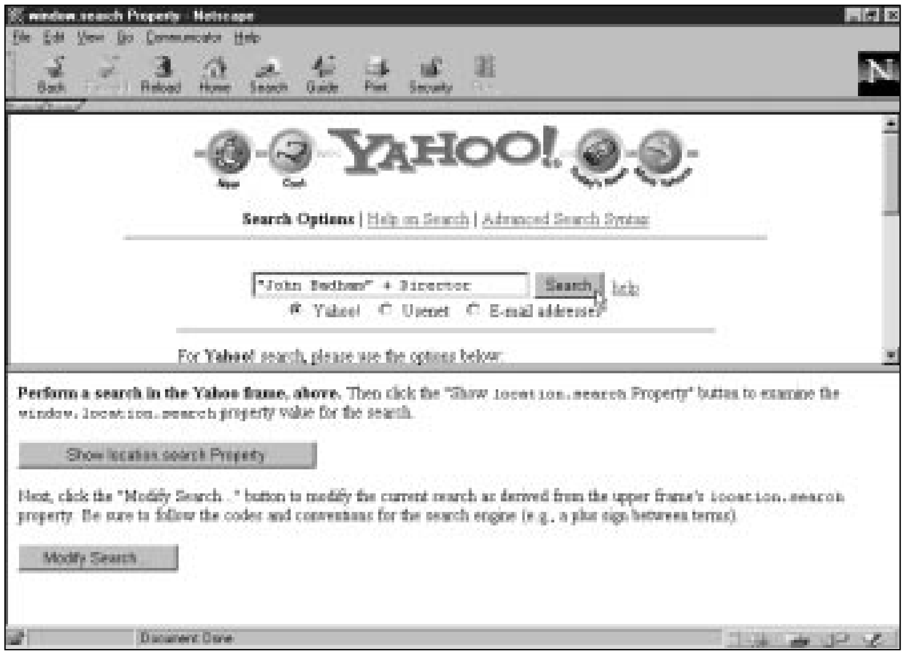
(continued)

Listing 15-7 *(continued)*

```
       }
       if (newSearch != null && newSearch != "") {
          if (isNav4) {

netscape.security.PrivilegeManager.enablePrivilege("UniversalBrowserRea
d ")
          }
          top.frames[0].location.search = newSearch
          if (isNav4) {
             netscape.security.PrivilegeManager. revertPrivilege
("UniversalBrowserRead ")
          }
       }
}
function showSearchData() {
       if (isNav4) {

netscape.security.PrivilegeManager.enablePrivilege("UniversalBrowserRea
d")
       }
       var msg = "location.href: " + top.frames[0].location.href +
"\n\n"
       msg += "location.search: " + top.frames[0].location.search
       if (isNav4) {
          netscape.security.PrivilegeManager. revertPrivilege
("UniversalBrowserRead")
       }
       alert(msg)
}
</SCRIPT>
</HEAD>
<BODY>
<FORM>
<B>Perform a search in the Yahoo frame, above.</B> Then click the
 "Show <TT>location.search</TT> Property" button to examine the
<TT>window.location.search</TT> property value for the search.<P>
<INPUT TYPE="button" NAME="getProperties" VALUE="Show location.search
Property" onClick="showSearchData()">
</FORM>
Next, click the "Modify Search..." button to modify the current
search as derived from the upper frame's <TT>location.search</TT>
property. Be sure to follow the codes and conventions for the
 search engine (e.g., a plus sign between terms).
<FORM>
<INPUT TYPE="button" NAME="opener" VALUE="Modify Search..."
onClick="scriptedSearch()">
</FORM>
</BODY>
</HTML>
```


window.search Property - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Guide Print Security Stop
YAHOO!
Search Options | Help on Search | Advanced Search Syntax
"John Badham" + Director
Search
help
Yahoo! Usenet E-mail addresses
Show location.search Property
Modify Search...
Document: Done

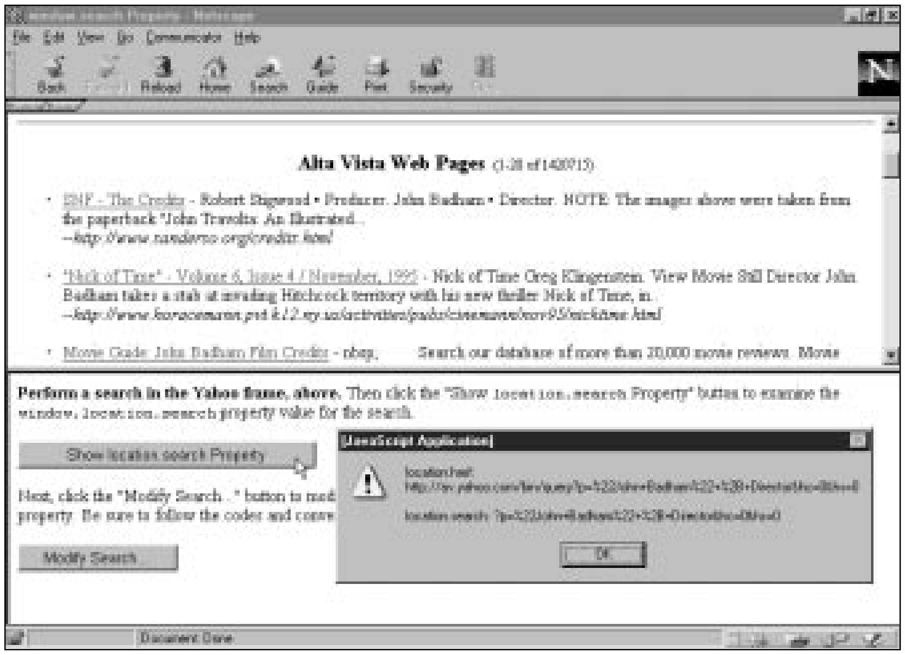
For **Yahoo!** search, please use the options below:

Perform a search in the Yahoo frame, above. Then click the "Show `location.search` Property" button to examine the `window.location.search` property value for the search.

Next, click the "Modify Search..." button to modify the current search as derived from the upper frame's `location.search` property. Be sure to follow the codes and conventions for the search engine (e.g., a plus sign between terms).

Figure 15-3: The two-frame window used to experiment with the location.search property. Yahoo!'s search page is in the upper frame.

After you perform a search in the upper frame, click the bottom button to view both the complete `location.href` value and the `location.search` portion (shown in Figure 15-4). Click the bottom button to edit the current `location.search` value (Figure 15-5).


File Edit View Go Communicator Help
Back Forward Reload Home Search Guide Print Security Stop
Show location.search Property
Modify Search...
OK
Document: Done

Alta Vista Web Pages (1-20 of 142071 3)

- SNF - The Credits - Robert Stigwood • Producer. John Badham • Director. NOTE: The images above were taken from the paperback "John Travolta: An Illustrated...
  --http://www.[illegible]/credits.html
- "Nick of Time" - Volume 6, Issue 4 / November, 1995 - Nick of Time Greg Klingenstein. View Movie Still Director John Badham takes a stab at invading Hitchcock territory with his new thriller Nick of Time, in...
  --http://www.[illegible].k12.ny.us/activities/[illegible]/nov95/nicktime.html
- Movie Guide: John Badham Film Credits - nbsp; Search our database of more than 20,000 movie reviews. Movie

Perform a search in the Yahoo frame, above. Then click the "Show location.search Property" button to examine the window.location.search property value for the search.

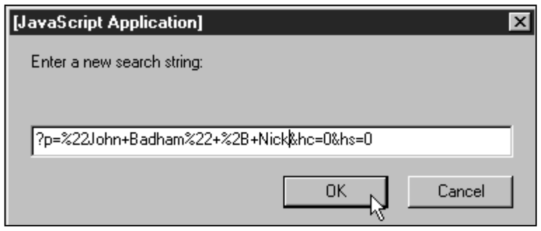
Next, click the "Modify Search..." button to mod[illegible] property. Be sure to follow the codes and conve[illegible]

[JavaScript Application]

location.href:
http://av.yahoo.com/bin/query?p=%22John+Badham%22+%2B+Director&hc=0&hs=0

location.search: ?p=%22John+Badham%22+%2B+Director&hc=0&hs=0

Figure 15-4: The alert dialog box shows both the full URL and the search property.



[JavaScript Application]

Enter a new search string:

?p=%22John+Badham%22+%2B+Nick&hc=0&hs=0

Figure 15-5: Using the existing search property as a model, make small changes to the search property to see how Yahoo! responds.

Although this interface is not as friendly as the one presented in the Yahoo! or other search engine pages, this illustration shows that you can control the search activities of a search engine or CGI parameters from a script. For example, you may prefer to invent a different user interface to search for specific keywords (or to present a limited selection to the user of your page). To do this, your script must gather the user's input in your document's form, construct the appropriate search query (in the search engine's lingo), and construct a URL that performs the search for the user. Because of security issues surrounding access to other sites, such customized interfaces to search engines are best suited to searches performed on intranet servers where all documents share a domain.

Related Items: `location.href` property.

Methods

`assign("URL")`

Returns: Nothing.

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

In earlier discussions about the location object, I said that you navigate to another page by assigning a new URL to the location object or `location.href` property. There also exists a method, `location.assign()`, that does the same thing. In fact, when you set the location object to a URL, JavaScript silently applies the `assign()` method. No particular penalty or benefit comes from using the `assign()` method, except perhaps to make your code more understandable to others. I don't recall the last time I used this method in a production document, but you are free to use it if you like.

Related Items: `location.href` property.

`reload(unconditionalGETBoolean)`

Returns: Nothing.

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

The `location.reload()` method may be inappropriately named, because it makes you think of the Reload button in the Navigator toolbar. The `reload()` method is actually more powerful than the Reload button.

Many form elements retain their screen states when you click Reload (except in Internet Explorer 3). Text and textarea objects maintain whatever text is inside them; radio buttons and checkboxes maintain their checked status; select objects remember which item is selected. About the only items the Reload button destroys are global variable values and any settable, but not visible, property (for example, the value of a hidden object). I call this kind of reload a *soft reload*.

A *hard reload,* as initiated by the `location.reload()` method, pushes aside any document settings the browser may have preserved in memory (in session history) and completely reopens the document as if you had chosen Open Location or Open File from the File menu. This method restores all default settings of form elements and ensures that users (and your scripts) are back at square one with the page.

You can script both types of reloading. For a soft reload, you invoke a window's `history.go()` method, using 0 as the parameter. For a hard reload, you use the `location.reload()` method.

By default, the `reload()` method performs what is known as a *conditional-GET*, which means that the file is retrieved from the server or the browser's cache according to the cache preferences in the browser. If your page must perform an *unconditional-GET* to retrieve continually updated server or CGI-based data, then add a `true` parameter to the `reload()` method.

Example

To experience the difference between the two loading styles, load the document in Listing 15-8. Click a radio button, enter some new text, and make a choice in the select object. Clicking the Soft Reload button invokes a method that reloads the document as if you had clicked on the browser's Reload button. It also preserves the visible properties of form elements. The Hard Reload button invokes the `location.reload()` method, which resets all objects to their default settings.

Listing 15-8: **Hard versus Soft Reloading**

```
<HTML>
<HEAD>
<TITLE>Reload Comparisons</TITLE>
<SCRIPT LANGUAGE="JavaScript1.1">
function hardReload() {
       location.reload()
}
function softReload() {
       history.go(0)
}
</SCRIPT>
</HEAD>
<BODY>
<FORM NAME="myForm">
<INPUT TYPE="radio" NAME="rad1" VALUE = 1>Radio 1<BR>
<INPUT TYPE="radio" NAME="rad1" VALUE = 2>Radio 2<BR>
<INPUT TYPE="radio" NAME="rad1" VALUE = 3>Radio 3<P>
<INPUT TYPE="text" NAME="entry" VALUE="Original"><P>
<SELECT NAME="theList">
<OPTION>Red
<OPTION>Green
<OPTION>Blue
</SELECT>
<HR>
<INPUT TYPE="button" VALUE="Soft Reload" onClick="softReload()">
<INPUT TYPE="button" VALUE="Hard Reload" onClick="hardReload()">
</FORM>
</BODY>
</HTML>
```

Related Items: `history.go()` method.

replace("URL")

Returns: Nothing.

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

In a complex Web site, you may have pages that you do not want to appear in the user's history list. For example, a registration sequence may lead the user to one or more intermediate HTML documents that won't make much sense to the user later: You especially don't want users to see these pages again if they use the Back button to return to a previous URL.

Although you cannot prevent a document from appearing in the history list (visible in the Go menu) while the user is looking at that page, you can instruct the browser to load another document into that window and replace the current history entry with the entry for the new document. This trick does not empty the history list but instead removes the current item from the list before the next URL is loaded. Removing the item from the history list prevents users from seeing the page again by clicking the Back button later.

Example

Calling the `location.replace()` method navigates to another URL, similar to assigning a URL to the location. The difference is that the document doing the calling won't appear in the history list after the new document loads. Check the history listing (in your browser's usual spot for this info) before and after clicking Replace Me in Listing 15-9.

Listing 15-9: Invoking the location.replace() Method

```
<HTML>
<HEAD>
<TITLE>location.replace() Demo</TITLE>
<SCRIPT LANGUAGE="JavaScript1.1">
function doReplace() {
       location.replace("lst15-01.htm")
}
</SCRIPT>
</HEAD>
<BODY>
<FORM NAME="myForm">
<INPUT TYPE="button" VALUE="Replace Me" onClick="doReplace()">
</FORM>
</BODY>
</HTML>
```

Related Items: history object.

History Object

Properties	*Methods*	*Event Handlers*
`current`	`back()`	(None)
`length`	`forward()`	
`next`	`go()`	
`previous`		

Syntax

Accessing history properties or methods:

```
[window.]history.property | method([parameters])
```

About this object

As a user surfs the Web, the browser maintains a list of URLs for the most recent stops. This list is represented in JavaScript by the history object. Actual URLs maintained in that list cannot be surreptitiously extracted by a script unless you are using signed scripts (in Navigator 4 — see Chapter 40) and the user grants permission. Under unsigned conditions, a script can methodically navigate to each URL in the history (by relative number or by stepping back one URL at a time), in which case the user sees the browser navigating on its own, as if possessed by a spirit. Good netiquette dictates that you do not navigate a user outside of your Web site without the user's explicit permission.

One application for the history object and its `back()` or `go()` methods is to provide the equivalent of a Back button in your HTML documents. That button triggers a script that checks for any items in the history list and then goes back one page. Your document doesn't have to know anything about the URL from which the user landed at your page.

The behavior of the Back and Forward buttons in Netscape Navigator underwent a significant change between Versions 2 and 3. In Navigator 2, there was one history list that applied to the entire browser window. You could load a frameset into the window and navigate the contents of each frame individually with wild abandon. But if you then clicked the Back button, Navigator unloaded the frameset and took you back to the page in history prior to that frameset.

In Navigator 3, each frame (window object) maintains its own history list. Thus, if you navigated within a frame, a click of the Back button steps you back out frame by frame. Only after the initial frameset documents appear in the window does the next Back button click unload the frameset. That behavior persists today in Navigator 4 and is the basis for Internet Explorer behavior from Version 3 onward.

How JavaScript reacted to the change of behavior over the generations is a bit murky. In Navigator 2, the `history.back()` and `history.forward()` methods acted like the toolbar buttons, since there was only one kind of history being tracked. In Navigator 3, however, there was a disconnect between JavaScript behavior and what the browser was doing internally with history: JavaScript failed to connect history entries to a particular frame. Therefore, a reference to `history.back()` built with a given frame name did not prevent the method from exceeding the history of that frame. Instead, the behavior was more like a global back operation, rather than frame-specific.

For Navigator 4 there is one more sea change in the relationship between JavaScript and these history object methods. The behavior of the Back and Forward buttons has been shifted to a new pair of window methods: `window.back()` and `window.forward()`. The history object methods are not specific to a frame that is part of the reference. When the `parent.frameName.history.back()` method reaches the end of history for that frame, further invocations of that method are ignored.

So much for the history of the history object. As the tale of history object method evolution indicates, you must use the history object and its methods with extreme care. Your design must be smart enough to "watch" what the user is doing with your pages (for example, by checking the current URL before navigating with these methods). Otherwise, you run the risk of completely confusing your user by navigating to unexpected places. Your script can also get into trouble because it cannot detect where the current document may be in the Back-Forward sequence in history.

Properties

current
next
previous

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

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

To know where to go when you click the Back and Forward buttons, the browser maintains a list of URLs visited. To someone trying to invade your privacy and see what sites and pages you frequent, this information is valuable. That's why the three properties that expose the actual URLs in the history list are restricted to pages with signed scripts and whose visitors have given permission to read sensitive browser data (see Chapter 40).

With signed scripts and permission, you can look through the entire array of history entries in any frame or window. Because the list is an array, you can extract individual items by index value. For example, if the array has 10 entries, you can see the fifth item by using normal array indexing methods:

```
var fifthEntry = window.history[4]
```

No property or method exists that directly reveals the index value of the currently loaded URL, but you could script an educated guess by comparing the values of the `current`, `next`, and `previous` properties of the history object against the entire list.

Since I personally don't like some unknown entity watching over my shoulder while I'm on the Net, I respect that same feeling in others and therefore discourage the use of these powers unless the user is given adequate warning in advance. The signed script permission dialog does not offer enough detail about the consequences of revealing this level of information.

Notice that in the compatibility chart these properties were available in some form in Navigator 3. Access to them required a short-lived security scheme called data tainting. That mechanism was never fully implemented and has been replaced by signed scripts.

Related Items: `history.length` property.

length

Value: Number **Gettable:** Yes **Settable:** No

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

Use the `history.length` property to count the items in the history list. Unfortunately, this nugget of information is not particularly helpful in scripting navigation relative to the current location, because your script cannot extract anything from the place in the history queue where the current document is located. If the current document is at the top of the list (the most recently loaded), you can calculate relative to that location. But users can use the Go menu to jump around the history list as they like. The position of a listing in the history list does not change by virtue of navigating back to that document. A `history.length` of 1, however, indicates that the current document is the first one the user loaded since starting the browser software.

Example

The simple function in Listing 15-10 displays one of two alert messages based on the number of items in the browser's history.

Listing 15-10: A Browser History Count

```
<HTML>
<HEAD>
<TITLE>History Object</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function showCount() {
        var histCount = window.history.length
        if (histCount > 5) {
            alert("My, my, you\'ve been busy. You have visited " +
histCount + " pages so far.")
        } else {
            alert("You have been to " + histCount + " Web pages this
session.")
        }
}
</SCRIPT>
</HEAD>

<BODY>
<FORM>
<INPUT TYPE="button" NAME="activity" VALUE="My Activity"
onClick="showCount()">
</FORM>
</BODY>
</HTML>
```

Related Items: None.

Methods

back()

Returns: Nothing.

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

The behavior of the `history.back()` method has changed in Netscape's browsers between Versions 3 and 4. Prior to Navigator 4, the method acted identically to clicking the Back button (and even this unscripted behavior changed between Navigator 2 and 3 to better accommodate frame navigation). Internet Explorer 3 and 4 follows this behavior. In Navigator 4, however, the `history.back()` method is window/frame-specific. Therefore, if you direct successive `back()` methods to a frame within a frameset, the method will be ignored once it has reached the first document to be loaded into that frame. The Back button (and the new `window.back()` method) would unload the frameset and continue taking you back through the browser's global history.

If you deliberately lead a user to a dead end in your Web site, you should make sure that the HTML document provides a way to navigate back to a recognizable spot. Because you can easily create a new window that has no toolbar or menu bar (non-Macintosh browsers), you may end up stranding your users, because they have no way to navigate out of a cul-de-sac in such a window. A button in your document should give the user a way back to the last location.

Unless you need to perform some additional processing prior to navigating back to the previous location, you can simply place this method as the parameter to the event handler attribute of a button definition.

Example

For the best demonstration of the differences between history and window object `back()` and `forward()` methods, see Listings 14-20 and 14-21 in the previous chapter.

Related Items: `history.forward()` method; `history.go()` method.

`forward()`

Returns: Nothing.

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

Less likely to be scripted than the `history.back()` action is the method that performs the opposite action: navigating forward one step in the browser's history list. Because the location of the current URL in the history list is impossible for a script to determine if the page is not part of your Web site, your script may not know exactly where it will be going. The only time you can confidently use the `history.forward()` method is to balance the use of the `history.back()` method in the same script — where your script closely keeps track of how many steps the script heads in either direction. Use the `history.forward()` method with extreme caution, and only after performing extensive user testing on your Web pages to make sure that you've covered all user possibilities. The same cautions about differences introduced in Navigator 4 for `history.back()` apply equally to `history.forward()`: Forward progress extends only through the history listing for a given window or frame, not the entire browser history list. See Listings 14-18 and 14-19 for a demonstration.

Related Items: `history.back()` method; `history.go()` method.

`go(relativeNumber | "URLOrTitleSubstring")`

Returns: Nothing.

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

Use the `history.go()` method if you have enough control over your user that you are confident of the destination before jumping there. This "go" command only accepts items that already exist in the history listing, so you cannot use it in place of setting the `window.location` object to a brand-new URL.

For navigating *n* steps in either direction along the history list, use the *relativeNumber* parameter of the `history.go()` method. This number is an integer value that indicates which item in the list to use, relative to the current location. For example, if the current URL is at the top of the list (that is, the Forward button in the toolbar is dimmed), then you need to use the following method to jump to the URL two items backward in the list:

```
history.go(-2)
```

In other words, the current URL is the equivalent of `history.go(0)` (a method that reloads the window). A positive integer indicates a jump that many items forward in the history list. Thus, `history.go(-1)` is the same as `history.back()`, whereas `history.go(1)` is the same as `history.forward()`.

Alternatively, you can specify one of the URLs or document *titles* stored in the browser's history list (titles are what appear in the Go menu). The method is a bit lenient with the string you specify as a parameter. It compares the string against all listings. The first item in the history list to contain the parameter string will be regarded as the match. But, again, no navigation takes place if the item you specify is not listed in the history.

Like most other history methods, your script will find it difficult to manage the history list or the current URL's spot in the queue. That fact makes it even more difficult for your script to intelligently determine how far to navigate in either direction or to which specific URL or title matches it should jump. Use this method only for situations in which your Web pages are in strict control of the user's activity (or for designing scripts for yourself that automatically crawl around sites according to a fixed regimen). Once you give the user control over navigation, you have no guarantee that the history list will be what you expect, and any scripts you write that depend on a history object will likely break.

In practice, this method is used mostly to perform a soft reload of the current window, using the 0 parameter.

Note

If you are developing a page for all scriptable browsers, be aware that Internet Explorer's `go()` method behaves a little differently than Netscape's. First, a bug in Internet Explorer 3 causes all invocations of `history.go()` with a non-zero value to behave as if the parameter were -1. Second, the string version does not work at all in Internet Explorer 3 (it generates an error alert); for Internet Explorer 4, the matching string must be part of the URL and not part of the document title, as in Navigator. Finally, the reloading of a page with `history.go(0)` takes a long time to complete.

Example

Fill in either the number or text field of the page in Listing 15-11 and then click the associated button. The script passes the appropriate kind of data to the `go()` method. Be sure to use negative numbers for visiting a page earlier in the history.

Listing 15-11: Navigating to an Item in History

```
<HTML>
<HEAD>
<TITLE>Go() Method</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function doGoNum(form) {
        window.history.go(parseInt(form.histNum.value))
}
function doGoTxt(form) {
        window.history.go(form.histWord.value)
}
</SCRIPT>
</HEAD>

<BODY>
<FORM>
<B>Calling the history.go() method:</B>
<HR>
Enter a number (+/-):<INPUT TYPE="text" NAME="histNum" SIZE=3
VALUE="0">
<INPUT TYPE="button" VALUE="Go to Offset"
onClick="doGoNum(this.form)"><P>
Enter a word in a title:<INPUT TYPE="text" NAME="histWord">
<INPUT TYPE="button" VALUE="Go to Match" onClick="doGoTxt(this.form)">
</FORM>
</BODY>
</HTML>
```

Related Items: `history.back()` method; `history.forward()` method; `location.reload()` method.

✦ ✦ ✦