Chapter 25. JavaScript Quick Reference

Topics in This Chapter

- Objects corresponding to the browser and its environment: Navigator, Plugin, Screen, and similar high-level objects
- Objects corresponding to HTML elements: Window, Document, Layer, Image, and similar objects directly associated with specific markup elements
- Objects corresponding to HTML forms and input elements: Form, Text, Button, Select, and similar objects used in CGI input forms
- Internal data structures: String, Array, Function, Math, Date, and similar utility libraries
- Regular expressions: RegExp

The previous chapter introduced JavaScript and gave examples of its use. This chapter provides a quick description of the core and client-side JavaScript 1.2 constructors, properties, methods, and event handlers. JavaScript 1.2 is supported in all Netscape and Internet Explorer versions 4.0 and greater. For specifics on later versions of JavaScript, see http://developer.netscape.com/docs/manuals/javascript.html.

25.1 The Array Object

In the first version of JavaScript, an array was implemented as a JavaScript object containing multiple property settings. Later, in JavaScript 1.1, an array was implemented as a separate `Array` object.

Constructors

new Array() This constructor builds a new zero-length array. Adding an element to a specified index automatically changes the length. For example,

```
var a = new Array();              // a.length = 0
a[12] = "foo";                    // a.length = 13
```

new Array(length) This constructor builds an array with indices from `0` to `length-1`. All the values will initially be `null`.

Note that this constructor is not properly recognized in Netscape's implementation of JavaScript 1.2 (Internet Explorer is fine); in Netscape this constructor produces an array with a single element where the value of the element is the constructor parameter. This Netscape problem is resolved in JavaScript 1.3.

new Array(entry0, entry1, … , entryN) This constructor creates an array of length *N* containing the specified elements.

[entry0, entry1, … , entryN] This constructor lets you create arrays by using a "literal" notation. For example, the following two statements create equivalent arrays.

```
var a1 = new Array("foo", "bar", "baz");
var a2 = [ "foo", "bar", "baz" ];
```

Properties

length This property gives the number of elements in the array. It is 1 greater than the index of the last element and is read/write. If you set `length` smaller, array elements beyond that point are lost. If you set `length` bigger, array elements beyond the old length have the special `undefined` value (which compares `==` to `null`).

Methods

concat(secondArray) This method returns a new array it forms by concatenating the current array with the specified array.

join()

join(delimiterString)

The first method returns a single large string it makes by converting all the array elements to strings and then concatenating the results. The second method is similar except that it inserts the delimiter string between each element (but not at the beginning or end).

reverse() This method changes the existing array so that the elements now appear in the opposite order. It does *not* create a new array.

slice(startIndex)

slice(startIndex, endIndex)

This method returns a new array formed by extracting the elements from `startIndex` (inclusive) to `endIndex` (exclusive).

sort()

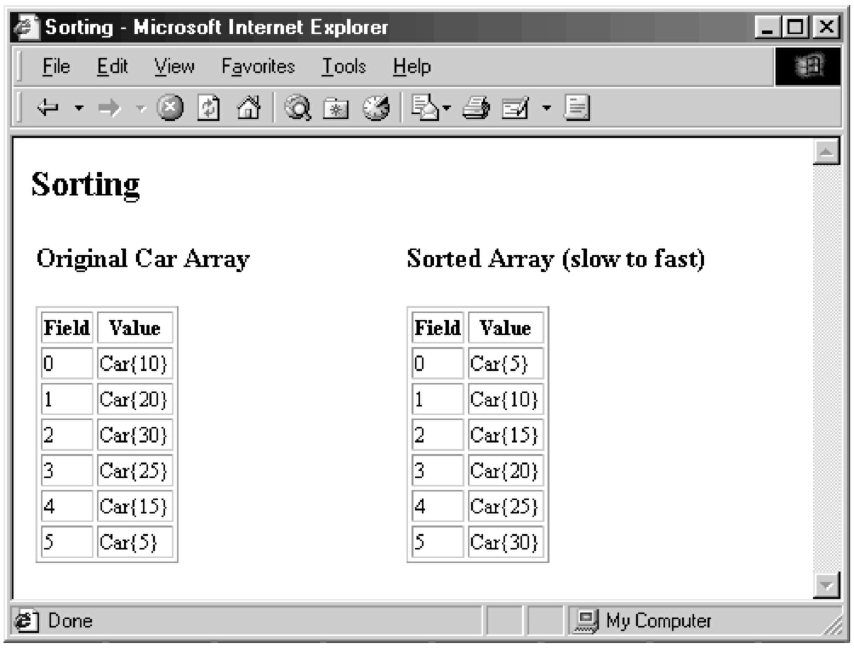
sort(comparisonFunction)

The first, i.e., no argument, method puts the array in alphabetical order without creating a new array. The second method puts the array in the order specified by the comparison function. This function should take two array elements as arguments, returning a negative number if the arguments are in order (the first is "less" than the second), zero if they are in order and would also be if they are swapped (the first is "equal to" the second in sorting value), and a positive number if they are out of order (the second is "less" than the first). For example, here is a comparison function that takes two `Car` objects and compares their `maxSpeed` properties.

```
function slower(car1, car2) {
  return(car1.maxSpeed - car2.maxSpeed);
}
```

Listing 25.1 uses this approach to create an array of `Car` objects and then sorts the array based on their `maxSpeed` property. Figure 25-1 shows the result.

Figure 25-1. In JavaScript you can sort arrays with user-defined ordering functions.



Sorting

Original Car Array

Field	Value
0	Car{10}
1	Car{20}
2	Car{30}
3	Car{25}
4	Car{15}
5	Car{5}

Sorted Array (slow to fast)

Field	Value
0	Car{5}
1	Car{10}
2	Car{15}
3	Car{20}
4	Car{25}
5	Car{30}

Sorting - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Done
My Computer

Listing 25.1 `Sort.html`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<HTML>
<HEAD>
  <TITLE>Sorting</TITLE>

<SCRIPT TYPE="text/javascript">
<!--

function makeObjectTable(name, object) {
  document.writeln("<H2>" + name + "</H2>");
  document.writeln("<TABLE BORDER=1>\n" +
                   "  <TR><TH>Field<TH>Value");
  for(field in object) {
    document.writeln("  <TR><TD>" + field + "<TD>" +
                     object[field]);
  }
  document.writeln("</TABLE>");
}
// -->
```

```
</SCRIPT>
</HEAD>
<BODY>
<H1>Sorting</H1>

<SCRIPT TYPE="text/javascript">
<!--

function carString() {
  return("Car{" + this.maxSpeed + "}");
}

function Car(maxSpeed) {
  this.maxSpeed = maxSpeed;
  this.toString = carString;
}

function slower(car1, car2) {
  return(car1.maxSpeed - car2.maxSpeed);
}

var cars = new Array(new Car(10), new Car(20),
                     new Car(30), new Car(25),
                     new Car(15), new Car(5));
// -->
</SCRIPT>

<TABLE>
  <TR><TD>
      <SCRIPT TYPE="text/javascript">
          <!--
          makeObjectTable("Original Car Array", cars);
          // -->
          </SCRIPT>
      <TD><PRE>           </PRE>
      <TD><SCRIPT TYPE="text/javascript">
          <!--
          cars.sort(slower);
          makeObjectTable("Sorted Array (slow to fast)", cars);
          // -->
          </SCRIPT>
</TABLE>

</BODY>
</HTML>
```

Event Handlers

None.

25.2 The Button Object

The `Button` object corresponds to form elements created through `<INPUT TYPE="BUTTON" ...>`. Most of its characteristics are shared by elements created through `<INPUT TYPE="SUBMIT" ...>` and `<INPUT TYPE="RESET" ...>`.

However, `SUBMIT` and `RESET` buttons have more specific types: `Submit` and `Reset`, respectively. A `Button` is typically accessed either through the `elements` array of the corresponding `Form` or, if both the form and the button are named, through `document.formName.buttonName`.

Properties

form This read-only property gives the `Form` object containing the button.

name If the button used the `NAME` attribute, this read-only property retrieves it.

type For pure `Button` objects, this property is always equal to `button`. For `SUBMIT` and `RESET` buttons, the type is `submit` and `reset`, respectively. All `Element` objects contain the `type` property, so it can be used to distinguish among the element types. The `type` property is read-only.

value This read/write property gives the label of the button and, for `SUBMIT` buttons, is transmitted with the button name when the button is used to trigger form submission.

Methods

blur() This method removes the keyboard focus from the button.

click() This method acts as though the button was clicked, but it does not trigger the `onClick` handler. For `SUBMIT` and `RESET` buttons, you can use the form's `submit` and `reset` methods instead of `click`.

focus() This method gives the keyboard focus to the button.

Event Handlers

onblur() This method is called when the button loses the input focus; it is normally set through the `onBlur` attribute, as below.

```
<INPUT TYPE="BUTTON" ...
       onBlur="doSomeAction()">
```

onclick() This method is called when the user clicks on the button, but not when the `click` method is called programmatically. It is normally set through the `onClick` attribute.

```
<INPUT TYPE="BUTTON" ...
       onClick="doSomeAction()">
```

If you return `false` from this method, then any additional action the button would trigger (i.e., submitting or resetting the form) is suppressed. For example,

```
<INPUT TYPE="RESET" ...
       onClick="return(maybeReset())">
```

The same effect can be achieved by the `onSubmit` and `onReset` handlers on the form containing the button.

ondblclick() This method is called on the second click of a double click. The `onclick` handler, if any, will be called first. It is set by the `onDblClick` attribute. This method is not supported on the Macintosh or in Netscape 6.

onfocus() This method is called when the button gains the input focus. It is normally set through the `onFocus` attribute.

25.3 The Checkbox Object

The `Checkbox` object corresponds to form elements created through `<INPUT TYPE="CHECKBOX" ...>`. A `Checkbox` is typically accessed either through the `elements` array of the corresponding `Form` or, if both the form and the check box are named, through `document.formName.checkboxName`.

Properties

checked This `Boolean` property specifies whether the box is currently checked. It is read/write.

defaultChecked This property is a `Boolean` specifying whether the box should be initially set. It is set through the `CHECKED` attribute and is read-only.

form This read-only property refers to the `Form` object containing the checkbox.

name This read-only property gives the name of the check box as given in the `NAME` attribute.

type This property contains the string `checkbox`. Since all `Element` objects have this property, it can be used to differentiate among them when the `form.elements` array is looked at. This property is read-only.

value This read/write property gives the value that is sent with the name to the CGI program if the form is submitted when the box is checked.

Methods

blur() This method removes the keyboard focus from the check box.

click() This method acts as though the check box was clicked, but it does not trigger the `onClick` handler.

focus() This method gives the keyboard focus to the check box.

Event Handlers

onblur() This method is called when the check box loses the input focus. It is normally set through the `onBlur` attribute, as below.

```
<INPUT TYPE="CHECKBOX" ...
       onBlur="doSomeAction()">
```

onclick() This method is called when the user clicks on the check box, but not when the `click` method is called programmatically. It is usually specified through the `onClick` attribute of the input element.

onfocus() This method is called when the check box gains the input focus. It is normally set through the `onFocus` attribute.

25.4 The Date Object

The `Date` object creates and manipulates dates and times.

Constructors

new Date() This constructor creates a `Date` object for the current time.

new Date(year, month, day) This constructor creates a `Date` object for midnight on the morning of the specified day.

new Date(year, month, day, hrs, mins, secs) This constructor creates a `Date` object for the specified time.

new Date("month day, year hrs:mins:secs") This constructor creates a `Date` object from the given string. The month should be the full name, not a number. For example,

```
var bDay = new Date("January 30, 1962 00:00:00");
```

new Date(millisecondsSinceEpoch) This constructor creates a `Date` object for the time corresponding to the specified number of milliseconds after midnight (GMT) on January 1, 1970.

Properties

None.

Methods

Note that `parseDate` and `UTC` are not really methods of `Date` objects but instead act like static methods of the `Date` "class" (really constructor). They *must* be invoked as `Date.parseDate` and `Date.UTC`, respectively, never through an individual `Date` object. The other methods should be invoked through `someDateObject.method(args)`.

getDate()

setDate(dayOfMonth)

These methods get and set the day of the month. The value is an integer from 1 to 31.

getDay() This method returns the day of the week as an integer from 0 (Sunday) to 6 (Saturday).

getHours()

setHours(hours)

These methods get and set the hour of the day as an integer from 0 to 23.

getMinutes()

setMinutes(minutes)

The `getMinutes` method returns the number of minutes past the hour given in `getHours`. Similarly, the `setMinutes` sets the number of minutes past the hour, specified as an integer from 0 to 59.

getMonth()

setMonth(monthIndex)

These methods get and set the month of the year as an integer from 0 (January) to 11 (December).

getSeconds()

setSeconds(seconds)

These methods get and set the number of seconds past the minute given by `getMinutes`. The seconds value is an integer from 0 to 59.

getTime()

setTime(millisecondsSinceEpoch)

These methods get and set the number of milliseconds after midnight January 1, 1970, GMT.

getTimezoneOffset() This method gives the difference in *minutes* between GMT and the local time.

getFullYear()

setFullYear(year)

The first method, `getFullYear`, returns the year of the `Date` object as a four-digit integer. The second method, `setFullYear`, sets the year to the `Date` object and returns the number of milliseconds since midnight January 1, 1970, GMT to the time specified in the `Date` object. Note that the `getYear`/`setYear` methods introduced in JavaScript 1.0 are deprecated because the year was represented as a two-digit or a four-digit number, depending on the browser.

parse(dateString) This method is not really a method of `Date` objects, but rather a method named `Date.parse`. It *must* be invoked that way, not on a `Date` object. It takes a string in any of a variety of formats as input and returns the corresponding number of milliseconds after midnight, January 1, 1970. It understands the standard IETF date formats used on the Internet (and generated by `toGMTString`), so, for instance, the following generates "Wed Sep 03 11:30:00 1997" on systems on the U.S. East Coast, which is in EDT (minus 4 hours offset from GMT).

```
// US Pacific Time
var dateString = "Wed, 3 Sep 1997 08:30:00 -0700";
var d1 = new Date(Date.parse(dateString));
// US Eastern Time
document.writeln(d1.toLocaleString());
```

The `parse` method also understands strings of the form `"Month Day, Year"`, where the month is spelled out completely or the first three letters are used (upper or lower case). In the IETF format, you can also use U.S. time zone abbreviations (e.g., EDT) instead of numeric offsets.

toGMTString() This method generates a string representing the date in GMT. It is formatted with IETF conventions; see `parse`.

toLocaleString() This method generates a string representing the date in the local time zone. It is formatted with local conventions.

UTC(year, month, day)

UTC(year, month, day, hrs)

UTC(year, month, day, hrs, mins)

UTC(year, month, day, hrs, mins, secs)

Each method is not really a method of `Date` objects, but rather a method named `Date.UTC`. It *must* be invoked that way, not on a `Date` object. It assumes that the input parameters are in GMT (also called UTC—Universal Coordinated Time) and returns the number of milliseconds since midnight, Jan 1, 1970, GMT.

Event Handlers

None. `Date` does not correspond to an HTML element.

25.5 The Document Object

Each `Window` object contains a `document` property referring to the document contained in the window. The top-level document is obtained through `window.document` or, more commonly, simply by `document`.

Properties

alinkColor This string specifies the color of activated links. It is initially set by the `ALINK` attribute of `BODY` and can only be modified by scripts that run in the `HEAD` of the document (which are parsed before the `BODY`). After that, it is read-only.

anchors This property returns an array of `Anchor` objects, one for each occurrence of `<A id=...>` in the document.

applets This is an array of `Applet` objects, one for each occurrence of `<APPLET ...>` in the document. If the `APPLET` tag includes a `MAYSCRIPT` attribute, you can call the applet's methods directly from JavaScript. See Section 24.9 (Accessing Java from JavaScript) for an example.

bgColor This string specifies the background color of the document. It is initially set by the `BGCOLOR` attribute of `BODY` but is read/write (anytime, not just in the `HEAD`, as with other colors). For example,

```
document.bgColor = "red";
document.bgColor = "#00FF00"; // green
```

cookie This string is the value of the cookie associated with the document. It is read/write; setting the value has the side effect of changing the cookie stored by the browser. For more details, see Section 24.7 (Using JavaScript to Store and Examine Cookies).

domain This string specifies the Internet domain that the document came from. It is read-only. JavaScript provides no standard way to find the domain or hostname of the *client* system (the one currently viewing the page), but you can do this with a little help from Java. See Section 24.9 (Accessing Java from JavaScript).

embeds This is an array of `JavaObject` objects corresponding to plug-in entries inside `EMBED` elements in the document. If the embedded object is not a Java-enabled plug-in, you cannot do anything with it, but if it is, you can call its public methods. Synonymous with the `plugins` array.

fgColor This string specifies the foreground color of the document. It is initially set by the `TEXT` attribute of `BODY` and can only be modified by scripts that run in the `HEAD` of the document. After that, it is read-only.

forms This is an array of `Form` objects, one for each occurrence of `<FORM ...>` in the document. See Section 25.8 for more on `Form`.

images This is an array of `Image` objects, one for each occurrence of `<IMG ...>` in the document. See Section 24.5 (Using JavaScript to Make Pages Dynamic) for examples.

lastModified This property gives the date of the most recent change to the document. Inserting this information near the top of the document is useful for readers who visit a page repeatedly, looking for new information. It is read-only.

linkColor This string specifies the color of unvisited links. It is initially set through the `LINK` attribute of `BODY` and can only be modified by scripts that run in the `HEAD` of the document.

After that, it is read-only.

links This is an array of `Link` objects, one for each occurrence of `<A HREF...>` in the document.

location This read-only property refers to the same `Location` object as `window.location` and contains the requested URL, which might have been redirected. The `document.URL` property contains the actual URL.

plugins This property is a synonym for the `embeds` array. Note that the array contains objects of type `JavaObject`, not `Plugin`, and describes embedded plug-ins in the current document, not plug-ins available to the browser. Use `navigator.plugins` to get an array describing the available plug-ins.

referrer This string, possibly empty, gives the URL of the document that contained the link to the current one. It is read-only.

title This property gives the string specified through `<TITLE>`. It is read-only.

URL This string gives the actual URL of the current document. It is read-only.

vlinkColor This string specifies the color of visited hypertext links. It is initially set through the `VLINK` attribute of `BODY` and can only be modified by scripts that run in the `HEAD` of the document. After that, it is read-only.

Methods

close() This method closes the output stream to the specified document, displaying any results that haven't already been displayed. It is used with `open` to build new documents.

getSelection() This method gives the text, if any, contained in the selected area.

open()

open(mimeType)

open(mimeType, "replace")

This method creates a new document in the current window. The most common usage is simply to call `open()`, then use `write` and `writeln` to add the content. However, you can optionally specify a MIME type, as follows:

- `text/html`—for regular HTML; the default.
- `text/plain`—for ASCII text with newline characters to delimit lines.
- `image/gif`—for encoded bytes representing a GIF file.
- `image/jpg`—for encoded bytes representing a JPEG file.
- `image/x-bitmap`—for encoded bytes representing an X bitmap.
- *pluginName*—for loading into plug-ins. For instance, specify `x-world/vrml`. Future `write`/`writeln` calls go to the plug-in.

If `replace` is specified, the new document replaces the previous one in the history list. Otherwise, a new history entry is created.

write(arg1, arg2, … , arg*N*)

writeln(arg1, arg2, ... , arg*N*)

These methods send output to the document, with or without a trailing newline character.

Event Handlers

Technically, `Document` has no event handlers; the `onLoad` and `onUnload` attributes of `BODY` set the `onload` and `onunload` event handlers of the `Window` object, not the `Document`.

25.6 The Element Object

The `Element` object corresponds to a form element and is contained in the `elements` array of the `Form` object. The `Form` objects are accessible through the `document.forms` array or, if named, by `document.formName`. Rather than treating elements of the elements array as `Element` objects, you are usually better off to treat them as `Button` objects, `Checkbox` objects, and so forth, based on their more specific types. In general, you should name objects and access them through their name.

Properties

The following only lists the more specific type of `Element` that uses these properties. See the sections describing those objects for details on the properties.

checked This property is used by `Checkbox` and `Radio` objects.

defaultChecked This property is used by `Checkbox` and `Radio` objects.

defaultValue This property is used by `FileUpload`, `Password`, `Text`, and `Textarea` objects.

form This property is used by all `Element` objects and refers to the HTML form containing the element.

length This property is used only by `Select` objects.

name This property is used by all `Element` objects and gives the value of the HTML `NAME` attribute.

options This property is used only by `Select` objects.

selectedIndex This property is used only by `Select` objects.

type In JavaScript 1.1, this property is used by all `Element` objects and can be used to differentiate among element types. The value of this property is one of `button`, `checkbox`, `file`, `hidden`, `password`, `radio`, `reset`, `select-one`, `select-multiple`, `submit`, `text`, or `textarea`.

value This property is used by all `Element` objects and gives the value that will be associated with the element's name when the form is submitted.

Methods

Again, the following lists only the more specific type of `Element` that uses these methods. See the sections describing those objects for more information on the methods.

blur() This method is used by all `Element` types except `Hidden`.

click() This method is used by `Button`, `Checkbox`, `Radio`, `Reset`, and `Submit` objects.

focus() This method is used by all `Element` types except `Hidden`.

select() This method is used by all elements that have textual values, namely, `FileUpload`, `Password`, `Text`, and `Textarea`.

Event Handlers

Again, the following only lists the more specific type of `Element` that uses these methods.

onblur() This method is used by all `Element` types except `Hidden`.

onchange() This method is used by `FileUpload`, `Password`, `Text`, `Textarea`, and `Select` objects.

onclick() This method is used by `Button`, `Checkbox`, `Radio`, `Reset`, and `Submit` objects.

ondblclick This method is used by `Button`, `Reset`, and `Submit`.

onfocus() This method is used by all `Element` types except `Hidden`.

25.7 The FileUpload Object

The `FileUpload` object corresponds to form elements declared through `<INPUT TYPE="FILE" ...>`. Objects of this type are generally accessed through the `elements` array of the `Form` object.

Properties

form This read-only property gives the `Form` object containing the element.

name If the element used the `NAME` attribute, this property retrieves it. The property is read-only.

type This read-only property is always equal to `file`. All `Element` objects contain this property, so it can be used to distinguish among the various types.

value This property gives the string set in the `VALUE` attribute. It is read-only.

Methods

blur() This method removes the keyboard focus from the element.

focus() This method gives the keyboard focus to the element.

select() This method selects the text in the element. Any user entry will replace the existing text.

Event Handlers]

onblur() This method is called when the element loses the input focus. It is normally set through the `onBlur` attribute, as follows:

```
<INPUT TYPE="FILE" ...
       onBlur="doSomeAction()">
```

onchange() This method is called when the element loses the focus after its value has been changed. It is normally set through the `onChange` attribute.

onfocus() This method, normally set through the `onFocus` attribute, is called when the element gains the input focus.

25.8 The Form Object

`Form` objects correspond to elements created with the HTML `FORM` element. They are normally accessed from the `document.forms` array or, if named, through `document.formName`.

Properties

action This string specifies the URL to which the form should be submitted. It is read/write.

elements This property is an array of `Element` objects corresponding to the input elements contained in the HTML form. See Section 25.6 for information on `Element`.

encoding This string specifies the form's encoding method, as initially set by the `ENCTYPE` attribute. It is read/write.

method This string is either `get` or `post`. It is initially set through the `METHOD` attribute but is read/write.

target This string specifies the frame in which the form results should be displayed. It is initially set through the `TARGET` attribute but is read/write.

Methods

reset() This method calls `onreset` and then, if the return value is not `false`, restores all input elements to the values originally specified in the document. The result is the same as if a `RESET` button was pressed.

submit() This method submits the form *without* first calling `onsubmit`.

Event Handlers

onreset() This method is called when the user presses a Reset button or the `reset` method is called. If `onReset` returns `false`, the form is not reset. It is normally specified through the `onReset` attribute, as follows:

```
<FORM ACTION="..." ...
      onReset="return(maybeReset())">
```

onsubmit() This method is called when the user presses a Submit button. It is *not* automatically called by `submit`. If `onSubmit` returns `false`, the form is not submitted. It is normally specified through the `onSubmit` attribute, as follows:

```
<FORM ACTION="..." ...
      onSubmit="return(validateEntries())">
```

For an example, see Section 24.6 (Using JavaScript to Validate HTML Forms).

25.9 The Function Object

The `Function` object corresponds to a JavaScript function, not to any HTML element.

Constructor

new Function(arg0Name, … , arg*N*Name, bodyString) This constructor builds a new function. For instance, the following two forms have the same effect, but the second can be performed inside another routine at run time.

```
function square(x) { return(x * x); }
square = new Function("x", "return(x * x)");
```

Properties

arguments From within the body of a function, this property gives an array of arguments used to call the function. Use this property to create variable-argument functions. For example, the following function adds any number of values together.

```
function sum() {
  var total = 0;
  for(var i=0; i<arguments.length; i++) {
    total = total + arguments[i];
  }
  return(total);
}
```

arity This read-only property gives the number of declared arguments a function expects. This may be different from the number it is actually called with, which is given from within the body by the `length` property.

caller From within the body of a function, this property gives the `Function` that called this one. The value is `null` if the function was called at the top level. It is read-only.

prototype For constructors, this property defines properties that are shared by all objects of the specified type. For an example, see Section 24.3 (Mastering JavaScript Syntax).

Methods

None, other than those defined for every `Object`.

Event Handlers

None. `Function` does not correspond to an HTML element.

25.10 The Hidden Object

The `HIDDEN` object corresponds to elements created through `<INPUT TYPE="HIDDEN" ...>`. Objects of this type are usually accessed through the `elements` property of `Form` or, if both the form and the hidden element are named, through `document.formName.elementName`.

Properties

form This read-only property refers to the `Form` object that holds this element.

name This read-only property gives the name of the element, as specified by the `NAME` attribute.

type This read-only property contains the value `hidden`.

value This property gives the string that is sent along with the name when the form is submitted. It is read/write.

Methods

None.

Event Handlers

None.

25.11 The History Object

The `HISTORY` object corresponds to the current window or frame's list of previously visited URLs. It is accessible through the `history` property of `Window`, which can be accessed through `window.history` or just `history`.

Properties

current In signed scripts you can read the URL of the current document. The property is a string and is read-only.

length This read-only property gives the number of URLs contained in the history list.

next This string specifies the URL of the next document in the history list. It is read-only and requires a signed script.

previous This string specifies the URL of the next document in the history list. It is read-only and requires a signed script.

Methods

back() This method instructs the browser to go back one entry in the history list.

forward() This method instructs the browser to go forward one entry in the history list.

go(n) This method instructs the browser to go *n* entries forward (if *n* is positive) or backward (if *n* is negative) in the history list.

Event Handlers

None. `History` does not correspond directly to an HTML element on the page.

25.12 The Image Object

The `Image` object corresponds to HTML elements inserted through `<IMG SRC="..." ...>`. An `Image` object is accessed through the `document.images` array or, if the image is named, through `document.imageName`. Manipulating images through JavaScript is an important capability; see Section 24.5 (Using JavaScript to Make Pages Dynamic) for examples.

Constructor

new Image(width, height) This constructor allocates a new `Image` object of the specified size. The main purpose for an `Image` object is to then set its `src` property in order to preload images that are used later. In such an application, the `Image` object is never actually used after its `src` is set; the purpose is to get the *browser* to cache the image. See Section 24.5 (Using JavaScript to Make Pages Dynamic) for an example.

Properties

border This property gives the size of the border around images that appear inside hypertext links. It is specified through the `BORDER` attribute of the `IMG` element. It is read-only.

complete This is a `Boolean` that determines whether the image has finished loading. It is read-only.

height This property gives the height of the image either as specified through the `HEIGHT` attribute (if present) or as it is in the actual image file. It is read-only.

hspace This property gives the number of empty pixels on the left and right of the image as given in the `HSPACE` attribute. It is read-only.

lowsrc Netscape (but not Internet Explorer) supports the nonstandard `LOWSRC` attribute in the `IMG` element, which gives an alternate image to show on low-resolution displays. This property gives that value (as a string). It is read/write.

name This read-only property gives the name of the image as given by the `NAME` attribute.

src This property gives a string representing the URL of the image file. It is read/write.

vspace This read-only property gives the number of empty pixels on the top and bottom of the image as given in the `VSPACE` attribute.

width This property gives the width of the image either as specified through the `HEIGHT` attribute (if present) or as it is in the actual image file. It is read-only.

Methods

None.

Event Handlers

onabort() This method is called when the user halts image loading by pressing the Stop button or by clicking on a hypertext link to go to another page. It is normally specified through the `onAbort` attribute, as below.

```
<IMG SRC="..." ...
     onAbort="takeSomeAction()">
```

onerror() This method is called when the image file cannot be found or is in illegal format. It is normally specified by the `onError` attribute, as follows:

```
<IMG SRC="..." ...
     onAbort="alert('Error loading image')">
```

Supplying a value of `null` suppresses error messages, as in this example.

```
<IMG SRC="..." ...
     onAbort="null">
```

onload() This method is called when the browser *finishes* loading the image. Every time the `src` is changed, this method is called again. It is normally set through the `onLoad` attribute, as follows:

```
<IMG SRC="..." ...
     onLoad="startImageAnimation()">
```

In an example like this, the `startImageAnimation` would change the `src` to a new image (perhaps after a fixed pause), which, when done, would trigger `startImageAnimation` all over again.

25.13 The JavaObject Object

A `JavaObject` is a JavaScript representation of either a real Java object (an applet) or a plug-in from the `document.embeds` array that is treated as a Java object. This object has no predefined properties or methods, but you can use `for`/`in` to look at the specific properties of any particular `JavaObject`. This behavior, known as *reflection,* is also available in Java 1.1 and later.

25.14 The JavaPackage Object

Objects of the `JavaPackage` type are accessed through the `java`, `netscape`, `sun`, and `Packages` properties of `Window`. They are used to access Java objects; for instance, you can call `java.lang.System.getProperty`. For an example, see Section 24.9 (Accessing Java from JavaScript).

25.15 The Layer Object

Netscape 4.0 supports layered HTML: HTML in separate, possibly overlapping regions. Layers can be defined with the `LAYER` or `ILAYER` element or through cascading style sheets; see Section 5.12, (Layers). JavaScript can access and manipulate layers; see Section 24.5 (Using JavaScript to Make Pages Dynamic) for examples. However, note that in Netscape 6, to be compliant with the HTML 4.0 specification, `LAYER` and `ILAYER` elements are no longer supported.

Constructors

new Layer(width) This constructor creates a new `Layer` object. You can specify its contents by setting the `src` property or by using the `load` method.

new Layer(width, parentLayer) This constructor builds a layer that is a child of the one specified.

Properties

above This read-only property specifies the layer above the current one.

background This property specifies the image to use for the layer of the background. It is read/write. For example,

```
someLayer.background.src = "bricks.gif";
```

below This property specifies the layer below the current one. It is read-only.

bgColor Layers are normally transparent, but the `bgColor` property can make them opaque. It is read/write. For instance,

```
someLayer.bgColor = "blue";
anotherLayer.bgColor = "#FF00FF";
thirdLayer.bgColor = null; // transparent
```

clip This property defines the clipping rectangle and is composed of `clip.top`, `clip.bottom`, `clip.left`, `clip.right`, `clip.width`, and `clip.height`. It is read/write.

document Each layer contains its own `Document` object. This property references the document object and is read-only.

left This property is the horizontal position of the layer with respect to the parent layer or, for floating layers, with respect to the natural document flow position. It is read/write.

name This read-only property gives the layer name as specified through the `ID` or `NAME` attributes.

pageX This property is the absolute horizontal position of the layer in the page. It is read/write.

pageY This property is the absolute vertical position of the layer in the page. It is read/write.

parentLayer This property returns the enclosing layer if there is one. Otherwise, it returns the enclosing `Window` object. It is read-only.

siblingAbove Of the layers that share the same parent as the current one, this property refers to the layer directly above the current one. It is read-only.

siblingBelow Of the layers that share the same parent as the current one, this property refers to the one directly below the current one. It is read-only.

src This property gives a URL specifying the content of the layer. It is read/write.

top This property is the vertical position of the layer with respect to the parent layer or, for floating layers, with respect to the natural document flow position. It is read/write.

visibility This property determines the layer's visibility. Legal values are `show` (layer is visible), `hide` or `hidden` (layer is invisible), or `inherit` (use parent's visibility). It is read/write.

zIndex This property specifies the stacking order relative to sibling layers. Lower numbers are underneath; higher numbers are on top. It is read/write.

Methods

load(sourceString, width) This method changes the source of the layer while simultaneously changing its width. See the `src` property.

moveAbove(layer) This method stacks the current layer above the one specified.

moveBelow(layer) This method stacks the current layer below the one specified.

moveBy(dx, dy) This method changes the layer's location by the specified number of pixels.

moveTo(x, y) This method moves the layer so that its top-left corner is at the specified location in the containing layer or document. See the `left` and `top` properties.

moveToAbsolute(x, y) This method moves the layer so that its top-left corner is at the specified location in the window. See the `pageX` and `pageY` properties.

resizeBy(dWidth, dHeight) This method changes the layer's width and height by the specified number of pixels. See the `clip.width` and `clip.height` properties.

resizeTo(width, height) This method changes the layer's width and height to the specified size in pixels. See the `clip.width` and `clip.height` properties.

Event Handlers

onblur() This method is called when the layer loses the keyboard focus. It is specified through the `onBlur` attribute of `LAYER` or `ILAYER`. If the layer is created with style sheets, you set this behavior by directly assigning a function, as below.

```
function blurHandler() { ... }
someLayer.onblur = blurHandler;
```

onfocus() This method is called when the layer gets the keyboard focus. It is normally specified by the `onFocus` attribute.

onload() This method is called when the layer is loaded (which may be before it is displayed). The method is normally specified by the `onLoad` attribute.

onmouseout() This method is called when the mouse moves off the layer. It is normally specified through the use of the `onMouseOut` attribute.

onmouseover() This method is called when the mouse moves onto the layer. It is normally specified through the use of the `onMouseOver` attribute.

25.16 The Link Object

The `Link` object corresponds to a hypertext link created through `<A HREF=...>`. On non-Windows platforms, the `AREA` client-side image map element also results in `Link` objects. Links are normally accessed through the `document.links` array. You cannot name links through the `NAME` attribute, since using `NAME` inside an `A` element creates an anchor for internal hypertext links.

Properties

hash This property gives the "section" part of the hypertext reference and includes the leading `#` (hash mark). It is read/write.

host This property returns a string of the form `hostname:port`. It is read/write.

hostname This property returns the hostname. It is read/write.

href This property gives the complete URL. It is read/write.

pathname This property gives the part of the URL that comes after the host and port. It is read/write.

port This property is a *string* (not integer) specifying the port. It is read/write.

protocol This property specifies the protocol. The colon is included as part of this property. It is read/write.

search This property gives the search part (e.g., `"?x,y"` from an `ISMAP` entry) or query part (e.g., `"?x=1&y=2"` from a form submission) of the URL. It is read/write.

target This property gives the name given as the value of the `TARGET` attribute. It is read/write. For instance, to redirect all hypertext links to a frame named `frame1`, you could do the following:

```
for(var i=0; i<document.links.length; i++) {
  document.links[i].target = "frame1";
}
```

Methods

None (other than the event handlers).

Event Handlers

onclick() This method is called when the user clicks on the hypertext link. Returning `false` prevents the browser from following the link. It is normally specified through the `onClick` attribute, as follows:

```
<A HREF="..." ...
   onClick="return(maybeCancel())">
```

ondblclick() This method is called on the second click of a double click. The `onclick` handler, if any, will be called first. It is set by the `onDblClick` attribute. This method is not supported on the Macintosh or in Netscape 6.

onmouseout() This method is called when the user moves the mouse off the link. Combined with `onmouseover`, this provides a good method to highlight images under the mouse. For an example, see Section 24.5 (Using JavaScript to Make Pages Dynamic). It is normally specified by the `onMouseOut` attribute.

onmouseover() This method is called when the user moves the mouse over a link. It is normally specified by using the `onMouseOver` attribute. If the method returns `true`, the browser does not display the associated URL in the status line. This behavior lets you use this method to display custom status-line messages.

25.17 The Location Object

The `Location` object corresponds to a window's current URL, as given in the `window.location` property.

Properties

hash This property gives the "section" part of the hypertext reference and includes the leading `#` (hash mark). It is read/write.

host This property returns a string of the form `hostname:port`. It is read/write.

hostname This property returns the hostname. It is read/write.

href This property gives the complete URL. It is read/write.

pathname This property gives the part of the URL that came after the host and port. It is read/write.

port This property is a *string* (not integer) specifying the port. It is read/write.

protocol This property specifies the protocol. The colon is included as part of this property. It is read/write.

search This property gives the search part (e.g., `"?x,y"` from an `ISMAP` entry) or query part (e.g., `"?x=1&y=2"` from a form submission) of the URL. It is read/write. You can use this property to implement self-processing "server"-side image maps or CGI forms by specifying the current document as the target URL and then checking `location.search` at the top of the document. See the `unescape` function, described in Section 25.31 (The String Object), for URL-decoding strings.

Methods

reload()

reload(true)

The first method reloads the document if the server reports it as having changed since last loaded. The second method always reloads the document.

replace(newURL) This method replaces the current document with a new one (just like setting `location` to a new value), but without adding a new entry in the history list.

Event Handlers

None.

25.18 The Math Object

The `Math` object does not correspond to an HTML element but is used for basic arithmetic operations. It supports substantially the same methods as does the `java.lang.Math` class in Java. You never make a new `Math` object, but instead you access the properties and methods through `Math.propertyName` or `Math.methodName(...)`.

Properties

Using these constants is faster than recalculating them each time.

E This is *e,* the base used for natural logarithms.

LN10 This is ln(10), that is, $\log_e(10)$.

LN2 This is ln(2), that is, $\log_e(2)$.

LOG10E This is $\log_{10}(e)$.

LOG2E This is lg(e), that is, $\log_2(e)$.

PI This is π.

SQRT1_2 This is

$\sqrt{1/2}$

, that is,

$1/\sqrt{2}$

SQRT2 This is

$\sqrt{2}$

Methods

General-Purpose Methods

abs(num) This method returns the absolute value of the specified number.

ceil(num) This method returns the smallest integer greater than or equal to the specified number.

exp(num) This method returns e^{num}.

floor(num) This method returns the greatest integer less than or equal to `num`.

log(num) This method returns the natural logarithm of the specified number. JavaScript does not provide a method to calculate logarithms using other common bases (e.g., 10 or 2), but following is a method that does this, using the relationship

$$\log_{b1}(n) = \frac{\log_{b2}(n)}{\log_{b2}(b1)}$$

```
function log(num, base) {
  return(Math.log(num) / Math.log(base));
}
```

max(num1, num2) This method returns the larger of the two numbers.

min(num1, num2) This method returns the smaller of the two numbers.

pow(base, exponent) This method returns *base*^{exponent}.

random() This method returns a random number from 0.0 (inclusive) to 1.0 (exclusive).

round(num) This method rounds toward the nearest number, rounding up if *num* is of the form xxx.5.

sqrt(num) This method returns the square root of *num*. Taking the square root of a negative number returns NaN.

Trigonometric Methods

All of these methods are in radians, not degrees. Convert degrees to radians with

```
function degreesToRadians(degrees) {
  return(degrees * Math.PI / 180);
}
```

acos(num) This method returns the arc cosine of the specified value. The result is expressed *in radians.*

asin(num) This method returns the arc sine of the specified number.

atan(num) This method returns the arc tangent of the specified number.

atan2(y, x) This method returns the θ part of the polar coordinate (r, θ) that corresponds to the Cartesian coordinate (x,y). This is the arc tangent of y/x that is in the range -π to π.

cos(radians) This method returns the cosine of the specified number, interpreted as a number in radians.

sin(radians) This method returns the sine of the specified number.

tan(radians) This method returns the tangent of the specified number.

Event Handlers

None. Math does not correspond to an HTML element.

25.19 The MimeType Object

The MimeType object describes a MIME type. The navigator.mimeTypes array lists all types supported by the browser, either directly by plug-ins or through external "helper" applications. For example, you could use code like the following to insert a link to an Adobe Acrobat file only if the current browser supports Acrobat; otherwise, you would use a plain text document.

```
document.writeln('For more information, see');
if (navigator.mimeTypes["application/pdf"] != null) {
  document.writeln
    ('<A HREF="manual.pdf">the widget manual</A>.');
} else {
  document.writeln
    ('<A HREF="manual.text">the widget manual</A>.');
}
```

For a list of common MIME types, see Table 19.1.

Properties

description This read-only property gives a textual description of the type.

enabledPlugin This property refers to the `Plugin` object that supports this MIME type if the plug-in is enabled. The property is `null` if no installed and enabled plug-in supports this type. It is read-only.

suffixes This property gives a comma-separated list of the filename extensions that are assumed to be of this type. It is read-only.

type This string is the type itself, e.g., `application/postscript`.

Methods

None.

Event Handlers

None. `MimeType` does not correspond directly to any HTML element.

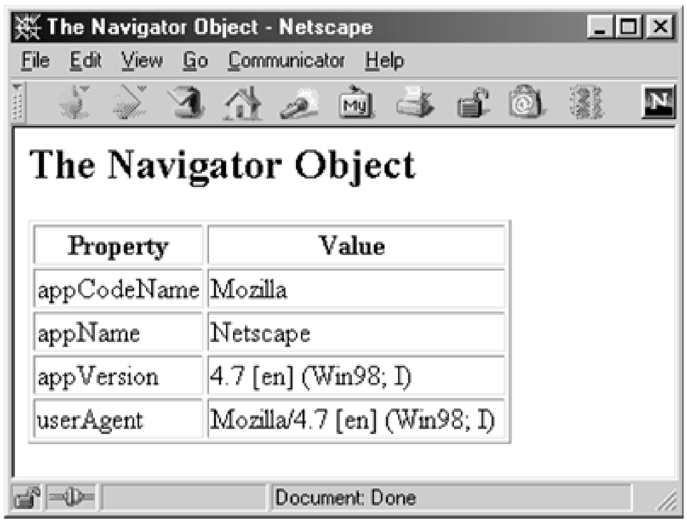
25.20 The Navigator Object

The `Navigator` object provides information about the browser. It is available through the `navigator` property of `Window`; i.e., through `window.navigator` or simply `navigator`.

Properties

Following the property descriptions, Listing 25.2 shows a page that makes a table of several properties. Results are shown in Figures25-2 and 25-3.

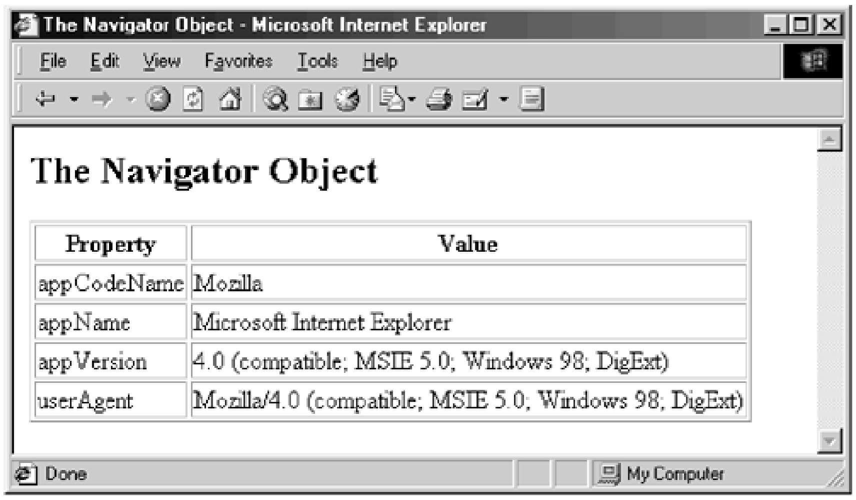
Figure 25-2. `Navigator` properties in Netscape 4.7 on Windows 98.


The Navigator Object - Netscape
File Edit View Go Communicator Help
Document: Done

The Navigator Object

Property	Value
appCodeName	Mozilla
appName	Netscape
appVersion	4.7 [en] (Win98; I)
userAgent	Mozilla/4.7 [en] (Win98; I)

Figure 25-3. `Navigator` properties in Internet Explorer 5.0 on Windows 98.


The Navigator Object - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Done
My Computer

The Navigator Object

Property	Value
appCodeName	Mozilla
appName	Microsoft Internet Explorer
appVersion	4.0 (compatible; MSIE 5.0; Windows 98; DigExt)
userAgent	Mozilla/4.0 (compatible; MSIE 5.0; Windows 98; DigExt)

appCodeName This property is intended to give the code name of the browser. Internet Explorer and Netscape use `Mozilla` for this property. It is read-only.

appName This is the browser name, for example, `Netscape` or `Microsoft Internet Explorer`. It is read-only.

appVersion This read-only property gives operating system and release number information.

language This property gives the browser translation. For English versions, the property is `en`. It is read-only.

mimeTypes This property is an array of `MimeType` objects supported by the browser either through plug-ins or helper applications. See Section 25.19 (The MimeType Object).

platform This property gives the machine type for which the browser was compiled. For instance, on Windows 95, 98, and NT, the type is `Win32`. It is read-only.

plugins This property is an array of `Plugin` objects supported by the browser. See Section 24.4 (Using JavaScript to Customize Web Pages) for an example of its use.

userAgent This is the string sent by the browser to the server in the `User-Agent` HTTP request header. It is read-only.

Figures25-2 and 25-3 show examples of some of these properties, based on a page generated from Listing 25.2.

Listing 25.2 `Navigator.html`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
  <TITLE>The Navigator Object</TITLE>

<SCRIPT TYPE="text/javascript">
<!--
function makePropertyTable(name, object, propertyList) {
  document.writeln("<H2>" + name + "</H2>");
  document.writeln("<TABLE BORDER=1>\n" +
                   "  <TR><TH>Property<TH>Value");
  var propertyName;
  for(var i=0; i<propertyList.length; i++) {
    propertyName = propertyList[i];
    document.writeln("  <TR><TD>" + propertyName +
                     "<TD>" + object[propertyName]);
  }
  document.writeln("</TABLE>");
}

// -->
</SCRIPT>
</HEAD>

<BODY BGCOLOR="WHITE">

<SCRIPT TYPE="text/javascript">
<!--

var propNames = new Array("appCodeName", "appName",
                          "appVersion", "userAgent");
makePropertyTable("The Navigator Object", navigator, propNames);

// -->
</SCRIPT>

</BODY>
</HTML>
```

Methods

javaEnabled() This method returns `true` if the browser supports Java *and* currently has it enabled. The method returns `false` otherwise.

taintEnabled() This method returns `true` if the browser has data tainting enabled, as when the user has set the `NS_ENABLE_TAINT` environment variable. That variable allows JavaScript on one page to discover privileged information about other pages. Netscape removed the capability to perform data tainting in JavaScript 1.2 and replaced it with signed scripts. For information on JavaScript security, see http://developer.netscape.com/docs/manuals/js/client/jsguide/sec.htm.

Event Handlers

None. `Navigator` does not correspond to an HTML element.

25.21 The Number Object

The `Number` object accesses information about numbers. You do not need to create an object of this type to access the properties; instead, you can access `Number.propertyName`. The main reason for making a `Number` object is to call `toString()`, which lets you specify a radix.

Constructor

new Number(value) This constructs a `Number` object for the specified primitive value.

Properties

MAX_VALUE This property specifies the largest number representable in JavaScript.

MIN_VALUE This property specifies the smallest number representable in JavaScript.

NaN This property is the special not-a-number value. Use the global `isNaN` function to compare to it, since all comparisons with `NaN` return `false`, including testing `(Number.NaN == Number.NaN)`.

NEGATIVE_INFINITY This property represents negative overflow values. For instance, `Number.MAX_VALUE` times –2 returns this value. This value times any number is this value. Any number divided by this value is 0, except that an infinite value divided by another infinite value is `NaN`.

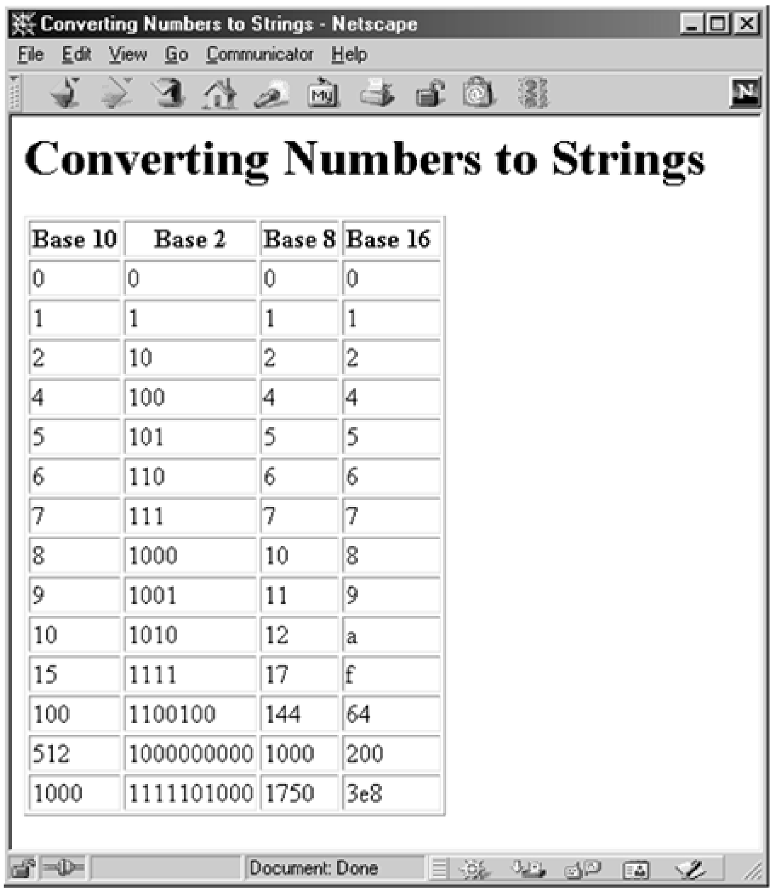
POSITIVE_INFINITY This property represents positive overflow values.

Methods

toString() This method is the same as calling `toString(10)`.

toString(radix) This method converts a number to a string in the specified radix. For example, Listing 25.3 creates a table of numbers in various radixes. Figure 25-4 shows the result in Netscape 4.7 on Windows 98.

Figure 25-4. `Number` object lets you print numbers in any radix.


Converting Numbers to Strings - Netscape
File Edit View Go Communicator Help
Document: Done

Converting Numbers to Strings

Base 10	Base 2	Base 8	Base 16
0	0	0	0
1	1	1	1
2	10	2	2
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	a
15	1111	17	f
100	1100100	144	64
512	1000000000	1000	200
1000	1111101000	1750	3e8

Listing 25.3 `NumberToString.html`

```
<!DOCTYPE HTML PUBLIC "--//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
  <TITLE>Converting Numbers to Strings</TITLE>

<SCRIPT TYPE="text/javascript">
<!--

function makeNumberTable(numberList, radixList) {
  document.write("<TABLE BORDER=1>\n<TR>");
  for(var i=0; i<radixList.length; i++) {
    document.write("<TH>Base " + radixList[i]);
  }
  var num;
  for(var i=0; i<numberList.length; i++) {
    document.write("\n<TR>");
    num = new Number(numberList[i]);
    for(var j=0; j<radixList.length; j++) {
      document.write("<TD>" + num.toString(radixList[j]));
     }
  }
  document.writeln("\n</TABLE>");
}

// -->
</SCRIPT>
</HEAD>

<BODY BGCOLOR="WHITE">
<H1>Converting Numbers to Strings</H1>

<SCRIPT TYPE="text/javascript">
<!--

var nums = new Array(0, 1, 2, 4, 5, 6, 7, 8, 9, 10, 15, 100,
                     512, 1000);
var radixes = new Array(10, 2, 8, 16);

makeNumberTable(nums, radixes);

// -->
</SCRIPT>

</BODY>
</HTML>
```

valueOf() This method returns the primitive number value associated with the `Number`.

Event Handlers

None. `Number` does not correspond to an HTML element.

25.22 The Object Object

`Object` is the object upon which all others are built. Properties and methods here are shared by *all* JavaScript objects.

Constructors

new Object() This constructor builds a generic `Object`.

new Object(primitiveValue) Depending on the argument type, this constructor creates a `Number`, `String`, `Boolean`, or `Function` "wrapper" object.

{prop1:val1, prop2:val2, … , propN:valN} You can also create objects by using "literal" notation.

Properties

constructor This read-only property refers to the JavaScript function that created the object instance.

prototype This property is not actually a property of `Object`, but rather of all constructor functions. It is mentioned here since it is used in a general-purpose way for all user-defined objects. See Section 24.3 (Mastering JavaScript Syntax) for more details.

Methods

assign(value) This method is called when an object of the type you define appears on the left side of an assignment operation. In most cases, the version of `assign` that you build calls `new` and fills in fields appropriately.

eval(javaScriptCode) This method takes an arbitrary string and evaluates the result.

toString() This method generates a string for the object. You define this in your classes to get custom string representations.

valueOf() This method returns the primitive value this `Object` represents if one is present. See the `Number` object (Section 25.21).

Methods

None.

Event Handlers

None. The `onblur`, `onfocus`, and `onchange` handlers are associated with the enclosing `Select` object, not with the `Option` object itself.

25.24 The Password Object

The `Password` object corresponds to HTML elements created through `<INPUT TYPE="PASSWORD" ...>`. `Password` objects are normally accessed through the `elements` array of the enclosing `Form` object or, if both it and the form are named, through `document.formName.passwordName`.

Properties

defaultValue This read-only property is the initial value as given in the `VALUE` attribute.

form This read-only property is the `Form` object containing the password field.

name This read-only property is the value of the `NAME` attribute.

type This read-only property contains the value `password`.

value This property gives the plain-text value of the password field. On a Unix system, a string of asterisks shows you the number of characters the user entered, but not the actual value.

Methods

blur() This method removes the keyboard focus from the element.

focus() This method gives the keyboard focus to the element.

select() This method highlights the text in the element. If the user types, the input replaces the existing text.

Event Handlers

onblur() This method is called when the password field loses the input focus. It is normally specified by means of the `onBlur` attribute.

onchange() This method is called when the password field loses the input focus after its value has changed. It is normally specified by means of the `onChange` attribute.

onfocus() This method is called when the password field gets the input focus. It is normally specified by means of the `onFocus` attribute.

onkeydown() This method is called when the user first presses any key in the password field. Returning `false` cancels the input of the character.

onkeypress() When the user first presses a key, this method is called after `onkeydown`. It is also called repeatedly when the key is held down, while `onkeydown` is not. Returning `false` cancels the input of the character.

onkeyup() This method is called when the user releases a key.

25.25 The Plugin Object

This section describes an installed plug-in, accessible through the `navigator.plugins` array. This array gives the plug-ins installed in the browser, not the objects in the current document that require plug-ins; for that information, see the `embeds` array of `Document`. `Plugin` is an unusual object in that it has normal properties and you can index it as an array (remember that JavaScript arrays are really just objects with number-valued property names). Each element of this array is a `MimeType` object. See Section 24.4 (Using JavaScript to Customize Web Pages) for an example of using `Plugin` and its associate `MimeType` objects.

Properties

description This property is a textual description of the plug-in, provided by the plug-in vendor. It is read-only.

filename This property gives the name of the file containing the code for the plug-in. It is read-only.

length This property specifies the number of `MimeType` objects in the array.

name This property gives a short name for the plug-in. You can use the name as an index into the `document.plugins` array.

Methods

None.

Event Handlers

None. A `Plugin` does not correspond to any HTML element.

25.26 The Radio Object

The `Radio` object corresponds to HTML input elements created inside a form through `<INPUT TYPE="RADIO" ...>`. `Radio` objects are normally accessed through the elements array of the enclosing `Form` object. If both the `Radio` object and the surrounding `Form` are named, you can access the `Radio` object through `document.formName.radioName`.

Properties

checked This is a `Boolean` property specifying whether the radio button is currently checked. It is read/write.

defaultChecked This is a `Boolean` specifying whether the radio button should be initially set. It is set through the `CHECKED` attribute and is read-only in JavaScript.

form This read-only property refers to the `Form` object containing the radio button.

name This property gives the name of the radio button as given in the `NAME` attribute. Remember that the whole point of radio buttons is that multiple entries share the same name but only one can be checked at any one time. The property is read-only.

type This property contains the string `radio`. Since all `Element` objects have this property, it can be used to differentiate among them when the `form.elements` array is looked at. It is read-only.

value This property gives the value that is sent with the name to the CGI program if the form is ubmitted when the radio button is checked. It is read/write.

Methods

blur() This method removes the keyboard focus from the radio button.

click() This method acts as though the radio button was clicked, but it does not trigger the `onClick` handler. Thus, calling this method is just like setting the `checked` property.

focus() This method gives the keyboard focus to the radio button.

Event Handlers

onblur() This method is called when the radio button loses the input focus. It is normally set through the `onBlur` attribute, as below.

```
<INPUT TYPE="RADIO" ...
       onBlur="doSomeAction()">
```

onclick() This method is called when the user clicks the radio button, but not when the `click` method is called programmatically. It is usually specified through the `onClick` attribute of the input element.

onfocus() This method is called when the radio button gains the input focus. It is normally set through the `onFocus` attribute.

25.27 The RegExp Object

Netscape 4.0 introduced the `RegExp` object to represent regular expressions and added support for it in the `String` object through the `match`, `replace`, `search`, and `split` methods.

Constructors

new RegExp("pattern") This constructor builds a regular expression. A regular expression is a string containing some special characters that Java uses to check for occurrences of certain patterns in strings. These characters are listed in Table 25.1, but the most important three are +, which means "match one or more occurrences of the previous character"; `*`, which means "match zero or more occurrences of the previous character"; and ?, which means "match zero or one occurrence of the previous character." In the absence of these special characters, characters in the regular expression are matched exactly against some comparison string. For example, the following regular expression means "a 'z', followed by one or more 'a's, followed by a 'b', followed by zero or more 'c's, followed by zero or one 'd', followed by an 'e'."

```
var re = new RegExp("za+bc*d?e");
```

`RegExp` has a `test` method that reads a string and returns `true` if and only if it contains the regular expression. Given the above definition of `re`, all of the following would return `true`:

```
re.test("zabcde");
re.test("xxxxxzabcdexxxxx");
re.test("zaaaabcde");
re.test("zaaaabde");
re.test("zaaaabe");
re.test("XXzaabcccccdeYY");
```

new RegExp("pattern", "g") This constructor builds a regular expression for global matches in a string. `String` has a `match` method that returns an array describing the matches against a particular string. If `g` is not specified, the first match is returned. With `g`, all matches are returned. For example, the first call to `exec` below returns an array containing `abc`, and the second returns an array containing `abc` and `abbbbc`.

```
var str = "abcabbbbcABCABBBBC";
var re1 = new RegExp("ab+c");
var re2 = new RegExp("ab+c", "g");
var result1 = str.match(re1);
var result2 = str.match(re2);
```

new RegExp("pattern", "i") This constructor builds a regular expression for case-insensitive matches.

new RegExp("pattern", "gi") This constructor builds a regular expression for global, case-insensitive matches. For example, the following builds an array containing `abc`, `abbbbc`, `ABC`, and `ABBBBC`.

```
var str = "abcabbbbcABCABBBBC";
var re = new RegExp("ab+c", "gi");
var result = str.match(re);
```

/pattern/ This notation is shorthand for `new RegExp("pattern")`. For example, the following two statements create equivalent regular expressions.

```
var re1 = /ab+c/;
var re2 = new RegExp("ab+c");
```

See further examples in Section 25.31 (The String Object).

/pattern/g This notation is shorthand for `new RegExp("pattern", "g")`.

/pattern/i This notation is shorthand for `new RegExp("pattern", "i")`.

/pattern/gi This notation is shorthand for `new RegExp("pattern", "gi")`.

Properties

These are properties of the global `RegExp` object, not of individual regular expressions. Thus, they are always accessed through `RegExp.propertyName`. The short version of the property names (`$_`, `$*`, etc.) are taken from the Perl language.

input

$_

If a regular expression's `exec` or `test` methods are called with no associated string, the expression uses the value of this global property. When an event handler for a `Text`, `TextArea`, `Select`, or `Link` object is invoked, this property is automatically filled in with the associated text. This property is read/write.

lastMatch

$&

This property gives the last matched substring. It is filled in after `exec` is called and is read-only.

lastParen

$+

This property gives the value of the last parenthesized match. It is filled in after `exec` is called and is read-only.

leftContext

$'

This property gives the left part of the string, up to but not including the most recent match. It is filled in after `exec` is called and is read-only.

multiline

$*

This property is a `Boolean` determining if matching should occur across line breaks. Set this property before calling `exec`; the `Textarea` event handler automatically sets this property to `true`. It is read/write.

rightcontext

$'

This property gives the right part of the string, starting after the most recent match. It is filled in after `exec` is called and is read-only.

$1

$2

…

$9

These properties give the values of the first nine parenthesized matches. They are filled in after `exec` is called and are read-only.

Methods

These methods belong to individual regular expression objects, not to the global `RegExp` object.

compile(pattern, flags) This method compiles a regular expression for faster execution.

exec(string) This method searches the string for the regular expression, filling in the fields of the RegExp object as described under Properties. As a shorthand, you can use someRegExp (string) instead of someRegExp.exec(string).

exec() This method is the same as exec(RegExp.input).

test(string) This method simply determines whether the string contains at least one occurrence of the regular expression, returning true or false. Using someRegExp.test(string) is equivalent to using string.search(someRegExp).

Event Handlers

None. RegExp does not correspond to an HTML element.

Special Patterns in Regular Expressions

The discussion of the RegExp constructors explained the purpose of the +, *, and ? characters. Table 25.1 gives a complete list of special characters and patterns. For clarity in the examples, we typically say something like "/ab+c/ matches abbbc", but note that /ab+c/ also matches XXabbbc and XXabbbc YYYY; i.e., the string only has to *contain* a match, not *be* a match.

Table 25.1. Special Regular Expression Patterns

Pattern	*Interpretation*
+	Match one or more occurrences of the previous character. For instance, /ab*c/ will match abc and abbbbbc, but not ac
*	Match zero or more occurrences of the previous character. For instance, /ab*c/ will match ac, abc, and abbbbbc.
?	Match zero or one occurrence of the previous character. For instance, /ab?c/ will match ac and abc, but not abbbbbc.
.	Match exactly one character. For instance, /a.c/ matches abc or aqc, but not ac or abbc. A newline does *not* match ".".
\	Treat the next character literally if it is a special character; treat it specially otherwise. For instance, /a**b/ matches ab, a*b, and a*****b.
(*pattern*)	Match *pattern,* but also "remember" the match for access through the $*N* properties of RegExp.
p1|*p2*	Match either *p1* or *p2.* For instance /foo|bar/ matches foot_ball and barstool.
{*n*}	Match exactly *n* occurrences of the previous character. For instance, /ab{3}c/ matches abbbc but not abbc or abbbbc.
{*n*,}	Match at least *n* occurrences of the previous character. For instance, /ab{3,}c/ matches abbbc and abbbbc but not abbc.
{*n1*,*n2*}	Match at least *n1* but no more than *n2* occurrences of the previous character.
$[c_1c_2...c_n]$	Match any one of the enclosed characters. For instance, /a[pl]*e/ matches ae, apple, and allpe. You can use dashes to represent series: e.g., [a-z] for any lowercase character, [0-7] for any digit from 0 to 7, and so forth.
$[\hat{}c_1c_2...c_n]$	Match any one character that is not part of the designated set. For instance, /a[^pl]*e/ matches ae and aqqxxe but not apple or allpe.
\b, \B	Match a word boundary (\b) or any one nonword-boundary character (\B). For example, /a\bc/ matches "a c" but not "abc"; /a\Bc/ matches "abc" but not "a c".
\w, \W	Match any word (\w) or nonword (\W) character. \w is equivalent to [A-Za-z0-9_], and \W is like [^A-Za-z0-9_].
\d, \D	Match any digit (\d) or nondigit (\D). Equivalent to [0-9] or [^0-9], respectively.
\f, \n, \r, \t, \v	Match formfeed, linefeed, carriage return, tab, and vertical tab, respectively.

`\s`, `\S`	Match any white space (`\s`) or non-white-space character (`\S`). `\s` is equivalent to `[\f\n\r\t\v]`, and `\S` is the same as `[^\f\n\r\t\v]`.
/xxx/	Match the character represented by the ASCII code *xxx.*

25.28 The Reset Object

The `Reset` object corresponds to buttons created through `<INPUT TYPE="RESET" ...>` in an HTML form. `Reset` objects are normally accessed through the elements array of the enclosing `Form` object. If both the button and the form are named, they can also be accessed through `document.formName.resetButtonName`.

Properties

form This property gives the `Form` object containing the button. It is read-only.

name If the button used the `NAME` attribute, this property retrieves it. The property is read-only.

type This property is always equal to `reset`. All `Element` objects contain this property, so it can be used to distinguish among the various types. It is read-only.

value This property gives the label of the button. It is read/write.

Methods

blur() This method removes the keyboard focus from the button.

click() This method acts as though the button was clicked, but without triggering the `onClick` handler. You can use the form's `reset` method instead of `click`.

focus() This method gives the keyboard focus to the button.

Event Handlers

onblur() This method is called when the button loses the input focus. It is normally set through the `onBlur` attribute, as below.

```
<INPUT TYPE="RESET" ...
       onBlur="doSomeAction()">
```

onclick() This method is called when the user clicks on the button, but not when the `click` method is called programmatically. It is normally set through the `onClick` attribute:

```
<INPUT TYPE="RESET" ...
       onClick="doSomeAction()">
```

If the method returns `false`, then the form is not actually reset. For example,

```
<INPUT TYPE="RESET" ...
       onClick="return(maybeReset())">
```

The same effect can be achieved by `onReset` handler on the form containing the button.

ondblclick() This method is called on the second click of a double click. The `onclick` handler, if any, will be called first. It is set by the `onDblClick` attribute. It is not supported on the Macintosh or in Netscape 6.

onfocus() This method is called when the button gains the input focus. It is normally set through the `onFocus` attribute.

25.29 The Screen Object

The `Screen` object, accessible through the global `screen` variable, contains information about the current screen's resolution and color.

Properties

availHeight This read-only property gives the height of the screen (in pixels), minus space occupied by semipermanent user interface elements such as the Windows 98 task bar.

availWidth This property gives the width of the screen (in pixels), minus space occupied by semipermanent user interface elements. It is read-only.

colorDepth This property specifies the number of simultaneous colors that can be displayed. It is read-only.

height This read-only property gives the height of the screen in pixels.

width This read-only property gives the width of the screen in pixels.

pixelDepth This property specifies the number of bits per pixel being used for color. It is read-only.

Methods

None.

Event Handlers

None. `Screen` does not correspond to an HTML element.

25.30 The Select Object

A `Select` object corresponds to an HTML element created through `<SELECT ...>`. It is normally accessed through the `elements` array of the enclosing `Form`. If the `Form` and `Select` objects both have names, you can also use `document.formName.selectName`.

Listing 25.4 shows a page that presents a pull-down menu of color choices. Choosing an entry changes the page's background color.

Listing 25.4 `SelectColor.html`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
  <TITLE>Changing the Background Color</TITLE>
<SCRIPT TYPE="text/javascript">
<!--

function setBackgroundColor() {
  var selection = document.colorForm.colorSelection;
  document.bgColor =
    selection.options[selection.selectedIndex].value;
}

// -->
</SCRIPT>
</HEAD>
```

```
<BODY BGCOLOR="WHITE">
<H1>Changing the Background Color</H1>

<FORM id="colorForm">
<SELECT id="colorSelection"
        onChange="setBackgroundColor()">
  <OPTION VALUE="#FFFFFF" SELECTED>White
  <OPTION VALUE="#C0C0C0">Gray
  <OPTION VALUE="#FF0000">Red
  <OPTION VALUE="#00FF00">Green
  <OPTION VALUE="#0000FF">Blue
</SELECT>

</FORM>

</BODY>
</HTML>
```

Properties

form This property refers to the `Form` object containing the selection element. It is read-only.

length This property specifies the number of `Option` elements contained in the selection. It is the same as `options.length` and is read-only.

name This property gives the name as specified through the `NAME` attribute. It is read-only.

options This property is an array of `Option` objects contained in the selection. You are permitted to add `Option` objects to the end of this array.

selectedIndex This property gives the index of the currently selected option. It will be `-1` if none is selected and will give the first selected index for `Select` elements that were created through `<SELECT ... MULTIPLE>`. It is read/write.

type This property contains either `select-one` or `select-multiple`, depending on whether the `MULTIPLE` attribute was included. It is read-only.

Methods

blur() This method removes the keyboard focus from the selection.

focus() This method gives the keyboard focus to the selection.

Event Handlers

onblur() This method is called when the selection loses the input focus. It is normally set through the `onBlur` attribute, as below.

```
<SELECT ... onBlur="doSomeAction()">
```

onchange() This method is called when the selection loses the input focus after the selected option has changed. See Listing 25.4 for an example of its use.

onfocus() This method is called when the selection gains the input focus. It is normally set through the `onFocus` attribute.

25.31 The String Object

`String` is an important datatype in JavaScript. It does not correspond directly to any particular HTML elements but is widely used.

Constructor

new String(value) This constructor builds a new `String`.

Properties

length This read-only property gives the number of characters in the string.

Methods

anchor(name) This method returns a copy of the current string, embedded between `<A id="name">` and `</A>`. For example,

```
"Chapter One".anchor("Ch1")
```

evaluates to

```
'<A id="Ch1">Chapter One</A>'
```

big() This method returns a copy of the string, embedded between `<BIG>` and `</BIG>`.

blink() This method returns a copy of the string, embedded between `<BLINK>` and `</BLINK>`.

bold() This method returns a copy of the string, embedded between `<B>` and `</B>`. For example,

```
"Wow".italics().bold()
```

evaluates to

```
"<B><I>Wow</I></B>"
```

charAt(index) This method returns a one-character string taken from the character at the specified location. Strings, like most datatypes in JavaScript, are zero indexed.

charCodeAt()

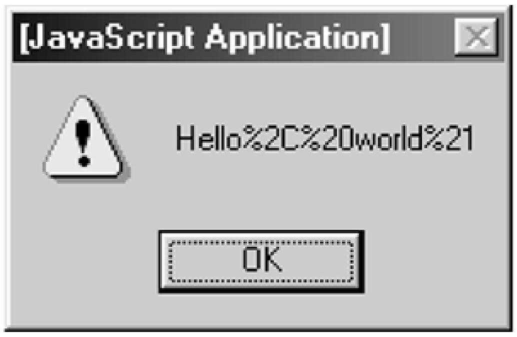
charCodeAt(index) The first method returns `charCodeAt(0)`. The second method returns the ISO Latin-1 number for the character at the designated location. The first 127 values correspond to ASCII values.

concat(suffixString) This method concatenates two strings. The following two forms are equivalent.

```
var newString = string1.concat(string2);
var newString = string1 + string2;
```

escape(string) The `escape` method is actually not a method of `String` but rather is a standard top-level function. However, because it is used for string manipulation, it is described here. It URL-encodes a string so that it can be attached to the query portion (`search` property) of a `Location` object. Note that this method replaces spaces with `%20`, not with `+`. For example, the following statement results in Figure 25-5.

Figure 25-5. The `escape` method is used to URL-encode.



Hello%2C%20world%21

See `unescape` for URL decoding.

```
alert(escape("Hello, world!"));
```

fixed() This method returns a copy of the string, embedded between `<TT>` and `</TT>`.

fontcolor(colorName) This method returns a copy of the string, embedded between `<FONT COLOR="colorName">` and `</FONT>`.

fontsize(size) This method returns a copy of the string, embedded between `<FONT SIZE=size>` and `</FONT>`.

fromCharCode(code0, code1, … , code*N*) This method creates a string composed of the designated ISO Latin-1 characters. It is not actually a method of individual string objects, but rather of the `String` constructor function itself. Thus, it is always called through `String.fromCharCode(...)`. For example, the following assigns the string `HELLO` to `helloString`.

```
var helloString =
  String.fromCharCode(72, 69, 76, 76, 79);
```

indexOf(substring)

indexOf(substring, startIndex)

If the specified substring is contained in the string, the first method returns the beginning index of the first match. Otherwise, `-1` is returned. For example, here is a `contains` predicate that returns `true` if and only if the second string is contained somewhere in the first.

```
function contains(string, possibleSubstring) {
  return(string.substring(possibleSubstring) != -1);
}
```

In the second method, if the specified substring is contained somewhere starting at or to the right of the specified starting point, the beginning index (relative to the whole string, not with respect to the starting point) of the first match is returned.

italics() This method returns a copy of the string, embedded between `<I>` and `</I>`.

lastIndexOf(substring)

lastIndexOf(substring, startIndex)

If the specified substring is contained in the string, the first method returns the beginning index of the last match. Otherwise, `-1` is returned. In the second method, if the specified substring is contained somewhere starting at or to the right of the specified starting point, then the beginning index of the last match is returned.

link(url) This method returns a copy of the string, embedded between `<A HREF="url">` and

`</A>`.

match(regExp) This method returns an array showing the matches of the `RegExp` argument in the string. For example, the following builds an array `result` containing the strings `abc`, `abbbbc`, `ABC`, and `ABBBBC`.

```
var str = "abcabbbbcABCABBBBC";
var re = /ab+c/gi;
var result = str.match(re);
```

Since the `g` in `re` means "find all" and the `i` means "case insensitive match", this match is interpreted as saying "find all occurrences of an 'a' or 'A' followed by one or more 'b's and/or 'B's followed by a 'c' or 'C'." See the `RegExp` object (Section 25.27) for more details.

replace(regExp, replacementString) This method returns a new string it formed by replacing the regular expression by the designated replacement string. All occurrences will be replaced if the regular expression includes the `g` (global) designation. For example, the following generates a `result` of `"We will use Java, Java, and Java"`.

```
var str = "We will use C, C++, and Java.";
var re = /C\+*/g;
var result = str.replace(re, "Java");
```

search(regExp) This method is invoked just like the `match` method but simply returns `true` or `false` depending on whether there was at least one match. If all you care about is whether the string appears, this method is faster than `match`.

slice(startIndex, endIndex) With a positive ending index, `slice` is just like `substring`. However, you can also supply a negative ending index, which is interpreted as an offset from the end of the string. Here are some examples.

```
var str = "0123456789";
var str2 = str.slice(1, 5);     //  "1234"
var str3 = str.substring(1, 5); //  "1234"
var str4 = str.slice(1, -2);    //  "1234567"
```

small() This method returns a copy of the string, embedded between `<SMALL>` and `</SMALL>`.

split() This method returns an array containing the string. Using `split` with a delimiter is much more useful.

split(delimChar) This method returns an array formed by breaking the string at each occurrence of the delimiter character. For instance, the following creates a three-element array containing the strings `foo`, `bar`, and `baz` (in that order).

```
var test = "foo,bar,baz".split(",");
```

If you use a space as the argument, `someString.split(" ")` returns an array of the strings that were separated by *any* number of white space characters (spaces, tabs, newlines). This method is the inverse of the `join` method of `Array`.

split(regExp) This variation splits on a regular expression. For example, the following creates a three-element array containing the strings `foo`, `bar`, and `baz` (in that order).

```
var str = "foo,bar,,,,,baz";
var re = /,+/;
var result = str.split(re);
```

split(separator, limit) This method extracts at most `limit` entries from the string. The separator can be a delimiter character or a `RegExp` object.

strike() This method returns a copy of the string, embedded between `<STRIKE>` and `</STRIKE>`.

sub() This method returns a copy of the string, embedded between `<SUB>` and `</SUB>`.

substr(startIndex, numChars) This method returns the substring of the current string that starts at `startIndex` and is `numChars` long.

substring(startIndex, endIndex) This method returns a new string taken from the characters from `startIndex` (inclusive) to `endIndex` (exclusive). For example, the following assigns `"is"` to the variable `test`.

```
var test = "this is a test".substring(5, 7);
```

sup() This method returns a copy of the string, embedded between `<SUP>` and `</SUP>`.

toLowerCase() This method returns a copy of the original string, converted to lower case.

toUpperCase() This method returns a copy of the original string, converted to upper case.

unescape(string) The `unescape` method is actually not a method of `String` but rather is a standard top-level function. However, since it is used for string manipulation, it is described here. It URL-decodes a string but has the unfortunate shortcoming that it does not map + to a space character.

Event Handlers

None. `String` does not correspond to an HTML element.

25.32 The Submit Object

The `Submit` object corresponds to buttons created through `<INPUT TYPE="SUBMIT" ...>` in an HTML form. `Submit` objects are normally accessed through the `elements` array of the enclosing `Form` object. If both the button and the form are named, they can also be accessed through `document.formName.submitButtonName`.

Properties

form This read-only property gives the `Form` object containing the button.

name If the button used the `NAME` attribute, this property retrieves it. The property is read-only.

type This property is always equal to `submit`. All `Element` objects contain this property, so it can be used to distinguish among the various types. It is read-only.

value This property gives the label of the button. It is read/write.

Methods

blur() This method removes the keyboard focus from the button.

click() This method acts as though the button was clicked, but it does not trigger the `onClick` handler. You can use the form's `submit` method instead of `click`.

focus() This method gives the keyboard focus to the button.

Event Handlers

onblur() This method is called when the button loses the input focus. It is normally set through the `onBlur` attribute, as below.

```
<INPUT TYPE="SUBMIT" ... onBlur="doSomeAction()">
```

onclick() This method is called when the user clicks on the button, but not when the `click` method is called programmatically. It is normally set through the `onClick` attribute.

```
<INPUT TYPE="SUBMIT" ... onClick="doSomeAction()">
```

If the method returns `false`, then the form is not actually submitted. For example,

```
<INPUT TYPE="SUBMIT" ... onClick="return(maybeSubmit())">
```

The same effect can be achieved by the `onSubmit` handler on the form containing the button.

ondblclick() This method is called on the second click of a double click. The `onclick` handler, if any, is called first. It is set by the `onDblClick` attribute. It is not supported on the Macintosh or in Netscape 6.

onfocus() This method is called when the button gains the input focus. It is normally set through the `onFocus` attribute.

25.33 The Text Object

The `Text` object corresponds to HTML elements created through `<INPUT TYPE="TEXT" ...>`. `Text` objects are normally accessed through the `elements` array of the enclosing `Form` object or, if both it and the form are named, through `document.formName.textfieldName`.

Properties

defaultValue This read-only property is the initial value as given in the `VALUE` attribute.

form This read-only property is the `Form` object containing the password field.

name This read-only property is the value of the `NAME` attribute.

type This read-only property contains the value `text`.

value This property gives the current text contained in the textfield. It is read/write.

Methods

blur() This method removes the keyboard focus from the textfield.

focus() This method gives the keyboard focus to the textfield.

select() This method highlights the text in the element. If the user types, the input replaces the existing text.

Event Handlers

onblur() This method is called when the textfield loses the input focus. It is normally specified by means of the `onBlur` attribute.

onchange() This method is called when the textfield loses the input focus after its value has been changed by the user. It is *not* called each time the user presses a key. It is normally specified by means of the `onChange` attribute.

onfocus() This method is called when the textfield gets the input focus. It is normally specified by means of the `onFocus` attribute.

onkeydown() This method is called when the user first presses any key in the textfield. Returning `false` cancels the input of the character, so it can be used to restrict the type of text that can be placed in the field.

onkeypress() When the user first presses a key, this method is called after `onkeydown`. It is also called repeatedly when the key is held down, whereas `onkeydown` is not. Returning `false` cancels the input of the character.

onkeyup() This method is called when the user releases a key.

25.34 The Textarea Object

The `Textarea` object corresponds to HTML elements created through `<TEXTAREA ...>` and `</TEXTAREA>`. `Textarea` objects are normally accessed through the `elements` array of the enclosing `Form` object or, if both it and the form are named, through `document.formName.textareaName`.

Properties

defaultValue This property is the initial value as given by the text that appears between `<TEXTAREA>` and`</TEXTAREA>`. It is read-only.

form This property is the `Form` object containing the text area. It is read-only.

name This property is the value of the `NAME` attribute. It is read-only.

type This read-only property contains the value `textarea`.

value This property gives the current text contained in the text area. It is read/write; however, there is no way to determine the number of rows or columns used by the textfield, so it may be difficult to insert properly formatted text.

Methods

blur() This method removes the keyboard focus from the text area.

focus() This method gives the keyboard focus to the text area.

select() This method highlights the text in the element. If the user types, the input replaces the existing text.

Event Handlers

onblur() This method is called when the text area loses the input focus. It is normally specified by means of the `onBlur` attribute.

onchange() This method is called when the text area loses the input focus after its value has been changed by the user. It is normally specified by means of the `onChange` attribute.

onfocus() This method is called when the text area gets the input focus. It is normally specified by means of the `onFocus` attribute.

onkeydown() This method is called when the user first presses any key in the text area. Returning `false` cancels the input of the character, so it can be used to restrict the type of text that can be placed in the field.

onkeypress() When the user first presses a key, this method is called after `onkeydown`. It is also called repeatedly when the key is held down, whereas `onkeydown` is not. Returning `false` cancels the input of the character.

onkeyup() This method is called when the user releases a key.

25.35 The Window Object

The `window` object describes a browser window or frame. The current window is available through the `window` reference, but you can omit that prefix when accessing its properties and methods. So, for instance, you can refer to the `Document` associated with the current window through `window.document` or simply by `document`. Similarly, to transfer the current window to a new page, you can set the `window.location` property or simply set `location`.

Properties

closed This `Boolean` property specifies whether the window has been closed. It is read-only.

defaultStatus This string specifies the default string that should appear in the status line. It is read/write.

document This property refers to the `Document` object contained in the window. See Section 25.5 for details on `Document`. It is read-only.

frames This array of `Window` objects refers to the entries contained in the frames of the current document.

history This property gives the `History` object associated with the window. It is read-only.

innerHeight This property gives the inner size of the browser window. It is read/write; changing it resizes the window.

innerWidth This property gives the inner width of the browser window. See Section 24.4 (Using JavaScript to Customize Web Pages) for an example of its use. It is read/write; changing it resizes the window.

java This property is a reference to the `JavaPackage` object that is the top of the `java.*` package hierarchy. For example, you can call `java.lang.Math.random()` to use Java's random number generator instead of JavaScript's or use `java.lang.System.out.println` to send output to the Java Console. It is read-only.

length This read-only property is the same as `frames.length`.

location This property refers to the `Location` object for this window, which is the *requested* URL. Due to redirection, this may be different from the *actual* URL. For that, see `document.URL`. This is a read/write variable; setting it changes the window to display a new document.

locationbar Signed scripts in Netscape can set the `visible` property of `locationbar` to hide or show the location bar. Legal values are `true` (or `1`) and `false` (or `0`).

Math This property is a reference to the `Math` object.

menubar Signed scripts in Netscape can set the `visible` property of `menubar` to hide or show the menu bar. Legal values are `true` (or `1`) and `false` (or `0`).

name When a window is created, you can specify a name. This property retrieves it. It is read/write.

navigator This property is a reference to the `Navigator` object. It is read-only.

netscape This property is a reference to the `JavaPackage` object corresponding to the `netscape.*` package. It is read-only.

opener This property is a reference to the `Window` object, if any, that created this window. It is read/write.

outerHeight This property gives the outside height of the browser window. It is read/write; changing it resizes the window. Windows smaller than 100 x 100 pixels can only be created from secure (signed) scripts.

outerWidth This property gives the outside width of the browser window. It is read/write.

Packages This property is a reference to the `JavaPackage` object that represents the top of the package hierarchy. It is read-only.

pageXOffset This property gives the x offset of the page with respect to the window's content area. It is useful when you are deciding how much to scroll. It is read-only; use `scrollTo` or `scrollBy` to change it.

pageYOffset This property gives the y offset of the page with respect to the window's content area. It is useful when you are deciding how much to scroll. It is read-only; use `scrollTo` or `scrollBy` to change it.

parent This property gives the parent window or frame. For a top-level window `win`, `win.parent` is simply `win`. It is read-only.

personalbar Signed scripts in Netscape can set the `visible` property of `personalbar` to hide or show the personal (directories) bar. Legal values are `true` (or `1`) and `false` (or `0`).

screen This property is actually a global variable, not a property of `Window`. However, it is mentioned here since most seemingly global variables (`document`, `Math`, etc.) are really properties of the current window. See Section 25.29.

scrollbars Signed scripts can set the `visible` property of `scrollbars` to hide or show scrollbars. Legal values are `true` (or `1`) and `false` (or `0`).

self This property is a reference to the window itself and is synonymous with `window`. It is read-only.

status This string represents the contents of the status bar. It is read/write. An ill-advised fad in the early JavaScript days was to put scrolling messages in the status bar through this property.

statusbar Signed scripts can set the `visible` property of `personalbar` to hide or show the status bar. Legal values are `true` (or `1`) and `false` (or `0`).

sun This property is a reference to the `JavaPackage` object that is the top of the `sun.*` package hierarchy. It is read-only.

tags This property can be used by JavaScript style sheets to set style sheet properties. See Section 5.2 (Using External and Local Style Sheets) for an example.

toolbar Signed scripts can set the `visible` property of `toolbar` to hide or show the Netscape toolbar. Legal values are `true` (or `1`) and `false` (or `0`).

top This property refers to the top-level window containing the current one. It is the same as the current one if frames are not being used. It is read-only.

window This property is a reference to the window itself and is synonymous with `self`. It is read-only.

Methods

alert(message) This method displays a message in a pop-up dialog box.

back() This method switches the window to the previous entry in the history list, as if the user clicked on the Back button.

blur() This method removes the keyboard focus from the current window, usually by putting the window in the background.

captureEvents(eventType) This method sets the window to capture all events of the specified type.

clearInterval(intervalID) The `setInterval` method returns an ID. Supplying the ID to `clearInterval` kills the interval routine.

clearTimeout(timeoutID) The `setTimeout` method returns an ID. Supplying the ID to `clearTimeout` kills the timeout routine.

close() This method closes a window. You aren't supposed to be able to close windows that you didn't create, but there are some bugs that let you do this anyhow.

confirm(questionString) This methods pops up a dialog box displaying your question. If the user presses OK, `true` is returned. Cancel results in `false` being returned. You can embed `\n` (newline) characters in the string to split the question across multiple lines.

enableExternalCapture()

disableExternalCapture()

Signed scripts can capture events in external pages. These two methods enable and disable this capability, respectively.

find()

find(searchString)

find(searchString, caseSensitivityFlag, backwardFlag)

The `find` method searches for strings in the current document. If you omit the search string, the Find dialog box pops up to let the user enter a string. Alternatively, you can supply a search string and optionally two boolean flags. These flags determine if a case-sensitive match should be used (`true` for the second parameter) or if the file should be searched from the end going backward (`true` for the third parameter). The methods return `true` if the string was found, `false` otherwise.

focus() This method gives the specified window the keyboard focus. On most platforms, getting the focus brings the window to the front.

forward() This method moves the window forward in the history list.

handleEvent(event) If `captureEvents` has been set, then events of the specified type get passed to `handleEvent`.

home() This method switches the window to the home document, as if the user clicked on the Home button.

moveBy(x, y) This method moves the window on the screen by the specified number of pixels.

In Netscape, moving the window off the screen requires a signed script, but even so, this method can easily be abused.

moveTo(x, y) This method moves the window to an absolute location on the screen. In Netscape, moving the window off the screen requires a signed script, but even so, this method can easily be abused.

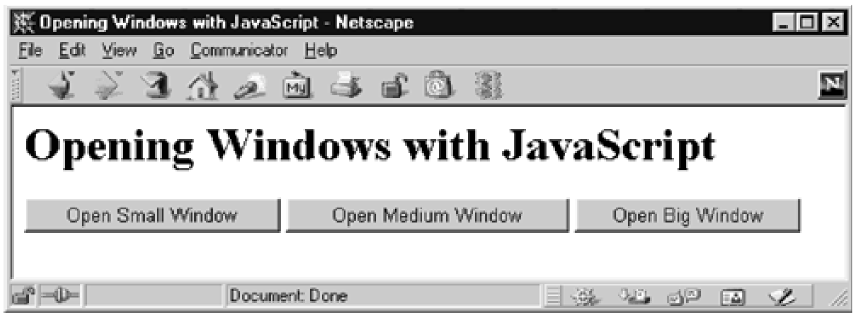
open(url, name)

open(url, name, features)

open(url, name, features, replaceFlag)

This method can be used to find an existing window or to open a new one. To avoid confusion with `document.open`, it is common practice to use `window.open(...)` instead of simply `open(...)`. Specifying an empty string for the URL opens a blank window. You can then write into it using that window's `document` property. The name can be used for other JavaScript methods or as the `TARGET` attribute in `A`, `BASE`, `AREA`, and `FORM` elements. The `replaceFlag` specifies whether the new window replaces the old entry in the history list (`true`) or whether a new entry should be created (`false`). The `features` string gives comma-separated *feature*=*value* entries (with *no* spaces!) and determines what browser features the window should include (all if `features` is omitted). Using *feature* is shorthand for *feature*`=yes`. If the `features` entry is omitted, *all* standard features are used, regardless of what the user has set in the preferences. Legal feature names are summarized in Table 25.2. Listing 25.5 (at the end of this section) gives a couple of examples of using `window.open`, with results shown in Figures25-6 through 25-9. Also see the window-creation example in Section 24.10.

Figure 25-6. OpenWindows.html before any buttons are clicked.


Opening Windows with JavaScript - Netscape

Opening Windows with JavaScript

Figure 25-7. Using `window.open` with a width and height but no other features results in a bare-bones, undecorated browser window. [© 2001 Netscape Communications Corp. Used with permission. All rights reserved.]

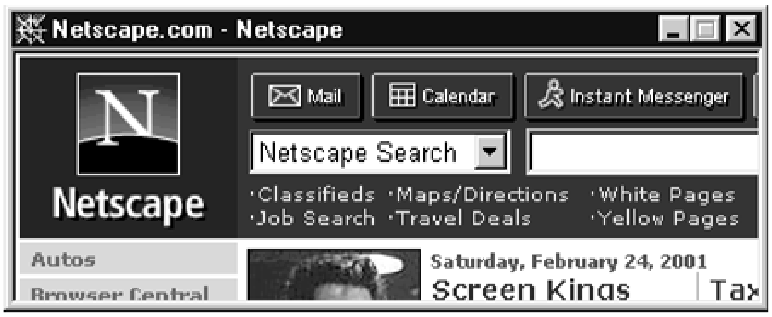
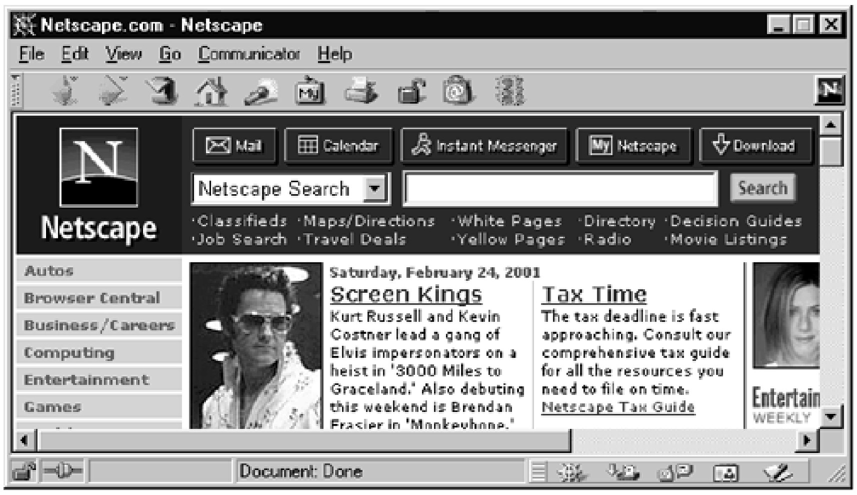


Figure 25-8. This version creates a moderately decorated browser window. [© 2001 Netscape Communications Corp. Used with permission. All rights reserved.]

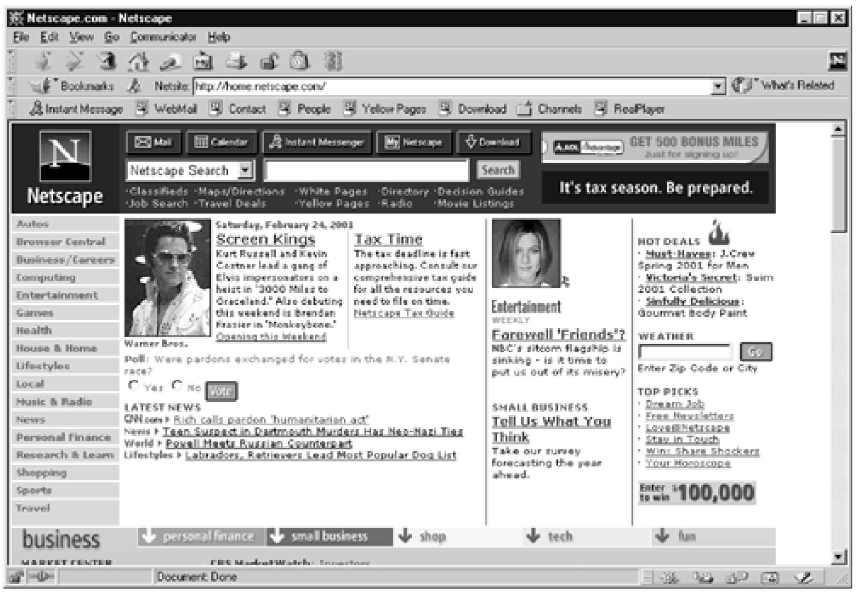


Saturday, February 24, 2001

Screen Kings
Kurt Russell and Kevin Costner lead a gang of Elvis impersonators on a heist in '3000 Miles to Graceland.' Also debuting this weekend is Brendan Frasier in 'Monkeybone.'

Tax Time
The tax deadline is fast approaching. Consult our comprehensive tax guide for all the resources you need to file on time.
Netscape Tax Guide

Figure 25-9. By specifying enough features, you can make a fully loaded browser window. [© 2001 Netscape Communications Corp. Used with permission. All rights reserved.]



Saturday, February 24, 2001

Screen Kings
Kurt Russell and Kevin Costner lead a gang of Elvis impersonators on a heist in '3000 Miles to Graceland.' Also debuting this weekend is Brendan Frasier in 'Monkeybone.'
Opening this Weekend

Warner Bros.

Tax Time
The tax deadline is fast approaching. Consult our comprehensive tax guide for all the resources you need to file on time.
Netscape Tax Guide

Poll: Were pardons exchanged for votes in the N.Y. Senate race?

LATEST NEWS
CNN.com ▸ Rich calls pardon 'humanitarian act'
News ▸ Teen Suspect in Dartmouth Murders Has Neo-Nazi Ties
World ▸ Powell Meets Russian Counterpart
Lifestyles ▸ Labradors, Retrievers Lead Most Popular Dog List

Farewell 'Friends'?
NBC's sitcom flagship is sinking - is it time to put us out of its misery?

SMALL BUSINESS
Tell Us What You Think
Take our survey forecasting the year ahead.

HOT DEALS
· Must Haves: J.Crew Spring 2001 for Men
· Victoria's Secret: Swim 2001 Collection
· Sinfully Delicious: Gourmet Body Paint

WEATHER
Enter Zip Code or City

TOP PICKS
· Dream Job
· Free Newsletters
· Love@Netscape
· Stay in Touch
· Win: Share Shockers
· Your Horoscope

MARKET CENTER
CBS MarketWatch: Investors

Core Warning

Feature lists should not have any blank spaces, or they will not be parsed properly.

Table 25.2. Features available in the open method

Feature	*Legal Values*	*Meaning*
`alwaysLowered`	yes/no	Should window always be below others? Available only with signed scripts.
`alwaysRaised`	yes/no	Should window always be above others? Available only with signed scripts.
`dependent`	yes/no	Is window a child of creating window? That is, should it close when parent window closes and be omitted from window's task bar?
`directories`	yes/no	Show the directory buttons ("What's Cool?", etc.)?
`hotkeys`	yes/no	Disable most hotkeys?
`innerHeight`	pixels	Sets the content area height. Unix users should note that `.Xdefaults` entries can override this value.
`innerWidth`	pixels	Sets the content area width. Unix users should note that `.Xdefaults` entries can override this value.

`location`	yes/no	Show the current location textfield?
`menubar`	yes/no	Show the menu bar?
`outerHeight`	pixels	Sets the outside window height.
`outerWidth`	pixels	Sets the outside window width.
`resizable`	yes/no	Let the user stretch the window?
`screenX`	pixels	Sets the location of the left side of the window.
`screenY`	pixels	Sets the location of the top side of the window.
`scrollbars`	yes/no	Use scrollbars if necessary?
`status`	yes/no	Show the status line at the bottom?
`titlebar`	yes/no	Include title bar? Disabling it requires a signed script.
`toolbar`	yes/no	Show the toolbar that contains back/forward/home/stop buttons?
`z-lock`	yes/no	Prevent window from being raised/lowered? Available only in signed scripts.

print() This method prints the document as though by the Print button. Note that the method brings up a dialog box; there is (fortunately) no way to print documents without user confirmation.

prompt(message)

prompt(message, defaultText)

These methods pop up a dialog box with a simple textfield, returning the value entered when it is closed. You can supply the initial string as the second argument if desired.

releaseEvents(eventType) This method tells JavaScript to stop capturing the specified event type.

resizeBy(x, y) This method lets you change the size of the browser window by the specified amount.

resizeTo(x, y) This method changes the *outer* width and height to the specified size.

routeEvent(event) This method is used by `handleEvent` to send the event along the normal event-handling path.

scrollBy(x, y) This method scrolls by the specified number of pixels.

scrollTo(x, y) This method scrolls the document so that the upper-left corner of the window shows the specified location of the document.

setInterval(code, delay) This method *repeatedly* executes a string representing code until the window is destroyed or `clearInterval` is called. See `setTimeout`.

setTimeout(code, delay) Given a string specifying JavaScript code and a delay time in milliseconds, this method executes the code after the specified delay unless `clearTimeout` is called with the `setTimeout` return value in the meantime. Note that `setTimeout` *returns* immediately; it just doesn't *execute* the code until later.

stop() This method stops the current document download, as if through the Stop button.

Event Handlers

onblur() This is the method called when the window loses the keyboard focus. It is normally set through the `onBlur` attribute of `BODY` or `FRAMESET`, as in the following example.

```
<BODY onBlur="alert('We will miss you')">
...
```

```
</BLUR>
```

A more useful application might be to halt certain processing when the user leaves the window, restarting it through `onfocus`.

ondragdrop() This Netscape method is called when a file or shortcut is dragged onto the Navigator window and released. If the method returns `false`, the normal action of loading the file is canceled. It is set by the `onDragDrop` attribute.

onerror() This method is called when a JavaScript error occurs. This error handler has no associated HTML attribute, so you have to set it directly, as in the example below.

```
function reportError() {
  return(!confirm("An error occurred.\n" +
                  "Please report it to\n" +
                  "gates@microsoft.com.\n\n" +
                  "See more details?"));
}

onerror = reportError;
```

Returning `true` prevents the browser from also reporting the error, so in the preceding example, users only see the standard error report if they click OK in the confirmation dialog box. Setting the value of `onerror` to `null` suppresses error reporting altogether.

onfocus() This method is called when the window gets the keyboard focus. It is normally set through the `onFocus` attribute of `BODY` or `FRAMESET`, as in the following example.

```
<FRAMESET ROWS=...
          onFocus="alert('Welcome back')">
...
</FRAMESET>
```

onload() This method is called when the browser finishes loading the page. It is normally set through the `onLoad` attribute of `BODY` or `FRAMESET`. It is useful for recording that the document finished loading so that functions that depend on various pieces of the document will operate correctly.

onmove() This method is called *after* the window is moved (either by the user or programmatically). It is set through the `onMove` attribute, as follows:

```
<BODY onMove="alert('Hey, move me back!')" ...>
...
</BODY>
```

onresize() This method is called when the user or JavaScript code stretches or shrinks the window. It is normally set by the `onResize` attribute.

onunload() This method is called when the user leaves the page. It is normally set through the `onUnload` attribute of `BODY` or `FRAMESET`.

An Example of the open Method

Listing 25.5 gives examples of several different features used in the `window.open` method. Figures 25-6 through 25-9 show the results.

Listing 25.5 `OpenWindows.html`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
```

```
<HTML>
<HEAD>
  <TITLE>Opening Windows with JavaScript</TITLE>

<SCRIPT TYPE="text/javascript">
<!--
function openSmallWindow() {
  window.open("http://home.netscape.com/",
              "smallWindow",
              "width=375,height=125");
}

function openMediumWindow() {
  window.open("http://home.netscape.com/",
              "mediumWindow",
              "width=550,height=225," +
              "menubar,scrollbars,status,toolbar");
}

function openBigWindow() {
  window.open("http://home.netscape.com/",
              "bigWindow",
              "width=850,height=450," +
              "directories,location,menubar," +
              "scrollbars,status,toolbar");
}

// -->
</SCRIPT>
</HEAD>

<BODY>
<H1>Opening Windows with JavaScript</H1>

<FORM>
  <INPUT TYPE="BUTTON" VALUE="Open Small Window"
         onClick="openSmallWindow()">
  <INPUT TYPE="BUTTON" VALUE="Open Medium Window"
         onClick="openMediumWindow()">
  <INPUT TYPE="BUTTON" VALUE="Open Big Window"
         onClick="openBigWindow()">
</FORM>

</BODY>
</HTML>
```

25.36 Summary

Whew! You finished the book. Congratulations. We hope you are now comfortable with the basics of HTML, Java, Servlets, and JavaScript, so you can develop Web applications from beginning to end. Now you can go back and focus on specific areas that you skimmed earlier. You're familiar with standard HTML; maybe now you should go back and look at style sheets or layers. You have a handle on Java; maybe this is the time to try out threading, RMI, or JDBC. Perhaps you've used CGI but haven't seen what servlets can buy you. Or maybe you now want to see how JavaScript regular expressions can help you. But don't worry, you don't have to be an expert in everything; few people are. No matter where you choose to concentrate, a solid base will serve you well.

But before you move on, relax and take a day off. Oh, and tell your boss that we said you deserve a raise. Hmm, 20% ought to do it, don't you think?

Have fun!

CONTENTS