JavaScript Reference

December 12, 1997

Printed in USA
97 96 95 10 9 8 7 6 5 4 3 2 1

Netscape Communications Corporation 501 East Middlefield Road, Mountain View, CA 94043

Contents

What's in this Reference

This reference is organized around the functionality of the JavaScript language. Sometimes you already know the name of an object or method, but don't know precisely where to look for it. This chapter contains tables of links to aid in this situation.

Table 1, "Operators," is a list of all JavaScript operators, grouped by type of operator.

Table 2, "Statements," is an alphabetical list of all JavaScript statements.

Table 3, "Objects with their methods and properties," is an alphabetical list of all of JavaScript's predefined classes and objects. The predefined methods and properties for each object are listed.

Table 4, "Methods," is an alphabetical list of all predefined methods, regardless of the object to which they belong. The second column indicates the object with which the method is associated. There are separate entries for methods of the same name used in different objects. Each method name links to the method in the indicated object.

Similarly, Table 5, "Properties," is an alphabetical list of all predefined properties, regardless of the object to which they belong. The second column indicates the object with which the property is associated.

Table 6, "Global functions," is an alphabetical list of JavaScript's global functions. These are functions which aren't associated with any object.

Table 7, "Event handlers," is an alphabetical list of all JavaScript event handlers.

Key to the versions

If there is an entry in both the Client Version and the Server Version columns for a single construct, that construct is part of the core language. Otherwise, it is defined only for the client or for the server, as indicated.

The version number indicates the versions of Netscape Navigator (Nav), LiveWire (LW), or the Netscape servers (Svr), such as Enterprise Server and FastTrack Server), for which the construct is defined.

- A plus sign after a version number (as in Nav 3+) indicates that the construct is defined for that version and all later versions (In the case of server constructs, LW 1+ indicates the construct was defined for LiveWire 1.0 and continues to be defined in Netscape 3.x servers.)
- If there is no plus sign (Nav 3) or there is a range (Nav 2-3), the construct was only defined for the named releases.
- A construct that has existed for more than one release may have had changes between releases. For this information, see the entry for the construct.

Table 1 Operators

Operator Category	Operator	Client version	Server version
Arithmetic Operators	+	Nav 2	LW 1
	++	Nav 2	LW 1
	-	Nav 2	LW 1
	--	Nav 2	LW 1
	*	Nav 2	LW 1
	/	Nav 2	LW 1
	%	Nav 2	LW 1
String Operators	+	Nav 2	LW 1
	+=	Nav 2	LW 1
Logical Operators	&&	Nav 2	LW 1
	||	Nav 2	LW 1
	!	Nav 2	LW 1

Table 1 Operators (Continued)

Operator Category	Operator	Client version	Server version
Bitwise Operators	&	Nav 2	LW 1
	^	Nav 2	LW 1
	|	Nav 2	LW 1
	~	Nav 2	LW 1
	<<	Nav 2	LW 1
	>>	Nav 2	LW 1
	>>>	Nav 2	LW 1
Assignment Operators	=	Nav 2	LW 1
	+=	Nav 2	LW 1
	-=	Nav 2	LW 1
	*=	Nav 2	LW 1
	/=	Nav 2	LW 1
	%=	Nav 2	LW 1
	&=	Nav 2	LW 1
	^=	Nav 2	LW 1
	|=	Nav 2	LW 1
	<<=	Nav 2	LW 1
	>>=	Nav 2	LW 1
	>>>=	Nav 2	LW 1
Comparison Operators	==	Nav 2	LW 1
	!=	Nav 2	LW 1
	>	Nav 2	LW 1
	>=	Nav 2	LW 1
	<	Nav 2	LW 1
	<=	Nav 2	LW 1

Table 1 Operators (Continued)

Operator Category	Operator	Client version	Server version
Special Operators	`?:`	Nav 2	LW 1
	`,`	Nav 2	LW 1
	`delete`	Nav 2	LW 1
	`new`	Nav 2	LW 1
	`this`	Nav 2	LW 1
	`typeof`	Nav 3	LW 1
	`void`	Nav 3	LW 1

Table 2 Statements

Statement	Client version	Server version
`break`	Nav 2+	LW 1+
`comment`	Nav 2+	LW 1+
`continue`	Nav 2+	LW 1+
`delete`	Nav 4	Svr 3
`do...while`	Nav 4	Svr 3
`export`	Nav 4	Svr 3
`for`	Nav 2+	LW 1+
`for...in`	Nav 2+	LW 1+
`function`	Nav 2+	LW 1+
`if...else`	Nav 2+	LW 1+
`import`	Nav 4	Svr 3
`labeled`	Nav 4	Svr 3
`return`	Nav 2+	LW 1+
`switch`	Nav 4	Svr 3

Table 2 Statements (Continued)

Statement	Client version	Server version
`var`	Nav 2+	LW 1+
`while`	Nav 2+	LW 1+
`with`	Nav 2+	LW 1+

Table 3 Objects with their methods and properties

Object	Client version	Server version	Methods	Properties	Event handlers
`Anchor`	Nav 2+				
`Applet`	Nav 3+				
`Area` (see `Link`)	Nav 3+				

`Array`	Nav 3+ (2 as non-object)	LW 1+	`concat`
			`join`
			`pop`
			`push`
			`reverse`
			`shift`
			`slice`
			`splice`
			`sort`
			`toString`
			`unshift`
				`index`
				`input`
				`length`
				`prototype`	

`blob`		LW 1+	`blobImage` `blobLink`		
`Boolean`	Nav 3+	LW 1+	`toString`	`prototype`	

`Button`	Nav 2+		`blur`
			`click`
			`focus`
			`handleEvent`
				`form`
				`name`
				`type`
				`value`
					`onBlur`
					`onClick`
					`onFocus`
					`onMouseDown`
					`onMouseUp`

Table 3 Objects with their methods and properties (Continued)

Object	Client version	Server version	Methods	Properties	Event handlers

`Checkbox`	Nav 2+		`blur`
			`click`
			`focus`
			`handleEvent`
				`checked`
				`defaultChecked`
				`form`
				`name`
				`type`
				`value`
					`onBlur`
					`onClick`
					`onFocus`

`client`		LW 1+	`destroy` `expiration`		

`Connection`		Svr 3	`beginTransaction`
			`commitTransaction`
			`connected`
			`cursor`
			`execute`
			`majorErrorCode`
			`majorErrorMessage`
			`minorErrorCode`
			`minorErrorMessage`
			`release`
			`rollbackTransaction`
			`SQLTable`
			`storedProc`
			`toString`
				`prototype`	

`Cursor`		LW 1+	`close`
			`columnName`
			`columns`
			`deleteRow`
			`insertRow`
			`next`
			`updateRow`
				`cursorColumn`
				`prototype`	

Table 3 Objects with their methods and properties (Continued)

Object	Client version	Server version	Methods	Properties	Event handlers

`database`		LW 1+	`beginTransaction`
			`commitTransaction`
			`connect`
			`connected`
			`cursor`
			`disconnect`
			`execute`
			`majorErrorCode`
			`majorErrorMessage`
			`minorErrorCode`
			`minorErrorMessage`
			`rollbackTransaction`
			`SQLTable`
			`storedProc`
			`storedProcArgs`
			`toString`
				`prototype`	

`Date`	Nav 2+	LW 1+	`getDate`
			`getDay`
			`getHours`
			`getMinutes`
			`getMonth`
			`getSeconds`
			`getTime`
			`getTimezoneOffset`
			`getYear`
			`parse`
			`setDate`
			`setHours`
			`setMinutes`
			`setMonth`
			`setSeconds`
			`setTime`
			`setYear`
			`toGMTString`
			`toLocaleString`
			`UTC`
				`prototype`	

Table 3 Objects with their methods and properties (Continued)

Object	Client version	Server version	Methods	Properties	Event handlers
DbPool		Svr 3	DbPool		
			connect		
			connected		
			connection		
			disconnect		
			majorErrorCode		
			majorErrorMessage		
			minorErrorCode		
			minorErrorMessage		
			storedProcArgs		
			toString		
document	Nav 2+		captureEvents	alinkColor	onClick
			close	anchors	onDblClick
			getSelection	applets	onKeyDown
			handleEvent	bgColor	onKeyPress
			open	cookie	onKeyUp
			releaseEvents	domain	onMouseDown
			routeEvent	embeds	onMouseUp
			write	fgColor	
			writeln	*formName*	
				forms	
				images	
				lastModified	
				layers	
				linkColor	
				links	
				plugins	
				referrer	
				title	
				URL	
				vlinkColor	

Table 3 Objects with their methods and properties (Continued)

Object	Client version	Server version	Methods	Properties	Event handlers

`event`	Nav 4			`data`
				`height`
				`layerX`
				`layerY`
				`modifiers`
				`pageX`
				`pageY`
				`screenX`
				`screenY`
				`target`
				`type`
				`which`
				`width`	

`File`		LW 1+	`byteToString`
			`clearError`
			`close`
			`eof`
			`error`
			`exists`
			`flush`
			`getLength`
			`getPosition`
			`open`
			`read`
			`readByte`
			`readln`
			`setPosition`
			`stringToByte`
			`write`
			`writeByte`
			`writeln`
				`prototype`	

`FileUpload`	Nav 2+		`blur`
			`focus`
			`handleEvent`
			`select`
				`form`
				`name`
				`type`
				`value`
					`onBlur`
					`onChange`
					`onFocus`

Table 3 Objects with their methods and properties (Continued)

Object	Client version	Server version	Methods	Properties	Event handlers

`Form`	Nav 2+		`handleEvent`
			`reset`
			`submit`
				`action`
				`elements`
				`encoding`
				`length`
				`method`
				`name`
				`target`
					`onReset`
					`onSubmit`

`Frame` (see `Window`)	Nav 2+				

`Function`	Nav 3+	LW 1+	`toString`	`arguments`
				`arity`
				`caller`
				`prototype`	

`Hidden`	Nav 2+			`form`
				`name`
				`type`
				`value`	

`History`	Nav 2+		`back`
			`forward`
			`go`
				`current`
				`length`
				`next`
				`previous`	

`Image`	Nav 3+		`handleEvent`	`border`
				`complete`
				`height`
				`hspace`
				`lowsrc`
				`name`
				`prototype`
				`src`
				`vspace`
				`width`
					`onAbort`
					`onError`
					`onKeyDown`
					`onKeyPress`
					`onKeyUp`
					`onLoad`

Table 3 Objects with their methods and properties (Continued)

Object	Client version	Server version	Methods	Properties	Event handlers
`Layer`	Nav 4		`captureEvents`	`above`	`onBlur`
			`handleEvent`	`background`	`onFocus`
			`load`	`bgColor`	`onLoad`
			`moveAbove`	`below`	`onMouseOut`
			`moveBelow`	`clip.bottom`	`onMouseOver`
			`moveBy`	`clip.height`	
			`moveTo`	`clip.left`	
			`moveToAbsolute`	`clip.right`	
			`releaseEvents`	`clip.top`	
			`resizeBy`	`clip.width`	
			`resizeTo`	`document`	
			`routeEvent`	`left`	
				`name`	
				`pageX`	
				`pageY`	
				`parentLayer`	
				`siblingAbove`	
				`siblingBelow`	
				`src`	
				`top`	
				`visibility`	
				`zIndex`	
`Link`	Nav 2+		`handleEvent`	`hash`	`onClick`
				`host`	`onDblClick`
				`hostname`	`onKeyDown`
				`href`	`onKeyPress`
				`pathname`	`onKeyUp`
				`port`	`onMouseDown`
				`protocol`	`onMouseOut`
				`search`	`onMouseUp`
				`target`	`onMouseOver`
				`text`	
`Location`	Nav 2+		`reload`	`hash`	
			`replace`	`host`	
				`hostname`	
				`href`	
				`pathname`	
				`port`	
				`protocol`	
				`search`	

Table 3 Objects with their methods and properties (Continued)

Object	Client version	Server version	Methods	Properties	Event handlers

`Lock`		Svr 3	`isValid`
			`lock`
			`unlock`		

`Math`	Nav 2+	LW 1+	`abs`
			`acos`
			`asin`
			`atan`
			`atan2`
			`ceil`
			`cos`
			`exp`
			`floor`
			`log`
			`max`
			`min`
			`pow`
			`random`
			`round`
			`sin`
			`sqrt`
			`tan`
				`E`
				`LN10`
				`LN2`
				`LOG10E`
				`LOG2E`
				`PI`
				`SQRT1_2`
				`SQRT2`	

`MimeType`	Nav 3+			`description`
				`enabledPlugin`
				`suffixes`
				`type`	

`navigator`	Nav 2+		`javaEnabled`
			`plugins.refresh`
			`preference`
			`taintEnabled`
				`appCodeName`
				`appName`
				`appVersion`
				`language`
				`mimeTypes`
				`platform`
				`plugins`
				`userAgent`	

Table 3 Objects with their methods and properties (Continued)

Object	Client version	Server version	Methods	Properties	Event handlers

`Number`	Nav 3+	LW 1+		`MAX_VALUE`
				`MIN_VALUE`
				`NaN`
				`NEGATIVE_INFINITY`
				`POSITIVE_INFINITY`
				`prototype`	

`Object`	Nav 2+	LW 1+	`eval`
			`toString`
			`unwatch`
			`valueOf`
			`watch`
				`constructor`
				`prototype`	

`Option`	Nav 2+			`defaultSelected`
				`selected`
				`text`
				`value`	

`Password`	Nav 2+		`blur`
			`focus`
			`handleEvent`
			`select`
				`defaultValue`
				`form`
				`name`
				`type`
				`value`
					`onBlur`
					`onFocus`

`Plugin`	Nav 3+			`description`
				`filename`
				`length`
				`name`	

`project`		LW 1+	`lock` `unlock`		

`Radio`	Nav 2+		`blur`
			`click`
			`focus`
			`handleEvent`
				`checked`
				`defaultChecked`
				`form`
				`name`
				`type`
				`value`
					`onBlur`
					`onClick`
					`onFocus`

Table 3 Objects with their methods and properties (Continued)

Object	Client version	Server version	Methods	Properties	Event handlers

`RegExp`	Nav 4	Svr 3	`compile`
			`exec`
			`test`
				`$1, ..., $9`
				`global`
				`ignoreCase`
				`input` ($_)
				`lastIndex`
				`lastMatch` ($&)
				`lastParen` ($+)
				`leftContext` ($‘)
				`multiline` ($*)
				`rightContext` ($’)
				`source`	

`request`		LW 1+		`agent`
				`imageX`
				`imageY`
				`inputName`
				`ip`
				`method`
				`protocol`	

`Reset`	Nav 2+		`blur`
			`click`
			`focus`
			`handleEvent`
				`form`
				`name`
				`type`
				`value`
					`onBlur`
					`onClick`
					`onFocus`

`Resultset`		Svr 3	`close`
			`columnName`
			`columns`
			`next`
				`prototype`	

`screen`	Nav 4			`availHeight`
				`availWidth`
				`colorDepth`
				`height`
				`pixelDepth`
				`width`	

`Select`	Nav 2+		`blur`
			`focus`
			`handleEvent`
				`form`
				`length`
				`name`
				`options`
				`selectedIndex`
				`type`
					`onBlur`
					`onChange`
					`onFocus`

Table 3 Objects with their methods and properties (Continued)

Object	Client version	Server version	Methods	Properties	Event handlers

`SendMail`		Svr 3	`errorCode`
			`errorMessage`
			`send`
				`Bcc`
				`Body`
				`Cc`
				`Errorsto`
				`From`
				`Organization`
				`Replyto`
				`Smtpserver`
				`Subject`
				`To`	

`server`		LW 1+	`lock`
			`unlock`
				`host`
				`hostname`
				`port`
				`protocol`	

`Stproc`		Svr 3	`close`
			`outParamCount`
			`outParameters`
			`resultSet`
			`returnValue`
				`prototype`	

Table 3 Objects with their methods and properties (Continued)

Object	Client version	Server version	Methods	Properties	Event handlers

`String`	Nav 2+	LW 1+	`anchor`
			`big`
			`blink`
			`bold`
			`charAt`
			`charCodeAt`
			`concat`
			`fixed`
			`fontcolor`
			`fontsize`
			`fromCharCode`
			`indexOf`
			`italics`
			`lastIndexOf`
			`link`
			`match`
			`replace`
			`search`
			`slice`
			`small`
			`split`
			`strike`
			`sub`
			`substr`
			`substring`
			`sup`
			`toLowerCase`
			`toUpperCase`
				`length`
				`prototype`	

`Submit`	Nav 2+		`blur`
			`click`
			`focus`
			`handleEvent`
				`form`
				`name`
				`type`
				`value`
					`onBlur`
					`onClick`
					`onFocus`

`Text`	Nav 2+		`blur`
			`focus`
			`handleEvent`
			`select`
				`defaultValue`
				`form`
				`name`
				`type`
				`value`
					`onBlur`
					`onChange`
					`onFocus`
					`onSelect`

Table 3 Objects with their methods and properties (Continued)

Object	Client version	Server version	Methods	Properties	Event handlers
`Textarea`	Nav 2+		`blur`	`defaultValue`	`onBlur`
			`focus`	`form`	`onChange`
			`handleEvent`	`name`	`onFocus`
			`select`	`type`	`onKeyDown`
				`value`	`onKeyPress`
					`onKeyUp`
					`onSelect`
`Window`	Nav 2+		`alert`	`closed`	`onBlur`
			`back`	`defaultStatus`	`onDragDrop`
			`blur`	`document`	`onError`
			`captureEvents`	`frames`	`onFocus`
			`clearInterval`	`history`	`onLoad`
			`clearTimeout`	`innerHeight`	`onMove`
			`close`	`innerWidth`	`onResize`
			`confirm`	`length`	`onUnload`
			`disableExternalCapture`	`location`	
			`enableExternalCapture`	`locationbar`	
			`find`	`menubar`	
			`focus`	`name`	
			`forward`	`opener`	
			`handleEvent`	`outerHeight`	
			`home`	`outerWidth`	
			`moveBy`	`pageXOffset`	
			`moveTo`	`pageYOffset`	
			`open`	`parent`	
			`print`	`personalbar`	
			`prompt`	`scrollbars`	
			`releaseEvents`	`self`	
			`resizeBy`	`status`	
			`resizeTo`	`statusbar`	
			`routeEvent`	`toolbar`	
			`scroll`	`top`	
			`scrollBy`	`window`	
			`scrollTo`		
			`setInterval`		
			`setTimeout`		
			`stop`		

Table 4 Methods

Method	Of object	Client version	Server Version
abs	Math	Nav 2+	LW 1+
acos	Math	Nav 2+	LW 1+
alert	Window	Nav 2+	
anchor	String	Nav 2+	LW 1+
asin	Math	Nav 2+	LW 1+
atan	Math	Nav 2+	LW 1+
atan2	Math	Nav 2+	LW 1+
back	History	Nav 2+	
back	Window	Nav 4	
beginTransaction	Connection		Svr 3
beginTransaction	database		LW 1+
big	String	Nav 2+	LW 1+
blink	String	Nav 2+	LW 1+
blobImage	blob		LW 1+
blobLink	blob		LW 1+
blur	Button	Nav 2+	
blur	Checkbox	Nav 2+	
blur	FileUpload	Nav 2+	
blur	Password	Nav 2+	
blur	Radio	Nav 2+	
blur	Reset	Nav 2+	
blur	Select	Nav 2+	
blur	Submit	Nav 2+	
blur	Text	Nav 2+	
blur	Textarea	Nav 2+	

Table 4 Methods (Continued)

Method	Of object	Client version	Server Version
`blur`	`Window`	Nav 3+	
`bold`	`String`	Nav 2+	LW 1+
`byteToString`	`File`		LW 1+
`captureEvents`	`document`	Nav 4	
`captureEvents`	`Layer`	Nav 4	
`captureEvents`	`Window`	Nav 4	
`ceil`	`Math`	Nav 2+	LW 1+
`charAt`	`String`	Nav 2+	LW 1+
`charCodeAt`	`String`	Nav 4	Svr 3
`clearError`	`File`		LW 1+
`clearInterval`	`Window`	Nav 4	
`clearTimeout`	`Window`	Nav 2+	
`click`	`Button`	Nav 2+	
`click`	`Checkbox`	Nav 2+	
`click`	`Radio`	Nav 2+	
`click`	`Reset`	Nav 2+	
`click`	`Submit`	Nav 2+	
`close`	`Cursor`		LW 1+
`close`	`document`	Nav 2+	
`close`	`File`		LW 1+
`close`	`Resultset`		Svr 3
`close`	`Stproc`		Svr 3
`close`	`Window`	Nav 2+	
`columnName`	`Cursor`		LW 1+
`columnName`	`Resultset`		Svr 3
`columns`	`Cursor`		LW 1+

Table 4 Methods (Continued)

Method	Of object	Client version	Server Version
columns	Resultset		Svr 3
commitTransaction	Connection		Svr 3
commitTransaction	database		LW 1+
compile	RegExp	Nav 4	Svr 3
concat	Array	Nav 4	Svr 3
concat	String	Nav 4	Svr 3
confirm	Window	Nav 2+	
connect	database		LW 1+
connect	DbPool		Svr 3
connected	Connection		Svr 3
connected	database		LW 1+
connected	DbPool		Svr 3
connection	DbPool		Svr 3
cos	Math	Nav 2+	LW 1+
cursor	Connection		Svr 3
cursor	database		LW 1+
DbPool	DbPool		Svr 3
deleteRow	Cursor		LW 1+
destroy	client		LW 1+
disableExternalCapture	Window	Nav 4	
disconnect	database		LW 1+
disconnect	DbPool		Svr 3
enableExternalCapture	Window	Nav 4	
eof	File		LW 1+
error	File		LW 1+
errorCode	SendMail		Svr 3

Table 4 Methods (Continued)

Method	Of object	Client version	Server Version
`errorMessage`	`SendMail`		Svr 3
`eval`	`Object`	Nav 3	LW 1+
`exec`	`RegExp`	Nav 4	Svr 3
`execute`	`Connection`		Svr 3
`execute`	`database`		LW 1+
`exists`	`File`		LW 1+
`exp`	`Math`	Nav 2+	LW 1+
`expiration`	`client`		LW 1+
`find`	`Window`	Nav 4	
`fixed`	`String`	Nav 2+	LW 1+
`floor`	`Math`	Nav 2+	LW 1+
`flush`	`File`		LW 1+
`focus`	`Button`	Nav 2+	
`focus`	`Checkbox`	Nav 2+	
`focus`	`FileUpload`	Nav 2+	
`focus`	`Password`	Nav 2+	
`focus`	`Radio`	Nav 2+	
`focus`	`Reset`	Nav 2+	
`focus`	`Select`	Nav 2+	
`focus`	`Submit`	Nav 2+	
`focus`	`Text`	Nav 2+	
`focus`	`Textarea`	Nav 2+	
`focus`	`Window`	Nav 3+	
`fontcolor`	`String`	Nav 2+	LW 1+
`fontsize`	`String`	Nav 2+	LW 1+
`forward`	`History`	Nav 2+	

Table 4 Methods (Continued)

Method	Of object	Client version	Server Version
forward	Window	Nav 4	
fromCharCode	String	Nav 4	Svr 3
getDate	Date	Nav 2+	LW 1+
getDay	Date	Nav 2+	LW 1+
getHours	Date	Nav 2+	LW 1+
getLength	File		LW 1+
getMinutes	Date	Nav 2+	LW 1+
getMonth	Date	Nav 2+	LW 1+
getPosition	File		LW 1+
getSeconds	Date	Nav 2+	LW 1+
getSelection	document	Nav 4	
getTime	Date	Nav 2+	LW 1+
getTimezoneOffset	Date	Nav 2+	LW 1+
getYear	Date	Nav 2+	LW 1+
go	History	Nav 2+	
handleEvent	Button	Nav 4	
handleEvent	Checkbox	Nav 4	
handleEvent	document	Nav 4	
handleEvent	FileUpload	Nav 4	
handleEvent	Form	Nav 4	
handleEvent	Image	Nav 4	
handleEvent	Layer	Nav 4	
handleEvent	Link	Nav 4	
handleEvent	Password	Nav 4	
handleEvent	Radio	Nav 4	
handleEvent	Reset	Nav 4	

Table 4 Methods (Continued)

Method	Of object	Client version	Server Version
`handleEvent`	`Select`	Nav 4	
`handleEvent`	`Submit`	Nav 4	
`handleEvent`	`Text`	Nav 4	
`handleEvent`	`Textarea`	Nav 4	
`handleEvent`	`Window`	Nav 4	
`home`	`Window`	Nav 4	
`indexOf`	`String`	Nav 2+	LW 1+
`insertRow`	`Cursor`		LW 1+
`isValid`	`Lock`		Svr 3
`italics`	`String`	Nav 2+	LW 1+
`javaEnabled`	`navigator`	Nav 3+	
`join`	`Array`	Nav 3+	LW 1+
`lastIndexOf`	`String`	Nav 2+	LW 1+
`link`	`String`	Nav 2+	LW 1+
`load`	`Layer`	Nav 4	
`lock`	`Lock`		Svr 3
`lock`	`project`		LW 1+
`lock`	`server`		LW 1+
`log`	`Math`	Nav 2+	LW 1+
`majorErrorCode`	`Connection`		Svr 3
`majorErrorCode`	`database`		LW 1+
`majorErrorCode`	`DbPool`		Svr 3
`majorErrorMessage`	`Connection`		Svr 3
`majorErrorMessage`	`database`		LW 1+
`majorErrorMessage`	`DbPool`		Svr 3
`match`	`String`	Nav 4	Svr 3

Table 4 Methods (Continued)

Method	Of object	Client version	Server Version
max	Math	Nav 2+	LW 1+
min	Math	Nav 2+	LW 1+
minorErrorCode	Connection		Svr 3
minorErrorCode	database		LW 1+
minorErrorCode	DbPool		Svr 3
minorErrorMessage	Connection		Svr 3
minorErrorMessage	database		LW 1+
minorErrorMessage	DbPool		Svr 3
moveAbove	Layer	Nav 4	
moveBelow	Layer	Nav 4	
moveBy	Layer	Nav 4	
moveBy	Window	Nav 4	
moveTo	Layer	Nav 4	
moveTo	Window	Nav 4	
moveToAbsolute	Layer	Nav 4	
next	Cursor		LW 1+
next	Resultset		Svr 3
open	document	Nav 2+	
open	File		LW 1+
open	Window	Nav 2+	
outParamCount	Stproc		Svr 3
outParameters	Stproc		Svr 3
parse	Date	Nav 2+	LW 1+
plugins.refresh	navigator	Nav 3+	
pop	Array	Nav 4	Svr 3
pow	Math	Nav 2+	LW 1+

Table 4 Methods (Continued)

Method	Of object	Client version	Server Version
preference	navigator	Nav 4	
print	Window	Nav 4	
prompt	Window	Nav 2+	
push	Array	Nav 4	Svr 3
random	Math	Nav 2+	LW 1+
read	File		LW 1+
readByte	File		LW 1+
readln	File		LW 1+
refresh	navigator.plugins	Nav 3+	
release	Connection		Svr 3
releaseEvents	document	Nav 4	
releaseEvents	Layer	Nav 4	
releaseEvents	Window	Nav 4	
reload	Location	Nav 3+	
replace	Location	Nav 3+	
replace	String	Nav 4	Svr 3
reset	Form	Nav 3+	
resizeBy	Layer	Nav 4	
resizeBy	Window	Nav 4	
resizeTo	Layer	Nav 4	
resizeTo	Window	Nav 4	
resultSet	Stproc		Svr 3
returnValue	Stproc		Svr 3
reverse	Array	Nav 3+	LW 1+
rollbackTransaction	Connection		Svr 3
rollbackTransaction	database		LW 1+

Table 4 Methods (Continued)

Method	Of object	Client version	Server Version
`round`	`Math`	Nav 2+	LW 1+
`routeEvent`	`document`	Nav 4	
`routeEvent`	`Layer`	Nav 4	
`routeEvent`	`Window`	Nav 4	
`scroll`	`Window`	Nav 2-3	
`scrollBy`	`Window`	Nav 4	
`scrollTo`	`Window`	Nav 4	
`search`	`String`	Nav 4	Svr 3
`select`	`FileUpload`	Nav 2+	
`select`	`Password`	Nav 2+	
`select`	`Text`	Nav 2+	
`select`	`Textarea`	Nav 2+	
`send`	`SendMail`		Svr 3
`setDate`	`Date`	Nav 2+	LW 1+
`setHours`	`Date`	Nav 2+	LW 1+
`setInterval`	`Window`	Nav 4	
`setMinutes`	`Date`	Nav 2+	LW 1+
`setMonth`	`Date`	Nav 2+	LW 1+
`setPosition`	`File`		LW 1+
`setSeconds`	`Date`	Nav 2+	LW 1+
`setTime`	`Date`	Nav 2+	LW 1+
`setTimeout`	`Window`	Nav 2+	
`setYear`	`Date`	Nav 2+	LW 1+
`shift`	`Array`	Nav 4	Svr 3
`sin`	`Math`	Nav 2+	LW 1+
`slice`	`Array`	Nav 4	Svr 3

Table 4 Methods (Continued)

Method	Of object	Client version	Server Version
`slice`	`String`	Nav 4	Svr 3
`small`	`String`	Nav 2+	LW 1+
`sort`	`Array`	Nav 3+	LW 1+
`splice`	`Array`	Nav 4	Svr 3
`split`	`String`	Nav 3+	LW 1+
`SQLTable`	`Connection`		Svr 3
`SQLTable`	`database`		LW 1+
`sqrt`	`Math`	Nav 2+	LW 1+
`stop`	`Window`	Nav 4	
`storedProc`	`Connection`		Svr 3
`storedProc`	`database`		Svr 3
`storedProcArgs`	`database`		Svr 3
`storedProcArgs`	`DbPool`		Svr 3
`strike`	`String`	Nav 2+	LW 1+
`stringToByte`	`File`		LW 1+
`sub`	`String`	Nav 2+	LW 1+
`submit`	`Form`	Nav 2+	
`substr`	`String`	Nav 4	Svr 3
`substring`	`String`	Nav 2+	LW 1+
`sup`	`String`	Nav 2+	LW 1+
`taintEnabled`	`navigator`	Nav 3	LW 1
`tan`	`Math`	Nav 2+	LW 1+
`test`	`RegExp`	Nav 4	Svr 3
`toGMTString`	`Date`	Nav 2+	LW 1+
`toLocaleString`	`Date`	Nav 2+	LW 1+
`toLowerCase`	`String`	Nav 2+	LW 1+

Table 4 Methods (Continued)

Method	Of object	Client version	Server Version
`toString`	`Array`	Nav 3+	LW 1+
`toString`	`Boolean`	Nav 3+	LW 1+
`toString`	`Connection`		Svr 3
`toString`	`database`		LW 1+
`toString`	`DbPool`		Svr 3
`toString`	`Number`	Nav 3+	LW 1+
`toString`	`Object`	Nav 2+	LW 1+
`toUpperCase`	`String`	Nav 2+	LW 1+
`unlock`	`Lock`		Svr 3
`unlock`	`project`		LW 1+
`unlock`	`server`		LW 1+
`unshift`	`Array`	Nav 4	Svr 3
`unwatch`	`Object`	Nav 4	Svr 3
`updateRow`	`Cursor`		LW 1+
`UTC`	`Date`	Nav 2+	LW 1+
`valueOf`	`Object`	Nav 3+	LW 1+
`watch`	`Object`	Nav 4	Svr 3
`write`	`document`	Nav 2+	
`write`	`File`		LW 1+
`writeByte`	`File`		LW 1+
`writeln`	`document`	Nav 2+	
`writeln`	`File`		LW 1+

Table 5 Properties

Property	Of object	Client version	Server version
$1, ..., $9	RegExp	Nav 4	Svr 3
$_	RegExp	Nav 4	Svr 3
$*	RegExp	Nav 4	Svr 3
$&	RegExp	Nav 4	Svr 3
$+	RegExp	Nav 4	Svr 3
$‘	RegExp	Nav 4	Svr 3
$’	RegExp	Nav 4	Svr 3
above	Layer	Nav 4	
action	Form	Nav 2+	
agent	request		LW 1+
alinkColor	document	Nav 2+	
anchors	document	Nav 2+	
appCodeName	navigator	Nav 2+	
applets	document	Nav 3+	
appName	navigator	Nav 2+	
appVersion	navigator	Nav 2+	
arguments	Function	Nav 3+	LW 1+
arity	Function	Nav 4	LW 1+
background	Layer	Nav 4	
below	Layer	Nav 4	
bgColor	document	Nav 2+	
bgColor	Layer	Nav 4	
border	Image	Nav 3+	
caller	Function	Nav 3+	LW 1+
checked	Checkbox	Nav 2+	

Table 5 Properties (Continued)

Property	Of object	Client version	Server version
`checked`	`Radio`	Nav 2+	
`clip.bottom`	`Layer`	Nav 4	
`clip.height`	`Layer`	Nav 4	
`clip.left`	`Layer`	Nav 4	
`clip.right`	`Layer`	Nav 4	
`clip.top`	`Layer`	Nav 4	
`clip.width`	`Layer`	Nav 4	
`closed`	`Window`	Nav 3+	
`colorDepth`	`screen`	Nav 4	
`complete`	`Image`	Nav 3+	
`constructor`	`Object`	Nav 3+	LW 1+
`cookie`	`document`	Nav 2+	
`current`	`History`	Nav 3+	
`cursorColumn`	`Cursor`		LW 1+
`data`	`event`	Nav 4	
`defaultChecked`	`Checkbox`	Nav 2+	
`defaultChecked`	`Radio`	Nav 2+	
`defaultStatus`	`Window`	Nav 2+	
`defaultSelected`	`Option`	Nav 3+	
`defaultValue`	`Password`	Nav 2+	
`defaultValue`	`Text`	Nav 2+	
`defaultValue`	`Textarea`	Nav 2+	
`description`	`MimeType`	Nav 3+	
`description`	`Plugin`	Nav 3+	
`document`	`Layer`	Nav 4	
`document`	`Window`	Nav 2+	

Table 5 Properties (Continued)

Property	Of object	Client version	Server version
`domain`	`document`	Nav 3+	
`E`	`Math`	Nav 2+	LW 1+
`elements`	`Form`	Nav 2+	
`embeds`	`document`	Nav 3+	
`enabledPlugin`	`MimeType`	Nav 3+	
`encoding`	`Form`	Nav 2+	
`fgColor`	`document`	Nav 2+	
`filename`	`Plugin`	Nav 3+	
`form`	`Button`	Nav 2+	
`form`	`Checkbox`	Nav 2+	
`form`	`FileUpload`	Nav 2+	
`form`	`Hidden`	Nav 2+	
`form`	`Password`	Nav 2+	
`form`	`Radio`	Nav 2+	
`form`	`Reset`	Nav 2+	
`form`	`Select`	Nav 2+	
`form`	`Submit`	Nav 2+	
`form`	`Text`	Nav 2+	
`form`	`Textarea`	Nav 2+	
`formName`	`document`	Nav 3+	
`forms`	`document`	Nav 3+	
`frames`	`Window`	Nav 2+	
`global`	`RegExp`	Nav 4	Svr 3
`hash`	`Link`	Nav 2+	
`hash`	`Location`	Nav 2+	
`height`	`event`	Nav 4	

Table 5 Properties (Continued)

Property	Of object	Client version	Server version
`height`	`Image`	Nav 3+	
`height`	`screen`	Nav 4	
`history`	`Window`	Nav 2+	
`host`	`Link`	Nav 2+	
`host`	`Location`	Nav 2+	
`host`	`server`		LW 1+
`hostname`	`Link`	Nav 2+	
`hostname`	`Location`	Nav 2+	
`hostname`	`server`		LW 1+
`href`	`Link`	Nav 2+	
`href`	`Location`	Nav 2+	
`hspace`	`Image`	Nav 3+	
`ignoreCase`	`RegExp`	Nav 4	Svr 3
`images`	`document`	Nav 3+	
`imageX`	`request`		LW 1+
`imageY`	`request`		LW 1+
`index`	`Array`	Nav 4	Svr 3
`input`	`Array`	Nav 4	Svr 3
`innerHeight`	`Window`	Nav 4	
`innerWidth`	`Window`	Nav 4	
`input`	`RegExp`	Nav 4	Svr 3
`inputName`	`request`		LW 1+
`ip`	`request`		LW 1+
`language`	`navigator`	Nav 4	
`lastIndex`	`RegExp`	Nav 4	Svr 3
`lastMatch`	`RegExp`	Nav 4	Svr 3

Table 5 Properties (Continued)

Property	Of object	Client version	Server version
`lastModified`	`document`	Nav 2+	
`lastParen`	`RegExp`	Nav 4	Svr 3
`layerX`	`event`	Nav 4	
`layerY`	`event`	Nav 4	
`layers`	`document`	Nav 4	
`left`	`Layer`	Nav 4	
`leftContext`	`RegExp`	Nav 4	Svr 3
`length`	`Array`	Nav 3+	LW 1+
`length`	`Form`	Nav 2+	
`length`	`History`	Nav 2+	
`length`	`Plugin`	Nav 3+	
`length`	`Select`	Nav 2+	
`length`	`String`	Nav 2+	LW 1+
`length`	`Window`	Nav 2+	
`linkColor`	`document`	Nav 2+	
`links`	`document`	Nav 2+	
`location`	`Window`	Nav 2+	
`locationbar`	`Window`	Nav 4	
`LN10`	`Math`	Nav 2+	LW 1+
`LN2`	`Math`	Nav 2+	LW 1+
`LOG10E`	`Math`	Nav 2+	LW 1+
`LOG2E`	`Math`	Nav 2+	LW 1+
`lowsrc`	`Image`	Nav 3+	
`MAX_VALUE`	`Number`	Nav 3+	LW 1+
`menubar`	`Window`	Nav 4	
`method`	`Form`	Nav 2+	

Table 5 Properties (Continued)

Property	Of object	Client version	Server version
`method`	`request`		LW 1+
`mimeTypes`	`navigator`	Nav 3+	
`modifiers`	`event`	Nav 4	
`MIN_VALUE`	`Number`	Nav 3+	LW 1+
`multiline`	`RegExp`	Nav 4	Svr 3
`name`	`Button`	Nav 2+	
`name`	`Checkbox`	Nav 2+	
`name`	`FileUpload`	Nav 2+	
`name`	`Form`	Nav 2+	
`name`	`Hidden`	Nav 2+	
`name`	`Image`	Nav 3+	
`name`	`Layer`	Nav 4	
`name`	`Password`	Nav 2+	
`name`	`Plugin`	Nav 3+	
`name`	`Radio`	Nav 2+	
`name`	`Reset`	Nav 2+	
`name`	`Select`	Nav 2+	
`name`	`Submit`	Nav 2+	
`name`	`Text`	Nav 2+	
`name`	`Textarea`	Nav 2+	
`name`	`Window`	Nav 2+	
`NaN`	`Number`	Nav 3+	LW 1+
`NEGATIVE_INFINITY`	`Number`	Nav 3+	LW 1+
`next`	`History`	Nav 3+	
`opener`	`Window`	Nav 3+	
`options`	`Select`	Nav 2+	

Table 5 Properties (Continued)

Property	Of object	Client version	Server version
outerHeight	Window	Nav 4	
outerWidth	Window	Nav 4	
pageX	event	Nav 4	
pageX	Layer	Nav 4	
pageXOffset	Window	Nav 4	
pageY	event	Nav 4	
pageY	Layer	Nav 4	
pageYOffset	Window	Nav 4	
parent	Window	Nav 2+	
parentLayer	Layer	Nav 4	
pathname	Link	Nav 2+	
pathname	Location	Nav 2+	
personalbar	Window	Nav 4	
PI	Math	Nav 2+	LW 1+
pixelDepth	screen	Nav 4	
platform	navigator	Nav 4	
plugins	document	Nav 3+	
plugins	navigator	Nav 3+	
port	Link	Nav 2+	
port	Location	Nav 2+	
port	server		LW 1+
POSITIVE_INFINITY	Number	Nav 3+	LW 1+
previous	History	Nav 3+	
protocol	Link	Nav 2+	
protocol	Location	Nav 2+	
protocol	request		LW 1+

Table 5 Properties (Continued)

Property	Of object	Client version	Server version
`protocol`	`server`		LW 1+
`prototype`	`Array`	Nav 3+	LW 1+
`prototype`	`Boolean`	Nav 3+	LW 1+
`prototype`	`Connection`		Svr 3
`prototype`	`Cursor`	Nav 3+	LW 1+
`prototype`	`database`		LW 1+
`prototype`	`Date`	Nav 3+	LW 1+
`prototype`	`DbPool`		Svr 3
`prototype`	`File`		LW 1+
`prototype`	`Function`	Nav 3+	LW 1+
`prototype`	`Image`	Nav 3+	LW 1+
`prototype`	`Number`	Nav 3+	LW 1+
`prototype`	`Object`	Nav 3+	LW 1+
`prototype`	`Resultset`		Svr 3
`prototype`	`SendMail`		Svr 3
`prototype`	`Stproc`		Svr 3
`prototype`	`String`	Nav 3+	LW 1+
`referrer`	`document`	Nav 2+	
`rightContext`	`RegExp`	Nav 4	Svr 3
`screenX`	`event`	Nav 4	
`screenY`	`event`	Nav 4	
`scrollbars`	`Window`	Nav 4	
`search`	`Link`	Nav 2+	
`search`	`Location`	Nav 2+	
`selected`	`Option`	Nav 2+	
`selectedIndex`	`Select`	Nav 2+	

Table 5 Properties (Continued)

Property	Of object	Client version	Server version
`self`	`Window`	Nav 2+	
`siblingAbove`	`Layer`	Nav 4	
`siblingBelow`	`Layer`	Nav 4	
`source`	`RegExp`	Nav 4	Svr 3
`SQRT1_2`	`Math`	Nav 2+	LW 1+
`SQRT2`	`Math`	Nav 2+	LW 1+
`src`	`Image`	Nav 3+	
`src`	`Layer`	Nav 4	
`status`	`Window`	Nav 2+	
`statusbar`	`Window`	Nav 4	
`suffixes`	`MimeType`	Nav 3+	
`target`	`event`	Nav 4	
`target`	`Form`	Nav 2+	
`target`	`Link`	Nav 2+	
`text`	`Option`	Nav 2+	
`text`	`Link`	Nav 4	
`title`	`document`	Nav 2+	
`toolbar`	`Window`	Nav 4	
`top`	`Layer`	Nav 4	
`top`	`Window`	Nav 2+	
`type`	`Button`	Nav 3+	
`type`	`Checkbox`	Nav 3+	
`type`	`event`	Nav 4	
`type`	`FileUpload`	Nav 3+	
`type`	`Hidden`	Nav 3+	
`type`	`Password`	Nav 3+	

Table 5 Properties (Continued)

Property	Of object	Client version	Server version
`type`	`MimeType`	Nav 3+	
`type`	`Radio`	Nav 3+	
`type`	`Reset`	Nav 3+	
`type`	`Select`	Nav 3+	
`type`	`Submit`	Nav 3+	
`type`	`Text`	Nav 3+	
`type`	`Textarea`	Nav 3+	
`URL`	`document`	Nav 2+	
`userAgent`	`navigator`	Nav 2+	
`value`	`Button`	Nav 2+	
`value`	`Checkbox`	Nav 2+	
`value`	`FileUpload`	Nav 2+	
`value`	`Hidden`	Nav 2+	
`value`	`Option`	Nav 2+	
`value`	`Password`	Nav 2+	
`value`	`Radio`	Nav 2+	
`value`	`Reset`	Nav 2+	
`value`	`Submit`	Nav 2+	
`value`	`Text`	Nav 2+	
`value`	`Textarea`	Nav 2+	
`visibility`	`Layer`	Nav 4	
`vlinkColor`	`document`	Nav 2+	
`vspace`	`Image`	Nav 3+	
`which`	`event`	Nav 4	
`width`	`event`	Nav 4	
`width`	`Image`	Nav 3+	

Table 5 Properties (Continued)

Property	Of object	Client version	Server version
`width`	`screen`	Nav 4	
`window`	`Window`	Nav 2+	
`zIndex`	`Layer`	Nav 4	

Table 6 Global functions

Function	Client version	Server version
`addClient`		LW 1+
`addResponseHeader`		Svr 3
`blob`		LW 1+
`callC`		LW 1+
`debug`		LW 1+
`deleteResponseHeader`		Svr 3
`escape`	Nav 2+	LW 1+
`eval`	Nav 2+	LW 1+
`flush`		LW 1+
`getOptionValue`		LW 1+
`getOptionValueCount`		LW 1+
`isNaN`	Nav 3+	LW 1+
`Number`	Nav 4	Svr 3
`parseFloat`	Nav 3+	LW 1+
`parseInt`	Nav 3+	LW 1+
`redirect`		LW 1+
`registerCFunction`		LW 1+
`ssjs_generateClientID`		Svr 3

Table 6 Global functions (Continued)

Function	Client version	Server version
`ssjs_getCGIVariable`		Svr 3
`ssjs_getClientID`		Svr 3
`String`	Nav 4	Svr 3
`taint`	Nav 3	LW 1+
`unescape`	Nav 2+	LW 1+
`untaint`	Nav 3	LW 1+
`write`		LW 1+

Table 7 Event handlers

Event handler	Client version	Handler for
`onAbort`	Nav 3+	`Image`
`onBlur`	Nav 3+	`Button`, `Checkbox`, `FileUpload`, `Layer`, `Password`, `Radio`, `Reset`, `Select`, `Submit`, `Text`, `Textarea`, `Window`
`onChange`	Nav 3+	`FileUpload`, `Select`, `Text`, `Textarea`
`onClick`	Nav 3+	`Button`, `Checkbox`, `document`, `Link`, `Radio`, `Reset`, `Submit`
`onDblClick`	Nav 4	`document`, `Link`
`onDragDrop`	Nav 4	`Window`
`onError`	Nav 3+	`Image`, `Window`
`onFocus`	Nav 3+	`Button`, `Checkbox`, `FileUpload`, `Layer`, `Password`, `Radio`, `Reset`, `Select`, `Submit`, `Text`, `Textarea`, `Window`
`onKeyDown`	Nav 4	`document`, `Image`, `Link`, `Textarea`
`onKeyPress`	Nav 4	`document`, `Image`, `Link`, `Textarea`
`onKeyUp`	Nav 4	`document`, `Image`, `Link`, `Textarea`

Table 7 Event handlers (Continued)

Event handler	Client version	Handler for
`onLoad`	Nav 3+	`Image`, `Layer`, `Window`
`onMouseDown`	Nav 4	`Button`, `document`, `Link`
`onMouseMove`	Nav 4	
`onMouseOut`	Nav 3+	`Layer`, `Link`
`onMouseOver`	Nav 3+	`Layer`, `Link`
`onMouseUp`	Nav 4	`Button`, `document`, `Link`
`onMove`	Nav 4	`Window`
`onReset`	Nav 3+	`Form`
`onResize`	Nav 4	`Window`
`onSelect`	Nav 3+	`Text`, `Textarea`
`onSubmit`	Nav 3+	`Form`
`onUnload`	Nav 3+	`Window`

Getting Started

This book is a reference manual for the JavaScript language, including objects in the core language and both client-side and server-side extensions. JavaScript is Netscape's cross-platform, object-based scripting language for client and server applications.

Sections:

- What You Should Already Know
- Where to Find JavaScript Information
- Document Conventions

What You Should Already Know

This book assumes you have this basic background:

- A general understanding of the Internet and the World Wide Web (WWW).
- Good working knowledge of HyperText Markup Language (HTML). Experience with forms and the Common Gateway Interface (CGI) is also useful.
- If you're going to use the LiveWire Database Service, relational databases and a working knowledge of Structured Query Language (SQL).

Where to Find JavaScript Information

Because JavaScript can be approached on several levels, its documentation has been split across several books to facilitate your introduction. The suite of online JavaScript books includes:

- *JavaScript Guide*[1] provides information about the core JavaScript language and about its client-side objects.
- *Writing Server-Side JavaScript Applications*[2] provides information about JavaScript's server-side objects and functions. In some cases, core language features work differently on the client than on the server. These differences are also discussed in this book. Finally, this book provides extra information you need to create an entire JavaScript application.
- *JavaScript Reference*[3] (this manual) provides reference material for the entire JavaScript language, including both client-side and server-side JavaScript.
- The JavaScript page[4] of the DevEdge library[5] contains several other documents of interest about JavaScript. The contents of this page change frequently. You should revisit it periodically to get the newest information.

In addition, other Netscape books discuss certain aspects of JavaScript particularly relevant to their topic area.

The Netscape web site contains much information that can be useful when you're creating JavaScript applications. Some URLs of particular interest include:

- `http://home.netscape.com/one_stop/intranet_apps/index.html`

  This is the Netscape AppFoundry Online home page. Netscape AppFoundry Online is a source for starter applications, technical information, tools, and expert forums for quickly building and dynamically deploying open intranet applications. This site also includes troubleshooting information in the resources section and extra help on setting up your JavaScript environment.

---

1. http://developer.netscape.com/library/documentation/communicator/jsguide4/index.htm
2. http://developer.netscape.com/library/documentation/enterprise/wrijsap/index.htm
3. http://developer.netscape.com/library/documentation/communicator/jsref/index.htm
4. http://developer.netscape.com/library/documentation/javascript.html
5. http://developer.netscape.com/library/documentation/

- `http://help.netscape.com/products/tools/livewire`

  This is Netscape's technical support page for information on the LiveWire Database Service. It contains lots of useful pointers to information on using LiveWire in your JavaScript applications.

- `http://developer.netscape.com/library/one/sdk/livewire/`

  This is Netscape's support page for information on server-side JavaScript. This URL is also available by clicking the Documentation link on the Netscape Server Application Manager

Document Conventions

Occasionally this book tells you where to find things in the user interface of Netscape Navigator. In these cases, the book describes the user interface in Navigator 4.0. This interface may be different in earlier versions of the browser.

JavaScript applications run on many operating systems; the information here applies to all versions. File and directory paths are given in Windows format (with backslashes separating directory names). For Unix versions, the directory paths are the same, except that you use slashes instead of backslashes to separate directories.

This book uses uniform resource locators (URLs) of the form

`http://`*`server.domain/path/file`*`.html`

In these URLs, *server* represents the name of the server on which you run your application, such as `research1` or `www`; *domain* represents your Internet domain name, such as `netscape.com` or `uiuc.edu`; *path* represents the directory structure on the server; and *file*`.html` represents an individual filename. In general, items in italics in URLs are placeholders and items in normal monospace font are literals. If your server has Secure Sockets Layer (SSL) enabled, you would use `https` instead of `http` in the URL.

This book uses the following font conventions:

- `The monospace font` is used for sample code and code listings, API and language elements (such as function names and class names), filenames, pathnames, directory names, HTML tags, and any text that must be typed on the screen. (*`Monospace italic font`* is used for placeholders embedded in code.)

- *Italic type* is used for book titles, emphasis, variables and placeholders, and words used in the literal sense.
- **Boldface type** is used for glossary terms.

Chapter

1

Introduction

This chapter briefly introduces JavaScript, Netscape's cross-platform, object-based scripting language for client and server applications.

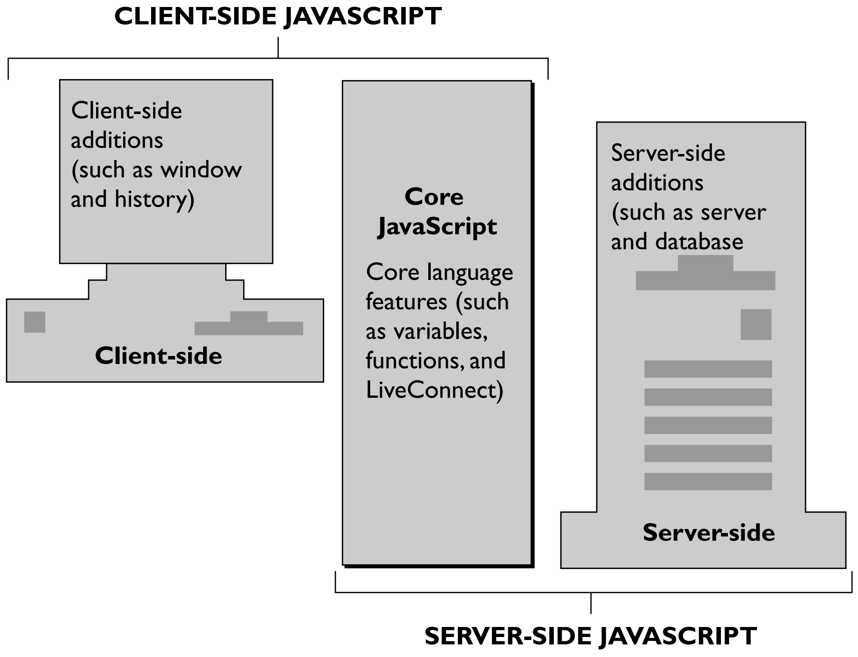
JavaScript lets you create applications that run over the Internet. Using JavaScript, you can create dynamic HTML pages that process user input and maintain persistent data using special objects, files, and relational databases. You can build applications ranging from internal corporate information management and intranet publishing to mass-market electronic transactions and commerce. Through JavaScript's LiveConnect functionality, your applications can access Java and CORBA distributed object applications.

Server-side and client-side JavaScript share the same core language. This core language corresponds to ECMA-262, the scripting language standardized by the European standards body, with some additions. The core language contains a set of core objects, such as the `Array` and `Date` objects. It also defines other language features such as its expressions, statements, and operators. Although server-side and client-side JavaScript use the same core functionality, in some cases they use them differently. You can download a PDF version of the ECMA-262 specification[1].

The components of JavaScript are illustrated in Figure 1.1.

1. http://developer.netscape.com/library/javascript/e262-pdf.pdf

Figure 1.1 The JavaScript language.


CLIENT-SIDE JAVASCRIPT
Client-side additions (such as window and history)
Client-side
Core JavaScript
Core language features (such as variables, functions, and LiveConnect)
Server-side additions (such as server and database
Server-side
SERVER-SIDE JAVASCRIPT

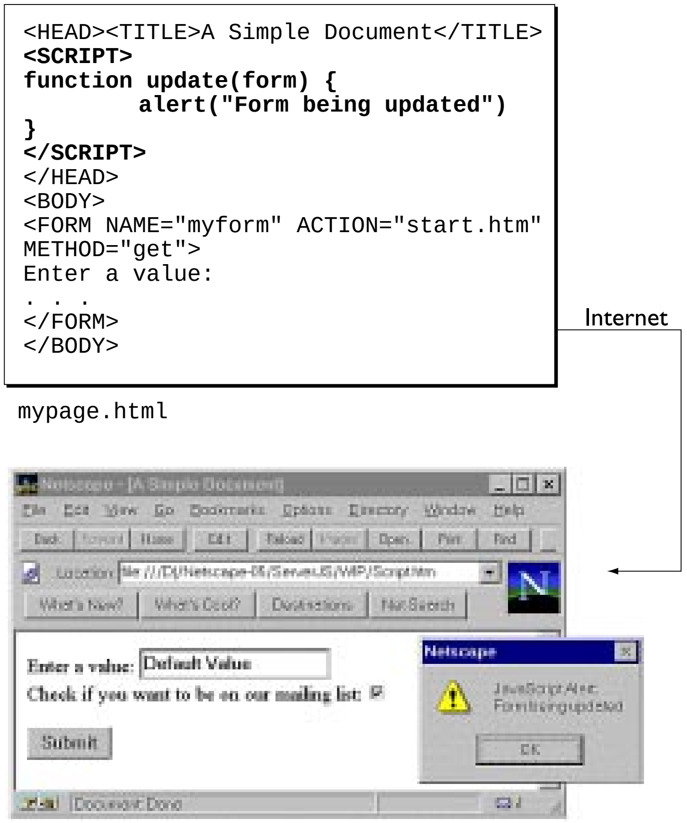
Client-side JavaScript (or *Navigator JavaScript*) encompasses the core language plus extras such as the predefined objects only relevant to running JavaScript in a browser. *Server-side JavaScript* encompasses the same core language plus extras such as the predefined objects and functions only relevant to running JavaScript on a server.

Client-side JavaScript is embedded directly in HTML pages and is interpreted by the browser completely at runtime. Because production applications frequently have greater performance demands upon them, JavaScript applications that take advantage of its server-side capabilities are compiled before they are deployed. The next two sections introduce you to how JavaScript works on the client and on the server.

Client-Side JavaScript

Web browsers such as Netscape Navigator 2.0 (and later versions) can interpret client-side JavaScript statements embedded in an HTML page. When the browser (or client) requests such a page, the server sends the full content of the document, including HTML and JavaScript statements, over the network to the client. The client reads the page from top to bottom, displaying the results of the HTML and executing JavaScript statements as it goes. This process produces the results that the user sees and is illustrated in Figure 1.2.

Figure 1.2 Client-side JavaScript.



```
<HEAD><TITLE>A Simple Document</TITLE>
<SCRIPT>
function update(form) {
        alert("Form being updated")
}
</SCRIPT>
</HEAD>
<BODY>
<FORM NAME="myform" ACTION="start.htm"
METHOD="get">
Enter a value:
. . .
</FORM>
</BODY>
```

mypage.html

Internet
Netscape - [A Simple Document]
File Edit View Go Bookmarks Options Directory Window Help
Back Forward Home Edit Reload Images Open Print Find
Location:
What's New? What's Cool? Destinations Net Search
Enter a value: Default Value
Check if you want to be on our mailing list:
Submit
Netscape
JavaScript Alert: Form being updated
OK
Document: Done

Client-side JavaScript statements embedded in an HTML page can respond to user events such as mouse clicks, form input, and page navigation. For example, you can write a JavaScript function to verify that users enter valid information into a form requesting a telephone number or zip code. Without any network transmission, the embedded JavaScript on the HTML page can check the entered data and display a dialog box to the user who enters invalid data.

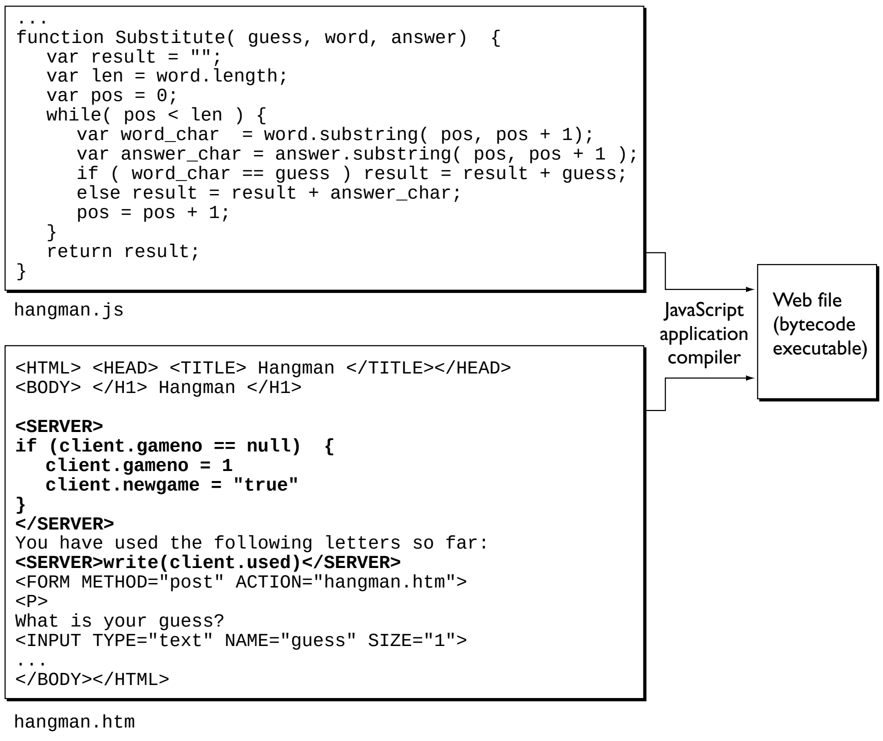
Server-Side JavaScript

On the server, JavaScript is also embedded in HTML pages. The server-side statements can connect to relational databases from different vendors, share information across users of an application, access the file system on the server, or communicate with other applications through LiveConnect and Java. A compiled JavaScript application can also include client-side JavaScript in addition to server-side JavaScript.

In contrast to pure client-side JavaScript scripts, JavaScript applications that use server-side JavaScript are compiled into bytecode executable files. These application executables are run in concert with a web server that contains the JavaScript runtime engine. For this reason, creating JavaScript applications is a two-stage process.

In the first stage, shown in Figure 1.3, you (the developer) create HTML pages (which can contain both client-side and server-side JavaScript statements) and JavaScript files. You then compile all of those files into a single executable.

Figure 1.3 Server-side JavaScript during development.



```
...
function Substitute( guess, word, answer)  {
   var result = "";
   var len = word.length;
   var pos = 0;
   while( pos < len ) {
      var word_char  = word.substring( pos, pos + 1);
      var answer_char = answer.substring( pos, pos + 1 );
      if ( word_char == guess ) result = result + guess;
      else result = result + answer_char;
      pos = pos + 1;
   }
   return result;
}
```

hangman.js

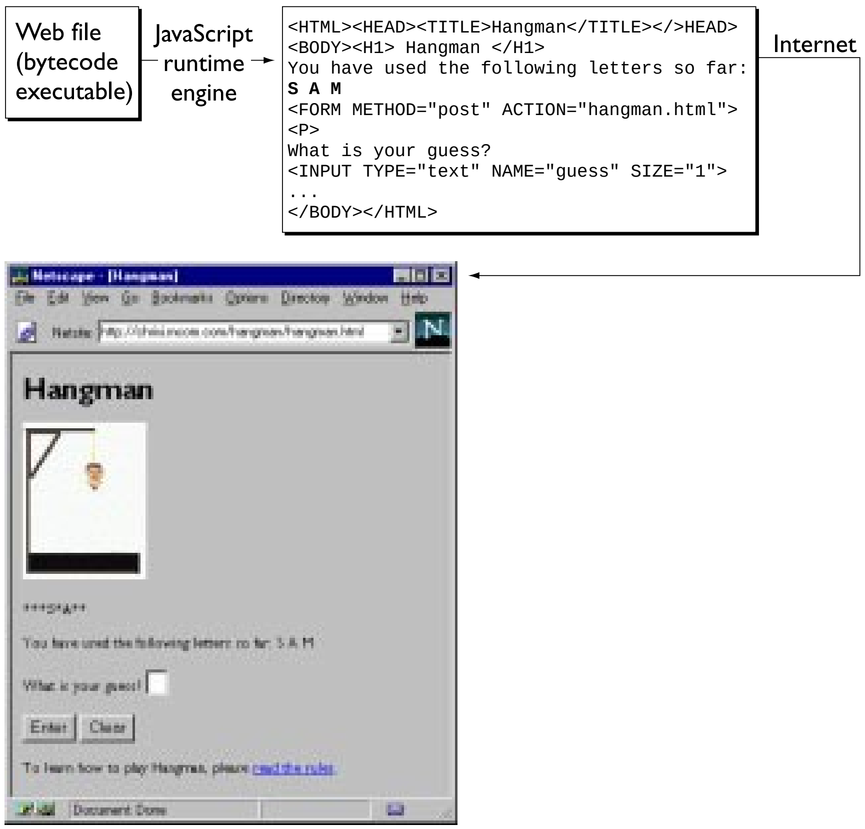
```
<HTML> <HEAD> <TITLE> Hangman </TITLE></HEAD>
<BODY> </H1> Hangman </H1>

<SERVER>
if (client.gameno == null)  {
   client.gameno = 1
   client.newgame = "true"
}
</SERVER>
You have used the following letters so far:
<SERVER>write(client.used)</SERVER>
<FORM METHOD="post" ACTION="hangman.htm">
<P>
What is your guess?
<INPUT TYPE="text" NAME="guess" SIZE="1">
...
</BODY></HTML>
```

hangman.htm

JavaScript application compiler
Web file (bytecode executable)

In the second stage, shown in Figure 1.4, a page in the application is requested by a client browser. The runtime engine uses the application executable to look up the source page and dynamically generate the HTML page to return. It runs any server-side JavaScript statements found on the page. The result of those statements might add new HTML or client-side JavaScript statements to the HTML page. It then sends the resulting page over the network to the Navigator client, which displays the results.

Figure 1.4 Server-side JavaScript during runtime.


Web file (bytecode executable)
JavaScript runtime engine
Internet
Netscape - [Hangman]
File Edit View Go Bookmarks Options Directory Window Help
Netsite:
Enter
Clear
Document: Done

```
<HTML><HEAD><TITLE>Hangman</TITLE></>HEAD>
<BODY><H1> Hangman </H1>
You have used the following letters so far:
S A M
<FORM METHOD="post" ACTION="hangman.html">
<P>
What is your guess?
<INPUT TYPE="text" NAME="guess" SIZE="1">
...
</BODY></HTML>
```

Hangman

***SAM**

You have used the following letters so far: S A M

What is your guess?

To learn how to play Hangman, please read the rules.

In contrast to standard Common Gateway Interface (CGI) programs, all JavaScript is integrated directly into HTML pages, facilitating rapid development and easy maintenance. JavaScript's Session Management Service contains objects you can use to maintain data that persists across client requests, multiple clients, and multiple applications. JavaScript's LiveWire Database Service provides objects for database access that serve as an interface to Structured Query Language (SQL) database servers.

JavaScript Objects

JavaScript has predefined objects for the core language, as well as additions for client-side and server-side JavaScript.

JavaScript has the following core objects:

`Array`, `Boolean`, `Date`, `Function`, `Math`, `Number`, `Object`, `String`

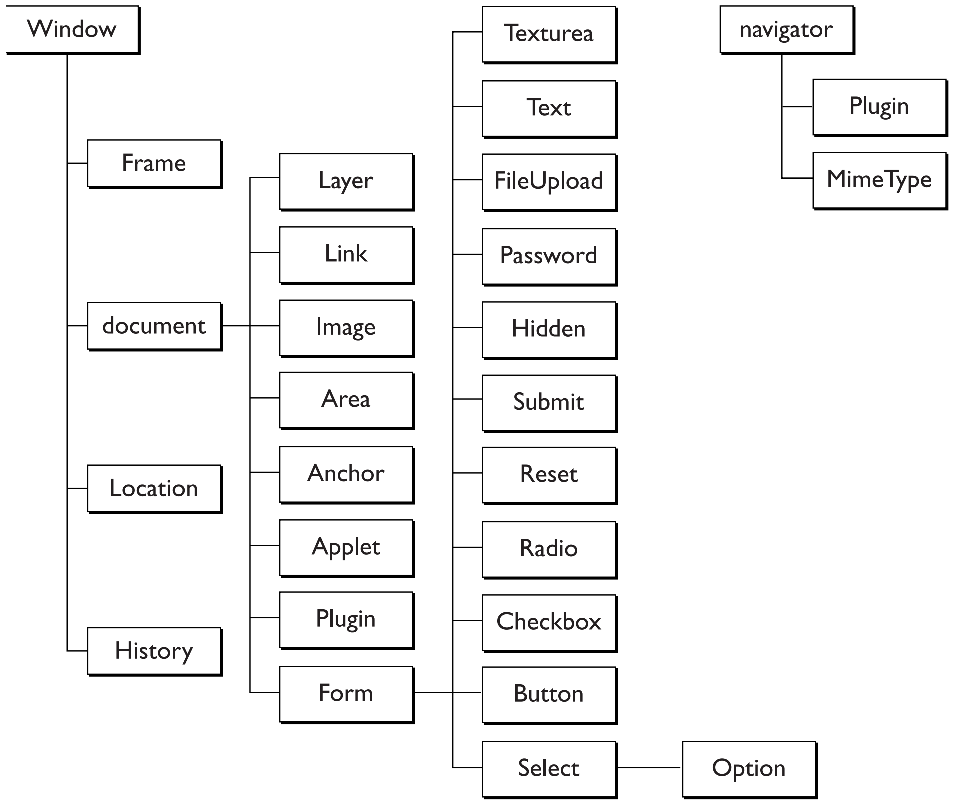
The additional client-side objects are as follows:

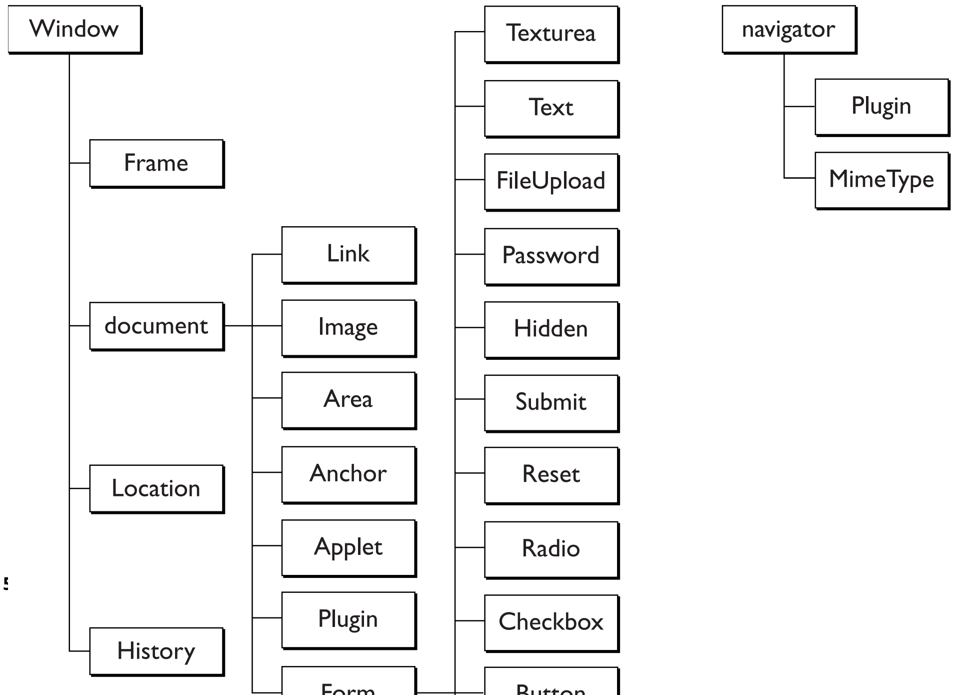
The objects available within the DOM are as follows:

`Anchor`, `Applet`, `Area`, `Button`, `Checkbox`, `document`, `event`, `FileUpload`, `Form`, `Frame`, `Hidden`, `History`, `Image`, `Layer`, `Link`, `Location`, `MimeType`, `navigator`, `Option`, `Password`, `Plugin`, `Radio`, `Reset`, `screen`, `Select`, `Submit`, `Text`, `Textarea`, `Window`

These objects represent information relevant to working with JavaScript in a web browser. Many of these objects are related to each other by occurring as property values. For example, to access the images in a document, you use the `document.images` array, each of whose elements is a `Image` object. Figure 1.5 shows the client-side object containment hierarchy.

Figure 1.5 Containment relationships among client-side objects



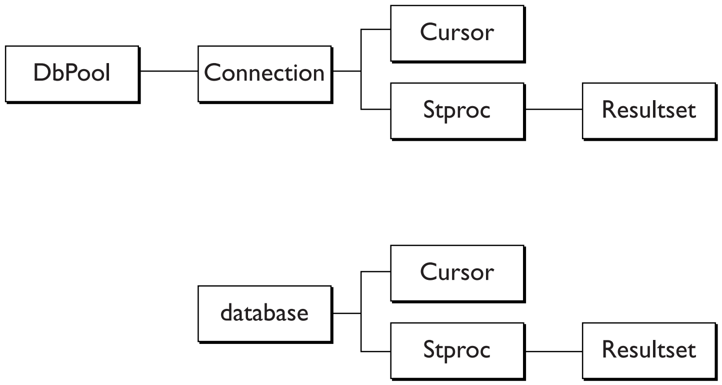


The server-side objects are:

`blob`, `client`, `Connection`, `Cursor`, `database`, `DbPool`, `File`, `Lock`, `project`, `request`, `Resultset`, `SendMail`, `server`, `Stproc`

As shown in Figure 1.6, some of the additional server-side objects also have a containment hierarchy.

Figure 1.6 Containment relationships among LiveWire objects



Security

Navigator version 2.02 and later automatically prevents scripts on one server from accessing properties of documents on a different server. This restriction prevents scripts from fetching private information such as directory structures or user session history.

JavaScript for Navigator 3.0 has a feature called data tainting that retains the security restriction but provides a means of secure access to specific components on a page.

- When data tainting is enabled, JavaScript in one window can see properties of another window, no matter what server the other window's document was loaded from. However, the author of the other window *taints* (marks) property values or other data that should be secure or private, and JavaScript cannot pass these tainted values on to any server without the user's permission.

- When data tainting is disabled, a script cannot access any properties of a window on another server.

In Navigator 4.0, data tainting has been removed. Instead, Navigator 4.0 provides signed JavaScript scripts for more reliable and more flexible security.

For information on data tainting and on signed scripts, see Chapter 7, "JavaScript Security," in the *JavaScript Guide*.

Chapter

2

Operators

JavaScript has assignment, comparison, arithmetic, bitwise, logical, string, and special operators. This chapter describes the operators and contains information about operator precedence.

Table 2.1 summarizes all of the JavaScript operators.

Table 2.1 JavaScript operators.

Operator Category	Operator	Description
Arithmetic Operators	+	(Addition) Adds 2 numbers.
	++	(Increment) Adds one to a variable representing a number (returning either the new or old value of the variable)
	-	(Unary negation, subtraction) As a unary operator, negates the value of its argument. As a binary operator, subtracts 2 numbers.
	--	(Decrement) Subtracts one from a variable representing a number (returning either the new or old value of the variable)
	*	(Multiplication) Multiplies 2 numbers.
	/	(Division) Divides 2 numbers.
	%	(Modulus) Computes the integer remainder of dividing 2 numbers.
String Operators	+	(String addition) Concatenates 2 strings.
	+=	Concatenates 2 strings and assigns the result to the first operand.

Table 2.1 JavaScript operators. (Continued)

Operator Category	Operator	Description
Logical Operators	&&	(Logical AND) Returns true if both logical operands are true. Otherwise, returns false.
	||	(Logical OR) Returns true if either logical expression is true. If both are false, returns false.
	!	(Logical negation) If its single operand is true, returns false; otherwise, returns true.
Bitwise Operators	&	(Bitwise AND) Returns a one in each bit position if bits of both operands are ones.
	^	(Bitwise XOR) Returns a one in a bit position if bits of one but not both operands are one.
	|	(Bitwise OR) Returns a one in a bit if bits of either operand is one.
	~	(Bitwise NOT) Flips the bits of its operand.
	<<	(Left shift) Shifts its first operand in binary representation the number of bits to the left specified in the second operand, shifting in zeros from the right.
	>>	(Sign-propagating right shift) Shifts the first operand in binary representation the number of bits to the right specified in the second operand, discarding bits shifted off.
	>>>	(Zero-fill right shift) Shifts the first operand in binary representation the number of bits to the right specified in the second operand, discarding bits shifted off, and shifting in zeros from the left.

Table 2.1 JavaScript operators. (Continued)

Operator Category	Operator	Description
Assignment Operators	=	Assigns the value of the second operand to the first operand.
	+=	Adds 2 numbers and assigns the result to the first.
	-=	Subtracts 2 numbers and assigns the result to the first.
	*=	Multiplies 2 numbers and assigns the result to the first.
	/=	Divides 2 numbers and assigns the result to the first.
	%=	Computes the modulus of 2 numbers and assigns the result to the first.
	&=	Performs a bitwise AND and assigns the result to the first operand.
	^=	Performs a bitwise XOR and assigns the result to the first operand.
	|=	Performs a bitwise OR and assigns the result to the first operand.
	<<=	Performs a left shift and assigns the result to the first operand.
	>>=	Performs a sign-propagating right shift and assigns the result to the first operand.
	>>>=	Performs a zero-fill right shift and assigns the result to the first operand.
Comparison Operators	==	Returns true if the operands are equal.
	!=	Returns true if the operands are not equal.
	>	Returns true if left operand is greater than right operand.
	>=	Returns true if left operand is greater than or equal to right operand.
	<	Returns true if left operand is less than right operand.
	<=	Returns true if left operand is less than or equal to right operand.

Table 2.1 JavaScript operators. (Continued)

Operator Category	Operator	Description
Special Operators	`?:`	Lets you perform a simple `"if...then...else"`
	`,`	Evaluates two expressions and returns the result of the second expression.
	`delete`	Lets you delete an object property or an element at a specified index in an array.
	`new`	Lets you create an instance of a user-defined object type or of one of the built-in object types.
	`this`	Keyword that you can use to refer to the current object.
	`typeof`	Returns a string indicating the type of the unevaluated operand.
	`void`	The void operator specifies an expression to be evaluated without returning a value.

Assignment Operators

An assignment operator assigns a value to its left operand based on the value of its right operand.

Implemented in Navigator 2.0

The basic assignment operator is equal (=), which assigns the value of its right operand to its left operand. That is, x = y assigns the value of y to x. The other assignment operators are shorthand for standard operations, as shown in Table 2.2.

Table 2.2 Assignment operators

Shorthand operator	Meaning
`x += y`	`x = x + y`
`x -= y`	`x = x - y`
`x *= y`	`x = x * y`
`x /= y`	`x = x / y`
`x %= y`	`x = x % y`

Table 2.2 Assignment operators

Shorthand operator	Meaning
`x <<= y`	`x = x << y`
`x >>= y`	`x = x >> y`
`x >>>= y`	`x = x >>> y`
`x &= y`	`x = x & y`
`x ^= y`	`x = x ^ y`
`x |= y`	`x = x | y`

Comparison Operators

A comparison operator compares its operands and returns a logical value based on whether the comparison is true or not. The operands can be numerical or string values. When used on string values, the comparisons are based on the standard lexicographical ordering.

Implemented in	Navigator 2.0

They are described in Table 2.3. In the examples in this table, assume `var1` has been assigned the value 3 and `var2` had been assigned the value 4.

Table 2.3 Comparison operators

Operator	Description	Examples returning true
Equal (`==`)	Returns true if the operands are equal.	`3 == var1`
Not equal (`!=`)	Returns true if the operands are not equal.	`var1 != 4`
Greater than (`>`)	Returns true if left operand is greater than right operand.	`var2 > var1`

Greater than or equal (`>=`)	Returns true if left operand is greater than or equal to right operand.	`var2 >= var1`
		`var1 >= 3`

Less than (`<`)	Returns true if left operand is less than right operand.	`var1 < var2`

Less than or equal (`<=`)	Returns true if left operand is less than or equal to right operand.	`var1 <= var2`
		`var2 <= 5`

Arithmetic Operators

Arithmetic operators take numerical values (either literals or variables) as their operands and return a single numerical value. The standard arithmetic operators are addition (+), subtraction (-), multiplication (*), and division (/). These operators work as they do in other programming languages.

Implemented in	Navigator 2.0

% (Modulus)

The modulus operator is used as follows:

```
var1 % var2
```

The modulus operator returns the first operand modulo the second operand, that is, `var1` modulo `var2`, in the preceding statement, where `var1` and `var2` are variables. The modulo function is the integer remainder of dividing `var1` by `var2`. For example, 12 % 5 returns 2.

++ (Increment)

The increment operator is used as follows:

`var++` or `++var`

This operator increments (adds one to) its operand and returns a value. If used postfix, with operator after operand (for example, x++), then it returns the value before incrementing. If used prefix with operator before operand (for example, ++x), then it returns the value after incrementing.

For example, if x is three, then the statement `y = x++` sets `y` to 3 and increments `x` to 4. If `x` is 3, then the statement `y = ++x` increments `x` to 4 and sets `y` to 4.

-- (Decrement)

The decrement operator is used as follows:

`var--` or `--var`

This operator decrements (subtracts one from) its operand and returns a value. If used postfix (for example, x--), then it returns the value before decrementing. If used prefix (for example, --x), then it returns the value after decrementing.

For example, if x is three, then the statement `y = x--` sets `y` to 3 and decrements `x` to 2. If `x` is 3, then the statement `y = --x` decrements `x` to 2 and sets `y` to 2.

- (Unary Negation)

The unary negation operator precedes its operand and negates it. For example, `y = -x` negates the value of `x` and assigns that to `y`; that is, if `x` were 3, `y` would get the value -3 and `x` would retain the value 3.

Bitwise Operators

Bitwise operators treat their operands as a set of bits (zeros and ones), rather than as decimal, hexadecimal, or octal numbers. For example, the decimal number nine has a binary representation of 1001. Bitwise operators perform their operations on such binary representations, but they return standard JavaScript numerical values.

Table 2.4 summarizes JavaScript's bitwise operators

Table 2.4 Bitwise operators

Operator	Usage	Description
Bitwise AND	`a & b`	Returns a one in each bit position if bits of both operands are ones.
Bitwise OR	`a | b`	Returns a one in a bit if bits of either operand is one.
Bitwise XOR	`a ^ b`	Returns a one in a bit position if bits of one but not both operands are one.
Bitwise NOT	`~ a`	Flips the bits of its operand.

Table 2.4 Bitwise operators

Operator	Usage	Description
Left shift	`a << b`	Shifts `a` in binary representation `b` bits to left, shifting in zeros from the right.
Sign-propagating right shift	`a >> b`	Shifts `a` in binary representation `b` bits to right, discarding bits shifted off.
Zero-fill right shift	`a >>> b`	Shifts `a` in binary representation `b` bits to the right, discarding bits shifted off, and shifting in zeros from the left.

Bitwise Logical Operators

Implemented in Navigator 2.0

Conceptually, the bitwise logical operators work as follows:

- The operands are converted to thirty-two-bit integers and expressed by a series of bits (zeros and ones).
- Each bit in the first operand is paired with the corresponding bit in the second operand: first bit to first bit, second bit to second bit, and so on.
- The operator is applied to each pair of bits, and the result is constructed bitwise.

For example, the binary representation of nine is 1001, and the binary representation of fifteen is 1111. So, when the bitwise operators are applied to these values, the results are as follows:

- 15 & 9 yields 9 (1111 & 1001 = 1001)
- 15 | 9 yields 15 (1111 | 1001 = 1111)
- 15 ^ 9 yields 6 (1111 ^ 1001 = 0110)

Bitwise Shift Operators

Implemented in Navigator 2.0

The bitwise shift operators take two operands: the first is a quantity to be shifted, and the second specifies the number of bit positions by which the first operand is to be shifted. The direction of the shift operation is controlled by the operator used.

Shift operators convert their operands to thirty-two-bit integers and return a result of the same type as the left operator.

<< (Left Shift)

This operator shifts the first operand the specified number of bits to the left. Excess bits shifted off to the left are discarded. Zero bits are shifted in from the right.

For example, `9<<2` yields thirty-six, because 1001 shifted two bits to the left becomes 100100, which is thirty-six.

>> (Sign-Propagating Right Shift)

This operator shifts the first operand the specified number of bits to the right. Excess bits shifted off to the right are discarded. Copies of the leftmost bit are shifted in from the left.

For example, 9>>2 yields two, because 1001 shifted two bits to the right becomes 10, which is two. Likewise, -9>>2 yields -3, because the sign is preserved.

>>> (Zero-Fill Right Shift)

This operator shifts the first operand the specified number of bits to the right. Excess bits shifted off to the right are discarded. Zero bits are shifted in from the left.

For example, 19>>>2 yields four, because 10011 shifted two bits to the right becomes 100, which is four. For non-negative numbers, zero-fill right shift and sign-propagating right shift yield the same result.

Logical Operators

Logical operators take Boolean (logical) values as operands and return a Boolean value.

Implemented in	Navigator 2.0

They are described in Table 2.5.

Table 2.5 Logical operators

Operator	Usage	Description
and (&&)	`expr1 && expr2`	Returns `expr1` if it converts to `false`. Otherwise, returns `expr2`.
or (||)	`expr1 || expr2`	Returns `expr1` if it converts to `true`. Otherwise, returns `expr2`.
not (!)	`!expr`	If `expr` is true, returns false; if `expr` is false, returns true.

Examples Consider the following script:

```
<script language="JavaScript1.2">"
v1 = "Cat";
v2 = "Dog";
v3 = false;

document.writeln("t && t returns " + (v1 && v2));
document.writeln("f && t returns " + (v3 && v1));
document.writeln("t && f returns " + (v1 && v3));
document.writeln("f && f returns " + (v3 && (3 == 4)));

document.writeln("t || t returns " + (v1 || v2));
document.writeln("f || t returns " + (v3 || v1));
document.writeln("t || f returns " + (v1 || v3));
document.writeln("f || f returns " + (v3 || (3 == 4)));

document.writeln("!t returns " + (!v1));
document.writeln("!f returns " + (!v3));

</script>
```

This script displays the following:

```
t && t returns Dog
f && t returns false
t && f returns false
```

```
f && f returns false
t || t returns Cat
f || t returns Cat
t || f returns Cat
f || f returns false
!t returns false
!f returns true
```

Short-Circuit Evaluation

As logical expressions are evaluated left to right, they are tested for possible "short-circuit" evaluation using the following rules:

- `false` && *anything* is short-circuit evaluated to false.
- `true` || *anything* is short-circuit evaluated to true.

The rules of logic guarantee that these evaluations are always correct. Note that the *anything* part of the above expressions is not evaluated, so any side effects of doing so do not take effect.

String Operators

In addition to the comparison operators, which can be used on string values, the concatenation operator (+) concatenates two string values together, returning another string that is the union of the two operand strings. For example, `"my " + "string"` returns the string `"my string"`.

Implemented in	Navigator 2.0

The shorthand assignment operator += can also be used to concatenate strings. For example, if the variable `mystring` has the value "alpha," then the expression `mystring += "bet"` evaluates to "alphabet" and assigns this value to `mystring`.

Special Operators

?: (Conditional operator)

The conditional operator is the only JavaScript operator that takes three operands. This operator is frequently used as a shortcut for the `if` statement.

Implemented in Navigator 2.0

Syntax `condition ? expr1 : expr2`

Parameters

`condition`	an expression that evaluates to `true` or `false`
`expr1, expr2`	expressions with values of any type.

Description If `condition` is `true`, the operator returns the value of `expr1`; otherwise, it returns the value of `expr2`. For example, to display a different message based on the value of the `isMember` variable, you could use this statement:

```
document.write ("The fee is " + (isMember ? "$2.00" : "$10.00"))
```

, (Comma operator)

The comma operator is very simple. It evaluates both of its operands and returns the value of the second operand.

Implemented in Navigator 2.0

Syntax `expr1, expr2`

Parameters

`expr1, expr2`	Any expressions

Description You can use the comma operator when you want to include multiple expressions in a location that requires a single expression. The most common usage of this operator is to supply multiple parameters in a `for` loop.

For example, if `a` is a 2-dimensional array with 10 elements on a side, the following code uses the comma operator to increment two variables at once. The code prints the values of the diagonal elements in the array:

```
for (var i=0, j=10; i <= 10; i++, j--)
   document.writeln("a["+i+","+j+"]= " + a[i,j])
```

delete

Deletes an object's property or an element at a specified index in an array.

Implemented in	Navigator 2.0

Syntax

```
delete objectName.property
delete objectName[index]
delete property
```

Parameters

`objectName`	The name of an object.
`property`	An existing property.
`index`	An integer representing the location of an element in an array

Description The third form is legal only within a `with` statement.

If the deletion succeeds, the `delete` operator sets the property or element to `undefined`. `delete` always returns undefined.

new

An operator that lets you create an instance of a user-defined object type or of one of the built-in object types that has a constructor function.

Implemented in	Navigator 2.0

Syntax

```
objectName = new objectType (param1 [,param2] ...[,paramN])
```

Arguments

`objectName`	Name of the new object instance.

`objectType`	Object type. It must be a function that defines an object type.
`param1...paramN`	Property values for the object. These properties are parameters defined for the `objectType` function.

Description Creating a user-defined object type requires two steps:

1. Define the object type by writing a function.

2. Create an instance of the object with `new`.

To define an object type, create a function for the object type that specifies its name, properties, and methods. An object can have a property that is itself another object. See the examples below.

You can always add a property to a previously defined object. For example, the statement `car1.color = "black"` adds a property `color` to `car1`, and assigns it a value of `"black"`. However, this does not affect any other objects. To add the new property to all objects of the same type, you must add the property to the definition of the `car` object type.

You can add a property to a previously defined object type by using the `Function.prototype` property. This defines a property that is shared by all objects created with that function, rather than by just one instance of the object type. The following code adds a `color` property to all objects of type `car`, and then assigns a value to the `color` property of the object `car1`. For more information, see `prototype`

```
Car.prototype.color=null
car1.color="black"
birthday.description="The day you were born"
```

Examples **Example 1: object type and object instance.** Suppose you want to create an object type for cars. You want this type of object to be called `car`, and you want it to have properties for make, model, and year. To do this, you would write the following function:

```
function car(make, model, year) {
   this.make = make
   this.model = model
   this.year = year
}
```

Now you can create an object called `mycar` as follows:

```
mycar = new car("Eagle", "Talon TSi", 1993)
```

This statement creates `mycar` and assigns it the specified values for its properties. Then the value of `mycar.make` is the string `"Eagle"`, `mycar.year` is the integer `1993`, and so on.

You can create any number of `car` objects by calls to `new`. For example,

```
kenscar = new car("Nissan", "300ZX", 1992)
```

Example 2: object property that is itself another object. Suppose you define an object called `person` as follows:

```
function person(name, age, sex) {
   this.name = name
   this.age = age
   this.sex = sex
}
```

And then instantiate two new `person` objects as follows:

```
rand = new person("Rand McNally", 33, "M")
ken = new person("Ken Jones", 39, "M")
```

Then you can rewrite the definition of `car` to include an owner property that takes a `person` object, as follows:

```
function car(make, model, year, owner) {
   this.make = make;
   this.model = model;
   this.year = year;
   this.owner = owner;
}
```

To instantiate the new objects, you then use the following:

```
car1 = new car("Eagle", "Talon TSi", 1993, rand);
car2 = new car("Nissan", "300ZX", 1992, ken)
```

Instead of passing a literal string or integer value when creating the new objects, the above statements pass the objects `rand` and `ken` as the parameters for the owners. To find out the name of the owner of `car2`, you can access the following property:

```
car2.owner.name
```

this

A keyword that you can use to refer to the current object. In general, in a method `this` refers to the calling object.

Implemented in Navigator 2.0

Syntax `this[.propertyName]`

Examples Suppose a function called `validate` validates an object's value property, given the object and the high and low values:

```
function validate(obj, lowval, hival) {
   if ((obj.value < lowval) || (obj.value > hival))
      alert("Invalid Value!")
}
```

You could call `validate` in each form element's `onChange` event handler, using `this` to pass it the form element, as in the following example:

```
<B>Enter a number between 18 and 99:</B>
<INPUT TYPE = "text" NAME = "age" SIZE = 3
   onChange="validate(this, 18, 99)">
```

typeof

The `typeof` operator is used in either of the following ways:

```
1. typeof operand
2. typeof (operand)
```

The `typeof` operator returns a string indicating the type of the unevaluated operand. `operand` is the string, variable, keyword, or object for which the type is to be returned. The parentheses are optional.

Implemented in Navigator 3.0

Suppose you define the following variables:

```
var myFun = new Function("5+2")
var shape="round"
var size=1
var today=new Date()
```

The `typeof` operator returns the following results for these variables:

```
typeof myFun is object
typeof shape is string
typeof size is number
typeof today is object
typeof dontExist is undefined
```

For the keywords `true` and `null`, the `typeof` operator returns the following results:

```
typeof true is boolean
typeof null is object
```

For a number or string, the `typeof` operator returns the following results:

```
typeof 62 is number
typeof 'Hello world' is string
```

For property values, the `typeof` operator returns the type of value the property contains:

```
typeof document.lastModified is string
typeof window.length is number
typeof Math.LN2 is number
```

For methods and functions, the `typeof` operator returns results as follows:

```
typeof blur is function
typeof eval is function
typeof parseInt is function
typeof shape.split is function
```

For predefined objects, the `typeof` operator returns results as follows:

```
typeof Date is function
typeof Function is function
typeof Math is function
typeof Option is function
typeof String is function
```

void

The void operator is used in either of the following ways:

```
1. javascript:void (expression)
2. javascript:void expression
```

The void operator specifies an expression to be evaluated without returning a value. `expression` is a JavaScript expression to evaluate. The parentheses surrounding the expression are optional, but it is good style to use them.

Implemented in Navigator 3.0

You can use the `void` operator to specify an expression as a hypertext link. The expression is evaluated but is not loaded in place of the current document.

The following code creates a hypertext link that does nothing when the user clicks it. When the user clicks the link, `void(0)` evaluates to 0, but that has no effect in JavaScript.

```
<A HREF="javascript:void(0)">Click here to do nothing</A>
```

The following code creates a hypertext link that submits a form when the user clicks it.

```
<A HREF="javascript:void(document.form.submit())">
Click here to submit</A>
```

Chapter

3

Statements

This chapter describes all JavaScript statements. JavaScript statements consist of keywords used with the appropriate syntax. A single statement may span multiple lines. Multiple statements may occur on a single line if each statement is separated by a semicolon.

Syntax conventions: All keywords in syntax statements are in bold. Words in italics represent user-defined names or statements. Any portions enclosed in square brackets, [ ], are optional. {statements} indicates a block of statements, which can consist of a single statement or multiple statements delimited by a curly braces { }.

Table 3.1 lists statements available in JavaScript.

Table 3.1 JavaScript statements.

`break`	Statement that terminates the current while or for loop and transfers program control to the statement following the terminated loop.
`comment`	Notations by the author to explain what a script does. Comments are ignored by the interpreter.
`continue`	Statement that terminates execution of the block of statements in a while or for loop, and continues execution of the loop with the next iteration.
`delete`	Deletes an object's property or an element of an array.

Table 3.1 JavaScript statements. (Continued)

`do...while`	Executes its statements until the test condition evaluates to false. Statement is executed at least once.
`export`	Allows a signed script to provide properties, functions, and objects to other signed or unsigned scripts.
`for`	Statement that creates a loop that consists of three optional expressions, enclosed in parentheses and separated by semicolons, followed by a block of statements executed in the loop.
`for...in`	Statement that iterates a specified variable over all the properties of an object. For each distinct property, JavaScript executes the specified statements.
`function`	Statement that declares a JavaScript function name with the specified parameters. Acceptable parameters include strings, numbers, and objects.
`if...else`	Statement that executes a set of statements if a specified condition is true. If the condition is false, another set of statements can be executed.
`import`	Allows a script to import properties, functions, and objects from a signed script which has exported the information.
`labeled`	Provides an identifier that can be used with break or continue to indicate where the program should continue execution.
`return`	Statement that specifies the value to be returned by a function.
`switch`	Allows a program to evaluate an expression and attempt to match the expression's value to a case label.
`var`	Statement that declares a variable, optionally initializing it to a value.
`while`	Statement that creates a loop that evaluates an expression, and if it is true, executes a block of statements.
`with`	Statement that establishes the default object for a set of statements.

break

Terminates the current `while` or `for` loop and transfers program control to the statement following the terminated loop.

Implemented in	Navigator 2.0, LiveWire 1.0

Syntax

```
break
break label
```

Parameter

`label`	Identifier associated with the label of the statement.

Description The `break` statement can now include an optional label that allows the program to break out of a labeled statement. This type of break must be in a statement identified by the label used by break.

The statements in a labeled statement can be of any type.

Examples The following function has a `break` statement that terminates the `while` loop when `e` is 3, and then returns the value 3 * `x`.

```
function testBreak(x) {
   var i = 0
   while (i < 6) {
      if (i == 3)
         break
      i++
   }
   return i*x
}
```

In the following example, a statement labeled `checkiandj` contains a statement labeled `checkj`. If `break` is encountered, the program breaks out of the `checkj` statement and continues with the remainder of the `checkiandj` statement. If `break` had a label of `checkiandj`, the program would break out of the `checkiandj` statement and continue at the statement following `checkiandj`.

```
checkiandj :
   if (4==i) {
      document.write("You've entered " + i + ".<BR>");
      checkj :
         if (2==j) {
            document.write("You've entered " + j + ".<BR>");
            break checkj;
            document.write("The sum is " + (i+j) + ".<BR>");
         }
      document.write(i + "-" + j + "=" + (i-j) + ".<BR>");
   }
```

See also `labeled`, `switch`

comment

Notations by the author to explain what a script does. Comments are ignored by the interpreter.

Implemented in	Navigator 2.0, LiveWire 1.0

Syntax

```
// comment text
/* multiple line comment text */
```

Description JavaScript supports Java-style comments:

- Comments on a single line are preceded by a double-slash (//).
- Comments that span multiple lines are preceded by a /* and followed by a */.

Examples

```
// This is a single-line comment.
/* This is a multiple-line comment. It can be of any length, and
you can put whatever you want here. */
```

continue

Terminates execution of the block of statements in a `while` or `for` loop, and continues execution of the loop with the next iteration.

Implemented in	Navigator 2.0, LiveWire 1.0

Syntax

```
continue
continue label
```

Parameter

`label`	Identifier associated with the label of the statement.

Description In contrast to the `break` statement, `continue` does not terminate the execution of the loop entirely: instead,

- In a `while` loop, it jumps back to the `condition`.
- In a `for` loop, it jumps to the `update` expression.

The `continue` statement can now include an optional label that allows the program to terminate execution of a labeled statement and continue to the specified labeled statement. This type of continue must be in a looping statement identified by the label used by `continue`.

Examples The following example shows a `while` loop that has a `continue` statement that executes when the value of `i` is 3. Thus, `n` takes on the values 1, 3, 7, and 12.

```
i = 0
n = 0
while (i < 5) {
   i++
   if (i == 3)
      continue
   n += i
}
```

In the following example, a statement labeled `checkiandj` contains a statement labeled `checkj`. If `continue` is encountered, the program continues at the top of the `checkj` statement. Each time `continue` is encountered, `checkj` reiterates until its condition returns false. When false is returned, the remainder of the `checkiandj` statement is completed. `checkiandj` reiterates until its condition returns false. When false is returned, the program continues at the statement following checkiandj.

If `continue` had a label of `checkiandj`, the program would continue at the top of the checkiandj statement.

```
checkiandj :
while (i<4) {
   document.write(i + "<BR>");
   i+=1;

   checkj :
   while (j>4) {
      document.write(j + "<BR>");
      j-=1;
      if ((j%2)==0)
         continue checkj;
      document.write(j + " is odd.<BR>");
   }
   document.write("i = " + i + "<br>");
   document.write("j = " + j + "<br>");
}
```

See also `labeled`

delete

Deletes an object's property or an element at a specified index in an array.

Implemented in Navigator 4.0, Netscape Server 3.0

Syntax

```
delete objectName.property
delete objectName[index]
delete property
```

Parameters

`objectName`	An object from which to delete the specified property or value.
`property`	The property to delete.
`index`	An integer index into an array.

Description If the `delete` operator succeeds, it sets the property of element to `undefined`; the operator always returns `undefined`.

You can only use the `delete` operator to delete object properties and array entries. You cannot use this operator to delete objects or variables. Consequently, you can only use the third form within a `with` statement, to delete a property from the object.

do...while

Executes its statements until the test condition evaluates to false. Statement is executed at least once.

Implemented in Navigator 4.0, Netscape Server 3.0

Syntax

```
do
   statement
while (condition);
```

Parameters

`statement`	Block of statements that is executed at least once and is re-executed each time the condition evaluates to true.
`condition`	Evaluated after each pass through the loop. If `condition` evaluates to true, the statements in the preceding block are re-executed. When `condition` evaluates to false, control passes to the statement following `do while`.

Example In the following example, the `do` loop iterates at least once and reiterates until i is no longer less than 5.

```
do {
   i+=1
   document.write(i);
while (i<5);
```

export

Allows a signed script to provide properties, functions, and objects to other signed or unsigned scripts.

Implemented in Navigator 4.0, Netscape Server 3.0

Syntax

```
export name1, name2, ..., nameN
export *
```

Parameters

`nameN`	List of properties, functions, and objects to be exported.

*	Exports all properties, functions, and objects from the script.

Description Typically, information in a signed script is available only to scripts signed by the same principals. By exporting properties, functions, or objects, a signed script makes this information available to any script (signed or unsigned). The receiving script uses the companion import statement to access the information.

See also `import`

for

Creates a loop that consists of three optional expressions, enclosed in parentheses and separated by semicolons, followed by a block of statements executed in the loop.

Implemented in Navigator 2.0, LiveWire 1.0

Syntax

```
for ([initial-expression;] [condition;] [increment-expression])
{
   statements
}
```

Parameters

`initial-expression`	Statement or variable declaration. Typically used to initialize a counter variable. This expression may optionally declare new variables with the `var` keyword.
`condition`	Evaluated on each pass through the loop. If this condition evaluates to true, the statements in `statements` are performed. This conditional test is optional. If omitted, the condition always evaluates to true.
`increment-expression`	Generally used to update or increment the counter variable.
`statements`	Block of statements that are executed as long as condition evaluates to true. This can be a single statement or multiple statements. Although not required, it is good practice to indent these statements from the beginning of the `for` statement.

Examples The following `for` statement starts by declaring the variable `i` and initializing it to 0. It checks that `i` is less than nine, performs the two succeeding statements, and increments `i` by 1 after each pass through the loop.

```
for (var i = 0; i < 9; i++) {
   n += i
   myfunc(n)
}
```

for...in

Iterates a specified variable over all the properties of an object. For each distinct property, JavaScript executes the specified statements.

Implemented in Navigator 2.0, LiveWire 1.0

Syntax

```
for (variable in object) {
   statements}
```

Parameters

`variable`	Variable to iterate over every property.
`object`	Object for which the properties are iterated.
`statements`	Specifies the statements to execute for each property.

Examples The following function takes as its argument an object and the object's name. It then iterates over all the object's properties and returns a string that lists the property names and their values.

```
function dump_props(obj, objName) {
   var result = ""
   for (var i in obj) {
      result += objName + "." + i + " = " + obj[i] + "<BR>"
   }
   result += "<HR>"
   return result
}
```

function

Declares a JavaScript function with the specified parameters. Acceptable parameters include strings, numbers, and objects.

Implemented in Navigator 2.0, LiveWire 1.0

Syntax
```
function name([param] [, param] [..., param]) {
   statements}
```

Parameters

`name`	The function name.
`param`	The name of an argument to be passed to the function. A function can have up to 255 arguments.

Description To return a value, the function must have a `return` statement that specifies the value to return. You cannot nest a function statement in another statement or in itself.

All parameters are passed to functions, *by value*. In other words, the value is passed to the function, but if the function changes the value of the parameter, this change is not reflected globally or in the calling function.

In addition to defining functions as described here, you can also define `Function` objects.

Examples
```
//This function returns the total dollar amount of sales, when
//given the number of units sold of products a, b, and c.
function calc_sales(units_a, units_b, units_c) {
   return units_a*79 + units_b*129 + units_c*699
}
```

if...else

Executes a set of statements if a specified condition is true. If the condition is false, another set of statements can be executed.

Implemented in Navigator 2.0, LiveWire 1.0

Syntax
```
if (condition) {
   statements1}
[else {
   statements2}]
```

Parameters

`condition`	Can be any JavaScript expression that evaluates to true or false. Parentheses are required around the condition. If condition evaluates to true, the statements in `statements1` are executed.
`statements1` `statements2`	Can be any JavaScript statements, including further nested `if` statements. Multiple statements must be enclosed in braces.

Examples

```
if (cipher_char == from_char) {
   result = result + to_char
   x++}
else
   result = result + clear_char
```

import

Allows a script to import properties, functions, and objects from a signed script which has exported the information.

Implemented in Navigator 4.0, Netscape Server 3.0

Syntax

```
import objectName.name1, objectName.name2, ..., objectName.nameN
import objectName.*
```

Parameters

`nameN`	List of properties, functions, and objects to import from the export file.
`objectName`	Name of the object that will receive the imported names.
`*`	imports all properties, functions, and objects from the export script.

Description The `objectName` parameter is the name of the object that will receive the imported names. For example, if `f` and `p` have been exported, and if `obj` is an object from the importing script, then

```
import obj.f, obj.p
```

makes `f` and `p` accessible in the importing script as properties of `obj`.

Typically, information in a signed script is available only to scripts signed by the same principals. By exporting (using the `export` statement) properties, functions, or objects, a signed script makes this information available to any script (signed or unsigned). The receiving script uses the `import` statement to access the information.

The script must load the export script into a window, frame, or layer before it can import and use any exported properties, functions, and objects.

See also `export`

labeled

Provides an identifier that can be used with `break` or `continue` to indicate where the program should continue execution.

Implemented in	Navigator 4.0, Netscape Server 3.0

In a labeled statement, `break` or `continue` must be followed with a label, and the label must be the identifier of the labeled statement containing `break` or `continue`.

Syntax

```
label :
   statement
```

Parameter

`statement`	Block of statements. `break` can be used with any labeled statement, and continue can be used with looping labeled statements.

Example For an example of a labeled statement using `break`, see `break`. For an example of a labeled statement using `continue`, see `continue`.

See also `break`, `continue`

return

Specifies the value to be returned by a function.

Implemented in Navigator 2.0, LiveWire 1.0

Syntax

```
return expression
```

Parameters

`expression`	The expression to return.

Examples

The following function returns the square of its argument, `x`, where `x` is a number.

```
function square(x) {
   return x * x
}
```

switch

Allows a program to evaluate an expression and attempt to match the expression's value to a case label.

Implemented in Navigator 4.0, Netscape Server 3.0

Syntax

```
switch (expression){
   case label :
      statement;
      break;
   case label :
      statement;
      break;
   ...
   default : statement;
}
```

Parameters

`expression`	Value matched against label.
`label`	Identifier used to match against expression.
`statement`	Any statement.

Description If a match is found, the program executes the associated statement.

The program first looks for a label matching the value of expression and then executes the associated statement. If no matching label is found, the program looks for the optional default statement, and if found, executes the associated statement. If no default statement is found, the program continues execution at the statement following the end of `switch`.

The optional `break` statement associated with each case label ensures that the program breaks out of switch once the matched statement is executed and continues execution at the statement following switch. If `break` is omitted, the program continues execution at the next statement in the `switch` statement.

Example In the following example, if `expression` evaluates to "Bananas," the program matches the value with case "Bananas" and executes the associated statement. When `break` is encountered, the program breaks out of `switch` and executes the statement following `switch`. If `break` were omitted, the statement for case "Cherries" would also be executed.

```
switch (i) {
   case "Oranges" :
      document.write("Oranges are $0.59 a pound.<BR>");
      break;
   case "Apples" :
      document.write("Apples are $0.32 a pound.<BR>");
      break;
   case "Bananas" :
      document.write("Bananas are $0.48 a pound.<BR>");
      break;
   case "Cherries" :
      document.write("Cherries are $3.00 a pound.<BR>");
      break;
   default :
      document.write("Sorry, we are out of " + i + ".<BR>");
}
document.write("Is there anything else you'd like?<BR>");
```

var

Declares a variable, optionally initializing it to a value.

Implemented in Navigator 2.0, LiveWire 1.0

Syntax `var varname [= value] [..., varname [= value] ]`

Parameters

`varname`	Variable name. It can be any legal identifier.
`value`	Initial value of the variable and can be any legal expression.

Description The scope of a variable is the current function or, for variables declared outside a function, the current application.

Using `var` outside a function is optional; you can declare a variable by simply assigning it a value. However, it is good style to use `var`, and it is necessary in functions if a global variable of the same name exists.

Examples

```
var num_hits = 0, cust_no = 0
```

while

Creates a loop that evaluates an expression, and if it is true, executes a block of statements. The loop then repeats, as long as the specified condition is true.

Implemented in	Navigator 2.0, LiveWire 1.0

Syntax

```
while (condition) {
   statements
}
```

Parameters

`condition`	Evaluated before each pass through the loop. If this condition evaluates to true, the statements in the succeeding block are performed. When `condition` evaluates to false, execution continues with the statement following `statements`.
`statements`	Block of statements that are executed as long as the condition evaluates to true. Although not required, it is good practice to indent these statements from the beginning of the statement.

Examples The following `while` loop iterates as long as `n` is less than three.

```
n = 0
x = 0
while(n < 3) {
   n ++
   x += n
}
```

Each iteration, the loop increments `n` and adds it to `x`. Therefore, `x` and `n` take on the following values:

- After the first pass: n = 1 and x = 1
- After the second pass: n = 2 and x = 3
- After the third pass: n = 3 and x = 6

After completing the third pass, the condition `n < 3` is no longer true, so the loop terminates.

with

Establishes the default object for a set of statements. Within the set of statements, any property references that do not specify an object are assumed to be for the default object.

Implemented in Navigator 2.0, LiveWire 1.0

Syntax

```
with (object){
   statements
}
```

Parameters

`object`	Specifies the default object to use for the statements. The parentheses around object are required.
`statements`	Any block of statements.

Examples The following `with` statement specifies that the `Math` object is the default object. The statements following the `with` statement refer to the `PI` property and the `cos` and `sin` methods, without specifying an object. JavaScript assumes the `Math` object for these references.

```
var a, x, y
var r=10
with (Math) {
   a = PI * r * r
   x = r * cos(PI)
   y = r * sin(PI/2)
}
```

with

Chapter

4

Core

This chapter includes the JavaScript core objects `Array`, `Boolean`, `Date`, `Function`, `Math`, `Number`, `Object`, and `String`. These objects are used in both client-side and server-side JavaScript.

Table 4.1 summarizes the objects in this chapter.

Table 4.1 Core objects

Object	Description
`Array`	Represents an array.
`Boolean`	Represents a Boolean value.
`Date`	Represents a date.
`Function`	Specifies a string of JavaScript code to be compiled as a function.
`Math`	Provides basic math constants and functions; for example, its `PI` property contains the value of pi.
`Number`	Represents primitive numeric values.
`Object`	Contains the base functionality shared by all JavaScript objects.
`RegExp`	Represents a regular expression; also contains static properties that are shared among all regular expression objects.
`String`	Represents a JavaScript string.

Array

Represents an array of elements.

Core object

Implemented in Navigator 3.0, LiveWire 1.0

Created by The `Array` object constructor:

```
new Array(arrayLength);
new Array(element0, element1, ..., elementN);
```

Parameters

`arrayLength`	(Optional) The initial length of the array. You can access this value using the `length` property.
`elementN`	(Optional) A list of values for the array's elements. When this form is specified, the array is initialized with the specified values as its elements, and the array's `length` property is set to the number of arguments.

Description In Navigator 3.0, you can specify an initial length when you create the array. The following code creates an array of five elements:

```
billingMethod = new Array(5)
```

When you create an array, all of its elements are initially null. The following code creates an array of 25 elements, then assigns values to the first three elements:

```
musicTypes = new Array(25)
musicTypes[0] = "R&B"
musicTypes[1] = "Blues"
musicTypes[2] = "Jazz"
```

However, in Navigator 4.0, if you specify `LANGUAGE="JavaScript1.2"` in the `<SCRIPT>` tag, using `new Array(1)` creates a new array with `a[0]=1`.

An array's length increases if you assign a value to an element higher than the current length of the array. The following code creates an array of length 0, then assigns a value to element 99. This changes the length of the array to 100.

```
colors = new Array()
colors[99] = "midnightblue"
```

You can construct a *dense* array of two or more elements starting with index 0 if you define initial values for all elements. A dense array is one in which each element has a value. The following code creates a dense array with three elements:

```
myArray = new Array("Hello", myVar, 3.14159)
```

In Navigator 2.0, you must index an array by its ordinal number, for example `document.forms[0]`. In Navigator 3.0 and later, you can index an array by either its ordinal number or by its name (if defined). For example, assume you define the following array:

```
myArray = new Array("Wind","Rain","Fire")
```

You can then refer to the first element of the array as `myArray[0]` or `myArray["Wind"]`.

In Navigator 4.0, the result of a match between a regular expression and a string can create an array. This array has properties and elements that provide information about the match. An array is the return value of `RegExp.exec`, `String.match`, and `String.replace`. To help explain these properties and elements, look at the following example and then refer to the table below:

```
<SCRIPT LANGUAGE="JavaScript1.2">
//Match one d followed by one or more b's followed by one d
//Remember matched b's and the following d
//Ignore case

myRe=/d(b+)(d)/i;
myArray = myRe.exec("cdbBdbsbz");

</SCRIPT>
```

The properties and elements returned from this match are as follows:

Property/Element	Description	Example
`input`	A read-only property that reflects the original string against which the regular expression was matched.	cdbBdbsbz
`index`	A read-only property that is the zero-based index of the match in the string.	1

Property/Element	Description	Example
`[0]`	A read-only element that specifies the last matched characters.	dbBd
`[1], ...[n]`	Read-only elements that specify the parenthesized substring matches, if included in the regular expression. The number of possible parenthesized substrings is unlimited.	[1]=bB [2]=d

Property Summary

Property	Description
`index`	For an array created by a regular expression match, the zero-based index of the match in the string.
`input`	For an array created by a regular expression match, reflects the original string against which the regular expression was matched.
`length`	Reflects the number of elements in an array
`prototype`	Allows the addition of properties to an `Array` object.

Method Summary

Method	Description
`concat`	Joins two arrays and returns a new array.
`join`	Joins all elements of an array into a string.
`pop`	Removes the last element from an array and returns that element.
`push`	Adds one or more elements to the end of an array and returns that last element added.
`reverse`	Transposes the elements of an array: the first array element becomes the last and the last becomes the first.
`shift`	Removes the first element from an array and returns that element
`slice`	Extracts a section of an array and returns a new array.
`splice`	Adds and/or removes elements from an array.
`sort`	Sorts the elements of an array.

Method	Description
`toString`	Returns a string representing the specified object.
`unshift`	Adds one or more elements to the front of an array and returns the new length of the array.

Examples

Example 1. The following example creates an array, `msgArray`, with a length of 0, then assigns values to `msgArray[0]` and `msgArray[99]`, changing the length of the array to 100.

```
msgArray = new Array()
msgArray [0] = "Hello"
msgArray [99] = "world"
// The following statement is true,
// because defined msgArray [99] element.
if (msgArray .length == 100)
   document.write("The length is 100.")
```

See also examples for `onError`.

Example 2: Two-dimensional array. The following code creates a two-dimensional array and displays the results.

```
a = new Array(4)
for (i=0; i < 4; i++) {
   a[i] = new Array(4)
   for (j=0; j < 4; j++) {
      a[i][j] = "["+i+","+j+"]"
   }
}
for (i=0; i < 4; i++) {
   str = "Row "+i+":"
   for (j=0; j < 4; j++) {
      str += a[i][j]
   }
   document.write(str,"<p>")
}
```

This example displays the following results:

```
Multidimensional array test
Row 0:[0,0][0,1][0,2][0,3]
Row 1:[1,0][1,1][1,2][1,3]
Row 2:[2,0][2,1][2,2][2,3]
Row 3:[3,0][3,1][3,2][3,3]
```

See also `Image`

Properties

index

For an array created by a regular expression match, the zero-based index of the match in the string.

Property of	`Array`
Static	
Implemented in	Navigator 4.0, Netscape Server 3.0

input

For an array created by a regular expression match, reflects the original string against which the regular expression was matched.

Property of	`Array`
Static	
Implemented in	Navigator 4.0, Netscape Server 3.0

length

An integer that specifies the number of elements in an array. You can set the `length` property to truncate an array at any time. You cannot extend an array; for example, if you set length to 3 when it is currently 2, the array will still contain only 2 elements.

Property of	`Array`
Implemented in	Navigator 3.0, LiveWire 1.0

Examples In the following example, the `getChoice` function uses the `length` property to iterate over every element in the `musicType` array. `musicType` is a select element on the `musicForm` form.

```
function getChoice() {
   for (var i = 0; i < document.musicForm.musicType.length; i++) {
      if (document.musicForm.musicType.options[i].selected == true) {
         return document.musicForm.musicType.options[i].text
      }
   }
}
```

The following example shortens the array `statesUS` to a length of 50 if the current length is greater than 50.

```
if (statesUS.length > 50) {
   statesUS.length=50
   alert("The U.S. has only 50 states. New length is " +
statesUS.length)
}
```

prototype

Represents the prototype for this class. You can use the prototype to add properties or methods to all instances of a class. For information on prototypes, see `Function.prototype`.

Property of	`Array`
Implemented in	Navigator 3.0, LiveWire 1.0

Methods

concat

Joins two arrays and returns a new array.

Method of	`Array`
Implemented in	Navigator 4.0, Netscape Server 3.0

Syntax `concat(arrayName2)`

Parameters

`arrayName2`	Name of the array to concatenate to this array.

Description `concat` does not alter the original arrays, but returns a “one level deep” copy that contains copies of the same elements combined from the original arrays. Elements of the original arrays are copied into the new array as follows:

- Object references (and not the actual object) -- `concat` copies object references into the new array. Both the original and new array refer to the same object. If a referenced object changes, the changes are visible to both the new and original arrays.
- Strings and numbers (not `String` and `Number` objects)-- `concat` copies strings and numbers into the new array. Changes to the string or number in one array does not affect the other arrays.

If a new element is added to either array, the other array is not affected.

join

Joins all elements of an array into a string.

Method of	`Array`
Implemented in	Navigator 3.0, LiveWire 1.0

Syntax `join(separator)`

Parameters

`separator`	Specifies a string to separate each element of the array. The separator is converted to a string if necessary. If omitted, the array elements are separated with a comma.

Description The string conversion of all array elements are joined into one string.

Examples The following example creates an array, `a` with three elements, then joins the array three times: using the default separator, then a comma and a space, and then a plus.

```
a = new Array("Wind","Rain","Fire")
document.write(a.join() +"<BR>")
document.write(a.join(", ") +"<BR>")
document.write(a.join(" + ") +"<BR>")
```

This code produces the following output:

```
Wind,Rain,Fire
Wind, Rain, Fire
Wind + Rain + Fire
```

See also `Array.reverse`

pop

Removes the last element from an array and returns that element. This method changes the length of the array.

Method of	`Array`
Implemented in	Navigator 4.0, Netscape Server 3.0

Syntax `pop()`

Parameters None.

Example The following code displays the `myFish` array before and after removing its last element. It also displays the removed element:

```
myFish = ["angel", "clown", "mandarin", "surgeon"];
document.writeln("myFish before: " + myFish);
popped = myFish.pop();
document.writeln("myFish after: " + myFish);
document.writeln("popped this element: " + popped);
```

This example displays the following:

```
myFish before: ["angel", "clown", "mandarin", "surgeon"]
myFish after: ["angel", "clown", "mandarin"]
popped this element: surgeon
```

See also `push`, `shift`, `unshift`

push

Adds one or more elements to the end of an array and returns that last element added. This method changes the length of the array.

Method of	`Array`
Implemented in	Navigator 4.0, Netscape Server 3.0

Syntax `push(elt1, ..., eltN)`

Parameters

`elt1, ..., eltN`	The elements to add to the end of the array.

Description The behavior of the `push` method is analogous to the push function in Perl 4. Note that this behavior is different in Perl 5.

Example The following code displays the `myFish` array before and after adding elements to its end. It also displays the last element added:

```
myFish = ["angel", "clown"];
document.writeln("myFish before: " + myFish);
pushed = myFish.push("drum", "lion");
document.writeln("myFish after: " + myFish);
document.writeln("pushed this element last: " + pushed);
```

This example displays the following:

myFish before: ["angel", "clown"]
myFish after: ["angel", "clown", "drum", "lion"]
pushed this element last: lion

See also `pop`, `shift`, `unshift`

reverse

Transposes the elements of an array: the first array element becomes the last and the last becomes the first.

Method of	`Array`
Implemented in	Navigator 3.0, LiveWire 1.0

Syntax `reverse()`

Parameters None

Description The `reverse` method transposes the elements of the calling array object.

Examples The following example creates an array `myArray`, containing three elements, then reverses the array.

```
myArray = new Array("one", "two", "three")
myArray.reverse()
```

This code changes `myArray` so that:

- `myArray[0]` is `"three"`
- `myArray[1]` is `"two"`
- `myArray[2]` is `"one"`

See also `Array.join`, `Array.sort`

shift

Removes the first element from an array and returns that element. This method changes the length of the array.

Method of	`Array`
Implemented in	Navigator 4.0, Netscape Server 3.0

Syntax `shift()`

Parameters None.

Example The following code displays the `myFish` array before and after removing its first element. It also displays the removed element:

```
myFish = ["angel", "clown", "mandarin", "surgeon"];
document.writeln("myFish before: " + myFish);
shifted = myFish.shift();
document.writeln("myFish after: " + myFish);
document.writeln("Removed this element: " + shifted);
```

This example displays the following:

myFish before: ["angel", "clown", "mandarin", "surgeon"]
myFish after: ["clown", "mandarin", "surgeon"]
Removed this element: angel

See also `pop`, `push`, `unshift`

slice

Extracts a section of an array and returns a new array.

Method of	`Array`
Implemented in	Navigator 4.0, Netscape Server 3.0

Syntax `slice(begin,end)`

Parameters

`begin`	Zero-based index at which to begin extraction.

`end`	(Optional) Zero-based index at which to end extraction: • `slice` extracts up to but not including `end`. `slice(1,4)` extracts the second element through the fourth element (elements indexed 1, 2, and 3) • As a negative index, `end` indicates an offset from the end of the sequence. `slice(2,-1)` extracts the third element through the second to last element in the sequence. • If `end` is omitted, `slice` extracts to the end of the sequence.

Description `slice` does not alter the original array, but returns a new "one level deep" copy that contains copies of the elements sliced from the original array. Elements of the original array are copied into the new array as follows:

Object references (and not the actual object) -- `slice` copies object references into the new array. Both the original and new array refer to the same object. If a referenced object changes, the changes are visible to both the new and original arrays.

Strings and numbers (not `String` and `Number` objects)-- `slice` copies strings and numbers into the new array. Changes to the string or number in one array does not affect the other array.

If a new element is added to either array, the other array is not affected.

Example In the following example, `slice` creates a new array, `newCar`, from `myCar`. Both include a reference to the object `myHonda`. When the color of `myHonda` is changed to `purple`, both arrays reflect the change.

```
<SCRIPT LANGUAGE="JavaScript1.2">

//Using slice, create newCar from myCar.
myHonda = {color:"red",wheels:4,engine:{cylinders:4,size:2.2}}
myCar = [myHonda, 2, "cherry condition", "purchased 1997"]
newCar = myCar.slice(0,2)

//Write the values of myCar, newCar, and the color of myHonda
// referenced from both arrays.
document.write("myCar = " + myCar + "<BR>")
document.write("newCar = " + newCar + "<BR>")
document.write("myCar[0].color = " + myCar[0].color + "<BR>")
document.write("newCar[0].color = " + newCar[0].color + "<BR><BR>")

//Change the color of myHonda.
myHonda.color = "purple"
document.write("The new color of my Honda is " + myHonda.color +
"<BR><BR>")
```

```
//Write the color of myHonda referenced from both arrays.
document.write("myCar[0].color = " + myCar[0].color + "<BR>")
document.write("newCar[0].color = " + newCar[0].color + "<BR>")

</SCRIPT>
```

This script writes:

```
myCar = [{color:"red", wheels:4, engine:{cylinders:4, size:2.2}}, 2,
   "cherry condition", "purchased 1997"]
newCar = [{color:"red", wheels:4, engine:{cylinders:4, size:2.2}}, 2]
myCar[0].color = red newCar[0].color = red
The new color of my Honda is purple
myCar[0].color = purple
newCar[0].color = purple
```

splice

Changes the content of an array, adding new elements while removing old elements.

Method of	`Array`
Implemented in	Navigator 4.0, Netscape Server 3.0

Syntax `splice(index, howMany, newElt1, ..., newEltN)`

Parameters

`index`	Index at which to start changing the array.
`howMany`	An integer indicating the number of old array elements to remove. If `howMany` is 0, no elements are removed. In this case, you should specify at least one new element.
`newElt1, ..., newEltN`	(Optional) The elements to add to the array. If you don't specify any elements, splice simply removes elements from the array.

Description If you specify a different number of elements to insert than the number you're removing, the array will have a different length at the end of the call.

If `howMany` is 1, this method returns the single element that it removes. If `howMany` is more than 1, the method returns an array containing the removed elements.

Examples The following script illustrate the use of `splice`:

```
<SCRIPT LANGUAGE="JavaScript1.2">

myFish = ["angel", "clown", "mandarin", "surgeon"];
document.writeln("myFish: " + myFish + "<BR>");

removed = myFish.splice(2, 0, "drum");
document.writeln("After adding 1: " + myFish);
document.writeln("removed is: " + removed + "<BR>");

removed = myFish.splice(3, 1)
document.writeln("After removing 1: " + myFish);
document.writeln("removed is: " + removed + "<BR>");

removed = myFish.splice(2, 1, "trumpet")
document.writeln("After replacing 1: " + myFish);
document.writeln("removed is: " + removed + "<BR>");

removed = myFish.splice(0, 2, "parrot", "anemone", "blue")
document.writeln("After replacing 2: " + myFish);
document.writeln("removed is: " + removed);

</SCRIPT>
```

This script displays:

```
myFish: ["angel", "clown", "mandarin", "surgeon"]

After adding 1: ["angel", "clown", "drum", "mandarin", "surgeon"]
removed is: undefined

After removing 1: ["angel", "clown", "drum", "surgeon"]
removed is: mandarin

After replacing 1: ["angel", "clown", "trumpet", "surgeon"]
removed is: drum

After replacing 2: ["parrot", "anemone", "blue", "trumpet", "surgeon"]
removed is: ["angel", "clown"]
```

sort

Sorts the elements of an array.

Method of	`Array`
Implemented in	Navigator 3.0, LiveWire 1.0 Navigator 4.0: modified behavior.

Syntax `sort(compareFunction)`

Parameters

`compareFunction`	Specifies a function that defines the sort order. If omitted, the array is sorted lexicographically (in dictionary order) according to the string conversion of each element.

Description If `compareFunction` is not supplied, elements are sorted by converting them to strings and comparing strings in lexicographic ("dictionary" or "telephone book," *not* numerical) order. For example, "80" comes before "9" in lexicographic order, but in a numeric sort 9 comes before 80.

If `compareFunction` is supplied, the array elements are sorted according to the return value of the compare function. If a and b are two elements being compared, then:

- If `compareFunction(a, b)` is less than 0, sort `b` to a lower index than `a`.
- If `compareFunction(a, b)` returns 0, leave `a` and `b` unchanged with respect to each other, but sorted with respect to all different elements.
- If `compareFunction(a, b)` is greater than 0, sort `b` to a higher index than `a`.

So, the compare function has the following form:

```
function compare(a, b) {
   if (a is less than b by some ordering criterion)
      return -1
   if (a is greater than b by the ordering criterion)
      return 1
   // a must be equal to b
   return 0
}
```

To compare numbers instead of strings, the compare function can simply subtract b from a:

```
function compareNumbers(a, b) {
   return a - b
}
```

JavaScript uses a stable sort: the index partial order of a and b does not change if a and b are equal. If a's index was less than b's before sorting, it will be after sorting, no matter how a and b move due to sorting.

The behavior of the `sort` method changed between Navigator 3.0 and Navigator 4.0.

In Navigator 3.0, on some platforms, the `sort` method does not work. This method works on all platforms for Navigator 4.0.

In Navigator 4.0, this method no longer converts undefined elements to null; instead it sorts them to the high end of the array. For example, assume you have this script:

```
<SCRIPT>
a = new Array();
a[0] = "Ant";
a[5] = "Zebra";

function writeArray(x) {
   for (i = 0; i < x.length; i++) {
      document.write(x[i]);
      if (i < x.length-1) document.write(", ");
   }
}

writeArray(a);
a.sort();
document.write("<BR><BR>");
writeArray(a);
</SCRIPT>
```

In Navigator 3.0, JavaScript prints:

```
ant, null, null, null, null, zebra
ant, null, null, null, null, zebra
```

In Navigator 4.0, JavaScript prints:

```
ant, undefined, undefined, undefined, undefined, zebra
ant, zebra, undefined, undefined, undefined, undefined
```

Examples The following example creates four arrays and displays the original array, then the sorted arrays. The numeric arrays are sorted without, then with, a compare function.

```
<SCRIPT>
stringArray = new Array("Blue","Humpback","Beluga")
numericStringArray = new Array("80","9","700")
numberArray = new Array(40,1,5,200)
mixedNumericArray = new Array("80","9","700",40,1,5,200)

function compareNumbers(a, b) {
   return a - b
}

document.write("<B>stringArray:</B> " + stringArray.join() +"<BR>")
document.write("<B>Sorted:</B> " + stringArray.sort() +"<P>")
document.write("<B>numberArray:</B> " + numberArray.join() +"<BR>")
document.write("<B>Sorted without a compare function:</B> " + numberArray.sort() +"<BR>")
document.write("<B>Sorted with compareNumbers:</B> " + numberArray.sort(compareNumbers)
+"<P>")

document.write("<B>numericStringArray:</B> " + numericStringArray.join() +"<BR>")
document.write("<B>Sorted without a compare function:</B> " + numericStringArray.sort()
+"<BR>")
document.write("<B>Sorted with compareNumbers:</B> " +
numericStringArray.sort(compareNumbers) +"<P>")

document.write("<B>mixedNumericArray:</B> " + mixedNumericArray.join() +"<BR>")
document.write("<B>Sorted without a compare function:</B> " + mixedNumericArray.sort()
+"<BR>")
document.write("<B>Sorted with compareNumbers:</B> " +
mixedNumericArray.sort(compareNumbers) +"<BR>")
</SCRIPT>
```

This example produces the following output. As the output shows, when a compare function is used, numbers sort correctly whether they are numbers or numeric strings.

```
stringArray: Blue,Humpback,Beluga
Sorted: Beluga,Blue,Humpback

numberArray: 40,1,5,200
Sorted without a compare function: 1,200,40,5
Sorted with compareNumbers: 1,5,40,200

numericStringArray: 80,9,700
Sorted without a compare function: 700,80,9
Sorted with compareNumbers: 9,80,700

mixedNumericArray: 80,9,700,40,1,5,200
Sorted without a compare function: 1,200,40,5,700,80,9
Sorted with compareNumbers: 1,5,9,40,80,200,700
```

See also `Array.join`, `Array.reverse`

toString

Returns a string representing the specified object.

Method of	`Array`
Implemented in	Navigator 3.0, LiveWire 1.0

Syntax `toString()`

Parameters None.

Description Every object has a `toString` method that is automatically called when it is to be represented as a text value or when an object is referred to in a string concatenation.

You can use `toString` within your own code to convert an object into a string, and you can create your own function to be called in place of the default `toString` method.

For `Array` objects, the built-in `toString` method joins the array and returns one string containing each array element separated by commas. For example, the following code creates an array and uses `toString` to convert the array to a string while writing output.

```
var monthNames = new Array("Jan","Feb","Mar","Apr")
document.write("monthNames.toString() is " + monthNames.toString())
```

The output is as follows:

```
monthNames.toString() is Jan,Feb,Mar,Apr
```

For information on defining your own `toString` method, see the `Object.toString` method.

unshift

Adds one or more elements to the beginning of an array and returns the new length of the array.

Method of	`Array`
Implemented in	Navigator 4.0, Netscape Server 3.0

Syntax `arrayName.unshift(elt1,..., eltN)`

Parameters

`elt1,...,eltN`	The elements to add to the front of the array.

Example The following code displays the `myFish` array before and after adding elements to it.

```
myFish = ["angel", "clown"];
document.writeln("myFish before: " + myFish);
unshifted = myFish.unshift("drum", "lion");
document.writeln("myFish after: " + myFish);
document.writeln("New length: " + unshifted);
```

This example displays the following:

```
myFish before: ["angel", "clown"]
myFish after: ["drum", "lion", "angel", "clown"]
New length: 4
```

See also `pop`, `push`, `shift`

Boolean

The `Boolean` object is an object wrapper for a boolean value.

Core object

Implemented in Navigator 3.0, LiveWire 1.0

Created by The `Boolean` constructor:

```
new Boolean(value)
```

Parameters

`value`	The initial value of the `Boolean` object. The value is converted to a `boolean` value, if necessary. If value is omitted or is 0, null, false, or the empty string (`""`), the object has an initial value of false. All other values, including the string `"false"`, create an object with an initial value of true.

Description Use a `Boolean` object when you need to convert a non-boolean value to a boolean value. You can use the `Boolean` object any place JavaScript expects a primitive boolean value. JavaScript returns the primitive value of the `Boolean` object by automatically invoking the `valueOf` method.

Property Summary

Property	Description
`prototype`	Defines a property that is shared by all Boolean objects.

Method Summary

Method	Description
`toString`	Returns a string representing the specified object.

Examples The following examples create `Boolean` objects with an initial value of false:

```
bNoParam = new Boolean()
bZero = new Boolean(0)
bNull = new Boolean(null)
bEmptyString = new Boolean("")
bfalse = new Boolean(false)
```

The following examples create `Boolean` objects with an initial value of true:

```
btrue = new Boolean(true)
btrueString = new Boolean("true")
bfalseString = new Boolean("false")
bSuLin = new Boolean("Su Lin")
```

Properties

prototype

Represents the prototype for this class. You can use the prototype to add properties or methods to all instances of a class. For information on prototypes, see `Function.prototype`.

Property of	`Boolean`
Implemented in	Navigator 3.0, LiveWire 1.0

Methods

toString

Returns a string representing the specified object.

Method of	`Boolean`
Implemented in	Navigator 3.0, LiveWire 1.0

Syntax `toString()`

Parameters None.

Description Every object has a `toString` method that is automatically called when it is to be represented as a text value or when an object is referred to in a string concatenation.

You can use `toString` within your own code to convert an object into a string, and you can create your own function to be called in place of the default `toString` method.

For `Boolean` objects and values, the built-in `toString` method returns `"true"` or `"false"` depending on the value of the boolean object. In the following code, `flag.toString` returns `"true"`.

```
flag = new Boolean(true)
document.write("flag.toString() is " + flag.toString() + "<BR>")
```

For information on defining your own `toString` method, see the `Object.toString` method.

Date

Lets you work with dates and times.

Core object	
Implemented in	Navigator 2.0, LiveWire 1.0 Navigator 3.0: added `prototype` property

Created by The `Date` constructor:

```
new Date()
new Date("month day, year hours:minutes:seconds")
new Date(yr_num, mo_num, day_num)
new Date(yr_num, mo_num, day_num, hr_num, min_num, sec_num)
```

Parameters

`month, day, year, hours, minutes, seconds`	String values representing part of a date.
`yr_num, mo_num, day_num, hr_num, min_num, sec_num`	Integer values representing part of a date. As an integer value, the month is represented by 0 to 11 with 0=January and 11=December.

Description If you supply no arguments, the constructor creates a `Date` object for today's date and time. If you supply some arguments, but not others, the missing arguments are set to 0. If you supply any arguments, you must supply at least the year, month, and day. You can omit the hours, minutes, and seconds.

The way JavaScript handles dates is very similar to the way Java handles dates: both languages have many of the same date methods, and both store dates internally as the number of milliseconds since January 1, 1970 00:00:00. Dates prior to 1970 are not allowed.

Property Summary

Property	Description
`prototype`	Allows the addition of properties to a `Date` object.

Method Summary

Method	Description
`getDate`	Returns the day of the month for the specified date.
`getDay`	Returns the day of the week for the specified date.
`getHours`	Returns the hour in the specified date.
`getMinutes`	Returns the minutes in the specified date.
`getMonth`	Returns the month in the specified date.
`getSeconds`	Returns the seconds in the specified date.
`getTime`	Returns the numeric value corresponding to the time for the specified date.
`getTimezoneOffset`	Returns the time-zone offset in minutes for the current locale.
`getYear`	Returns the year in the specified date.
`parse`	Returns the number of milliseconds in a date string since January 1, 1970, 00:00:00, local time.
`setDate`	Sets the day of the month for a specified date.
`setHours`	Sets the hours for a specified date.
`setMinutes`	Sets the minutes for a specified date.
`setMonth`	Sets the month for a specified date.

Method	Description
`setSeconds`	Sets the seconds for a specified date.
`setTime`	Sets the value of a Date object.
`setYear`	Sets the year for a specified date.
`toGMTString`	Converts a date to a string, using the Internet GMT conventions.
`toLocaleString`	Converts a date to a string, using the current locale's conventions.
`UTC`	Returns the number of milliseconds in a `Date` object since January 1, 1970, 00:00:00, Universal Coordinated Time (GMT).

Examples The following examples show several ways to assign dates:

```
today = new Date()
birthday = new Date("December 17, 1995 03:24:00")
birthday = new Date(95,11,17)
birthday = new Date(95,11,17,3,24,0)
```

Properties

prototype

Represents the prototype for this class. You can use the prototype to add properties or methods to all instances of a class. For information on prototypes, see `Function.prototype`.

Property of	`Date`
Implemented in	Navigator 3.0, LiveWire 1.0

Methods

getDate

Returns the day of the month for the specified date.

Method of	`Date`
Implemented in	Navigator 2.0, LiveWire 1.0

Syntax `getDate()`

Parameters None

Description The value returned by `getDate` is an integer between 1 and 31.

Examples The second statement below assigns the value 25 to the variable `day`, based on the value of the `Date` object `Xmas95`.

```
Xmas95 = new Date("December 25, 1995 23:15:00")
day = Xmas95.getDate()
```

See also `Date.setDate`

getDay

Returns the day of the week for the specified date.

Method of	`Date`
Implemented in	Navigator 2.0, LiveWire 1.0

Syntax `getDay()`

Parameters None

Description The value returned by `getDay` is an integer corresponding to the day of the week: 0 for Sunday, 1 for Monday, 2 for Tuesday, and so on.

Examples The second statement below assigns the value 1 to `weekday`, based on the value of the `Date` object `Xmas95`. December 25, 1995, is a Monday.

```
Xmas95 = new Date("December 25, 1995 23:15:00")
weekday = Xmas95.getDay()
```

getHours

Returns the hour for the specified date.

Method of	`Date`
Implemented in	Navigator 2.0, LiveWire 1.0

Syntax `getHours()`

Parameters None

Description The value returned by `getHours` is an integer between 0 and 23.

Examples The second statement below assigns the value 23 to the variable `hours`, based on the value of the `Date` object `Xmas95`.

```
Xmas95 = new Date("December 25, 1995 23:15:00")
hours = Xmas95.getHours()
```

See also `Date.setHours`

getMinutes

Returns the minutes in the specified date.

Method of	`Date`
Implemented in	Navigator 2.0, LiveWire 1.0

Syntax `getMinutes()`

Parameters None

Description The value returned by `getMinutes` is an integer between 0 and 59.

Examples The second statement below assigns the value 15 to the variable `minutes`, based on the value of the `Date` object `Xmas95`.

```
Xmas95 = new Date("December 25, 1995 23:15:00")
minutes = Xmas95.getMinutes()
```

See also `Date.setMinutes`

getMonth

Returns the month in the specified date.

Method of	`Date`
Implemented in	Navigator 2.0, LiveWire 1.0

Syntax `getMonth()`

Parameters None

Description The value returned by `getMonth` is an integer between 0 and 11. 0 corresponds to January, 1 to February, and so on.

Examples The second statement below assigns the value 11 to the variable `month`, based on the value of the `Date` object `Xmas95`.

```
Xmas95 = new Date("December 25, 1995 23:15:00")
month = Xmas95.getMonth()
```

See also `Date.setMonth`

getSeconds

Returns the seconds in the current time.

Method of	`Date`
Implemented in	Navigator 2.0, LiveWire 1.0

Syntax `getSeconds()`

Parameters None

Description The value returned by `getSeconds` is an integer between 0 and 59.

Examples The second statement below assigns the value 30 to the variable `secs`, based on the value of the `Date` object `Xmas95`.

```
Xmas95 = new Date("December 25, 1995 23:15:30")
secs = Xmas95.getSeconds()
```

See also `Date.setSeconds`

getTime

Returns the numeric value corresponding to the time for the specified date.

Method of	`Date`
Implemented in	Navigator 2.0, LiveWire 1.0

Syntax `getTime()`

Parameters None

Description The value returned by the `getTime` method is the number of milliseconds since 1 January 1970 00:00:00. You can use this method to help assign a date and time to another `Date` object.

Examples The following example assigns the date value of `theBigDay` to `sameAsBigDay`:

```
theBigDay = new Date("July 1, 1999")
sameAsBigDay = new Date()
sameAsBigDay.setTime(theBigDay.getTime())
```

See also `Date.setTime`

getTimezoneOffset

Returns the time-zone offset in minutes for the current locale.

Method of	`Date`
Implemented in	Navigator 2.0, LiveWire 1.0

Syntax `getTimezoneOffset()`

Parameters None

Description The time-zone offset is the difference between local time and Greenwich Mean Time (GMT). Daylight savings time prevents this value from being a constant.

Examples

```
x = new Date()
currentTimeZoneOffsetInHours = x.getTimezoneOffset()/60
```

getYear

Returns the year in the specified date.

Method of	`Date`
Implemented in	Navigator 2.0, LiveWire 1.0

Syntax `getYear()`

Parameters None

Description The `getYear` method returns either a 2-digit or 4-digit year:

- For years between and including 1900 and 1999, the value returned by `getYear` is the year minus 1900. For example, if the year is 1976, the value returned is 76.
- For years less than 1900 or greater than 1999, the value returned by `getYear` is the four-digit year. For example, if the year is 1856, the value returned is 1856. If the year is 2026, the value returned is 2026.

Examples **Example 1.** The second statement assigns the value 95 to the variable `year`.

```
Xmas = new Date("December 25, 1995 23:15:00")
year = Xmas.getYear()
```

Example 2. The second statement assigns the value 2000 to the variable `year`.

```
Xmas = new Date("December 25, 2000 23:15:00")
year = Xmas.getYear()
```

Example 3. The second statement assigns the value 95 to the variable `year`, representing the year 1995.

```
Xmas.setYear(95)
year = Xmas.getYear()
```

See also `Date.setYear`

parse

Returns the number of milliseconds in a date string since January 1, 1970, 00:00:00, local time.

Method of	`Date`

Static

Implemented in	Navigator 2.0, LiveWire 1.0

Syntax `Date.parse(dateString)`

Parameters :

`dateString`	A string representing a date.

Description The `parse` method takes a date string (such as `"Dec 25, 1995"`) and returns the number of milliseconds since January 1, 1970, 00:00:00 (local time). This function is useful for setting date values based on string values, for example in conjunction with the `setTime` method and the `Date` object.

Given a string representing a time, `parse` returns the time value. It accepts the IETF standard date syntax: `"Mon, 25 Dec 1995 13:30:00 GMT"`. It understands the continental US time-zone abbreviations, but for general use, use a time-zone offset, for example, `"Mon, 25 Dec 1995 13:30:00 GMT+0430"` (4 hours, 30 minutes west of the Greenwich meridian). If you do not specify a time zone, the local time zone is assumed. GMT and UTC are considered equivalent.

Because `parse` is a static method of `Date`, you always use it as `Date.parse()`, rather than as a method of a `Date` object you created.

Examples If `IPOdate` is an existing `Date` object, then you can set it to August 9, 1995 as follows:

```
IPOdate.setTime(Date.parse("Aug 9, 1995"))
```

See also `Date.UTC`

setDate

Sets the day of the month for a specified date.

Method of	`Date`
Implemented in	Navigator 2.0, LiveWire 1.0

Syntax `setDate(dayValue)`

Parameters

`dayValue`	An integer from 1 to 31, representing the day of the month.

Examples The second statement below changes the day for `theBigDay` to July 24 from its original value.

```
theBigDay = new Date("July 27, 1962 23:30:00")
theBigDay.setDate(24)
```

See also `Date.getDate`

setHours

Sets the hours for a specified date.

Method of	`Date`
Implemented in	Navigator 2.0, LiveWire 1.0

Syntax `setHours(hoursValue)`

Parameters

`hoursValue`	An integer between 0 and 23, representing the hour.

Examples `theBigDay.setHours(7)`

See also `Date.getHours`

setMinutes

Sets the minutes for a specified date.

Method of	`Date`
Implemented in	Navigator 2.0, LiveWire 1.0

Syntax `setMinutes(minutesValue)`

Parameters

`minutesValue`	An integer between 0 and 59, representing the minutes.

Examples `theBigDay.setMinutes(45)`

See also `Date.getMinutes`

setMonth

Sets the month for a specified date.

Method of	`Date`
Implemented in	Navigator 2.0, LiveWire 1.0

Syntax `setMonth(monthValue)`

Parameters

`monthValue`	An integer between 0 and 11 (representing the months January through December).

Examples `theBigDay.setMonth(6)`

See also `Date.getMonth`

setSeconds

Sets the seconds for a specified date.

Method of	`Date`
Implemented in	Navigator 2.0, LiveWire 1.0

Syntax `setSeconds(secondsValue)`

Parameters

`secondsValue`	An integer between 0 and 59.

Examples `theBigDay.setSeconds(30)`

See also `Date.getSeconds`

setTime

Sets the value of a `Date` object.

Method of	`Date`
Implemented in	Navigator 2.0, LiveWire 1.0

Syntax `setTime(timevalue)`

Parameters

`timevalue`	An integer representing the number of milliseconds since 1 January 1970 00:00:00.

Description Use the `setTime` method to help assign a date and time to another `Date` object.

Examples

```
theBigDay = new Date("July 1, 1999")
sameAsBigDay = new Date()
sameAsBigDay.setTime(theBigDay.getTime())
```

See also `Date.getTime`

setYear

Sets the year for a specified date.

Method of	`Date`
Implemented in	Navigator 2.0, LiveWire 1.0

Syntax `setYear(yearValue)`

Parameters

`yearValue`	An integer.

Description If `yearValue` is a number between 0 and 99 (inclusive), then the year for `dateObjectName` is set to 1900 + `yearValue`. Otherwise, the year for `dateObjectName` is set to `yearValue`.

Examples Note that there are two ways to set years in the 20th century.

Example 1. The year is set to 1996.

```
theBigDay.setYear(96)
```

Example 2. The year is set to 1996.

```
theBigDay.setYear(1996)
```

Example 3. The year is set to 2000.

```
theBigDay.setYear(2000)
```

See also `Date.getYear`

toGMTString

Converts a date to a string, using the Internet GMT conventions.

Method of	`Date`
Implemented in	Navigator 2.0, LiveWire 1.0

Syntax `toGMTString()`

Parameters None

Description The exact format of the value returned by `toGMTString` varies according to the platform.

Examples In the following example, `today` is a `Date` object:

```
today.toGMTString()
```

In this example, the `toGMTString` method converts the date to GMT (UTC) using the operating system's time-zone offset and returns a string value that is similar to the following form. The exact format depends on the platform.

```
Mon, 18 Dec 1995 17:28:35 GMT
```

See also `Date.toLocaleString`

toLocaleString

Converts a date to a string, using the current locale's conventions.

Method of	`Date`
Implemented in	Navigator 2.0, LiveWire 1.0

Syntax `toLocaleString()`

Parameters None

Description If you pass a date using `toLocaleString`, be aware that different platforms assemble the string in different ways. Using methods such as `getHours`, `getMinutes`, and `getSeconds` gives more portable results.

Examples In the following example, `today` is a `Date` object:

```
today = new Date(95,11,18,17,28,35) //months are represented by 0 to 11
today.toLocaleString()
```

In this example, `toLocaleString` returns a string value that is similar to the following form. The exact format depends on the platform.

```
12/18/95 17:28:35
```

See also `Date.toGMTString`

UTC

Returns the number of milliseconds in a `Date` object since January 1, 1970, 00:00:00, Universal Coordinated Time (GMT).

Method of	`Date`
Static	
Implemented in	Navigator 2.0, LiveWire 1.0

Syntax `Date.UTC(year, month, day, hrs, min, sec)`

Parameters

`year`	A year after 1900.
`month`	A month between 0 and 11.
`date`	A day of the month between 1 and 31.

`hrs`	(Optional) A number of hours between 0 and 23.
`min`	(Optional) A number of minutes between 0 and 59.
`sec`	(Optional) A number of seconds between 0 and 59.

Description `UTC` takes comma-delimited date parameters and returns the number of milliseconds since January 1, 1970, 00:00:00, Universal Coordinated Time (GMT).

Because UTC is a static method of `Date`, you always use it as `Date.UTC()`, rather than as a method of a `Date` object you created.

Examples The following statement creates a `Date` object using GMT instead of local time:

```
gmtDate = new Date(Date.UTC(96, 11, 1, 0, 0, 0))
```

See also `Date.parse`

Function

Specifies a string of JavaScript code to be compiled as a function.

Core object

Implemented in	Navigator 3.0, LiveWire 1.0 Navigator 4.0: added `arity` property.

Created by The `Function` constructor:

```
new Function (arg1, arg2, ... argN, functionBody)
```

Parameters

`arg1, arg2, ... argN`	(Optional) Names to be used by the function as formal argument names. Each must be a string that corresponds to a valid JavaScript identifier; for example `"x"` or `"theForm"`.
`functionBody`	A string containing the JavaScript statements comprising the function definition.

Description `Function` objects are evaluated each time they are used. This is less efficient than declaring a function and calling it within your code, because declared functions are compiled.

In addition to defining functions as described here, you can also use the `function` statement, as described in the *JavaScript Guide*.

Property Summary

Property	Description
`arguments`	An array corresponding to the arguments passed to a function.
`arity`	Indicates the number of arguments expected by the function.
`caller`	Specifies which function called the current function.
`prototype`	Allows the addition of properties to a `Function` object.

Method Summary

Method	Description
`toString`	Returns a string representing the specified object.

Specifying a variable value with a Function object

The following code assigns a function to the variable `setBGColor`. This function sets the current document's background color.

```
var setBGColor = new Function("document.bgColor='antiquewhite'")
```

To call the `Function` object, you can specify the variable name as if it were a function. The following code executes the function specified by the `setBGColor` variable:

```
var colorChoice="antiquewhite"
if (colorChoice=="antiquewhite") {setBGColor()}
```

You can assign the function to an event handler in either of the following ways:

```
document.form1.colorButton.onclick=setBGColor
```

```
<INPUT NAME="colorButton" TYPE="button"
   VALUE="Change background color"
   onClick="setBGColor()">
```

Creating the variable `setBGColor` shown above is similar to declaring the following function:

```
function setBGColor() {
   document.bgColor='antiquewhite'
}
```

Assigning a function to a variable is similar to declaring a function, but they have differences:

- When you assign a function to a variable using `var setBGColor = new Function("...")`, `setBGColor` is a variable for which the current value is a reference to the function created with `new Function()`.
- When you create a function using `function setBGColor() {...}`, `setBGColor` is not a variable, it is the name of a function.

Specifying arguments in a Function object

The following code specifies a `Function` object that takes two arguments.

```
var multFun = new Function("x", "y", "return x * y")
```

The string arguments `"x"` and `"y"` are formal argument names that are used in the function body, `"return x * y"`.

The following code shows several ways to call the function `multFun`:

```
var theAnswer = multFun(7,6)

document.write("15*2 = " + multFun(15,2))

<INPUT NAME="operand1" TYPE="text" VALUE="5" SIZE=5>
<INPUT NAME="operand2" TYPE="text" VALUE="6" SIZE=5>
<INPUT NAME="result" TYPE="text" VALUE="" SIZE=10>
<INPUT NAME="buttonM" TYPE="button" VALUE="Multiply"
   onClick="document.form1.result.value=
      multFun(document.form1.operand1.value,
         document.form1.operand2.value)">
```

You cannot call the function `multFun` in an object's event handler property, because event handler properties cannot take arguments. For example, you cannot call the function `multFun` by setting a button's `onclick` property as follows:

```
document.form1.button1.onclick=multFun(5,10)
```

Specifying an event handler with a Function object

The following code assigns a function to a window's `onFocus` event handler (the event handler must be spelled in all lowercase):

```
window.onfocus = new Function("document.bgColor='antiquewhite'")
```

Once you have a reference to a function object, you can use it like a function and it will convert from an object to a function:

```
window.onfocus()
```

Event handlers do not take arguments, so you cannot declare any arguments in the `Function` constructor for an event handler.

Examples **Example 1.** The following example creates `onFocus` and `onBlur` event handlers for a frame. This code exists in the same file that contains the `FRAMESET` tag. Note that this is the only way to create `onFocus` and `onBlur` event handlers for a frame, because you cannot specify the event handlers in the `FRAME` tag.

```
frames[0].onfocus = new Function("document.bgColor='antiquewhite'")
frames[0].onblur = new Function("document.bgColor='lightgrey'")
```

Example 2. You can determine whether a function exists by comparing the function name to null. In the following example, `func1` is called if the function `noFunc` does not exist; otherwise `func2` is called. Notice that the window name is needed when referring to the function name `noFunc`.

```
if (window.noFunc == null)
   func1()
else func2()
```

Properties

arguments

An array corresponding to the arguments passed to a function.

Property of	`Function`
Implemented in	Navigator 3.0, LiveWire 1.0 Navigator 4.0

Description You can call a function with more arguments than it is formally declared to accept by using the `arguments` array. This technique is useful if a function can be passed a variable number of arguments. You can use `arguments.length` to determine the number of arguments passed to the function, and then treat each argument by using the `arguments` array.

The `arguments` array is available only within a function declaration. Attempting to access the `arguments` array outside a function declaration results in an error.

The `this` keyword does not refer to the currently executing function, so you must refer to functions and `Function` objects by name, even within the function body. In JavaScript 1.2, `arguments` includes these additional properties:

- formal arguments—each formal argument of a function is a property of the `arguments` array.
- local variables—each local variable of a function is a property of the `arguments` array.
- `caller`—a property whose value is the `arguments` array of the outer function. If there is no outer function, the value is undefined.
- `callee`—a property whose value is the function reference.

For example, the following script demonstrates several of the `arguments` properties:

```
<SCRIPT>

function b(z) {
   document.write(arguments.z + "<BR>")
   document.write (arguments.caller.x + "<BR>")
   return 99
}

function a(x, y) {
   return  b(534)
}

document.write (a(2,3) + "<BR>")

</SCRIPT>
```

This displays:

534
2
99

`534 is the actual parameter to b, so it is the value of arguments.z.`

2 is a's actual x parameter, so (viewed within b) it is the value of `arguments.caller.x`.

99 is what `a(2,3)` returns.

Examples This example defines a function that creates HTML lists. The only formal argument for the function is a string that is `"U"` if the list is to be unordered (bulleted), or `"O"` if the list is to be ordered (numbered). The function is defined as follows:

```
function list(type) {
   document.write("<" + type + "L>")
   for (var i=1; i<list.arguments.length; i++) {
      document.write("<LI>" + list.arguments[i])
      document.write("</" + type + "L>")
   }
}
```

You can pass any number of arguments to this function, and it displays each argument as an item in the type of list indicated. For example, the following call to the function

```
list("U", "One", "Two", "Three")
```

results in this output:

```
<UL>
<LI>One
<LI>Two
<LI>Three
</UL>
```

In server-side JavaScript, you can display the same output by calling the `write` function instead of using `document.write`.

arity

When the `LANGUAGE` attribute of the `SCRIPT` tag is "JavaScript1.2", this property indicates the number of arguments expected by a function.

Property of	`Function`

Implemented in Navigator 4.0, Netscape Server 3.0

Description `arity` is external to the function, and indicates how many arguments the function expects. By contrast, `arguments.length` provides the number of arguments actually passed to the function.

Example The following example demonstrates the use of `arity` and `arguments.length`.

```
<SCRIPT LANGUAGE = "JavaScript1.2">
function addNumbers(x,y){
   document.write("length = " + arguments.length + "<BR>")
   z = x + y
}
document.write("arity = " + addNumbers.arity + "<BR>")
addNumbers(3,4,5)
</SCRIPT>
```

This script writes:

arity = 2
length = 3

caller

Returns the name of the function that invoked the currently executing function.

Property of	`Function`
Implemented in	Navigator 3.0, LiveWire 1.0

Description The `caller` property is available only within the body of a function. If used outside a function declaration, the `caller` property is null.

If the currently executing function was invoked by the top level of a JavaScript program, the value of `caller` is null.

The `this` keyword does not refer to the currently executing function, so you must refer to functions and `Function` objects by name, even within the function body.

The `caller` property is a reference to the calling function, so

- If you use it in a string context, you get the result of calling `functionName.toString`. That is, the decompiled canonical source form of the function.
- You can also call the calling function, if you know what arguments it might want. Thus, a called function can call its caller without knowing the name of the particular caller, provided it knows that all of its callers have the same form and fit, and that they will not call the called function again unconditionally (which would result in infinite recursion).

Examples The following code checks the value of a function's `caller` property.

```
function myFunc() {
   if (myFunc.caller == null) {
      alert("The function was called from the top!")
   } else alert("This function's caller was " + myFunc.caller)
}
```

See also `Function.arguments`

prototype

A value from which instances of a particular class are created. Every object that can be created by calling a constructor function has an associated `prototype` property.

Property of	`Object`
Implemented in	Navigator 3.0, LiveWire 1.0

Description You can add new properties or methods to an existing class by adding them to the prototype associated with the constructor function for that class. The syntax for adding a new property or method is:

`fun.prototype.name = value`

where

`fun`	The name of the constructor function object you want to change.
`name`	The name of the property or method to be created.
`value`	The value initially assigned to the new property or method.

If you add a new property to the prototype for an object, then all objects created with that object's constructor function will have that new property, even if the objects existed before you created the new property. For example, assume you have the following statements:

```
var array1 = new Array();
var array2 = new Array(3);
Array.prototype.description=null;
array1.description="Contains some stuff"
array2.description="Contains other stuff"
```

After you set a property for the prototype, all subsequent objects created with `Array` will have the property:

```
anotherArray=new Array()
anotherArray.description="Currently empty"
```

Example The following example creates a method, `str_rep`, and uses the statement `String.prototype.rep = str_rep` to add the method to all `String` objects. All objects created with `new String()` then have that method, even objects already created. The example then creates an alternate method and adds that to one of the `String` objects using the statement `s1.rep = fake_rep`. The `str_rep` method of the remaining `String` objects is not altered.

```
var s1 = new String("a")
var s2 = new String("b")
var s3 = new String("c")

// Create a repeat-string-N-times method for all String objects
function str_rep(n) {
   var s = "", t = this.toString()
   while (--n >= 0) s += t
   return s
}
String.prototype.rep = str_rep

// Display the results
document.write("<P>s1.rep(3) is " + s1.rep(3)) // "aaa"
document.write("<BR>s2.rep(5) is " + s2.rep(5)) // "bbbbb"
document.write("<BR>s3.rep(2) is " + s3.rep(2)) // "cc"

// Create an alternate method and assign it to only one String variable
function fake_rep(n) {
   return "repeat " + this + n + " times."
}

s1.rep = fake_rep

document.write("<P>s1.rep(1) is " + s1.rep(1)) // "repeat a 1 times."
document.write("<BR>s2.rep(4) is " + s2.rep(4)) // "bbbb"
document.write("<BR>s3.rep(6) is " + s3.rep(6)) // "cccccc"
```

This example produces the following output:

```
s1.rep(3) is aaa
s2.rep(5) is bbbbb
s3.rep(2) is cc
```

```
s1.rep(1) is repeat a1 times.
s2.rep(4) is bbbb
s3.rep(6) is cccccc
```

The function in this example also works on `String` objects not created with the `String` constructor. The following code returns `"zzz"`.

```
"z".rep(3)
```

Methods

toString

Returns a string representing the specified object.

Method of	`Function`
Implemented in	Navigator 3.0, LiveWire 1.0

Syntax `toString()`

Parameters None.

Description Every object has a `toString` method that is automatically called when it is to be represented as a text value or when an object is referred to in a string concatenation.

You can use `toString` within your own code to convert an object into a string, and you can create your own function to be called in place of the default `toString` method.

For `Function` objects, the built-in `toString` method decompiles the function back into the JavaScript source that defines the function. This string includes the `function` keyword, the argument list, curly braces, and function body.

For example, assume you have the following code that defines the `Dog` object type and creates `theDog,` an object of type `Dog`:

```
function Dog(name,breed,color,sex) {
   this.name=name
   this.breed=breed
   this.color=color
   this.sex=sex
}
```

```
theDog = new Dog("Gabby","Lab","chocolate","girl")
```

Any time `Dog` is used in a string context, JavaScript automatically calls the `toString` function, which returns the following string:

> function Dog(name, breed, color, sex) { this.name = name; this.breed = breed; this.color = color; this.sex = sex; }

For information on defining your own `toString` method, see the `Object.toString` method.

Math

A built-in object that has properties and methods for mathematical constants and functions. For example, the `Math` object's `PI` property has the value of pi.

Core object.

Implemented in	Navigator 2.0, LiveWire 1.0

Created by The `Math` object is a top-level, predefined JavaScript object. You can automatically access it without using a constructor or calling a method.

Description All properties and methods of `Math` are static. You refer to the constant PI as `Math.PI` and you call the sine function as `Math.sin(x)`, where `x` is the method's argument. Constants are defined with the full precision of real numbers in JavaScript.

It is often convenient to use the `with` statement when a section of code uses several `Math` constants and methods, so you don't have to type "Math" repeatedly. For example,

```
with (Math) {
   a = PI * r*r
   y = r*sin(theta)
   x = r*cos(theta)
}
```

Property Summary

Property	Description
`E`	Euler's constant and the base of natural logarithms, approximately 2.718.
`LN10`	Natural logarithm of 10, approximately 2.302.
`LN2`	Natural logarithm of 2, approximately 0.693.
`LOG10E`	Base 10 logarithm of E (approximately 0.434).
`LOG2E`	Base 2 logarithm of E (approximately 1.442).
`PI`	Ratio of the circumference of a circle to its diameter, approximately 3.14159.
`SQRT1_2`	Square root of 1/2; equivalently, 1 over the square root of 2, approximately 0.707.
`SQRT2`	Square root of 2, approximately 1.414.

Method Summary

Method	Description
`abs`	Returns the absolute value of a number.
`acos`	Returns the arccosine (in radians) of a number.
`asin`	Returns the arcsine (in radians) of a number.
`atan`	Returns the arctangent (in radians) of a number.
`atan2`	Returns the arctangent of the quotient of its arguments.
`ceil`	Returns the smallest integer greater than or equal to a number.
`cos`	Returns the cosine of a number.
`exp`	Returns E^{number}, where `number` is the argument, and E is Euler's constant, the base of the natural logarithms.
`floor`	Returns the largest integer less than or equal to a number.
`log`	Returns the natural logarithm (base E) of a number.
`max`	Returns the greater of two numbers.
`min`	Returns the lesser of two numbers.
`pow`	Returns `base` to the `exponent` power, that is, $\texttt{base}^{\texttt{exponent}}$.

Method	Description
`random`	Returns a pseudo-random number between 0 and 1.
`round`	Returns the value of a number rounded to the nearest integer.
`sin`	Returns the sine of a number.
`sqrt`	Returns the square root of a number.
`tan`	Returns the tangent of a number.

Properties

E

Euler's constant and the base of natural logarithms, approximately 2.718.

Property of	`Math`
Static, Read-only	
Implemented in	Navigator 2.0, LiveWire 1.0

Examples The following function returns Euler's constant:

```
function getEuler() {
   return Math.E
}
```

Description Because `E` is a static property of `Math`, you always use it as `Math.E`, rather than as a property of a `Math` object you created.

LN10

The natural logarithm of 10, approximately 2.302.

Property of	`Math`
Static, Read-only	
Implemented in	Navigator 2.0, LiveWire 1.0

Examples The following function returns the natural log of 10:

```
function getNatLog10() {
   return Math.LN10
}
```

Description Because `LN10` is a static property of `Math`, you always use it as `Math.LN10`, rather than as a property of a `Math` object you created.

LN2

The natural logarithm of 2, approximately 0.693.

Property of	`Math`
Static, Read-only	
Implemented in	Navigator 2.0, LiveWire 1.0

Examples The following function returns the natural log of 2:

```
function getNatLog2() {
   return Math.LN2
}
```

Description Because `LN2` is a static property of `Math`, you always use it as `Math.LN2`, rather than as a property of a `Math` object you created.

LOG10E

The base 10 logarithm of E (approximately 0.434).

Property of	`Math`
Static, Read-only	
Implemented in	Navigator 2.0, LiveWire 1.0

Examples The following function returns the base 10 logarithm of `E`:

```
function getLog10e() {
   return Math.LOG10E
}
```

Description Because `LOG10E` is a static property of `Math`, you always use it as `Math.LOG10E`, rather than as a property of a `Math` object you created.

LOG2E

The base 2 logarithm of E (approximately 1.442).

Property of	`Math`
Static, Read-only	
Implemented in	Navigator 2.0, LiveWire 1.0

Examples The following function returns the base 2 logarithm of E:

```
function getLog2e() {
   return Math.LOG2E
}
```

Description Because `LOG2E` is a static property of `Math`, you always use it as `Math.LOG2E`, rather than as a property of a `Math` object you created.

PI

The ratio of the circumference of a circle to its diameter, approximately 3.14159.

Property of	`Math`
Static, Read-only	
Implemented in	Navigator 2.0, LiveWire 1.0

Examples The following function returns the value of pi:

```
function getPi() {
   return Math.PI
}
```

Description Because `PI` is a static property of `Math`, you always use it as `Math.PI`, rather than as a property of a `Math` object you created.

SQRT1_2

The square root of 1/2; equivalently, 1 over the square root of 2, approximately 0.707.

Property of	`Math`
Static, Read-only	

Implemented in Navigator 2.0, LiveWire 1.0

Examples The following function returns 1 over the square root of 2:

```
function getRoot1_2() {
   return Math.SQRT1_2
}
```

Description Because `SQRT1_2` is a static property of `Math`, you always use it as `Math.SQRT1_2`, rather than as a property of a `Math` object you created.

SQRT2

The square root of 2, approximately 1.414.

Property of	`Math`
Static, Read-only	
Implemented in	Navigator 2.0, LiveWire 1.0

Examples The following function returns the square root of 2:

```
function getRoot2() {
   return Math.SQRT2
}
```

Description Because `SQRT2` is a static property of `Math`, you always use it as `Math.SQRT2`, rather than as a property of a `Math` object you created.

Methods

abs

Returns the absolute value of a number.

Method of	`Math`
Static	
Implemented in	Navigator 2.0, LiveWire 1.0

Syntax `abs(x)`

Parameters

`x`	A number

Examples The following function returns the absolute value of the variable `x`:

```
function getAbs(x) {
   return Math.abs(x)
}
```

Description Because `abs` is a static method of `Math`, you always use it as `Math.abs()`, rather than as a method of a `Math` object you created.

acos

Returns the arccosine (in radians) of a number.

Method of	`Math`
Static	
Implemented in	Navigator 2.0, LiveWire 1.0

Syntax `acos(x)`

Parameters

`x`	A number

Description The `acos` method returns a numeric value between 0 and pi radians. If the value of `number` is outside this range, it returns 0.

Because `acos` is a static method of `Math`, you always use it as `Math.acos()`, rather than as a method of a `Math` object you created.

Examples The following function returns the arccosine of the variable `x`:

```
function getAcos(x) {
   return Math.acos(x)
}
```

If you pass -1 to `getAcos`, it returns 3.141592653589793; if you pass 2, it returns 0 because 2 is out of range.

See also `Math.asin`, `Math.atan`, `Math.atan2`, `Math.cos`, `Math.sin`, `Math.tan`

asin

Returns the arcsine (in radians) of a number.

Method of	`Math`
Static	
Implemented in	Navigator 2.0, LiveWire 1.0

Syntax `asin(x)`

Parameters

`x`	A number

Description The `asin` method returns a numeric value between -pi/2 and pi/2 radians. If the value of `number` is outside this range, it returns 0.

Because `asin` is a static method of `Math`, you always use it as `Math.asin()`, rather than as a method of a `Math` object you created.

Examples The following function returns the arcsine of the variable `x`:

```
function getAsin(x) {
   return Math.asin(x)
}
```

If you pass `getAsin` the value 1, it returns 1.570796326794897 (pi/2); if you pass it the value 2, it returns 0 because 2 is out of range.

See also `Math.acos`, `Math.atan`, `Math.atan2`, `Math.cos`, `Math.sin`, `Math.tan`

atan

Returns the arctangent (in radians) of a number.

Method of	`Math`
Static	
Implemented in	Navigator 2.0, LiveWire 1.0

Syntax `atan(x)`

Parameters

`x`	A number

Description The `atan` method returns a numeric value between -pi/2 and pi/2 radians.

Because `atan` is a static method of `Math`, you always use it as `Math.atan()`, rather than as a method of a `Math` object you created.

Examples The following function returns the arctangent of the variable `x`:

```
function getAtan(x) {
   return Math.atan(x)
}
```

If you pass `getAtan` the value 1, it returns 0.7853981633974483; if you pass it the value .5, it returns 0.4636476090008061.

See also `Math.acos`, `Math.asin`, `Math.atan2`, `Math.cos`, `Math.sin`, `Math.tan`

atan2

Returns the arctangent of the quotient of its arguments.

Method of	`Math`
Static	
Implemented in	Navigator 2.0, LiveWire 1.0

Syntax `atan2(y, x)`

Parameters

`y, x`	Number

Description The `atan2` method returns a numeric value between -pi and pi representing the angle theta of an (`x,y`) point. This is the counterclockwise angle, measured in radians, between the positive X axis, and the point (`x,y`). Note that the arguments to this function pass the y-coordinate first and the x-coordinate second.

`atan2` is passed separate `x` and `y` arguments, and `atan` is passed the ratio of those two arguments.

Because `atan2` is a static method of `Math`, you always use it as `Math.atan2()`, rather than as a method of a `Math` object you created.

Examples The following function returns the angle of the polar coordinate:

```
function getAtan2(x,y) {
   return Math.atan2(x,y)
}
```

If you pass `getAtan2` the values (90,15), it returns 1.4056476493802699; if you pass it the values (15,90), it returns 0.16514867741462683.

See also `Math.acos`, `Math.asin`, `Math.atan`, `Math.cos`, `Math.sin`, `Math.tan`

ceil

Returns the smallest integer greater than or equal to a number.

Method of	`Math`
Static	
Implemented in	Navigator 2.0, LiveWire 1.0

Syntax `ceil(x)`

Parameters

`x`	A number

Description Because `ceil` is a static method of `Math`, you always use it as `Math.ceil()`, rather than as a method of a `Math` object you created.

Examples The following function returns the ceil value of the variable `x`:

```
function getCeil(x) {
   return Math.ceil(x)
}
```

If you pass 45.95 to `getCeil`, it returns 46; if you pass -45.95, it returns -45.

See also `Math.floor`

cos

Returns the cosine of a number.

Method of	`Math`
Static	
Implemented in	Navigator 2.0, LiveWire 1.0

Syntax `cos(x)`

Parameters

`x`	A number

Description The `cos` method returns a numeric value between -1 and 1, which represents the cosine of the angle.

Because `cos` is a static method of `Math`, you always use it as `Math.cos()`, rather than as a method of a `Math` object you created.

Examples The following function returns the cosine of the variable `x`:

```
function getCos(x) {
   return Math.cos(x)
}
```

If `x` equals `Math.PI/2`, `getCos` returns 6.123031769111886e-017; if `x` equals `Math.PI`, `getCos` returns -1.

See also `Math.acos`, `Math.asin`, `Math.atan`, `Math.atan2`, `Math.sin`, `Math.tan`

exp

Returns E^{x}, where `x` is the argument, and `E` is Euler's constant, the base of the natural logarithms.

Method of	`Math`
Static	
Implemented in	Navigator 2.0, LiveWire 1.0

Syntax `exp(x)`

Parameters

`x`	A number

Description Because `exp` is a static method of `Math`, you always use it as `Math.exp()`, rather than as a method of a `Math` object you created.

Examples The following function returns the exponential value of the variable `x`:

```
function getExp(x) {
   return Math.exp(x)
}
```

If you pass `getExp` the value 1, it returns 2.718281828459045.

See also `Math.E`, `Math.log`, `Math.pow`

floor

Returns the largest integer less than or equal to a number.

Method of	`Math`
Static	
Implemented in	Navigator 2.0, LiveWire 1.0

Syntax `floor(x)`

Parameters

`x`	A number

Description Because `floor` is a static method of `Math`, you always use it as `Math.floor()`, rather than as a method of a `Math` object you created.

Examples The following function returns the floor value of the variable `x`:

```
function getFloor(x) {
   return Math.floor(x)
}
```

If you pass 45.95 to `getFloor`, it returns 45; if you pass -45.95, it returns -46.

See also `Math.ceil`

log

Returns the natural logarithm (base `E`) of a number.

Method of	`Math`
Static	
Implemented in	Navigator 2.0, LiveWire 1.0

Syntax `log(x)`

Parameters

`x`	A number

Description If the value of `number` is outside the suggested range, the return value is always -1.797693134862316e+308.

Because `log` is a static method of `Math`, you always use it as `Math.log()`, rather than as a method of a `Math` object you created.

Examples The following function returns the natural log of the variable `x`:

```
function getLog(x) {
   return Math.log(x)
}
```

If you pass `getLog` the value 10, it returns 2.302585092994046; if you pass it the value 0, it returns -1.797693134862316e+308 because 0 is out of range.

See also `Math.exp`, `Math.pow`

max

Returns the larger of two numbers.

Method of	`Math`
Static	
Implemented in	Navigator 2.0, LiveWire 1.0

Syntax `max(x,y)`

Parameters

`x, y`	Numbers.

Description Because `max` is a static method of `Math`, you always use it as `Math.max()`, rather than as a method of a `Math` object you created.

Examples The following function evaluates the variables `x` and `y`:

```
function getMax(x,y) {
   return Math.max(x,y)
}
```

If you pass `getMax` the values 10 and 20, it returns 20; if you pass it the values -10 and -20, it returns -10.

See also `Math.min`

min

Returns the smaller of two numbers.

Method of	`Math`
Static	
Implemented in	Navigator 2.0, LiveWire 1.0

Syntax `min(x,y)`

Parameters

`x, y`	Numbers.

Description Because `min` is a static method of `Math`, you always use it as `Math.min()`, rather than as a method of a `Math` object you created.

Examples The following function evaluates the variables `x` and `y`:

```
function getMin(x,y) {
   return Math.min(x,y)
}
```

If you pass `getMin` the values 10 and 20, it returns 10; if you pass it the values -10 and -20, it returns -20.

See also `Math.max`

pow

Returns `base` to the `exponent` power, that is, `base`^{exponent}.

Method of	`Math`
Static	
Implemented in	Navigator 2.0, LiveWire 1.0

Syntax `pow(x,y)`

Parameters

`base`	The base number
`exponent`	The exponent to which to raise `base`

Description Because `pow` is a static method of `Math`, you always use it as `Math.pow()`, rather than as a method of a `Math` object you created.

Examples

```
function raisePower(x,y) {
   return Math.pow(x,y)
}
```

If `x` is 7 and `y` is 2, `raisePower` returns 49 (7 to the power of 2).

See also `Math.exp`, `Math.log`

random

Returns a pseudo-random number between 0 and 1. The random number generator is seeded from the current time, as in Java.

Method of	`Math`
Static	
Implemented in	Navigator 2.0, LiveWire 1.0: Unix only Navigator 3.0, LiveWire 1.0: all platforms

Syntax `random()`

Parameters None.

Description Because `random` is a static method of `Math`, you always use it as `Math.random()`, rather than as a method of a `Math` object you created.

Examples

```
//Returns a random number between 0 and 1
function getRandom() {
   return Math.random()
}
```

round

Returns the value of a number rounded to the nearest integer.

Method of	`Math`
Static	
Implemented in	Navigator 2.0, LiveWire 1.0

Syntax `round(x)`

Parameters

`x`	A number

Description If the fractional portion of `number` is .5 or greater, the argument is rounded to the next highest integer. If the fractional portion of `number` is less than .5, the argument is rounded to the next lowest integer.

Because `round` is a static method of `Math`, you always use it as `Math.round()`, rather than as a method of a `Math` object you created.

Examples

```
//Displays the value 20
document.write("The rounded value is " + Math.round(20.49))

//Displays the value 21
document.write("<P>The rounded value is " + Math.round(20.5))

//Displays the value -20
document.write("<P>The rounded value is " + Math.round(-20.5))

//Displays the value -21
document.write("<P>The rounded value is " + Math.round(-20.51))
```

In server-side JavaScript, you can display the same output by calling the `write` function instead of using `document.write`.

sin

Returns the sine of a number.

Method of	`Math`
Static	
Implemented in	Navigator 2.0, LiveWire 1.0

Syntax `sin(x)`

Parameters

`x`	A number

Description The `sin` method returns a numeric value between -1 and 1, which represents the sine of the argument.

Because `sin` is a static method of `Math`, you always use it as `Math.sin()`, rather than as a method of a `Math` object you created.

Examples The following function returns the sine of the variable `x`:

```
function getSine(x) {
   return Math.sin(x)
}
```

If you pass `getSine` the value `Math.PI/2`, it returns 1.

See also `Math.acos`, `Math.asin`, `Math.atan`, `Math.atan2`, `Math.cos`, `Math.tan`

sqrt

Returns the square root of a number.

Method of	`Math`
Static	
Implemented in	Navigator 2.0, LiveWire 1.0

Syntax `sqrt(x)`

Parameters

`x`	A number

Description If the value of `number` is outside the required range, `sqrt` returns 0.

Because `sqrt` is a static method of `Math`, you always use it as `Math.sqrt()`, rather than as a method of a `Math` object you created.

Examples The following function returns the square root of the variable `x`:

```
function getRoot(x) {
   return Math.sqrt(x)
}
```

If you pass `getRoot` the value 9, it returns 3; if you pass it the value 2, it returns 1.414213562373095.

tan

Returns the tangent of a number.

Method of	`Math`
Static	
Implemented in	Navigator 2.0, LiveWire 1.0

Syntax `tan(x)`

Parameters

`x`	A number

Description The `tan` method returns a numeric value that represents the tangent of the angle.

Because `tan` is a static method of `Math`, you always use it as `Math.tan()`, rather than as a method of a `Math` object you created.

Examples The following function returns the tangent of the variable `x`:

```
function getTan(x) {
   return Math.tan(x)
}
```

If you pass `Math.PI/4` to `getTan`, it returns 0.9999999999999999.

See also `Math.acos`, `Math.asin`, `Math.atan`, `Math.atan2`, `Math.cos`, `Math.sin`

Number

Lets you work with numeric values. The `Number` object is an object wrapper for primitive numeric values.

Core object	
Implemented in	Navigator 3.0, LiveWire 1.0 Navigator 4.0: modified behavior of `Number` constructor

Created by

The `Number` constructor:

```
new Number(value);
```

Parameters

`value`	The numeric value of the object being created.

Description

The primary uses for the `Number` object are:

- To access its constant properties, which represent the largest and smallest representable numbers, positive and negative infinity, and the Not-a-Number value.
- To create numeric objects that you can add properties to. Most likely, you will rarely need to create a `Number` object.

The properties of `Number` are properties of the class itself, not of individual `Number` objects.

Navigator 4.0: `Number(x)` now produces `NaN` rather than an error if `x` is a string that does not contain a well-formed numeric literal. For example,

```
x=Number("three");
document.write(x + "<BR>");
```

prints `NaN`

Property Summary

Property	Description
`MAX_VALUE`	The largest representable number.
`MIN_VALUE`	The smallest representable number.

Property	Description
`NaN`	Special "not a number" value.
`NEGATIVE_INFINITY`	Special infinite value; returned on overflow.
`POSITIVE_INFINITY`	Special negative infinite value; returned on overflow.
`prototype`	Allows the addition of properties to a `Number` object.

Method Summary

Method	Description
`toString`	Returns a string representing the specified object.

Examples **Example 1.** The following example uses the `Number` object's properties to assign values to several numeric variables:

```
biggestNum = Number.MAX_VALUE
smallestNum = Number.MIN_VALUE
infiniteNum = Number.POSITIVE_INFINITY
negInfiniteNum = Number.NEGATIVE_INFINITY
notANum = Number.NaN
```

Example 2. The following example creates a `Number` object, `myNum`, then adds a `description` property to all `Number` objects. Then a value is assigned to the `myNum` object's `description` property.

```
myNum = new Number(65)
Number.prototype.description=null
myNum.description="wind speed"
```

Properties

MAX_VALUE

The maximum numeric value representable in JavaScript.

Property of `Number`

Static, Read-only

Implemented in Navigator 3,0, LiveWire 1.0

Description The `MAX_VALUE` property has a value of approximately 1.79E+308. Values larger than `MAX_VALUE` are represented as `"Infinity"`.

Because `MAX_VALUE` is a static property of `Number`, you always use it as `Number.MAX_VALUE`, rather than as a property of a `Number` object you created.

Examples The following code multiplies two numeric values. If the result is less than or equal to `MAX_VALUE`, the `func1` function is called; otherwise, the `func2` function is called.

```
if (num1 * num2 <= Number.MAX_VALUE)
   func1()
else
   func2()
```

MIN_VALUE

The smallest positive numeric value representable in JavaScript.

Property of	`Number`
Static, Read-only	
Implemented in	Navigator 3,0, LiveWire 1.0

Description The `MIN_VALUE` property is the number closest to 0, not the most negative number, that JavaScript can represent.

`MIN_VALUE` has a value of approximately 2.22E-308. Values smaller than `MIN_VALUE` (“underflow values”) are converted to 0.

Because `MIN_VALUE` is a static property of `Number`, you always use it as `Number.MIN_VALUE`, rather than as a property of a `Number` object you created.

Examples The following code divides two numeric values. If the result is greater than or equal to `MIN_VALUE`, the `func1` function is called; otherwise, the `func2` function is called.

```
if (num1 / num2 >= Number.MIN_VALUE)
   func1()
else
   func2()
```

NaN

A special value representing Not-A-Number. This value is represented as the unquoted literal NaN.

Property of	`Number`
Read-only	
Implemented in	Navigator 3,0, LiveWire 1.0

Description JavaScript prints the value `Number.NaN` as `NaN`.

`NaN` is always unequal to any other number, including NaN itself; you cannot check for the not-a-number value by comparing to `Number.NaN`. Use the `isNaN` function instead.

You might use the `NaN` property to indicate an error condition for a function that should return a valid number.

Examples In the following example, if `month` has a value greater than 12, it is assigned NaN, and a message is displayed indicating valid values.

```
var month = 13
if (month < 1 || month > 12) {
   month = Number.NaN
   alert("Month must be between 1 and 12.")
}
```

See also `isNaN`, `parseFloat`, `parseInt`

NEGATIVE_INFINITY

A special numeric value representing negative infinity. This value is displayed as `"-Infinity"`.

Property of	`Number`
Static, Read-only	
Implemented in	Navigator 3,0, LiveWire 1.0

Description This value behaves mathematically like infinity; for example, anything multiplied by infinity is infinity, and anything divided by infinity is 0.

Because `NEGATIVE_INFINITY` is a static property of `Number`, you always use it as `Number.NEGATIVE_INFINITY`, rather than as a property of a `Number` object you created.

Examples In the following example, the variable `smallNumber` is assigned a value that is smaller than the minimum value. When the `if` statement executes, `smallNumber` has the value `"-Infinity"`, so the `func1` function is called.

```
var smallNumber = -Number.MAX_VALUE*10
if (smallNumber == Number.NEGATIVE_INFINITY)
   func1()
else
   func2()
```

POSITIVE_INFINITY

A special numeric value representing infinity. This value is displayed as `"Infinity"`.

Property of	`Number`
Static, Read-only	
Implemented in	Navigator 3,0, LiveWire 1.0

Description This value behaves mathematically like infinity; for example, anything multiplied by infinity is infinity, and anything divided by infinity is 0.

JavaScript does not have a literal for Infinity.

Because `POSITIVE_INFINITY` is a static property of `Number`, you always use it as `Number.POSITIVE_INFINITY`, rather than as a property of a `Number` object you created.

Examples In the following example, the variable `bigNumber` is assigned a value that is larger than the maximum value. When the `if` statement executes, `bigNumber` has the value `"Infinity"`, so the `func1` function is called.

```
var bigNumber = Number.MAX_VALUE * 10
if (bigNumber == Number.POSITIVE_INFINITY)
   func1()
else
   func2()
```

prototype

Represents the prototype for this class. You can use the prototype to add properties or methods to all instances of a class. For information on prototypes, see `Function.prototype`.

Property of	`Number`
Implemented in	Navigator 3.0, LiveWire 1.0

Methods

toString

Returns a string representing the specified object.

Method of	`Number`
Implemented in	Navigator 3.0

Syntax

```
toString()
toString(radix)
```

Parameters

`radix`	(Optional) An integer between 2 and 16 specifying the base to use for representing numeric values.

Description Every object has a `toString` method that is automatically called when it is to be represented as a text value or when an object is referred to in a string concatenation.

You can use `toString` within your own code to convert an object into a string, and you can create your own function to be called in place of the default `toString` method.

You can use `toString` on numeric values, but not on numeric literals:

```
// The next two lines are valid
var howMany=10
document.write("howMany.toString() is " + howMany.toString() + "<BR>")

// The next line causes an error
document.write("45.toString() is " + 45.toString() + "<BR>")
```

For information on defining your own `toString` method, see the `Object.toString` method.

Object

`Object` is the primitive JavaScript object type. All JavaScript objects are descended from `Object`. That is, all JavaScript objects have the methods defined for `Object`.

Core object	
Implemented in	Navigator 2.0: `toString` method Navigator 3.0, LiveWire 1.0: added `eval` and `valueOf` methods; `constructor` property Navigator 3.0: removed `eval` method

Created by The `Object` constructor:

```
new Object();
```

Parameters None

Property Summary

Property	Description
`constructor`	Specifies the function that creates an object's prototype.
`prototype`	Allows the addition of properties to all objects.

Method Summary

Method	Description
`eval`	Evaluates a string of JavaScript code in the context of the specified object.
`toString`	Returns a string representing the specified object.
`unwatch`	Removes a watchpoint from a property of the object.
`valueOf`	Returns the primitive value of the specified object.
`watch`	Adds a watchpoint to a property of the object.

Properties

constructor

Specifies the function that creates an object's prototype. Note that the value of this property is a reference to the function itself, not a string containing the function's name.

Property of	`Object`
Implemented in	Navigator 3.0, LiveWire 1.0

Description All objects inherit a `constructor` property from their `prototype`:

```
o = new Object  // or o = {} in Navigator 4.0
o.constructor == Object
a = new Array   // or a = [] in Navigator 4.0
a.constructor == Array
n = new Number(3)
n.constructor == Number
```

Even though you cannot construct most HTML objects, you can do comparisons. For example,

```
document.constructor == Document
document.form3.constructor == Form
```

Examples The following example creates a prototype, `Tree`, and an object of that type, `theTree`. The example then displays the `constructor` property for the object `theTree`.

```
function Tree(name) {
   this.name=name
}
theTree = new Tree("Redwood")
document.writeln("<B>theTree.constructor is</B> " +
   theTree.constructor + "<P>")
```

This example displays the following output:

```
theTree.constructor is function Tree(name) { this.name = name; }
```

prototype

Represents the prototype for this class. You can use the prototype to add properties or methods to all instances of a class. For more information, see `Function.prototype`.

Property of	`Object`
Implemented in	Navigator 3.0

Methods

eval

Evaluates a string of JavaScript code in the context of this object.

Method of	`Object`
Implemented in	Navigator 3.0, LiveWire 1.0 Navigator 4.0, Netscape Server 3.0: removed as method of objects; retained as global function.

Syntax `eval(string)`

Parameters

`string`	Any string representing a JavaScript expression, statement, or sequence of statements. The expression can include variables and properties of existing objects.

Description The argument of the `eval` method is a string. If the string represents an expression, `eval` evaluates the expression. If the argument represents one or more JavaScript statements, `eval` performs the statements. Do not call `eval` to evaluate an arithmetic expression; JavaScript evaluates arithmetic expressions automatically.

If you construct an arithmetic expression as a string, you can use `eval` to evaluate it at a later time. For example, suppose you have a variable `x`. You can postpone evaluation of an expression involving `x` by assigning the string value of the expression, say `"3 * x + 2"`, to a variable, and then calling `eval` at a later point in your script.

`eval` is also a global function, not associated with any object.

Note In Navigator 2.0, `eval` was a top-level function. In Navigator 3.0 `eval` was also a method of every object. The ECMA-262 standard for JavaScript made eval available only as a top-level function. For this reason, in Navigator 4.0, `eval` is once again a top-level function. In Navigator 4.02, `obj.eval(str)` is equivalent in all scopes to `with(obj)eval(str)`, except of course that the latter is a statement, not an expression.

Examples **Example 1.** The following example creates `breed` as a property of the object `myDog`, and also as a variable. The first `write` statement uses `eval('breed')` without specifying an object; the string `"breed"` is evaluated without regard to any object, and the `write` method displays `"Shepherd"`, which is the value of the `breed` variable. The second `write` statement uses `myDog.eval('breed')` which specifies the object `myDog`; the string `"breed"` is evaluated with regard to the `myDog` object, and the `write` method displays `"Lab"`, which is the value of the `breed` property of the `myDog` object.

```
function Dog(name,breed,color) {
   this.name=name
   this.breed=breed
   this.color=color
}
myDog = new Dog("Gabby")
myDog.breed="Lab"
var breed='Shepherd'
document.write("<P>" + eval('breed'))
document.write("<BR>" + myDog.eval('breed'))
```

Example 2. The following example uses `eval` within a function that defines an object type, `stone`. The statement `flint = new stone("x=42")` creates the object `flint` with the properties `x`, `y`, `z`, and `z2`. The `write` statements display the values of these properties as 42, 43, 44, and 45, respectively.

```
function stone(str) {
   this.eval("this."+str)
   this.eval("this.y=43")
   this.z=44
   this["z2"] = 45
}
flint = new stone("x=42")
document.write("<BR>flint.x is " + flint.x)
document.write("<BR>flint.y is " + flint.y)
document.write("<BR>flint.z is " + flint.z)
document.write("<BR>flint.z2 is " + flint.z2)
```

See also `eval`

toString

Returns a string representing the specified object.

Method of	`Object`
Implemented in	Navigator 2.0

Syntax

```
toString()
toString(radix)
```

Parameters

`radix`	(Optional) An integer between 2 and 16 specifying the base to use for representing numeric values.

Security Navigator 3.0: This method is tainted by default for the following objects: `Button`, `Checkbox`, `FileUpload`, `Hidden`, `History`, `Link`, `Location`, `Password`, `Radio`, `Reset`, `Select`, `Submit`, `Text`, and `Textarea`.

For information on data tainting, see “Security” on page 55.

Description Every object has a `toString` method that is automatically called when it is to be represented as a text value or when an object is referred to in a string concatenation. For example, the following examples require `theDog` to be represented as a string:

```
document.write(theDog)
document.write("The dog is " + theDog)
```

You can use `toString` within your own code to convert an object into a string, and you can create your own function to be called in place of the default `toString` method.

Built-in toString methods

Every object type has a built-in `toString` method, which JavaScript calls whenever it needs to convert an object to a string. If an object has no string value and no user-defined `toString` method, `toString` returns `"[object type]"`, where `type` is the object type or the name of the constructor function that created the object. For example, if for an `Image` object named `sealife` defined as shown below, `sealife.toString()` returns `[object Image]`.

```
<IMG NAME="sealife" SRC="images\seaotter.gif" ALIGN="left" VSPACE="10">
```

Some built-in classes have special definitions for their toString methods. See the descriptions of this method for these objects:

`Array`, `Boolean`, `Connection`, `database`, `DbPool`, `Function`, `Number`

User-defined toString methods

You can create a function to be called in place of the default `toString` method. The `toString` method takes no arguments and should return a string. The `toString` method you create can be any value you want, but it will be most useful if it carries information about the object.

The following code defines the `Dog` object type and creates `theDog,` an object of type `Dog`:

```
function Dog(name,breed,color,sex) {
   this.name=name
   this.breed=breed
   this.color=color
   this.sex=sex
}
theDog = new Dog("Gabby","Lab","chocolate","girl")
```

The following code creates `dogToString,` the function that will be used in place of the default `toString` method. This function generates a string containing each property, of the form `"property = value;"`.

```
function dogToString() {
   var ret = "Dog " + this.name + " is ["
   for (var prop in this)
      ret += "  " + prop + " is " + this[prop] + ";"
   return ret + "]"
}
```

The following code assigns the user-defined function to the object's `toString` method:

```
Dog.prototype.toString = dogToString
```

With the preceding code in place, any time `theDog` is used in a string context, JavaScript automatically calls the `dogToString` function, which returns the following string:

> Dog Gabby is [name is Gabby; breed is Lab; color is chocolate; sex is girl; toString is function dogToString() { var ret = "Object " + this.name + " is ["; for (var prop in this) { ret += " " + prop + " is " + this[prop] + ";"; } return ret + "]"; } ;]

An object's `toString` method is usually invoked by JavaScript, but you can invoke it yourself as follows:

```
alert(theDog.toString())
```

Examples **Example 1: The location object.** The following example prints the string equivalent of the current location.

```
document.write("location.toString() is " + location.toString() + "<BR>")
```

The output is as follows:

```
location.toString() is file:///C|/TEMP/myprog.html
```

Example 2: Object with no string value. Assume you have an `Image` object named `sealife` defined as follows:

```
<IMG NAME="sealife" SRC="images\seaotter.gif" ALIGN="left" VSPACE="10">
```

Because the `Image` object itself has no special `toString` method, `sealife.toString()` returns the following:

```
[object Image]
```

Example 3: The radix parameter. The following example prints the string equivalents of the numbers 0 through 9 in decimal and binary.

```
for (x = 0; x < 10; x++) {
   document.write("Decimal: ", x.toString(10), " Binary: ",
      x.toString(2), "<BR>")
}
```

The preceding example produces the following output:

```
Decimal: 0 Binary: 0
Decimal: 1 Binary: 1
Decimal: 2 Binary: 10
Decimal: 3 Binary: 11
Decimal: 4 Binary: 100
Decimal: 5 Binary: 101
Decimal: 6 Binary: 110
Decimal: 7 Binary: 111
Decimal: 8 Binary: 1000
Decimal: 9 Binary: 1001
```

See also `Object.valueOf`

unwatch

Removes a watchpoint set with the `watch` method.

Method of	`Object`
Implemented in	Navigator 4.0, Netscape Server 3.0

Syntax `unwatch(prop)`

Parameters

`prop`	The name of a property of the object.

Description The JavaScript debugger has functionality similar to that provided by this method, as well as other debugging options. For information on the debugger, see *Getting Started with Netscape JavaScript Debugger*[1].

Example See `watch`.

valueOf

Returns the primitive value of the specified object.

Method of	`Object`
Implemented in	Navigator 3.0

Syntax `valueOf()`

Parameters None

Description Every object has a `valueOf` method that is automatically called when it is to be represented as a primitive value. If an object has no primitive value, `valueOf` returns the object itself.

You can use `valueOf` within your own code to convert an object into a primitive value, and you can create your own function to be called in place of the default `valueOf` method.

Every object type has a built-in `valueOf` method, which JavaScript calls whenever it needs to convert an object to a primitive value.

1. http://developer.netscape.com/library/documentation/jsdebug/index.htm

You rarely need to invoke the `valueOf` method yourself. JavaScript automatically invokes it when encountering an object where a primitive value is expected.

Table 4.2 shows the object types for which the `valueOf` method is most useful. Most other objects have no primitive value.

Table 4.2 Use `valueOf` for these object types

Object type	Value returned by **valueOf**
`Number`	Primitive numeric value associated with the object.
`Boolean`	Primitive boolean value associated with the object.
`String`	String associated with the object.
`Function`	Function reference associated with the object. For example, `typeof funObj` returns `"object"`, but `typeof funObj.valueOf()` returns `"function"`.

You can create a function to be called in place of the default `valueOf` method. Your function must take no arguments.

Suppose you have an object type `myNumberType` and you want to create a `valueOf` method for it. The following code assigns a user-defined function to the object's `valueOf` method:

```
myNumberType.prototype.valueOf = new Function(functionText)
```

With the preceding code in place, any time an object of type `myNumberType` is used in a context where it is to be represented as a primitive value, JavaScript automatically calls the function defined in the preceding code.

An object's `valueOf` method is usually invoked by JavaScript, but you can invoke it yourself as follows:

```
myNumber.valueOf()
```

Note Objects in string contexts convert via the `toString` method, which is different from `String` objects converting to string primitives using `valueOf`. All string objects have a string conversion, if only `"[object` *`type`*`]"`. But many objects do not convert to number, boolean, or function.

See also `parseInt`, `Object.toString`

watch

Watches for a property to be assigned a value and runs a function when that occurs.

Method of	`Object`
Implemented in	Navigator 4.0, Netscape Server 3.0

Syntax `watch(prop, handler)`

Parameters

`prop`	The name of a property of the object.
`handler`	A function to call.

Description Watches for assignment to a property named `prop` in this object, calling `handler(prop, oldval, newval)` whenever `prop` is set and storing the return value in that property. A watchpoint can filter (or nullify) the value assignment, by returning a modified `newval` (or `oldval`).

If you delete a property for which a watchpoint has been set, that watchpoint does not disappear. If you later recreate the property, the watchpoint is still in effect.

To remove a watchpoint, use the `unwatch` method.

The JavaScript debugger has functionality similar to that provided by this method, as well as other debugging options. For information on the debugger, see *Getting Started with Netscape JavaScript Debugger*[1].

Example

```
<script language="JavaScript1.2">
o = {p:1}
o.watch("p",
   function (id,oldval,newval) {
      document.writeln("o." + id + " changed from "
         + oldval + " to " + newval)
      return newval
   })

o.p = 2
o.p = 3
delete o.p
o.p = 4
```

1. http://developer.netscape.com/library/documentation/jsdebug/index.htm

```
o.unwatch('p')
o.p = 5

</script>
```

This script displays the following:

```
o.p changed from 1 to 2
o.p changed from 2 to 3
o.p changed from 3 to 4
```

String

An object representing a series of characters in a string.

Core object	
Implemented in	Navigator 2.0: Create a `String` object only by quoting characters. Navigator 3.0, LiveWire 1.0: added `String` constructor; added `prototype` property; added `split` method; added ability to pass strings among scripts in different windows or frames (in previous releases, you had to add an empty string to another window's string to refer to it) Navigator 4.0, Netscape Server 3.0: added `concat`, `match`, `replace`, `search`, `slice`, and `substr` methods.

Created by The `String` constructor:

```
new String(string);
```

Parameters

`string`	Any string.

Description The `String` object is a built-in JavaScript object. You an treat any JavaScript string as a `String` object.

A string can be represented as a literal enclosed by single or double quotation marks; for example, "Netscape" or 'Netscape'.

Property Summary

Property	Description
`length`	Reflects the length of the string.
`prototype`	Allows the addition of properties to a `String` object.

Method Summary

Method	Description
`anchor`	Creates an HTML anchor that is used as a hypertext target.
`big`	Causes a string to be displayed in a big font as if it were in a `BIG` tag.
`blink`	Causes a string to blink as if it were in a `BLINK` tag.
`bold`	Causes a string to be displayed as if it were in a `B` tag.
`charAt`	Returns the character at the specified `index`.
`charCodeAt`	Returns a number indicating the ISO-Latin-1 codeset value of the character at the given index.
`concat`	Combines the text of two strings and returns a new string.
`fixed`	Causes a string to be displayed in fixed-pitch font as if it were in a `TT` tag.
`fontcolor`	Causes a string to be displayed in the specified color as if it were in a `<FONT COLOR=color>` tag.
`fontsize`	Causes a string to be displayed in the specified font size as if it were in a `<FONT SIZE=size>` tag.
`fromCharCode`	Returns a string from the specified sequence of numbers that are ISO-Latin-1 codeset values.
`indexOf`	Returns the index within the calling `String` object of the first occurrence of the specified value.
`italics`	Causes a string to be italic, as if it were in an `I` tag.
`lastIndexOf`	Returns the index within the calling `String` object of the last occurrence of the specified value.
`link`	Creates an HTML hypertext link that requests another URL.
`match`	Used to match a regular expression against a string.

Method	Description
`replace`	Used to find a match between a regular expression and a string, and to replace the matched substring with a new substring.
`search`	Executes the search for a match between a regular expression and a specified string.
`slice`	Extracts a section of a string and returns a new string.
`small`	Causes a string to be displayed in a small font, as if it were in a `SMALL` tag.
`split`	Splits a `String` object into an array of strings by separating the string into substrings.
`strike`	Causes a string to be displayed as struck-out text, as if it were in a `STRIKE` tag.
`sub`	Causes a string to be displayed as a subscript, as if it were in a `SUB` tag.
`substr`	Returns the characters in a string beginning at the specified location through the specified number of characters.
`substring`	Returns the characters in a string between two indexes into the string.
`sup`	Causes a string to be displayed as a superscript, as if it were in a `SUP` tag.
`toLowerCase`	Returns the calling string value converted to lowercase.
`toUpperCase`	Returns the calling string value converted to uppercase.

Examples

Example 1: String variable. The following statement creates a string variable:

```
var last_name = "Schaefer"
```

Example 2: String object properties. The following statements evaluate to 8, `"SCHAEFER,"` and `"schaefer"`:

```
last_name.length
last_name.toUpperCase()
last_name.toLowerCase()
```

Example 3: Accessing individual characters in a string. You can think of a string as an array of characters. In this way, you can access the individual characters in the string by indexing that array. For example, the following code:

```
var myString = "Hello"
document.write ("The first character in the string is " + myString[0])
```

displays “The first character in the string is H”

Example 4: Pass a string among scripts in different windows or frames. The following code creates two string variables and opens a second window:

```
var lastName = new String("Schaefer")
var firstName = new String ("Jesse")
empWindow=window.open('string2.html','window1','width=300,height=300')
```

If the HTML source for the second window (`string2.html`) creates two string variables, `empLastName` and `empFirstName`, the following code in the first window assigns values to the second window’s variables:

```
empWindow.empFirstName=firstName
empWindow.empLastName=lastName
```

The following code in the first window displays the values of the second window’s variables:

```
alert('empFirstName in empWindow is ' + empWindow.empFirstName)
alert('empLastName in empWindow is ' + empWindow.empLastName)
```

Properties

length

The length of the string.

Property of	`String`
Read-only	
Implemented in	Navigator 2.0, LiveWire 1.0

Description For a null string, length is 0.

Examples The following example displays 8 in an Alert dialog box:

```
var x="Netscape"
alert("The string length is " + x.length)
```

prototype

Represents the prototype for this class. You can use the prototype to add properties or methods to all instances of a class. For information on prototypes, see `Function.prototype`.

Property of	`String`
Implemented in	Navigator 3.0, Netscape Server 3.0

Methods

anchor

Creates an HTML anchor that is used as a hypertext target.

Method of	`String`
Implemented in	Navigator 2.0, LiveWire 1.0

Syntax `anchor(nameAttribute)`

Parameters

`nameAttribute`	A string.

Description Use the `anchor` method with the `document.write` or `document.writeln` methods to programmatically create and display an anchor in a document. Create the anchor with the `anchor` method, and then call `write` or `writeln` to display the anchor in a document. In server-side JavaScript, use the `write` function to display the anchor.

In the syntax, the `text` string represents the literal text that you want the user to see. The `nameAttribute` string represents the `NAME` attribute of the `A` tag.

Anchors created with the `anchor` method become elements in the `document.anchors` array.

Examples The following example opens the `msgWindow` window and creates an anchor for the table of contents:

```
var myString="Table of Contents"
msgWindow.document.writeln(myString.anchor("contents_anchor"))
```

The previous example produces the same output as the following HTML:

```
<A NAME="contents_anchor">Table of Contents</A>
```

In server-side JavaScript, you can generate this HTML by calling the `write` function instead of using `document.writeln`.

See also `String.link`

big

Causes a string to be displayed in a big font as if it were in a `BIG` tag.

Method of	`String`
Implemented in	Navigator 2.0, LiveWire 1.0

Syntax `big()`

Parameters None

Description Use the `big` method with the `write` or `writeln` methods to format and display a string in a document. In server-side JavaScript, use the `write` function to display the string.

Examples The following example uses `string` methods to change the size of a string:

```
var worldString="Hello, world"

document.write(worldString.small())
document.write("<P>" + worldString.big())
document.write("<P>" + worldString.fontsize(7))
```

The previous example produces the same output as the following HTML:

```
<SMALL>Hello, world</SMALL>
<P><BIG>Hello, world</BIG>
<P><FONTSIZE=7>Hello, world</FONTSIZE>
```

See also `String.fontsize`, `String.small`

blink

Causes a string to blink as if it were in a `BLINK` tag.

Method of	`String`
Implemented in	Navigator 2.0, LiveWire 1.0

Syntax `blink()`

Parameters None

Description Use the `blink` method with the `write` or `writeln` methods to format and display a string in a document. In server-side JavaScript, use the `write` function to display the string.

Examples The following example uses `string` methods to change the formatting of a string:

```
var worldString="Hello, world"

document.write(worldString.blink())
document.write("<P>" + worldString.bold())
document.write("<P>" + worldString.italics())
document.write("<P>" + worldString.strike())
```

The previous example produces the same output as the following HTML:

```
<BLINK>Hello, world</BLINK>
<P><B>Hello, world</B>
<P><I>Hello, world</I>
<P><STRIKE>Hello, world</STRIKE>
```

See also `String.bold`, `String.italics`, `String.strike`

bold

Causes a string to be displayed as bold as if it were in a `B` tag.

Method of	`String`
Implemented in	Navigator 2.0, LiveWire 1.0

Syntax `bold()`

Parameters None

Description Use the `bold` method with the `write` or `writeln` methods to format and display a string in a document. In server-side JavaScript, use the `write` function to display the string.

Examples The following example uses `string` methods to change the formatting of a string:

```
var worldString="Hello, world"
document.write(worldString.blink())
document.write("<P>" + worldString.bold())
document.write("<P>" + worldString.italics())
document.write("<P>" + worldString.strike())
```

The previous example produces the same output as the following HTML:

```
<BLINK>Hello, world</BLINK>
<P><B>Hello, world</B>
<P><I>Hello, world</I>
<P><STRIKE>Hello, world</STRIKE>
```

See also `String.blink`, `String.italics`, `String.strike`

charAt

Returns the specified character from the string.

Method of	`String`
Implemented in	Navigator 2.0, LiveWire 1.0

Syntax `charAt(index)`

Parameters

`index`	An integer between 0 and 1 less than the length of the string.

Description Characters in a string are indexed from left to right. The index of the first character is 0, and the index of the last character in a string called `stringName` is `stringName.length - 1`. If the `index` you supply is out of range, JavaScript returns an empty string.

Examples The following example displays characters at different locations in the string `"Brave new world"`:

```
var anyString="Brave new world"

document.writeln("The character at index 0 is " + anyString.charAt(0))
document.writeln("The character at index 1 is " + anyString.charAt(1))
document.writeln("The character at index 2 is " + anyString.charAt(2))
document.writeln("The character at index 3 is " + anyString.charAt(3))
document.writeln("The character at index 4 is " + anyString.charAt(4))
```

These lines display the following:

```
The character at index 0 is B
The character at index 1 is r
The character at index 2 is a
The character at index 3 is v
The character at index 4 is e
```

In server-side JavaScript, you can display the same output by calling the `write` function instead of using `document.write`.

See also `String.indexOf`, `String.lastIndexOf`, `String.split`

charCodeAt

Returns a number indicating the ISO-Latin-1 codeset value of the character at the given index.

Method of	`String`
Implemented in	Navigator 4.0, Netscape Server 3.0

Syntax `charCodeAt(index)`

Parameters

`index`	(Optional) An integer between 0 and 1 less than the length of the string. The default value is 0.

Description The ISO-Latin-1 codeset ranges from 0 to 255. The first 0 to 127 are a direct match of the ASCII character set.

Example The following example returns 65, the ISO-Latin-1 codeset value for A.

```
"ABC".charCodeAt(0)
```

concat

Combines the text of two strings and returns a new string.

Method of	`String`
Implemented in	Navigator 4.0, Netscape Server 3.0

Syntax `concat(string2)`

Parameters

`string1`	The first string.
`string2`	The second string.

Description `concat` combines the text from two strings and returns a new string. Changes to the text in one string do not affect the other string.

Example The following example combines two strings into a new string.

```
<SCRIPT>
str1="The morning is upon us. "
str2="The sun is bright."
str3=str1.concat(str2)
document.writeln(str1)
document.writeln(str2)
document.writeln(str3)
</SCRIPT>
```

This writes:

The morning is upon us.
The sun is bright.
The morning is upon us. The sun is bright.

fixed

Causes a string to be displayed in fixed-pitch font as if it were in a `TT` tag.

Method of	`String`
Implemented in	Navigator 2.0, LiveWire 1.0

Syntax `fixed()`

Parameters None

Description Use the `fixed` method with the `write` or `writeln` methods to format and display a string in a document. In server-side JavaScript, use the `write` function to display the string.

Examples The following example uses the `fixed` method to change the formatting of a string:

```
var worldString="Hello, world"
document.write(worldString.fixed())
```

The previous example produces the same output as the following HTML:

```
<TT>Hello, world</TT>
```

fontcolor

Causes a string to be displayed in the specified color as if it were in a `<FONT COLOR=color>` tag.

Method of	`String`
Implemented in	Navigator 2.0, LiveWire 1.0

Syntax `fontcolor(color)`

Parameters

`color`	A string expressing the color as a hexadecimal RGB triplet or as a string literal. String literals for color names are listed in Appendix B, "Color Values," in the *JavaScript Guide*.

Description Use the `fontcolor` method with the `write` or `writeln` methods to format and display a string in a document. In server-side JavaScript, use the `write` function to display the string.

If you express `color` as a hexadecimal RGB triplet, you must use the format `rrggbb`. For example, the hexadecimal RGB values for salmon are `red=FA`, `green=80`, and `blue=72`, so the RGB triplet for `salmon` is `"FA8072"`.

The `fontcolor` method overrides a value set in the `fgColor` property.

Examples The following example uses the `fontcolor` method to change the color of a string:

```
var worldString="Hello, world"

document.write(worldString.fontcolor("maroon") +
   " is maroon in this line")
document.write("<P>" + worldString.fontcolor("salmon") +
   " is salmon in this line")
document.write("<P>" + worldString.fontcolor("red") +
   " is red in this line")

document.write("<P>" + worldString.fontcolor("8000") +
   " is maroon in hexadecimal in this line")
document.write("<P>" + worldString.fontcolor("FA8072") +
   " is salmon in hexadecimal in this line")
```

```
document.write("<P>" + worldString.fontcolor("FF00") +
   " is red in hexadecimal in this line")
```

The previous example produces the same output as the following HTML:

```
<FONT COLOR="maroon">Hello, world</FONT> is maroon in this line
<P><FONT COLOR="salmon">Hello, world</FONT> is salmon in this line
<P><FONT COLOR="red">Hello, world</FONT> is red in this line

<FONT COLOR="8000">Hello, world</FONT>
is maroon in hexadecimal in this line
<P><FONT COLOR="FA8072">Hello, world</FONT>
is salmon in hexadecimal in this line
<P><FONT COLOR="FF00">Hello, world</FONT>
is red in hexadecimal in this line
```

fontsize

Causes a string to be displayed in the specified font size as if it were in a `<FONT SIZE=size>` tag.

Method of	`String`
Implemented in	Navigator 2.0, LiveWire 1.0

Syntax `fontsize(size)`

Parameters

`size`	An integer between 1 and 7, a string representing a signed integer between 1 and 7.

Description Use the `fontsize` method with the `write` or `writeln` methods to format and display a string in a document. In server-side JavaScript, use the `write` function to display the string.

When you specify `size` as an integer, you set the size of `stringName` to one of the 7 defined sizes. When you specify `size` as a string such as `"-2"`, you adjust the font size of `stringName` relative to the size set in the `BASEFONT` tag.

Examples The following example uses `string` methods to change the size of a string:

```
var worldString="Hello, world"

document.write(worldString.small())
document.write("<P>" + worldString.big())
document.write("<P>" + worldString.fontsize(7))
```

The previous example produces the same output as the following HTML:

```
<SMALL>Hello, world</SMALL>
<P><BIG>Hello, world</BIG>
<P><FONTSIZE=7>Hello, world</FONTSIZE>
```

See also `String.big`, `String.small`

fromCharCode

Returns a string created by using the specified sequence ISO-Latin-1 codeset values.

Method of	`String`
Static	
Implemented in	Navigator 4.0, Netscape Server 3.0

Syntax `fromCharCode(num1, ..., numN)`

Parameters

`num1, ..., numN`	A sequence of numbers that are ISO-Latin-1 codeset values.

Description This method returns a string and not a `String` object.

Because `fromCharCode` is a static method of `String`, you always use it as `String.fromCharCode()`, rather than as a method of a `String` object you created.

Examples **Example 1**. The following example returns the string "ABC".

```
String.fromCharCode(65,66,67)
```

Example 2. The `which` property of the `KeyDown`, `KeyPress`, and `KeyUp` events contains the ASCII value of the key pressed at the time the event occurred. If you want to get the actual letter, number, or symbol of the key, you can use `fromCharCode`. The following example returns the letter, number, or symbol of the KeyPress event's `which` property.

```
String.fromCharCode(KeyPress.which)
```

indexOf

Returns the index within the calling `String` object of the first occurrence of the specified value, starting the search at `fromIndex`, or -1 if the value is not found.

Method of	`String`
Implemented in	Navigator 2.0, LiveWire 1.0

Syntax `indexOf(searchValue, fromIndex)`

Parameters

`searchValue`	A string representing the value to search for.
`fromIndex`	(Optional) The location within the calling string to start the search from. It can be any integer between 0 and 1 less than the length of the string. The default value is 0.

Description Characters in a string are indexed from left to right. The index of the first character is 0, and the index of the last character of a string called `stringName` is `stringName.length - 1`.

If `stringName` contains an empty string (`""`), `indexOf` returns an empty string.

The `indexOf` method is case sensitive. For example, the following expression returns -1:

```
"Blue Whale".indexOf("blue")
```

Examples **Example 1.** The following example uses `indexOf` and `lastIndexOf` to locate values in the string `"Brave new world."`

```
var anyString="Brave new world"

//Displays 8
document.write("<P>The index of the first w from the beginning is " +
   anyString.indexOf("w"))
//Displays 10
document.write("<P>The index of the first w from the end is " +
   anyString.lastIndexOf("w"))
//Displays 6
document.write("<P>The index of 'new' from the beginning is " +
   anyString.indexOf("new"))
//Displays 6
document.write("<P>The index of 'new' from the end is " +
   anyString.lastIndexOf("new"))
```

Example 2. The following example defines two string variables. The variables contain the same string except that the second string contains uppercase letters. The first `writeln` method displays 19. But because the `indexOf` method is case sensitive, the string `"cheddar"` is not found in `myCapString`, so the second `writeln` method displays -1.

```
myString="brie, pepper jack, cheddar"
myCapString="Brie, Pepper Jack, Cheddar"
document.writeln('myString.indexOf("cheddar") is ' +
   myString.indexOf("cheddar"))
document.writeln('<P>myCapString.indexOf("cheddar") is ' +
   myCapString.indexOf("cheddar"))
```

Example 3. The following example sets `count` to the number of occurrences of the letter `x` in the string `str`:

```
count = 0;
pos = str.indexOf("x");
while ( pos != -1 ) {
   count++;
   pos = str.indexOf("x",pos+1);
}
```

See also `String.charAt`, `String.lastIndexOf`, `String.split`

italics

Causes a string to be italic, as if it were in an `I` tag.

Method of	`String`
Implemented in	Navigator 2.0, LiveWire 1.0

Syntax `italics()`

Parameters None

Description Use the `italics` method with the `write` or `writeln` methods to format and display a string in a document. In server-side JavaScript, use the `write` function to display the string.

Examples The following example uses `string` methods to change the formatting of a string:

```
var worldString="Hello, world"

document.write(worldString.blink())
document.write("<P>" + worldString.bold())
document.write("<P>" + worldString.italics())
document.write("<P>" + worldString.strike())
```

The previous example produces the same output as the following HTML:

```
<BLINK>Hello, world</BLINK>
<P><B>Hello, world</B>
<P><I>Hello, world</I>
<P><STRIKE>Hello, world</STRIKE>
```

See also `String.blink`, `String.bold`, `String.strike`

lastIndexOf

Returns the index within the calling `String` object of the last occurrence of the specified value. The calling string is searched backward, starting at `fromIndex`, or -1 if not found.

Method of	`String`
Implemented in	Navigator 2.0, LiveWire 1.0

Syntax `lastIndexOf(searchValue, fromIndex)`

Parameters

`searchValue`	A string representing the value to search for.
`fromIndex`	(Optional) The location within the calling string to start the search from. It can be any integer between 0 and 1 less than the length of the string. The default value is 1 less than the length of the string.

Description Characters in a string are indexed from left to right. The index of the first character is 0, and the index of the last character is `stringName`.length - 1.

The `lastIndexOf` method is case sensitive. For example, the following expression returns -1:

```
"Blue Whale, Killer Whale".lastIndexOf("blue")
```

Examples The following example uses `indexOf` and `lastIndexOf` to locate values in the string `"Brave new world."`

```
var anyString="Brave new world"

//Displays 8
document.write("<P>The index of the first w from the beginning is " +
   anyString.indexOf("w"))
//Displays 10
document.write("<P>The index of the first w from the end is " +
   anyString.lastIndexOf("w"))
//Displays 6
document.write("<P>The index of 'new' from the beginning is " +
   anyString.indexOf("new"))
//Displays 6
document.write("<P>The index of 'new' from the end is " +
   anyString.lastIndexOf("new"))
```

In server-side JavaScript, you can display the same output by calling the `write` function instead of using `document.write`.

See also `String.charAt`, `String.indexOf`, `String.split`

link

Creates an HTML hypertext link that requests another URL.

Method of	`String`
Implemented in	Navigator 2.0, LiveWire 1.0

Syntax `link(hrefAttribute)`

Parameters

`hrefAttribute`	Any string that specifies the `HREF` attribute of the `A` tag; it should be a valid URL (relative or absolute).

Description Use the `link` method to programmatically create a hypertext link, and then call `write` or `writeln` to display the link in a document. In server-side JavaScript, use the `write` function to display the link.

Links created with the `link` method become elements in the `links` array of the `document` object. See `document.links`.

Examples The following example displays the word "Netscape" as a hypertext link that returns the user to the Netscape home page:

```
var hotText="Netscape"
var URL="http://home.netscape.com"

document.write("Click to return to " + hotText.link(URL))
```

The previous example produces the same output as the following HTML:

```
Click to return to <A HREF="http://home.netscape.com">Netscape</A>
```

See also `Anchor`

match

Used to match a regular expression against a string.

Method of	`String`
Implemented in	Navigator 4.0

Syntax `match(regexp)`

Parameters

`regexp`	Name of the regular expression. It can be a variable name or a literal.

Description If you want to execute a global match, or a case insensitive match, include the `g` (for global) and `i` (for ignore case) flags in the regular expression. These can be included separately or together. The following two examples below show how to use these flags with `match`.

Note If you execute a match simply to find true or false, use `String.search` or the regular expression `test` method.

Examples **Example 1**. In the following example, `match` is used to find 'Chapter' followed by 1 or more numeric characters followed by a decimal point and numeric character 0 or more times. The regular expression includes the `i` flag so that case will be ignored.

```
<SCRIPT>
str = "For more information, see Chapter 3.4.5.1";
re = /(chapter \d+(\.\d)*)/i;
found = str.match(re);
document.write(found);
</SCRIPT>
```

This returns the array containing Chapter 3.4.5.1,Chapter 3.4.5.1,.1

`'Chapter 3.4.5.1'` is the first match and the first value remembered from `(Chapter \d+(\.\d)*)`.

`'.1'` is the second value remembered from `(\.\d)`.

Example 2. The following example demonstrates the use of the global and ignore case flags with `match`.

```
<SCRIPT>
str = "abcDdcba";
newArray = str.match(/d/gi);
document.write(newArray);
</SCRIPT>
```

The returned array contains D, d.

replace

Used to find a match between a regular expression and a string, and to replace the matched substring with a new substring.

Method of	`String`
Implemented in	Navigator 4.0

Syntax `replace(regexp, newSubStr)`

Parameters

`regexp`	The name of the regular expression. It can be a variable name or a literal.
`newSubStr`	The string to put in place of the string found with `regexp`. This string can include the `RegExp` properties `$1, ..., $9`, `lastMatch`, `lastParen`, `leftContext`, and `rightContext`.

Description This method does not change the `String` object it is called on; it simply returns a new string.

If you want to execute a global search and replace, or a case insensitive search, include the `g` (for global) and `i` (for ignore case) flags in the regular expression. These can be included separately or together. The following two examples below show how to use these flags with `replace`.

Examples **Example 1**. In the following example, the regular expression includes the global and ignore case flags which permits `replace` to replace each occurrence of 'apples' in the string with 'oranges.'

```
<SCRIPT>
re = /apples/gi;
str = "Apples are round, and apples are juicy.";
newstr=str.replace(re, "oranges");
```

```
document.write(newstr)
</SCRIPT>
```

This prints "oranges are round, and oranges are juicy."

Example 2. In the following example, the regular expression is defined in `replace` and includes the ignore case flag.

```
<SCRIPT>
str = "Twas the night before Xmas...";
newstr=str.replace(/xmas/i, "Christmas");
document.write(newstr)
</SCRIPT>
```

This prints "Twas the night before Christmas..."

Example 3. The following script switches the words in the string. For the replacement text, the script uses the values of the `$1` and `$2` properties.

```
<SCRIPT LANGUAGE="JavaScript1.2">
re = /(\w+)\s(\w+)/;
str = "John Smith";
newstr = str.replace(re, "$2, $1");
document.write(newstr)
</SCRIPT>
```

This prints "Smith, John".

search

Executes the search for a match between a regular expression and this `String` object.

Method of	`String`
Implemented in	Navigator 4.0

Syntax `search(regexp)`

Parameters

`regexp`	Name of the regular expression. It can be a variable name or a literal.

Description If successful, `search` returns the index of the regular expression inside the string. Otherwise, it returns -1.

When you want to know whether a pattern is found in a string use `search` (similar to the regular expression `test` method); for more information (but slower execution) use `match` (similar to the regular expression `exec` method).

Example The following example prints a message which depends on the success of the test.

```
function testinput(re, str){
   if (str.search(re) != -1)
      midstring = " contains ";
   else
      midstring = " does not contain ";
   document.write (str + midstring + re.source);
}
```

slice

Extracts a section of a string and returns a new string.

Method of	`String`
Implemented in	Navigator 2.0, LiveWire 1.0

Syntax `slice(beginslice,endSlice)`

Parameters

`beginSlice`	The zero-based index at which to begin extraction.
`endSlice`	(Optional) The zero-based index at which to end extraction. If omitted, `slice` extracts to the end of the string.

Description `slice` extracts the text from one string and returns a new string. Changes to the text in one string do not affect the other string.

`slice` extracts up to but not including `endSlice`. `string.slice(1,4)` extracts the second character through the fourth character (characters indexed 1, 2, and 3).

As a negative index, `endSlice` indicates an offset from the end of the string. `string.slice(2,-1)` extracts the third character through the second to last character in the string.

Example The following example uses `slice` to create a new string.

```
<SCRIPT>
str1="The morning is upon us. "
```

```
str2=str1.slice(3,-5)
document.write(str2)
</SCRIPT>
```

This writes:

morning is upon

small

Causes a string to be displayed in a small font, as if it were in a `SMALL` tag.

Method of	`String`
Implemented in	Navigator 2.0, LiveWire 1.0

Syntax `small()`

Parameters None

Description Use the `small` method with the `write` or `writeln` methods to format and display a string in a document. In server-side JavaScript, use the `write` function to display the string.

Examples The following example uses `string` methods to change the size of a string:

```
var worldString="Hello, world"

document.write(worldString.small())
document.write("<P>" + worldString.big())
document.write("<P>" + worldString.fontsize(7))
```

The previous example produces the same output as the following HTML:

```
<SMALL>Hello, world</SMALL>
<P><BIG>Hello, world</BIG>
<P><FONTSIZE=7>Hello, world</FONTSIZE>
```

See also `String.big`, `String.fontsize`

split

Splits a `String` object into an array of strings by separating the string into substrings.

Method of	`String`

Implemented in Navigator 3.0, LiveWire 1.0

Syntax `split(separator, limit)`

Parameters

`separator`	(Optional) Specifies the character to use for separating the string. The `separator` is treated as a string. If `separator` is omitted, the array returned contains one element consisting of the entire string.
`limit`	(Optional) Integer specifying a limit on the number of splits to be found.

Description The `split` method returns the new array.

When found, `separator` is removed from the string and the substrings are returned in an array. If `separator` is omitted, the array contains one element consisting of the entire string.

In Navigator 4.0, `Split` has the following additions:

- It can take a regular expression argument, as well as a fixed string, by which to split the object string. If `separator` is a regular expression, any included parenthesis cause submatches to be included in the returned array.
- It can take a limit count so that it won't include trailing empty elements in the resulting array.
- If you specify `LANGUAGE="JavaScript1.2"` in the `SCRIPT` tag, `string.split(" ")` splits on any run of 1 or more white space characters including spaces, tabs, line feeds, and carriage returns.

Examples **Example 1**. The following example defines a function that splits a string into an array of strings using the specified separator. After splitting the string, the function displays messages indicating the original string (before the split), the separator used, the number of elements in the array, and the individual array elements.

```
function splitString (stringToSplit,separator) {
   arrayOfStrings = stringToSplit.split(separator)
   document.write ('<P>The original string is: "' + stringToSplit + '"')
   document.write ('<BR>The separator is: "' + separator + '"')
   document.write ("<BR>The array has " + arrayOfStrings.length + " elements: ")

   for (var i=0; i < arrayOfStrings.length; i++) {
      document.write (arrayOfStrings[i] + " / ")
   }
}

var tempestString="Oh brave new world that has such people in it."
var monthString="Jan,Feb,Mar,Apr,May,Jun,Jul,Aug,Sep,Oct,Nov,Dec"

var space=" "
var comma=","

splitString(tempestString,space)
splitString(tempestString)
splitString(monthString,comma)
```

This example produces the following output:

```
The original string is: "Oh brave new world that has such people in it."
The separator is: " "
The array has 10 elements: Oh / brave / new / world / that / has / such / people / in / it.
/

The original string is: "Oh brave new world that has such people in it."
The separator is: "undefined"
The array has 1 elements: Oh brave new world that has such people in it. /

The original string is: "Jan,Feb,Mar,Apr,May,Jun,Jul,Aug,Sep,Oct,Nov,Dec"
The separator is: ","
The array has 12 elements: Jan / Feb / Mar / Apr / May / Jun / Jul / Aug / Sep / Oct / Nov
/ Dec /
```

Example 2. Consider the following script:

```
<SCRIPT LANGUAGE="JavaScript1.2">
str="She sells    seashells \nby   the\n seashore"
document.write(str + "<BR>")
a=str.split(" ")
document.write(a)
</SCRIPT>
```

Using `LANGUAGE="JavaScript1.2"`, this script produces

```
"She", "sells", "seashells", "by", "the", "seashore"
```

Without `LANGUAGE="JavaScript1.2"`, this script splits only on single space characters, producing

```
"She", "sells", , , , "seashells", "by", , , "the", "seashore"
```

Example 3. In the following example, `split` looks for 0 or more spaces followed by a semicolon followed by 0 or more spaces and, when found, removes the spaces from the string. `nameList` is the array returned as a result of `split`.

```
<SCRIPT>
names = "Harry  Trump  ;Fred Barney; Helen   Rigby ;  Bill Abel ;Chris
Hand ";
document.write (names + "<BR>" + "<BR>");
re = /\s*;\s*/;
nameList = names.split (re);
document.write(nameList);
</SCRIPT>
```

This prints two lines; the first line prints the original string, and the second line prints the resulting array.

Harry Trump ;Fred Barney; Helen Rigby ; Bill Abel ;Chris Hand
Harry Trump,Fred Barney,Helen Rigby,Bill Abel,Chris Hand

Example 4. In the following example, `split` looks for 0 or more spaces in a string and returns the first 3 splits that it finds.

```
<SCRIPT LANGUAGE="JavaScript1.2">
myVar = "  Hello World. How are you doing?    ";
splits = myVar.split(" ", 3);
document.write(splits)
</SCRIPT>
```

This script displays the following:

```
["Hello", "World.", "How"]
```

See also `String.charAt`, `String.indexOf`, `String.lastIndexOf`

strike

Causes a string to be displayed as struck-out text, as if it were in a `STRIKE` tag.

Method of	`String`
Implemented in	Navigator 2.0, LiveWire 1.0

Syntax `strike()`

Parameters None

Description Use the `strike` method with the `write` or `writeln` methods to format and display a string in a document. In server-side JavaScript, use the `write` function to display the string.

Examples The following example uses `string` methods to change the formatting of a string:

```
var worldString="Hello, world"

document.write(worldString.blink())
document.write("<P>" + worldString.bold())
document.write("<P>" + worldString.italics())
document.write("<P>" + worldString.strike())
```

The previous example produces the same output as the following HTML:

```
<BLINK>Hello, world</BLINK>
<P><B>Hello, world</B>
<P><I>Hello, world</I>
<P><STRIKE>Hello, world</STRIKE>
```

See also `String.blink`, `String.bold`, `String.italics`

sub

Causes a string to be displayed as a subscript, as if it were in a `SUB` tag.

Method of	`String`
Implemented in	Navigator 2.0, LiveWire 1.0

Syntax `sub()`

Parameters None

Description Use the `sub` method with the `write` or `writeln` methods to format and display a string in a document. In server-side JavaScript, use the `write` function to generate the HTML.

Examples The following example uses the `sub` and `sup` methods to format a string:

```
var superText="superscript"
var subText="subscript"

document.write("This is what a " + superText.sup() + " looks like.")
document.write("<P>This is what a " + subText.sub() + " looks like.")
```

The previous example produces the same output as the following HTML:

```
This is what a <SUP>superscript</SUP> looks like.
<P>This is what a <SUB>subscript</SUB> looks like.
```

See also `String.sup`

substr

Returns the characters in a string beginning at the specified location through the specified number of characters.

Method of	`String`
Implemented in	Navigator 2.0, LiveWire 1.0

Syntax `substr(start, length)`

Parameters

`start`	Location at which to begin extracting characters.
`length`	(Optional) The number of characters to extract

Description `start` is a character index. The index of the first character is 0, and the index of the last character is 1 less than the length of the string. `substr` begins extracting characters at `start` and collects `length` number of characters.

If `start` is positive and is the length of the string or longer, `substr` returns no characters.

If `start` is negative, `substr` uses it as a character index from the end of the string. If `start` is negative and `abs(start)` is larger than the length of the string, `substr` uses 0 is the start index.

If `length` is 0 or negative, `substr` returns no characters. If `length` is omitted, `start` extracts characters to the end of the string.

Example Consider the following script:

```
<SCRIPT LANGUAGE="JavaScript1.2">

str = "abcdefghij"
document.writeln("(1,2): ", str.substr(1,2))
document.writeln("(-2,2): ", str.substr(-2,2))
document.writeln("(1): ", str.substr(1))
document.writeln("(-20, 2): ", str.substr(1,20))
document.writeln("(20, 2): ", str.substr(20,2))

</SCRIPT>
```

This script displays:

```
(1,2): bc
(-2,2): ij
```

```
(1): bcdefghij
(-20, 2): bcdefghij
(20, 2):
```

See also `substring`

substring

Returns a subset of a `String` object.

Method of	`String`
Implemented in	Navigator 2.0, LiveWire 1.0

Syntax `substring(indexA, indexB)`

Parameters

`indexA`	An integer between 0 and 1 less than the length of the string.
`indexB`	An integer between 0 and 1 less than the length of the string.

Description `substring` extracts characters from `indexA` up to but not including `indexB`. In particular:

- If `indexA` is less than 0, `indexA` is treated as if it were 0.
- If `indexB` is greater than `stringName.length`, `indexB` is treated as if it were `stringName.length`.
- If `indexA` equals `indexB`, `substring` returns an empty string.
- If `indexB` is omitted, `indexA` extracts characters to the end of the string.

Using `LANGUAGE="JavaScript1.2"` in the `SCRIPT` tag,

- If `indexA` is greater than `indexB`, JavaScript produces a runtime error (out of memory).

Without `LANGUAGE="JavaScript1.2"`,

- If `indexA` is greater than `indexB`, JavaScript returns a substring beginning with `indexB` and ending with `indexA - 1`.

Examples **Example 1.** The following example uses `substring` to display characters from the string `"Netscape"`:

```
var anyString="Netscape"

//Displays "Net"
document.write(anyString.substring(0,3))
document.write(anyString.substring(3,0))
//Displays "cap"
document.write(anyString.substring(4,7))
document.write(anyString.substring(7,4))
//Displays "Netscap"
document.write(anyString.substring(0,7))
//Displays "Netscape"
document.write(anyString.substring(0,8))
document.write(anyString.substring(0,10))
```

Example 2. The following example replaces a substring within a string. It will replace both individual characters and substrings. The function call at the end of the example changes the string `"Brave New World"` into `"Brave New Web"`.

```
function replaceString(oldS,newS,fullS) {
// Replaces oldS with newS in the string fullS
   for (var i=0; i<fullS.length; i++) {
      if (fullS.substring(i,i+oldS.length) == oldS) {
         fullS = fullS.substring(0,i)+newS+fullS.substring(i+oldS.length,fullS.length)
      }
   }
   return fullS
}

replaceString("World","Web","Brave New World")
```

Example 3. Using `LANGUAGE="JavaScript1.2"`, the following script produces a runtime error (out of memory).

```
<SCRIPT LANGUAGE="JavaScript1.2">
str="Netscape"
document.write(str.substring(0,3);
document.write(str.substring(3,0);
</SCRIPT>
```

Without `LANGUAGE="JavaScript1.2"`, the above script prints

Net Net

In the second `write`, the index numbers are swapped.

See also `substr`

sup

Causes a string to be displayed as a superscript, as if it were in a `SUP` tag.

Method of	`String`
Implemented in	Navigator 2.0, LiveWire 1.0

Syntax `sup()`

Parameters None

Description Use the `sup` method with the `write` or `writeln` methods to format and display a string in a document. In server-side JavaScript, use the `write` function to generate the HTML.

Examples The following example uses the `sub` and `sup` methods to format a string:

```
var superText="superscript"
var subText="subscript"

document.write("This is what a " + superText.sup() + " looks like.")
document.write("<P>This is what a " + subText.sub() + " looks like.")
```

The previous example produces the same output as the following HTML:

```
This is what a <SUP>superscript</SUP> looks like.
<P>This is what a <SUB>subscript</SUB> looks like.
```

See also `String.sub`

toLowerCase

Returns the calling string value converted to lowercase.

Method of	`String`
Implemented in	Navigator 2.0, LiveWire 1.0

Syntax `toLowerCase()`

Parameters None

Description The `toLowerCase` method returns the value of the string converted to lowercase. `toLowerCase` does not affect the value of the string itself.

Examples The following example displays the lowercase string `"alphabet"`:

```
var upperText="ALPHABET"
document.write(upperText.toLowerCase())
```

See also `String.toUpperCase`

toUpperCase

Returns the calling string value converted to uppercase.

Method of	`String`
Implemented in	Navigator 2.0, LiveWire 1.0

Syntax `toUpperCase()`

Parameters None

Description The `toUpperCase` method returns the value of the string converted to uppercase. `toUpperCase` does not affect the value of the string itself.

Examples The following example displays the string `"ALPHABET"`:

```
var lowerText="alphabet"
document.write(lowerText.toUpperCase())
```

See also `String.toLowerCase`

RegExp

A regular expression object contains the pattern of a regular expression. It has properties and methods for using that regular expression to find and replace matches in strings.

In addition to the properties of an individual regular expression object that you create using the `RegExp` constructor function, the predefined `RegExp` object has static properties that are set whenever any regular expression is used.

Core object	
Implemented in	Navigator 4.0, Netscape Server 3.0

Created by A literal text format or the `RegExp` constructor function.

The literal format is used as follows:

/pattern/flags

The constructor function is used as follows:

new RegExp("*pattern*", "*flags*")

Parameters

`pattern`	The text of the regular expression.
`flags`	(Optional) If specified, flags can have one of the following 3 values: • `g`: global match • `i`: ignore case • `gi`: both global match and ignore case

Notice that the parameters to the literal format do not use quotation marks to indicate strings, while the parameters to the constructor function do use quotation marks. So the following expressions create the same regular expression:

```
/ab+c/i
new RegExp("ab+c", "i")
```

Description When using the constructor function, the normal string escape rules (preceding special characters with \ when included in a string) are necessary. For example, the following are equivalent:

```
re = new RegExp("\\w+")
re = /\w+/
```

Table 4.3 provides a complete list and description of the special characters that can be used in regular expressions.

Table 4.3 Special characters in regular expressions.

Character	Meaning
`\`	For characters that are usually treated literally, indicates that the next character is special and not to be interpreted literally. For example, `/b/` matches the character 'b'. By placing a backslash in front of b, that is by using `/\b/`, the character becomes special to mean match a word boundary. -or- For characters that are usually treated specially, indicates that the next character is not special and should be interpreted literally. For example, `*` is a special character that means 0 or more occurrences of the preceding character should be matched; for example, `/a*/` means match 0 or more a's. To match `*` literally, precede the it with a backslash; for example, `/a\*/` matches 'a*'.
`^`	Matches beginning of input or line. For example, `/^A/` does not match the 'A' in "an A," but does match it in "An A."
`$`	Matches end of input or line. For example, `/t$/` does not match the 't' in "eater", but does match it in "eat"
`*`	Matches the preceding character 0 or more times. For example, `/bo*/` matches 'boooo' in "A ghost booooed" and 'b' in "A bird warbled", but nothing in "A goat grunted".
`+`	Matches the preceding character 1 or more times. Equivalent to `{1,}`. For example, `/a+/` matches the 'a' in "candy" and all the a's in "caaaaaaandy."
`?`	Matches the preceding character 0 or 1 time. For example, `/e?le?/` matches the 'el' in "angel" and the 'le' in "angle."
`.`	(The decimal point) matches any single character except the newline character. For example, `/.n/` matches 'an' and 'on' in "nay, an apple is on the tree", but not 'nay'.

Table 4.3 Special characters in regular expressions. (Continued)

Character	Meaning
`(x)`	Matches 'x' and remembers the match. For example, `/(foo)/` matches and remembers 'foo' in "foo bar." The matched substring can be recalled from the resulting array's elements `[1]`, ..., `[n]`, or from the predefined `RegExp` object's properties `$1`, ..., `$9`.
`x|y`	Matches either 'x' or 'y'. For example, `/green|red/` matches 'green' in "green apple" and 'red' in "red apple."
`{n}`	Where `n` is a positive integer. Matches exactly `n` occurrences of the preceding character. For example, `/a{2}/` doesn't match the 'a' in "candy," but it matches all of the a's in "caandy," and the first two a's in "caaandy."
`{n,}`	Where `n` is a positive integer. Matches at least `n` occurrences of the preceding character. For example, `/a{2,}` doesn't match the 'a' in "candy", but matches all of the a's in "caandy" and in "caaaaaaandy."
`{n,m}`	Where `n` and `m` are positive integers. Matches at least `n` and at most `m` occurrences of the preceding character. For example, `/a{1,3}/` matches nothing in "cndy", the 'a' in "candy," the first two a's in "caandy," and the first three a's in "caaaaaaandy" Notice that when matching "caaaaaaandy", the match is "aaa", even though the original string had more a's in it.
`[xyz]`	A character set. Matches any one of the enclosed characters. You can specify a range of characters by using a hyphen. For example, `[abcd]` is the same as `[a-c]`. They match the 'b' in "brisket" and the 'c' in "ache".
`[^xyz]`	A negated or complemented character set. That is, it matches anything that is not enclosed in the brackets. You can specify a range of characters by using a hyphen. For example, `[^abc]` is the same as `[^a-c]`. They initially match 'r' in "brisket" and 'h' in "chop."
`[\b]`	Matches a backspace. (Not to be confused with `\b`.)
`\b`	Matches a word boundary, such as a space. (Not to be confused with `[\b]`.) For example, `/\bn\w/` matches the 'no' in "noonday";`/\wy\b/` matches the 'ly' in "possibly yesterday."

Table 4.3 Special characters in regular expressions. (Continued)

Character	Meaning
`\B`	Matches a non-word boundary. For example, `/\w\Bn/` matches 'on' in "noonday", and `/y\B\w/` matches 'ye' in "possibly yesterday."
`\c`*X*	Where *X* is a control character. Matches a control character in a string. For example, `/\cM/` matches control-M in a string.
`\d`	Matches a digit character. Equivalent to `[0-9]`. For example, `/\d/` or `/[0-9]/` matches '2' in "B2 is the suite number."
`\D`	Matches any non-digit character. Equivalent to `[^0-9]`. For example, `/\D/` or `/[^0-9]/` matches 'B' in "B2 is the suite number."
`\f`	Matches a form-feed.
`\n`	Matches a linefeed.
`\r`	Matches a carriage return.
`\s`	Matches a single white space character, including space, tab, form feed, line feed. Equivalent to `[ \f\n\r\t\v]`. for example, `/\s\w*/` matches ' bar' in "foo bar."
`\S`	Matches a single character other than white space. Equivalent to `[^ \f\n\r\t\v]`. For example, `/\S/\w*` matches 'foo' in "foo bar."
`\t`	Matches a tab
`\v`	Matches a vertical tab.
`\w`	Matches any alphanumeric character including the underscore. Equivalent to `[A-Za-z0-9_]`. For example, `/\w/` matches 'a' in "apple," '5' in "$5.28," and '3' in "3D."
`\W`	Matches any non-word character. Equivalent to `[^A-Za-z0-9_]`. For example, `/\W/` or `/[^$A-Za-z0-9_]/` matches '%' in "50%."

Table 4.3 Special characters in regular expressions. (Continued)

Character	Meaning
`\`*n*	Where *n* is a positive integer. A back reference to the last substring matching the *n* parenthetical in the regular expression (counting left parentheses). For example, `/apple(,)\sorange\1/` matches 'apple, orange', in "apple, orange, cherry, peach." A more complete example follows this table. **Note:** If the number of left parentheses is less than the number specified in \n, the \n is taken as an octal escape as described in the next row.
`\o`*octal* `\x`*hex*	Where `\o`*octal* is an octal escape value or `\x`*hex* is a hexadecimal escape value. Allows you to embed ASCII codes into regular expressions.

The literal notation provides compilation of the regular expression when the expression is evaluated. Use literal notation when the regular expression will remain constant. For example, if you use literal notation to construct a regular expression used in a loop, the regular expression won't be recompiled on each iteration.

The constructor of the regular expression object, for example, `new RegExp("ab+c")`, provides runtime compilation of the regular expression. Use the constructor function when you know the regular expression pattern will be changing, or you don't know the pattern and are getting it from another source, such as user input. Once you have a defined regular expression, and if the regular expression is used throughout the script and may change, you can use the `compile` method to compile a new regular expression for efficient reuse.

A separate predefined `RegExp` object is available in each window; that is, each separate thread of JavaScript execution gets its own `RegExp` object. Because each script runs to completion without interruption in a thread, this assures that different scripts do not overwrite values of the `RegExp` object.

The predefined `RegExp` object contains the static properties `input`, `multiline`, `lastMatch`, `lastParen`, `leftContext`, `rightContext`, and `$1` through `$9`. The `input` and `multiline` properties can be preset. The values for the other static properties are set after execution of the `exec` and `test` methods of an individual regular expression object, and after execution of the `match` and `replace` methods of `String`.

Property Summary

Note that several of the `RegExp` properties have both long and short (Perl-like) names. Both names always refer to the same value. Perl is the programming language from which JavaScript modeled its regular expressions.

Property	Description
`$1, ..., $9`	Parenthesized substring matches, if any.
`$_`	See `input.`
`$*`	See `multiline.`
`$&`	See `lastMatch.`
`$+`	See `lastParen.`
`$‘`	See `leftContext.`
`$’`	See `rightContext.`
`global`	Whether or not to test the regular expression against all possible matches in a string, or only against the first.
`ignoreCase`	Whether or not to ignore case while attempting a match in a string.
`input`	The string against which a regular expression is matched.
`lastIndex`	The index at which to start the next match.
`lastMatch`	The last matched characters.
`lastParen`	The last parenthesized substring match, if any.
`leftContext`	The substring preceding the most recent match.
`multiline`	Whether or not to search in strings across multiple lines.
`rightContext`	The substring following the most recent match.
`source`	The text of the pattern.

Method Summary

Method	Description
`compile`	Compiles a regular expression object.
`exec`	Executes a search for a match in its string parameter.
`test`	Tests for a match in its string parameter.

Examples **Example 1.** The following script uses the `replace` method to switch the words in the string. For the replacement text, the script uses the values of the `$1` and `$2` properties of the global `RegExp` object. Note that the `RegExp` object name is not be prepended to the `$` properties when they are passed as the second argument to the `replace` method.

```
<SCRIPT LANGUAGE="JavaScript1.2">
re = /(\w+)\s(\w+)/;
str = "John Smith";
newstr=str.replace(re, "$2, $1");
document.write(newstr)
</SCRIPT>
```

This displays "Smith, John".

Example 2. In the following example, `RegExp.input` is set by the Change event. In the `getInfo` function, the `exec` method uses the value of `RegExp.input` as its argument. Note that `RegExp` is prepended to the `$` properties.

```
<HTML>

<SCRIPT LANGUAGE="JavaScript1.2">
function getInfo() {
   re = /(\w+)\s(\d+)/;
   re.exec();
   window.alert(RegExp.$1 + ", your age is " + RegExp.$2);
}
</SCRIPT>

Enter your first name and your age, and then press Enter.

<FORM>
<INPUT TYPE:"TEXT" NAME="NameAge" onChange="getInfo(this);">
</FORM>

</HTML>
```

Properties

$1, ..., $9

Properties that contain parenthesized substring matches, if any.

Property of	`RegExp`
Static, Read-only	

Implemented in Navigator 4.0, Netscape Server 3.0

Description Because `input` is static, it is not a property of an individual regular expression object. Instead, you always use it as `RegExp.input`.

The number of possible parenthesized substrings is unlimited, but the predefined `RegExp` object can only hold the last nine. You can access all parenthesized substrings through the returned array's indexes.

These properties can be used in the replacement text for the `String.replace` method. When used this way, do not prepend them with `RegExp`. The example below illustrates this. When parentheses are not included in the regular expression, the script interprets *`$n`*'s literally (where *`n`* is a positive integer).

Examples The following script uses the `replace` method to switch the words in the string. For the replacement text, the script uses the values of the `$1` and `$2` properties of the global `RegExp` object. Note that the `RegExp` object name is not be prepended to the `$` properties when they are passed as the second argument to the `replace` method.

```
<SCRIPT LANGUAGE="JavaScript1.2">
re = /(\w+)\s(\w+)/;
str = "John Smith";
newstr=str.replace(re, "$2, $1");
document.write(newstr)
</SCRIPT>
```

This displays "Smith, John".

$_

See `input`.

$*

See `multiline`.

$&

See `lastMatch`.

$+

See `lastParen`.

$‘

See `leftContext`.

$’

See `rightContext`.

global

Whether or not the `"g"` flag is used with the regular expression.

Property of	`RegExp`
Read-only	
Implemented in	Navigator 4.0, Netscape Server 3.0

Description `global` is a property of an individual regular expression object.

The value of `global` is `true` if the `"g"` flag was used; otherwise, `false`. The `"g"` flag indicates that the regular expression should be tested against all possible matches in a string.

You cannot change this property directly. However, calling the `compile` method changes the value of this property.

ignoreCase

Whether or not the `"i"` flag is used with the regular expression.

Property of	`RegExp`
Read-only	
Implemented in	Navigator 4.0, Netscape Server 3.0

Description `ignoreCase` is a property of an individual regular expression object.

The value of `ignoreCase` is `true` if the `"i"` flag was used; otherwise, `false`. The `"i"` flag indicates that case should be ignored while attempting a match in a string.

You cannot change this property directly. However, calling the `compile` method changes the value of this property.

input

The string against which a regular expression is matched. `$_` is another name for the same property.

Property of	`RegExp`
Static	
Implemented in	Navigator 4.0, Netscape Server 3.0

Description Because `input` is static, it is not a property of an individual regular expression object. Instead, you always use it as `RegExp.input`.

If no string argument is provided to a regular expression's `exec` or `test` methods, and if `RegExp.input` has a value, its value is used as the argument to that method.

The script or the browser can preset the `input` property. If preset and if no string argument is explicitly provided, the value of `input` is used as the string argument to the `exec` or `test` methods of the regular expression object. `input` is set by the browser in the following cases:

- When an event handler is called for a `TEXT` form element, `input` is set to the value of the contained text.
- When an event handler is called for a `TEXTAREA` form element, `input` is set to the value of the contained text. Note that `multiline` is also set to `true` so that the match can be executed over the multiple lines of text.
- When an event handler is called for a `SELECT` form element, `input` is set to the value of the selected text.
- When an event handler is called for a `Link` object, `input` is set to the value of the text between `<A HREF=...>` and `</A>`.

The value of the `input` property is cleared after the event handler completes.

lastIndex

A read/write integer property that specifies the index at which to start the next match.

Property of	`RegExp`
Implemented in	Navigator 4.0, Netscape Server 3.0

Description `lastIndex` is a property of an individual regular expression object.

This property is set only if the regular expression used the `"g"` flag to indicate a global search. The following rules apply:

- If `lastIndex` is greater than the length of the string, `regexp.test` and `regexp.exec` fail, and `lastIndex` is set to `0`.
- If `lastIndex` is equal to the length of the string and if the regular expression matches the empty string, then the regular expression matches input starting at `lastIndex`.
- If `lastIndex` is equal to the length of the string and if the regular expression does not match the empty string, then the regular expression mismatches input, and `lastIndex` is reset to 0.
- Otherwise, `lastIndex` is set to the next position following the most recent match.

For example, consider the following sequence of statements:

`re = /(hi)?/g`	Matches the empty string.
`re("hi")`	Returns `["hi", "hi"]` with `lastIndex` equal to `2`.
`re("hi")`	Returns `[""]`, an empty array whose zeroth element is the match string. In this case, the empty string because `lastIndex` was 2 (and still is 2) and `"hi"` has length 2.

lastMatch

The last matched characters. `$&` is another name for the same property.

Property of	`RegExp`

Static, Read-only

Implemented in Navigator 4.0, Netscape Server 3.0

Description Because `lastMatch` is static, it is not a property of an individual regular expression object. Instead, you always use it as `RegExp.lastMatch`.

lastParen

The last parenthesized substring match, if any. `$+` is another name for the same property.

Property of `RegExp`

Static, Read-only

Implemented in Navigator 4.0, Netscape Server 3.0

Description Because `lastParen` is static, it is not a property of an individual regular expression object. Instead, you always use it as `RegExp.lastParen`.

leftContext

The substring preceding the most recent match. `$‘` is another name for the same property.

Property of `RegExp`

Static, Read-only

Implemented in Navigator 4.0, Netscape Server 3.0

Description Because `leftContext` is static, it is not a property of an individual regular expression object. Instead, you always use it as `RegExp.leftContext`.

multiline

Reflects whether or not to search in strings across multiple lines. $* is another name for the same property.

Property of `RegExp`

Static

Implemented in Navigator 4.0, Netscape Server 3.0

Description Because `multiline` is static, it is not a property of an individual regular expression object. Instead, you always use it as `RegExp.multiline`.

The value of `multiline` is `true` if multiple lines are searched, `false` if searches must stop at line breaks.

The script or the browser can preset the `multiline` property. When an event handler is called for a `TEXTAREA` form element, the browser sets `multiline` to `true`. `multiline` is cleared after the event handler completes. This means that, if you've preset multiline to `true`, it is reset to `false` after the execution of any event handler.

rightContext

The substring following the most recent match. `$'` is another name for the same property.

Property of	`RegExp`
Static, Read-only	
Implemented in	Navigator 4.0, Netscape Server 3.0

Description Because `rightContext` is static, it is not a property of an individual regular expression object. Instead, you always use it as `RegExp.rightContext`.

source

A read-only property that contains the text of the pattern, excluding the forward slashes and `"g"` or `"i"` flags.

Property of	`RegExp`
Read-only	
Implemented in	Navigator 4.0, Netscape Server 3.0

Description `source` is a property of an individual regular expression object.

You cannot change this property directly. However, calling the `compile` method changes the value of this property.

Methods

compile

Compiles a regular expression object during execution of a script.

Method of	`RegExp`
Implemented in	Navigator 4.0, Netscape Server 3.0

Syntax `regexp.compile(pattern, flags)`

Parameters

`regexp`	`The` name of the regular expression. It can be a variable name or a literal.
`pattern`	A string containing the text of the regular expression.
`flags`	(Optional) If specified, flags can have one of the following 3 values: • `"g"`: global match • `"i"`: ignore case • `"gi"`: both global match and ignore case

Description Use the `compile` method to compile a regular expression created with the `RegExp` constructor function. This forces compilation of the regular expression once only which means the regular expression isn't compiled each time it is encountered. Use the `compile` method when you know the regular expression will remain constant (after getting its pattern) and will be used repeatedly throughout the script.

You can also use the `compile` method to change the regular expression during execution. For example, if the regular expression changes, you can use the `compile` method to recompile the object for more efficient repeated use.

Calling this method changes the value of the regular expression's `source`, `global`, and `ignoreCase` properties.

exec

Executes the search for a match in a specified string. Returns a result array.

Method of	`RegExp`
Implemented in	Navigator 4.0, Netscape Server 3.0

Syntax

```
regexp.exec(str)
regexp(str)
```

Parameters

`regexp`	`The` name of the regular expression. It can be a variable name or a literal.
`str`	(Optional) The string against which to match the regular expression. If omitted, the value of `RegExp.input` is used.

Description As shown in the syntax description, a regular expression's `exec` method call be called either directly, (with `regexp.exec(str)`) or indirectly (with `regexp(str)`).

If you are executing a match simply to find `true` or `false`, use the `test` method or the `String` `search` method.

If the match succeeds, the `exec` method returns an array and updates properties of the regular expression object and the predefined regular expression object, `RegExp`. If the match fails, the `exec` method returns `null`.

Consider the following example:

```
<SCRIPT LANGUAGE="JavaScript1.2">
//Match one d followed by one or more b's followed by one d
//Remember matched b's and the following d
//Ignore case
myRe=/d(b+)(d)/ig;
myArray = myRe.exec("cdbBdbsbz");
</SCRIPT>
```

The following table shows the results for this script:

Object	Property/Index	Description	Example
myArray		The contents of myArray	["dbBd", "bB", "d"]
	index	The 0-based index of the match in the string	1
	input	The original string	cdbBdbsbz
	[0]	The last matched characters	dbBd
	[1], ...[*n*]	The parenthesized substring matches, if any. The number of possible parenthesized substrings is unlimited.	[1] = bB [2] = d
myRe	lastIndex	The index at which to start the next match.	5
	ignoreCase	Indicates if the "i" flag was used to ignore case	true
	global	Indicates if the "g" flag was used for a global match	true
	source	The text of the pattern	d(b+)(d)
RegExp	lastMatch $&	The last matched characters	dbBd
	leftContext $`	The substring preceding the most recent match	c
	rightContext $'	The substring following the most recent match	bsbz
	$1, ...$9	The parenthesized substring matches, if any. The number of possible parenthesized substrings is unlimited, but RegExp can only hold the last nine.	$1 = bB $2 = d
	lastParen $+	The last parenthesized substring match, if any.	d

If your regular expression uses the "g" flag, you can use the exec method multiple times to find successive matches in the same string. When you do so, the search starts at the substring of str specified by the regular expression's lastIndex property. For example, assume you have this script:

```
<SCRIPT LANGUAGE="JavaScript1.2">
myRe=/ab*/g;
str = "abbcdefabh"
```

```
myArray = myRe.exec(str);
document.writeln("Found " + myArray[0] +
   ". Next match starts at " + myRe.lastIndex)
mySecondArray = myRe.exec(str);
document.writeln("Found " + mySecondArray[0] +
   ". Next match starts at " + myRe.lastIndex)
</SCRIPT>
```

This script displays the following text:

Found abb. Next match starts at 3
Found ab. Next match starts at 9

Examples In the following example, the user enters a name and the script executes a match against the input. It then cycles through the array to see if other names match the user's name.

This script assumes that first names of registered party attendees are preloaded into the array A, perhaps by gathering them from a party database.

```
<HTML>

<SCRIPT LANGUAGE="JavaScript1.2">
A = ["Frank", "Emily", "Jane", "Harry", "Nick", "Beth", "Rick",
      "Terrence", "Carol", "Ann", "Terry", "Frank", "Alice", "Rick",
      "Bill", "Tom", "Fiona", "Jane", "William", "Joan", "Beth"]

function lookup() {
   firstName = /\w+/i();
   if (!firstName)
      window.alert (RegExp.input + " isn't a name!");
   else {
      count = 0;
      for (i=0; i<A.length; i++)
         if (firstName[0].toLowerCase() == A[i].toLowerCase()) count++;
      if (count ==1)
         midstring = " other has ";
      else
         midstring = " others have ";
      window.alert ("Thanks, " + count + midstring + "the same name!")
   }
}

</SCRIPT>

Enter your first name and then press Enter.

<FORM> <INPUT TYPE:"TEXT" NAME="FirstName" onChange="lookup(this);"> </
FORM>

</HTML>
```

test

Executes the search for a match between a regular expression and a specified string. Returns `true` or `false`.

Method of	`RegExp`
Implemented in	Navigator 4.0, Netscape Server 3.0

Syntax `regexp.test(str)`

Parameters

`regexp`	`The` name of the regular expression. It can be a variable name or a literal.
`str`	(Optional) The string against which to match the regular expression. If omitted, the value of `RegExp.input` is used.

Description When you want to know whether a pattern is found in a string use the `test` method (similar to the `String.search` method); for more information (but slower execution) use the `exec` method (similar to the `String.match` method).

Example The following example prints a message which depends on the success of the test:

```
function testinput(re, str){
   if (re.test(str))
      midstring = " contains ";
   else
      midstring = " does not contain ";
   document.write (str + midstring + re.source);
}
```

Chapter

5

Document

This chapter deals with the document and its associated objects, `document`, `Layer`, `Link`, `Anchor`, `Area`, `Image`, and `Applet`.

Table 5.1 summarizes the objects in this chapter.

Table 5.1 Document objects

Object	Description
`Anchor`	A place in a document that is the target of a hypertext link.
`Applet`	Includes a Java applet in a web page.
`Area`	Defines an area of an image as an image map.
`document`	Contains information on the current document, and provides methods for displaying HTML output to the user.
`Image`	An image on an HTML form.
`Layer`	Corresponds to a layer in an HTML page and provides a means for manipulating that layer.
`Link`	A piece of text, an image, or an area of an image identified as a hypertext link.

document

Contains information about the current document, and provides methods for displaying HTML output to the user.

Client-side object

Implemented in Navigator 2.0
Navigator 3.0: added `onBlur` and `onFocus` syntax; added `applets`, `domain`, `embeds`, `forms`, *`formName`*, `images`, and `plugins` properties.
Navigator 4.0: added `layers` property; added `captureEvents`, `getSelection`, `handleEvent`, `releaseEvents`, and `routeEvent` methods.

Created by The HTML `BODY` tag. The JavaScript runtime engine creates a `document` object for each HTML page. Each `Window` object has a `document` property whose value is a `document` object.

To define a `document` object, use standard HTML syntax for the `BODY` tag with the addition of JavaScript event handlers.

Event handlers The `onBlur`, `onFocus`, `onLoad`, and `onUnload` event handlers are specified in the `BODY` tag but are actually event handlers for the `Window` object. The following are event handlers for the `document` object.

- `onClick`
- `onDblClick`
- `onKeyDown`
- `onKeyPress`
- `onKeyUp`
- `onMouseDown`
- `onMouseUp`

Description An HTML document consists of `HEAD` and `BODY` tags. The `HEAD` tag includes information on the document's title and base (the absolute URL base to be used for relative URL links in the document). The `BODY` tag encloses the body of a document, which is defined by the current URL. The entire body of the document (all other HTML elements for the document) goes within the `BODY` tag.

You can load a new document by setting the `Window.location` property.

You can clear the document pane (and remove the text, form elements, and so on so they do not redisplay) with these statements:

```
document.close();
document.open();
document.write();
```

You can omit the `document.open` call if you are writing text or HTML, since `write` does an implicit open of that MIME type if the document stream is closed.

You can refer to the anchors, forms, and links of a document by using the `anchors`, `forms`, and `links` arrays. These arrays contain an entry for each anchor, form, or link in a document and are properties of the `document` object.

Do not use `location` as a property of the `document` object; use the `document.URL` property instead. The `document.location` property, which is a synonym for `document.URL`, will be removed in a future release.

Property Summary

Property	Description
`alinkColor`	A string that specifies the `ALINK` attribute.
`anchors`	An array containing an entry for each anchor in the document.
`applets`	An array containing an entry for each applet in the document.
`bgColor`	A string that specifies the `BGCOLOR` attribute.
`cookie`	Specifies a cookie.
`domain`	Specifies the domain name of the server that served a document.
`embeds`	An array containing an entry for each plug-in in the document.
`fgColor`	A string that specifies the `TEXT` attribute.
`formName`	A separate property for each named form in the document.
`forms`	An array a containing an entry for each form in the document.
`images`	An array containing an entry for each image in the document.
`lastModified`	A string that specifies the date the document was last modified.
`layers`	Array containing an entry for each layer within the document.
`linkColor`	A string that specifies the `LINK` attribute.
`links`	An array containing an entry for each link in the document.

Property	Description
`plugins`	An array containing an entry for each plug-in in the document.
`referrer`	A string that specifies the URL of the calling document.
`title`	A string that specifies the contents of the `TITLE` tag.
`URL`	A string that specifies the complete URL of a document.
`vlinkColor`	A string that specifies the `VLINK` attribute.

Method Summary

Method	Description
`captureEvents`	Sets the document to capture all events of the specified type.
`close`	Closes an output stream and forces data to display.
`getSelection`	Returns a string containing the text of the current selection.
`handleEvent`	Invokes the handler for the specified event.
`open`	Opens a stream to collect the output of `write` or `writeln` methods.
`releaseEvents`	Sets the window or document to release captured events of the specified type, sending the event to objects further along the event hierarchy.
`routeEvent`	Passes a captured event along the normal event hierarchy.
`write`	Writes one or more HTML expressions to a document in the specified window.
`writeln`	Writes one or more HTML expressions to a document in the specified window and follows them with a newline character.

Examples The following example creates two frames, each with one document. The document in the first frame contains links to anchors in the document of the second frame. Each document defines its colors.

`doc0.html`, which defines the frames, contains the following code:

```
<HTML>
<HEAD>
<TITLE>Document object example</TITLE>
</HEAD>
```

```
<FRAMESET COLS="30%,70%">
<FRAME SRC="doc1.html" NAME="frame1">
<FRAME SRC="doc2.html" NAME="frame2">
</FRAMESET>
</HTML>
```

`doc1.html`, which defines the content for the first frame, contains the following code:

```
<HTML>
<SCRIPT>
</SCRIPT>
<BODY
   BGCOLOR="antiquewhite"
   TEXT="darkviolet"
   LINK="fuchsia"
   ALINK="forestgreen"
   VLINK="navy">
<P><B>Some links</B>
<LI><A HREF="doc2.html#numbers" TARGET="frame2">Numbers</A>
<LI><A HREF="doc2.html#colors" TARGET="frame2">Colors</A>
<LI><A HREF="doc2.html#musicTypes" TARGET="frame2">Music types</A>
<LI><A HREF="doc2.html#countries" TARGET="frame2">Countries</A>
</BODY>
</HTML>
```

`doc2.html`, which defines the content for the second frame, contains the following code:

```
<HTML>
<SCRIPT>
</SCRIPT>
<BODY
   BGCOLOR="oldlace" onLoad="alert('Hello, World.')"
   TEXT="navy">
<P><A NAME="numbers"><B>Some numbers</B></A>
<UL><LI>one
<LI>two
<LI>three
<LI>four</UL>
<P><A NAME="colors"><B>Some colors</B></A>
<UL><LI>red
<LI>orange
<LI>yellow
<LI>green</UL>
<P><A NAME="musicTypes"><B>Some music types</B></A>
<UL><LI>R&B
<LI>Jazz
<LI>Soul
<LI>Reggae</UL>
```

```
<P><A NAME="countries"><B>Some countries</B></A>
<UL><LI>Afghanistan
<LI>Brazil
<LI>Canada
<LI>Finland</UL>
</BODY>
</HTML>
```

See also `Frame`, `Window`

Properties

alinkColor

A string specifying the color of an active link (after mouse-button down, but before mouse-button up).

Property of	`document`
Implemented in	Navigator 2.0

Description The `alinkColor` property is expressed as a hexadecimal RGB triplet or as one of the string literals listed in Appendix B, "Color Values," in the *JavaScript Guide* This property is the JavaScript reflection of the `ALINK` attribute of the `BODY` tag. You cannot set this property after the HTML source has been through layout.

If you express the color as a hexadecimal RGB triplet, you must use the format `rrggbb`. For example, the hexadecimal RGB values for salmon are `red=FA`, `green=80`, and `blue=72`, so the RGB triplet for salmon is `"FA8072"`.

Examples The following example sets the color of active links using a string literal:

```
document.alinkColor="aqua"
```

The following example sets the color of active links to aqua using a hexadecimal triplet:

```
document.alinkColor="00FFFF"
```

See also `document.bgColor`, `document.fgColor`, `document.linkColor`, `document.vlinkColor`

anchors

An array of objects corresponding to named anchors in source order.

Property of	`document`
Read-only	
Implemented in	Navigator 2.0

Description You can refer to the `Anchor` objects in your code by using the `anchors` array. This array contains an entry for each `A` tag containing a `NAME` attribute in a document; these entries are in source order. For example, if a document contains three named anchors whose `NAME` attributes are `anchor1`, `anchor2`, and `anchor3`, you can refer to the anchors either as:

```
document.anchors["anchor1"]
document.anchors["anchor2"]
document.anchors["anchor3"]
```

or as:

```
document.anchors[0]
document.anchors[1]
document.anchors[2]
```

To obtain the number of anchors in a document, use the `length` property: `document.anchors.length`. If a document names anchors in a systematic way using natural numbers, you can use the `anchors` array and its `length` property to validate an anchor name before using it in operations such as setting `location.hash`.

applets

An array of objects corresponding to the applets in a document in source order.

Property of	`document`
Read-only	
Implemented in	Navigator 3.0

Description You can refer to the applets in your code by using the `applets` array. This array contains an entry for each `Applet` object (`APPLET` tag) in a document; these entries are in source order. For example, if a document contains three applets whose `NAME` attributes are `app1`, `app2`, and `app3`, you can refer to the anchors either as:

```
document.applets["app1"]
document.applets["app2"]
document.applets["app3"]
```

or as:

```
document.applets[0]
document.applets[1]
document.applets[2]
```

bgColor

A string specifying the color of the document background.

Property of	`document`
Implemented in	Navigator 2.0

Description The `bgColor` property is expressed as a hexadecimal RGB triplet or as one of the string literals listed in Appendix B, "Color Values," in the *JavaScript Guide*. This property is the JavaScript reflection of the `BGCOLOR` attribute of the `BODY` tag. The default value of this property is set by the user with the preferences dialog box.

If you express the color as a hexadecimal RGB triplet, you must use the format `rrggbb`. For example, the hexadecimal RGB values for salmon are `red=FA`, `green=80`, and `blue=72`, so the RGB triplet for `salmon` is `"FA8072"`.

Examples The following example sets the color of the document background to aqua using a string literal:

```
document.bgColor="aqua"
```

The following example sets the color of the document background to aqua using a hexadecimal triplet:

```
document.bgColor="00FFFF"
```

See also `document.alinkColor`, `document.fgColor`, `document.linkColor`, `document.vlinkColor`

cookie

String value representing all of the cookies associated with this document.

Property of	`document`
Implemented in	Navigator 2.0

Security Navigator 3.0: This property is tainted by default. For information on data tainting, see “Security” on page 55.

Description A cookie is a small piece of information stored by the web browser in the `cookies.txt` file. Use `string` methods such as `substring`, `charAt`, `indexOf`, and `lastIndexOf` to determine the value stored in the cookie. See the *JavaScript Guide* for a complete specification of the cookie syntax.

You can set the `cookie` property at any time.

The `"expires="` component in the cookie file sets an expiration date for the cookie, so it persists beyond the current browser session. This date string is formatted as follows:

```
Wdy, DD-Mon-YY HH:MM:SS GMT
```

This format represents the following values:

- `Wdy` is a string representing the full name of the day of the week.
- `DD` is an integer representing the day of the month.
- `Mon` is a string representing the three-character abbreviation of the month.
- `YY` is an integer representing the last two digits of the year.
- `HH`, `MM`, and `SS` are 2-digit representations of hours, minutes, and seconds, respectively.

For example, a valid cookie expiration date is

```
expires=Wednesday, 09-Nov-99 23:12:40 GMT
```

The cookie date format is the same as the date returned by `toGMTString`, with the following exceptions:

- Dashes are added between the day, month, and year.
- The year is a 2-digit value for cookies.

Examples The following function uses the `cookie` property to record a reminder for users of an application. The cookie expiration date is set to one day after the date of the reminder.

```
function RecordReminder(time, expression) {
   // Record a cookie of the form "@<T>=<E>" to map
   // from <T> in milliseconds since the epoch,
   // returned by Date.getTime(), onto an encoded expression,
   // <E> (encoded to contain no white space, semicolon,
   // or comma characters)
   document.cookie = "@" + time + "=" + expression + ";"
   // set the cookie expiration time to one day
   // beyond the reminder time
   document.cookie += "expires=" + cookieDate(time + 24*60*60*1000)
   // cookieDate is a function that formats the date
   //according to the cookie spec
}
```

See also `Hidden`

domain

Specifies the domain name of the server that served a document.

Property of	`document`
Implemented in	Navigator 3.0

Security Navigator 3.0: This property is tainted by default. For information on data tainting, see "Security" on page 55.

Description Navigator 3.0: The `domain` property lets scripts on multiple servers share properties when data tainting is not enabled. With tainting disabled, a script running in one window can read properties of another window only if both windows come from the same Web server. But large Web sites with multiple servers might need to share properties among servers. For example, a script on the host `www.royalairways.com` might need to share properties with a script on the host `search.royalairways.com`.

If scripts on two different servers change their `domain` property so that both scripts have the same domain name, both scripts can share properties. For example, a script loaded from `search.royalairways.com` could set its `domain` property to `"royalairways.com"`. A script from `www.royalairways.com` running in another window could also set its `domain` property to "royalairways.com". Then, since both scripts have the domain "royalairways.com", these two scripts can share properties, even though they did not originate from the same server.

You can change domain only in a restricted way. Initially, domain contains the hostname of the Web server from which the document was loaded. You can set domain only to a domain suffix of itself. For example, a script from search.royalairways.com can't set its domain property to "search.royalairways". And a script from IWantYourMoney.com cannot set its domain to "royalairways.com".

Once you change the domain property, you cannot change it back to its original value. For example, if you change domain from "search.royalairways.com" to "royalairways.com", you cannot reset it to "search.royalairways.com".

Examples The following statement changes the domain property to "braveNewWorld.com". This statement is valid only if "braveNewWorld.com" is a suffix of the current domain, such as "www.braveNewWorld.com".

```
document.domain="braveNewWorld.com"
```

embeds

An array containing an entry for each object embedded in the document.

Property of	document
Read-only	
Implemented in	Navigator 3.0

Description You can refer to embedded objects (created with the EMBED tag) in your code by using the embeds array. This array contains an entry for each EMBED tag in a document in source order. For example, if a document contains three embedded objects whose NAME attributes are e1, e2, and e3, you can refer to the objects either as:

```
document.embeds["e1"]
document.embeds["e2"]
document.embeds["e3"]
```

or as:

```
document.embeds[0]
document.embeds[1]
document.embeds[2]
```

Elements in the `embeds` array may have public callable functions, if they refer to a plug-in that uses LiveConnect. See the *JavaScript Guide*.

Use the elements in the `embeds` array to interact with the plug-in that is displaying the embedded object. If a plug-in is not Java-enabled, you cannot do anything with its element in the `embeds` array. The fields and methods of the elements in the `embeds` array vary from plug-in to plug-in; see the documentation supplied by the plug-in manufacturer.

When you use the `EMBED` tag to generate output from a plug-in application, you are not creating a `Plugin` object.

Examples The following code includes an audio plug-in in a document.

```
<EMBED SRC="train.au" HEIGHT=50 WIDTH=250>
```

See also `Plugin`

fgColor

A string specifying the color of the document (foreground) text.

Property of	`document`
Implemented in	Navigator 2.0

Description The `fgColor` property is expressed as a hexadecimal RGB triplet or as one of the string literals listed in Appendix B, “Color Values,” in the *JavaScript Guide*. This property is the JavaScript reflection of the `TEXT` attribute of the `BODY` tag. The default value of this property is set by the user with the preferences dialog box You cannot set this property after the HTML source has been through layout.

If you express the color as a hexadecimal RGB triplet, you must use the format `rrggbb`. For example, the hexadecimal RGB values for salmon are `red=FA`, `green=80`, and `blue=72`, so the RGB triplet for `salmon` is `"FA8072"`.

You can override the value set in the `fgColor` property in either of the following ways:

- Setting the `COLOR` attribute of the `FONT` tag.

- Using the `fontcolor` method.

formName

Property of	`document`
Implemented in	Navigator 3.0

The `document` object contains a separate property for each form in the document. The name of this property is the value of its `NAME` attribute. See `Form` for information on `Form` objects. You cannot add new forms to the document by creating new properties, but you can modify the form by modifying this object.

forms

An array containing an entry for each form in the document.

Property of	`document`
Read-only	
Implemented in	Navigator 3.0

Security Navigator 3.0: This property is tainted by default. For information on data tainting, see “Security” on page 55.

Description You can refer to the forms in your code by using the `forms` array (you can also use the form name). This array contains an entry for each `Form` object (`FORM` tag) in a document; these entries are in source order. For example, if a document contains three forms whose `NAME` attributes are `form1`, `form2`, and `form3`, you can refer to the objects in the `forms` array either as:

```
document.forms["form1"]
document.forms["form2"]
document.forms["form3"]
```

or as:

```
document.forms[0]
document.forms[1]
document.forms[2]
```

Additionally, the document object has a separate property for each named form, so you could refer to these forms also as:

```
document.form1
document.form2
document.form3
```

For example, you would refer to a `Text` object named `quantity` in the second form as `document.forms[1].quantity`. You would refer to the `value` property of this `Text` object as `document.forms[1].quantity.value`.

The value of each element in the `forms` array is `<object nameAttribute>`, where `nameAttribute` is the `NAME` attribute of the form.

images

An array containing an entry for each image in the document.

Property of	`document`
Read-only	
Implemented in	Navigator 3.0

You can refer to the images in a document by using the `images` array. This array contains an entry for each `Image` object (`IMG` tag) in a document; the entries are in source order. Images created with the `Image` constructor are not included in the `images` array. For example, if a document contains three images whose `NAME` attributes are `im1`, `im2`, and `im3`, you can refer to the objects in the `images` array either as:

```
document.images["im1"]
document.images["im2"]
document.images["im3"]
```

or as:

```
document.images[0]
document.images[1]
document.images[2]
```

lastModified

A string representing the date that a document was last modified.

Property of	`document`
Read-only	
Implemented in	Navigator 2.0

Security Navigator 3.0: This property is tainted by default. For information on data tainting, see "Security" on page 55.

Description The `lastModified` property is derived from the HTTP header data sent by the web server. Servers generally obtain this date by examining the file's modification date.

The last modified date is not a required portion of the header, and some servers do not supply it. If the server does not return the last modified information, JavaScript receives a 0, which it displays as January 1, 1970 GMT. The following code checks the date returned by `lastModified` and prints out a value that corresponds to unknown.

```
lastmod = document.lastModified // get string of last modified date
lastmoddate = Date.parse(lastmod)// convert modified string to date
if(lastmoddate == 0){// unknown date (or January 1, 1970 GMT)
   document.writeln("Lastmodified: Unknown")
   } else {
   document.writeln("LastModified: " + lastmod)
}
```

Examples In the following example, the `lastModified` property is used in a `SCRIPT` tag at the end of an HTML file to display the modification date of the page:

```
document.write("This page updated on " + document.lastModified)
```

layers

The layers property is an array containing an entry for each layer within the document.

Property of	`document`
Implemented in	Navigator 4.0

Description You can refer to the layers in your code by using the `layers` array. This array contains an entry for each `Layer` object (`LAYER` or `ILAYER` tag) in a document; these entries are in source order. For example, if a document contains three layers whose `NAME` attributes are `layer1`, `layer2`, and `layer3`, you can refer to the objects in the `layers` array either as:

```
document.layers["layer1"]
document.layers["layer2"]
document.layers["layer3"]
```

or as:

```
document.layers[0]
document.layers[1]
document.layers[2]
```

When accessed by integer index, array elements appear in z-order from back to front, where 0 is the bottommost layer and higher layers are indexed by consecutive integers. The index of a layer is not the same as its `zIndex` property, as the latter does not necessarily enumerate layers with consecutive integers. Adjacent layers can have the same `zIndex` property values.

These are valid ways of accessing layer objects:

```
document.layerName
document.layers[index]
document.layers["layerName"]
// example of using layers property to access nested layers:
document.layers["parentlayer"].layers["childlayer"]
```

Elements of a layers array are JavaScript objects that cannot be set by assignment, though their properties can be set. For example, the statement

```
document.layers[0]="music"
```

is invalid (and ignored) because it attempts to alter the `layers` array. However, the properties of the objects in the array readable and some are writable. For example, the statement

```
document.layers["suspect1"].left = 100;
```

is valid. This sets the layer's horizontal position to 100. The following example sets the background color to blue for the layer `bluehouse` which is nested in the layer `houses`.

```
document.layers["houses"].layers["bluehouse"].bgColor="blue";
```

linkColor

A string specifying the color of the document hyperlinks.

Property of	`document`
Implemented in	Navigator 2.0

Description The `linkColor` property is expressed as a hexadecimal RGB triplet or as one of the string literals listed in the *JavaScript Guide*. This property is the JavaScript reflection of the `LINK` attribute of the `BODY` tag. The default value of this property is set by the user with the preferences dialog box. You cannot set this property after the HTML source has been through layout.

If you express the color as a hexadecimal RGB triplet, you must use the format `rrggbb`. For example, the hexadecimal RGB values for salmon are `red=FA`, `green=80`, and `blue=72`, so the RGB triplet for salmon is `"FA8072"`.

Examples The following example sets the color of document links to aqua using a string literal:

```
document.linkColor="aqua"
```

The following example sets the color of document links to aqua using a hexadecimal triplet:

```
document.linkColor="00FFFF"
```

See also `document.alinkColor`, `document.bgColor`, `document.fgColor`, `document.vlinkColor`

links

An array of objects corresponding to `Area` and `Link` objects in source order.

Property of	`document`
Read-only	
Implemented in	Navigator 2.0

Security Navigator 3.0: This property is tainted by default. For information on data tainting, see “Security” on page 55.

Description You can refer to the `Area` and `Link` objects in your code by using the `links` array. This array contains an entry for each `Area` (`<AREA HREF="...">` tag) and `Link` (`<A HREF="...">` tag) object in a document in source order. It also contains links created with the `link` method. For example, if a document contains three links, you can refer to them as:

```
document.links[0]
document.links[1]
document.links[2]
```

plugins

An array of objects corresponding to `Plugin` objects in source order.

Property of	`document`
Read-only	
Implemented in	Navigator 3.0

You can refer to the `Plugin` objects in your code by using the `plugins` array. This array contains an entry for each `Plugin` object in a document in source order. For example, if a document contains three plugins, you can refer to them as:

```
document.plugins[0]
document.plugins[1]
document.plugins[2]
```

referrer

Specifies the URL of the calling document when a user clicks a link.

Property of	`document`
Read-only	
Implemented in	Navigator 2.0

Security Navigator 3.0: This property is tainted by default. For information on data tainting, see "Security" on page 55.

Description When a user navigates to a destination document by clicking a `Link` object on a source document, the `referrer` property contains the URL of the source document.

`referrer` is empty if the user typed a URL in the Location box, or used some other means to get to the current URL. `referrer` is also empty if the server does not provide environment variable information.

Examples In the following example, the `getReferrer` function is called from the destination document. It returns the URL of the source document.

```
function getReferrer() {
   return document.referrer
}
```

title

A string representing the title of a document.

Property of	`document`
Read-only	
Implemented in	Navigator 2.0

Security Navigator 3.0: This property is tainted by default. For information on data tainting, see "Security" on page 55.

Description The `title` property is a reflection of the value specified between the `TITLE` start and end tags. If a document does not have a title, the `title` property is null.

Examples In the following example, the value of the `title` property is assigned to a variable called `docTitle`:

```
var newWindow = window.open("http://home.netscape.com")
var docTitle = newWindow.document.title
```

URL

A string specifying the complete URL of the document.

Property of	`document`
Read-only	
Implemented in	Navigator 2.0

Security Navigator 3.0: This property is tainted by default. For information on data tainting, see "Security" on page 55.

Description `URL` is a string-valued property containing the full URL of the document. It usually matches what `window.location.href` is set to when you load the document, but redirection may change `location.href`.

Examples The following example displays the URL of the current document:

```
document.write("The current URL is " + document.URL)
```

See also `Location.href`

vlinkColor

A string specifying the color of visited links.

Property of	`document`
Implemented in	Navigator 2.0

Description The `vlinkColor` property is expressed as a hexadecimal RGB triplet or as one of the string literals listed in the *JavaScript Guide*. This property is the JavaScript reflection of the `VLINK` attribute of the `BODY` tag. The default value of this property is set by the user with the preferences dialog box. You cannot set this property after the HTML source has been through layout.

If you express the color as a hexadecimal RGB triplet, you must use the format `rrggbb`. For example, the hexadecimal RGB values for salmon are `red=FA`, `green=80`, and `blue=72`, so the RGB triplet for salmon is `"FA8072"`.

Examples The following example sets the color of visited links to aqua using a string literal:

```
document.vlinkColor="aqua"
```

The following example sets the color of active links to aqua using a hexadecimal triplet:

```
document.vlinkColor="00FFFF"
```

See also `document.alinkColor`, `document.bgColor`, `document.fgColor`, `document.linkColor`

Methods

captureEvents

Sets the document to capture all events of the specified type.

Method of	`document`
Implemented in	Navigator 4.0

Syntax `captureEvents(eventType)`

Parameters

`eventType`	The type of event to be captured. The available event types are listed with the `event` object.

Description When a window with frames wants to capture events in pages loaded from different locations (servers), you need to use `Window.captureEvents` in a signed script and precede it with `Window.enableExternalCapture`. For more information and an example, see `Window.enableExternalCapture`.

`captureEvents` works in tandem with `releaseEvents`, `routeEvent`, and `handleEvent`. For more information, see "Events in Navigator 4.0" on page 482.

close

Closes an output stream and forces data sent to layout to display.

Method of	`document`
Implemented in	Navigator 2.0

Syntax `close()`

Parameters None.

Description The `close` method closes a stream opened with the `document.open` method. If the stream was opened to layout, the `close` method forces the content of the stream to display. Font style tags, such as `BIG` and `CENTER`, automatically flush a layout stream.

The `close` method also stops the "meteor shower" in the Netscape icon and displays Document: Done in the status bar.

Examples The following function calls `document.close` to close a stream that was opened with `document.open`. The `document.close` method forces the content of the stream to display in the window.

```
function windowWriter1() {
   var myString = "Hello, world!"
   msgWindow.document.open()
   msgWindow.document.write(myString + "<P>")
   msgWindow.document.close()
}
```

See also `document.open`, `document.write`, `document.writeln`

getSelection

Returns a string containing the text of the current selection.

Method of	`document`
Implemented in	Navigator 4.0

Syntax `getSelection()`

Description This method works only on the current document.

Security You cannot determine selected areas in another window.

Examples If you have a form with the following code and you click on the button, JavaScript displays an alert box containing the currently selected text from the window containing the button:

```
<INPUT TYPE="BUTTON" NAME="getstring"
   VALUE="Show highlighted text (if any)"
   onClick="alert('You have selected:\n'+document.getSelection());">
```

handleEvent

Invokes the handler for the specified event.

Method of	`document`
Implemented in	Navigator 4.0

Syntax `handleEvent(event)`

Parameters

`event`	The name of an event for which the specified object has an event handler.

Description For information on handling events, see "General Information about Events" on page 481.

open

Opens a stream to collect the output of `write` or `writeln` methods.

Method of	`document`
Implemented in	Navigator 2.0 Navigator 3.0: added `"replace"` parameter; `document.open()` or `document.open("text/html")` clears the current document if it has finished loading

Syntax `open(mimeType, replace)`

Parameters

`mimeType`	(Optional) A string specifying the type of document to which you are writing. If you do not specify `mimeType`, `text/html` is the default.
`replace`	(Optional) The string `"replace"`. If you supply this parameter, `mimeType` must be `"text/html"`. Causes the new document to reuse the history entry that the previous document used.

Description Sample values for `mimeType` are:

- `text/html` specifies a document containing ASCII text with HTML formatting.
- `text/plain` specifies a document containing plain ASCII text with end-of-line characters to delimit displayed lines.
- `image/gif` specifies a document with encoded bytes constituting a GIF header and pixel data.
- `image/jpeg` specifies a document with encoded bytes constituting a JPEG header and pixel data.
- `image/x-bitmap` specifies a document with encoded bytes constituting a bitmap header and pixel data.
- `plugIn` loads the specified plug-in and uses it as the destination for `write` and `writeln` methods. For example, `"x-world/vrml"` loads the VR Scout VRML plug-in from Chaco Communications, and `"application/x-director"` loads the Macromedia Shockwave plug-in. Plug-in MIME types are only valid if the user has installed the required plug-in software.

The `open` method opens a stream to collect the output of `write` or `writeln` methods. If the `mimeType` is `text` or `image`, the stream is opened to layout; otherwise, the stream is opened to a plug-in. If a document exists in the target window, the `open` method clears it.

End the stream by using the `document.close` method. The `close` method causes text or images that were sent to layout to display. After using `document.close`, call `document.open` again when you want to begin another output stream.

In Navigator 3.0 and later, `document.open` or `document.open("text/html")` clears the current document if it has finished loading. This is because this type of `open` call writes a default `<BASE HREF=>` tag so you can generate relative URLs based on the generating script's document base.

The `"replace"` keyword causes the new document to reuse the history entry that the previous document used. When you specify `"replace"` while opening a document, the target window's history length is not incremented even after you write and close.

`"replace"` is typically used on a window that has a blank document or an `"about:blank"` URL. After `"replace"` is specified, the `write` method typically generates HTML for the window, replacing the history entry for the blank URL. Take care when using generated HTML on a window with a blank URL. If you do not specify `"replace"`, the generated HTML has its own history entry, and the user can press the Back button and back up until the frame is empty.

After `document.open("text/html","replace")` executes, `history.current` for the target window is the URL of document that executed `document.open`.

Examples **Example 1.** The following function calls `document.open` to open a stream before issuing a `write` method:

```
function windowWriter1() {
   var myString = "Hello, world!"
   msgWindow.document.open()
   msgWindow.document.write("<P>" + myString)
   msgWindow.document.close()
}
```

Example 2. The following function calls `document.open` with the `"replace"` keyword to open a stream before issuing `write` methods. The HTML code in the `write` methods is written to `msgWindow`, replacing the current history entry. The history length of `msgWindow` is not incremented.

```
function windowWriter2() {
   var myString = "Hello, world!"
   msgWindow.document.open("text/html","replace")
   msgWindow.document.write("<P>" + myString)
   msgWindow.document.write("<P>history.length is " +
      msgWindow.history.length)
   msgWindow.document.close()
}
```

The following code creates the `msgWindow` window and calls the function:

```
msgWindow=window.open('','',
   'toolbar=yes,scrollbars=yes,width=400,height=300')
windowWriter2()
```

Example 3. In the following example, the `probePlugIn` function determines whether a user has the Shockwave plug-in installed:

```
function probePlugIn(mimeType) {
   var havePlugIn = false
   var tiny = window.open("", "teensy", "width=1,height=1")
   if (tiny != null) {
      if (tiny.document.open(mimeType) != null)
         havePlugIn = true
      tiny.close()
   }
   return havePlugIn
}

var haveShockwavePlugIn = probePlugIn("application/x-director")
```

See also `document.close`, `document.write`, `document.writeln`, `Location.reload`, `Location.replace`

releaseEvents

Sets the document to release captured events of the specified type, sending the event to objects further along the event hierarchy.

Method of	`document`
Implemented in	Navigator 4.0

Note If the original target of the event is a window, the window receives the event even if it is set to release that type of event.

Syntax `releaseEvents(eventType)`

Parameters

`eventType`	Type of event to be captured.

Description `releaseEvents` works in tandem with `captureEvents`, `routeEvent`, and `handleEvent`. For more information, see "Events in Navigator 4.0" on page 482.

routeEvent

Passes a captured event along the normal event hierarchy.

Method of	`document`
Implemented in	Navigator 4.0

Syntax `routeEvent(event)`

Parameters

`event`	Name of the event to be routed.

Description If a subobject (document or layer) is also capturing the event, the event is sent to that object. Otherwise, it is sent to its original target.

`routeEvent` works in tandem with `captureEvents`, `releaseEvents`, and `handleEvent`. For more information, see "Events in Navigator 4.0" on page 482.

write

Writes one or more HTML expressions to a document in the specified window.

Method of	`document`
Implemented in	Navigator 2.0 Navigator 3.0: users can print and save generated HTML using the commands on the File menu

Syntax `document.write(expr1, ...,expr`*`N`*`)`

Parameters

`expr1, ... exprN`	Any JavaScript expressions.

Description The `write` method displays any number of expressions in the document window. You can specify any JavaScript expression with the `write` method, including numeric, string, or logical expressions.

The `write` method is the same as the `writeln` method, except the `write` method does not append a newline character to the end of the output.

Use the `write` method within any `SCRIPT` tag or within an event handler. Event handlers execute after the original document closes, so the `write` method implicitly opens a new document of `mimeType text/html` if you do not explicitly issue a `document.open` method in the event handler.

You can use the `write` method to generate HTML and JavaScript code. However, the HTML parser reads the generated code as it is being written, so you might have to escape some characters. For example, the following `write` method generates a comment and writes it to `window2`:

```
window2=window.open('','window2')
beginComment="\<!--"
endComment="--\>"
window2.document.write(beginComment)
window2.document.write(" This some text inside a comment. ")
window2.document.write(endComment)
```

Printing, saving, and viewing generated HTML

In Navigator 3.0 and later, users can print and save generated HTML using the commands on the File menu.

If you choose Document Source or Frame Source from the View menu, the web browser displays the content of the HTML file with the generated HTML. (This is what would be displayed using a `wysiwyg:` URL.)

If you instead want to view the HTML source showing the scripts which generate HTML (with the `document.write` and `document.writeln` methods), do not use the Document Source or Frame Source menu item. In this situation, use the `view-source:` protocol.

For example, assume the file `file://c|/test.html` contains this text:

```
<HTML>
<BODY>
Hello,
<SCRIPT>document.write(" there.")</SCRIPT>
</BODY>
</HTML>
```

If you load this URL into the web browser, it displays the following:

```
Hello, there.
```

If you choose View Document Source, the browser displays:

```
<HTML>
<BODY>
Hello,
 there.
</BODY>
</HTML>
```

If you load `view-source:file://c|/test.html`, the browser displays:

```
<HTML>
<BODY>
Hello,
<SCRIPT>document.write(" there.")</SCRIPT>
</BODY>
</HTML>
```

For information on specifying the `view-source:` protocol in the `location` object, see the `Location` object.

Examples In the following example, the `write` method takes several arguments, including strings, a numeric, and a variable:

```
var mystery = "world"
// Displays Hello world testing 123
msgWindow.document.write("Hello ", mystery, " testing ", 123)
```

In the following example, the `write` method takes two arguments. The first argument is an assignment expression, and the second argument is a string literal.

```
//Displays Hello world...
msgWindow.document.write(mystr = "Hello ", "world...")
```

In the following example, the `write` method takes a single argument that is a conditional expression. If the value of the variable `age` is less than 18, the method displays “Minor.” If the value of `age` is greater than or equal to 18, the method displays “Adult.”

```
msgWindow.document.write(status = (age >= 18) ? "Adult" : "Minor")
```

See also `document.close`, `document.open`, `document.writeln`

writeln

Writes one or more HTML expressions to a document in the specified window and follows them with a newline character.

Method of	`document`
Implemented in	Navigator 2.0 Navigator 3.0: users can print and save generated HTML using the commands on the File menu

Syntax `writeln(expr1, ... exprN)`

Parameters

`expr1, ... exprN`	Any JavaScript expressions.

Description The `writeln` method displays any number of expressions in a document window. You can specify any JavaScript expression, including numeric, string, or logical expressions.

The `writeln` method is the same as the `write` method, except the `writeln` method appends a newline character to the end of the output. HTML ignores the newline character, except within certain tags such as the `PRE` tag.

Use the `writeln` method within any `SCRIPT` tag or within an event handler. Event handlers execute after the original document closes, so the `writeln` method will implicitly open a new document of `mimeType` `text/html` if you do not explicitly issue a `document.open` method in the event handler.

Examples All the examples used for the `write` method are also valid with the `writeln` method.

See also `document.close`, `document.open`, `document.write`

Link

A piece of text, an image, or an area of an image identified as a hypertext link. When the user clicks the link text, image, or area, the link hypertext reference is loaded into its target window. `Area` objects are a type of `Link` object.

Client-side object

Implemented in	Navigator 2.0 Navigator 3.0: added `onMouseOut` event handler; added `Area` objects; `links` array contains areas created with `<AREA HREF="...">` Navigator 4.0: added `handleEvent` method

Created by

By using the HTML `A` or `AREA` tag or by a call to the `String.link` method. The JavaScript runtime engine creates a `Link` object corresponding to each `A` and `AREA` tag in your document that supplies the `HREF` attribute. It puts these objects as an array in the `document.links` property. You access a `Link` object by indexing this array.

To define a link with the `String.link` method:

```
theString.link(hrefAttribute)
```

where:

`theString`	A `String` object.
`hrefAttribute`	Any string that specifies the `HREF` attribute of the `A` tag; it should be a valid URL (relative or absolute).

To define a link with the `A` or `MAP` tag, use standard HTML syntax with the addition of JavaScript event handlers. If you're going to use the `onMouseOut` or `onMouseOver` event handlers, you must supply a value for the `HREF` attribute.

Event handlers

`Area` objects have the following event handlers:

- `onDblClick`
- `onMouseOut`
- `onMouseOver`

`Link` objects have the following event handlers:

- `onClick`
- `onDblClick`
- `onKeyDown`

- `onKeyPress`
- `onKeyUp`
- `onMouseDown`
- `onMouseOut`
- `onMouseUp`
- `onMouseOver`

Description Each `Link` object is a `location` object and has the same properties as a `location` object.

If a `Link` object is also an `Anchor` object, the object has entries in both the `anchors` and `links` arrays.

When a user clicks a `Link` object and navigates to the destination document (specified by `HREF="locationOrURL"`), the destination document's `referrer` property contains the URL of the source document. Evaluate the `referrer` property from the destination document.

You can use a `Link` object to execute a JavaScript function rather than link to a hypertext reference by specifying the `javascript:` URL protocol for the link's `HREF` attribute. You might want to do this if the link surrounds an `Image` object and you want to execute JavaScript code when the image is clicked. Or you might want to use a link instead of a button to execute JavaScript code.

For example, when a user clicks the following links, the `slower` and `faster` functions execute:

```
<A HREF="javascript:slower()">Slower</A>
<A HREF="javascript:faster()">Faster</A>
```

You can use a `Link` object to do nothing rather than link to a hypertext reference by specifying the `javascript:void(0)` URL protocol for the link's `HREF` attribute. You might want to do this if the link surrounds an `Image` object and you want to use the link's event handlers with the image. When a user clicks the following link or image, nothing happens:

```
<A HREF="javascript:void(0)">Click here to do nothing</A>

<A HREF="javascript:void(0)">
   <IMG SRC="images\globe.gif" ALIGN="top" HEIGHT="50" WIDTH="50">
</A>
```

Property Summary

Property	Description
`hash`	Specifies an anchor name in the URL.
`host`	Specifies the host and domain name, or IP address, of a network host.
`hostname`	Specifies the `host:port` portion of the URL.
`href`	Specifies the entire URL.
`pathname`	Specifies the URL-path portion of the URL.
`port`	Specifies the communications port that the server uses.
`protocol`	Specifies the beginning of the URL, including the colon.
`search`	Specifies a query string.
`target`	Reflects the `TARGET` attribute.
`text`	A string containing the content of the corresponding `A` tag.

Method Summary

Method	Description
`handleEvent`	Invokes the handler for the specified event.

Examples

Example 1. The following example creates a hypertext link to an anchor named `javascript_intro`:

```
<A HREF="#javascript_intro">Introduction to JavaScript</A>
```

Example 2. The following example creates a hypertext link to an anchor named `numbers` in the file `doc3.html` in the window `window2`. If `window2` does not exist, it is created.

```
<LI><A HREF=doc3.html#numbers TARGET="window2">Numbers</A>
```

Example 3. The following example takes the user back `x` entries in the history list:

```
<A HREF="javascript:history.go(-1 * x)">Click here</A>
```

Example 4. The following example creates a hypertext link to a URL. The user can use the set of radio buttons to choose between three URLs. The link's `onClick` event handler sets the URL (the link's `href` property) based on the selected radio button. The link also has an `onMouseOver` event handler that changes the window's `status` property. As the example shows, you must return true to set the `window.status` property in the `onMouseOver` event handler.

```
<SCRIPT>
var destHREF="http://home.netscape.com/"
</SCRIPT>
<FORM NAME="form1">
<B>Choose a destination from the following list, then click "Click me" below.</B>
<BR><INPUT TYPE="radio" NAME="destination" VALUE="netscape"
   onClick="destHREF='http://home.netscape.com/'"> Netscape home page
<BR><INPUT TYPE="radio" NAME="destination" VALUE="sun"
   onClick="destHREF='http://www.sun.com/'"> Sun home page
<BR><INPUT TYPE="radio" NAME="destination" VALUE="rfc1867"
   onClick="destHREF='http://www.ics.uci.edu/pub/ietf/html/rfc1867.txt'"> RFC 1867
<P><A HREF=""
   onMouseOver="window.status='Click this if you dare!'; return true"
   onClick="this.href=destHREF">
   <B>Click me</B></A>
</FORM>
```

Example 5: links array. In the following example, the `linkGetter` function uses the `links` array to display the value of each link in the current document. The example also defines several links and a button for running `linkGetter.`

```
function linkGetter() {
   msgWindow=window.open("","msg","width=400,height=400")
   msgWindow.document.write("links.length is " +
      document.links.length + "<BR>")
   for (var i = 0; i < document.links.length; i++) {
      msgWindow.document.write(document.links[i] + "<BR>")
   }
}

<A HREF="http://home.netscape.com">Netscape Home Page</A>
<A HREF="http://www.catalog.com/fwcfc/">China Adoptions</A>
<A HREF="http://www.supernet.net/~dugbrown/">Bad Dog Chronicles</A>
<A HREF="http://www.best.com/~doghouse/homecnt.shtml">Lab Rescue</A>
<P>
<INPUT TYPE="button" VALUE="Display links"
   onClick="linkGetter()">
```

Example 6: Refer to Area object with links array. The following code refers to the `href` property of the first `Area` object shown in Example 1.

```
document.links[0].href
```

Example 7: Area object with onMouseOver and onMouseOut event handlers. The following example displays an image, `globe.gif`. The image uses an image map that defines areas for the top half and the bottom half of the image. The `onMouseOver` and `onMouseOut` event handlers display different status bar messages depending on whether the mouse passes over or leaves the top half or bottom half of the image. The `HREF` attribute is required when using the `onMouseOver` and `onMouseOut` event handlers, but in this example the image does not need a hypertext link, so the `HREF` attribute executes `javascript:void(0)`, which does nothing.

```
<MAP NAME="worldMap">
   <AREA NAME="topWorld" COORDS="0,0,50,25" HREF="javascript:void(0)"
      onMouseOver="self.status='You are on top of the world';return true"
      onMouseOut="self.status='You have left the top of the world';return true">
   <AREA NAME="bottomWorld" COORDS="0,25,50,50" HREF="javascript:void(0)"
      onMouseOver="self.status='You are on the bottom of the world';return true"
      onMouseOut="self.status='You have left the bottom of the world';return true">
</MAP>
<IMG SRC="images\globe.gif" ALIGN="top" HEIGHT="50" WIDTH="50" USEMAP="#worldMap">
```

Example 8: Simulate an Area object's onClick using the HREF attribute. The following example uses an `Area` object's HREF attribute to execute a JavaScript function. The image displayed, `colors.gif`, shows two sample colors. The top half of the image is the color antiquewhite, and the bottom half is white. When the user clicks the top or bottom half of the image, the function `setBGColor` changes the document's background color to the color shown in the image.

```
<SCRIPT>
function setBGColor(theColor) {
   document.bgColor=theColor
}
</SCRIPT>
Click the color you want for this document's background color
<MAP NAME="colorMap">
   <AREA NAME="topColor" COORDS="0,0,50,25" HREF="javascript:setBGColor('antiquewhite')">
   <AREA NAME="bottomColor" COORDS="0,25,50,50" HREF="javascript:setBGColor('white')">
</MAP>
<IMG SRC="images\colors.gif" ALIGN="top" HEIGHT="50" WIDTH="50" USEMAP="#colorMap">
```

See also `Anchor`, `Image`, `link`

Properties

hash

A string beginning with a hash mark (#) that specifies an anchor name in the URL.

Property of	`Link`
Implemented in	Navigator 2.0

Security Navigator 3.0: This property is tainted by default. For information on data tainting, see "Security" on page 55.

Description The `hash` property specifies a portion of the URL. This property applies to HTTP URLs only.

Be careful using this property. Assume `document.links[0]` contains:

```
http://royalairways.com/fish.htm#angel
```

Then `document.links[0].hash` returns `#angel`. Assume you have this code:

```
hash = document.links[0].hash;
document.links[0].hash = hash;
```

Now, `document.links[0].hash` returns `##angel`.

This behavior may change in a future release.

You can set the `hash` property at any time, although it is safer to set the `href` property to change a location. If the hash that you specify cannot be found in the current location, you get an error.

Setting the `hash` property navigates to the named anchor without reloading the document. This differs from the way a document is loaded when other `link` properties are set.

See RFC 1738 (`http://www.cis.ohio-state.edu/htbin/rfc/rfc1738.html`) for complete information about the hash.

See also `Link.host`, `Link.hostname`, `Link.href`, `Link.pathname`, `Link.port`, `Link.protocol`, `Link.search`

host

A string specifying the server name, subdomain, and domain name.

Property of	`Link`
Implemented in	Navigator 2.0

Security Navigator 3.0: This property is tainted by default. For information on data tainting, see "Security" on page 55.

Description The `host` property specifies a portion of a URL. The `host` property is a substring of the `hostname` property. The `hostname` property is the concatenation of the `host` and `port` properties, separated by a colon. When the `port` property is null, the `host` property is the same as the `hostname` property.

You can set the `host` property at any time, although it is safer to set the `href` property to change a location. If the host that you specify cannot be found in the current location, you get an error.

See Section 3.1 of RFC 1738 (`http://www.cis.ohio-state.edu/htbin/rfc/rfc1738.html`) for complete information about the hostname and port.

See also `Link.hash`, `Link.hostname`, `Link.href`, `Link.pathname`, `Link.port`, `Link.protocol`, `Link.search`

hostname

A string containing the full hostname of the server, including the server name, subdomain, domain, and port number.

Property of	`Link`
Implemented in	Navigator 2.0

Security Navigator 3.0: This property is tainted by default. For information on data tainting, see "Security" on page 55.

Description The `hostname` property specifies a portion of a URL. The `hostname` property is the concatenation of the `host` and `port` properties, separated by a colon. When the `port` property is 80 (the default), the `host` property is the same as the `hostname` property.

You can set the `hostname` property at any time, although it is safer to set the `href` property to change a location. If the hostname that you specify cannot be found in the current location, you get an error.

See Section 3.1 of RFC 1738 (`http://www.cis.ohio-state.edu/htbin/rfc/rfc1738.html`) for complete information about the hostname.

See also `Link.host`, `Link.hash`, `Link.href`, `Link.pathname`, `Link.port`, `Link.protocol`, `Link.search`

href

A string specifying the entire URL.

Property of	`Link`
Implemented in	Navigator 2.0

Security Navigator 3.0: This property is tainted by default. For information on data tainting, see "Security" on page 55.

Description The `href` property specifies the entire URL. Other `link` object properties are substrings of the `href` property.

You can set the `href` property at any time.

See RFC 1738 (`http://www.cis.ohio-state.edu/htbin/rfc/rfc1738.html`) for complete information about the URL.

See also `Link.hash`, `Link.host`, `Link.hostname`, `Link.pathname`, `Link.port`, `Link.protocol`, `Link.search`

pathname

A string specifying the URL-path portion of the URL.

Property of	`Link`
Implemented in	Navigator 2.0

Security Navigator 3.0: This property is tainted by default. For information on data tainting, see "Security" on page 55.

Description The `pathname` property specifies a portion of the URL. The pathname supplies the details of how the specified resource can be accessed.

You can set the `pathname` property at any time, although it is safer to set the `href` property to change a location. If the pathname that you specify cannot be found in the current location, you get an error.

See Section 3.1 of RFC 1738 (`http://www.cis.ohio-state.edu/htbin/rfc/rfc1738.html`) for complete information about the pathname.

See also `Link.host`, `Link.hostname`, `Link.hash`, `Link.href`, `Link.port`, `Link.protocol`, `Link.search`

port

A string specifying the communications port that the server uses.

Property of	`Link`
Implemented in	Navigator 2.0

Security Navigator 3.0: This property is tainted by default. For information on data tainting, see “Security” on page 55.

Description The `port` property specifies a portion of the URL. The `port` property is a substring of the `hostname` property. The `hostname` property is the concatenation of the `host` and `port` properties, separated by a colon. When the `port` property is 80 (the default), the `host` property is the same as the `hostname` property.

You can set the `port` property at any time, although it is safer to set the `href` property to change a location. If the port that you specify cannot be found in the current location, you will get an error. If the `port` property is not specified, it defaults to 80 on the server.

See Section 3.1 of RFC 1738 (`http://www.cis.ohio-state.edu/htbin/rfc/rfc1738.html`) for complete information about the port.

See also `Link.host`, `Link.hostname`, `Link.hash`, `Link.href`, `Link.pathname`, `Link.protocol`, `Link.search`

protocol

A string specifying the beginning of the URL, up to and including the first colon.

Property of	`Link`
Implemented in	Navigator 2.0

Security Navigator 3.0: This property is tainted by default. For information on data tainting, see "Security" on page 55.

Description The `protocol` property specifies a portion of the URL. The protocol indicates the access method of the URL. For example, the value `"http:"` specifies HyperText Transfer Protocol, and the value `"javascript:"` specifies JavaScript code.

You can set the `protocol` property at any time, although it is safer to set the `href` property to change a location. If the protocol that you specify cannot be found in the current location, you get an error.

The `protocol` property represents the scheme name of the URL. See Section 2.1 of RFC 1738 (`http://www.cis.ohio-state.edu/htbin/rfc/rfc1738.html`) for complete information about the protocol.

See also `Link.host`, `Link.hostname`, `Link.hash`, `Link.href`, `Link.pathname`, `Link.port`, `Link.search`

search

A string beginning with a question mark that specifies any query information in the URL.

Property of	`Link`
Implemented in	Navigator 2.0

Security Navigator 3.0: This property is tainted by default. For information on data tainting, see "Security" on page 55.

Description The `search` property specifies a portion of the URL. This property applies to http URLs only.

The `search` property contains variable and value pairs; each pair is separated by an ampersand. For example, two pairs in a search string could look like the following:

```
?x=7&y=5
```

You can set the `search` property at any time, although it is safer to set the `href` property to change a location. If the search that you specify cannot be found in the current location, you get an error.

See Section 3.3 of RFC 1738 (`http://www.cis.ohio-state.edu/htbin/rfc/rfc1738.html`) for complete information about the search.

See also `Link.host`, `Link.hostname`, `Link.hash`, `Link.href`, `Link.pathname`, `Link.port`, `Link.protocol`

target

A string specifying the name of the window that displays the content of a clicked hypertext link.

Property of	`Link`
Implemented in	Navigator 2.0

Description The `target` property initially reflects the `TARGET` attribute of the `A` or `AREA` tags; however, setting `target` overrides this attribute.

You can set `target` using a string, if the string represents a window name. The `target` property cannot be assigned the value of a JavaScript expression or variable.

You can set the `target` property at any time.

Examples The following example specifies that responses to the `musicInfo` form are displayed in the `msgWindow` window:

```
document.musicInfo.target="msgWindow"
```

See also `Form`

text

A string containing the content of the corresponding `A` tag.

Property of	`Link`
Implemented in	Navigator 4.0

Methods

handleEvent

Invokes the handler for the specified event.

Method of	`Link`
Implemented in	Navigator 4.0

Syntax `handleEvent(event)`

Parameters

`event`	The name of an event for which the specified object has an event handler.

Description For information on handling events, see "General Information about Events" on page 481.

Area

Defines an area of an image as an image map. When the user clicks the area, the area's hypertext reference is loaded into its target window. `Area` objects are a type of `Link` object.

Client-side object	
Implemented in	Navigator 3.0:

For information on `Area` objects, see `Link`.

Anchor

A place in a document that is the target of a hypertext link.

Client-side object	
Implemented in	Navigator 2.0

Created by Using the HTML `A` tag or calling the `String.anchor` method. The JavaScript runtime engine creates an `Anchor` object corresponding to each `A` tag in your document that supplies the `NAME` attribute. It puts these objects in an array in the `document.anchors` property. You access an `Anchor` object by indexing this array.

To define an anchor with the `String.anchor` method:

```
theString.anchor(nameAttribute)
```

where:

`theString`	A `String` object.
`nameAttribute`	A string.

To define an anchor with the `A` tag, use standard HTML syntax. If you specify the `NAME` attribute, you can use the value of that attribute to index into the `anchors` array.

Description If an `Anchor` object is also a `Link` object, the object has entries in both the `anchors` and `links` arrays.

Properties None.

Methods None.

Examples **Example 1: An anchor.** The following example defines an anchor for the text "Welcome to JavaScript":

```
<A NAME="javascript_intro"><H2>Welcome to JavaScript</H2></A>
```

If the preceding anchor is in a file called `intro.html`, a link in another file could define a jump to the anchor as follows:

```
<A HREF="intro.html#javascript_intro">Introduction</A>
```

Example 2: anchors array. The following example opens two windows. The first window contains a series of buttons that set `location.hash` in the second window to a specific anchor. The second window defines four anchors named "0," "1," "2," and "3." (The anchor names in the document are therefore 0, 1, 2, ... (document.anchors.length-1).) When a button is pressed in the first window, the `onClick` event handler verifies that the anchor exists before setting `window2.location.hash` to the specified anchor name.

`link1.html`, which defines the first window and its buttons, contains the following code:

```
<HTML>
<HEAD>
<TITLE>Links and Anchors: Window 1</TITLE>
</HEAD>
<BODY>
<SCRIPT>
window2=open("link2.html","secondLinkWindow",
   "scrollbars=yes,width=250, height=400")
function linkToWindow(num) {
   if (window2.document.anchors.length > num)
      window2.location.hash=num
   else
      alert("Anchor does not exist!")
}
</SCRIPT>
<B>Links and Anchors</B>
<FORM>
<P>Click a button to display that anchor in window #2
<P><INPUT TYPE="button" VALUE="0" NAME="link0_button"
   onClick="linkToWindow(this.value)">
<INPUT TYPE="button" VALUE="1" NAME="link0_button"
   onClick="linkToWindow(this.value)">
<INPUT TYPE="button" VALUE="2" NAME="link0_button"
   onClick="linkToWindow(this.value)">
<INPUT TYPE="button" VALUE="3" NAME="link0_button"
   onClick="linkToWindow(this.value)">
<INPUT TYPE="button" VALUE="4" NAME="link0_button"
   onClick="linkToWindow(this.value)">
</FORM>
</BODY>
</HTML>
```

`link2.html`, which contains the anchors, contains the following code:

```
<HTML>
<HEAD>
<TITLE>Links and Anchors: Window 2</TITLE>
</HEAD>
```

```
<BODY>
<A NAME="0"><B>Some numbers</B> (Anchor 0)</A>
<UL><LI>one
<LI>two
<LI>three
<LI>four</UL>

<P><A NAME="1"><B>Some colors</B> (Anchor 1)</A>
<UL><LI>red
<LI>orange
<LI>yellow
<LI>green</UL>

<P><A NAME="2"><B>Some music types</B> (Anchor 2)</A>
<UL><LI>R&B
<LI>Jazz
<LI>Soul
<LI>Reggae
<LI>Rock</UL>

<P><A NAME="3"><B>Some countries</B> (Anchor 3)</A>
<UL><LI>Afghanistan
<LI>Brazil
<LI>Canada
<LI>Finland
<LI>India</UL>
</BODY>
</HTML>
```

See also `Link`

Image

An image on an HTML form.

Client-side object	
Implemented in	Navigator 3.0 Navigator 4.0: added `handleEvent` method

Created by The `Image` constructor or the `IMG` tag.

The JavaScript runtime engine creates an `Image` object corresponding to each `IMG` tag in your document. It puts these objects in an array in the `document.images` property. You access an `Image` object by indexing this array.

To define an image with the `IMG` tag, use standard HTML syntax with the addition of JavaScript event handlers. If specify a value for the `NAME` attribute, you can use that name when indexing the `images` array.

To define an image with its constructor, use the following syntax:

```
new Image(width, height)
```

Parameters

`width`	(Optional) The image width, in pixels.
`height`	(Optional) The image height, in pixels.

Event handlers

- `onAbort`
- `onError`
- `onKeyDown`
- `onKeyPress`
- `onKeyUp`
- `onLoad`

To define an event handler for an `Image` object created with the `Image` constructor, set the appropriate property of the object. For example, if you have an `Image` object named `imageName` and you want to set one of its event handlers to a function whose name is `handlerFunction`, use one of the following statements:

```
imageName.onabort = handlerFunction
imageName.onerror = handlerFunction
imageName.onkeydown = handlerFunction
imageName.onkeypress = handlerFunction
imageName.onkeyup = handlerFunction
imageName.onload = handlerFunction
```

`Image` objects do not have `onClick`, `onMouseOut`, and `onMouseOver` event handlers. However, if you define an `Area` object for the image or place the `IMG` tag within a `Link` object, you can use the `Area` or `Link` object's event handlers. See `Link`.

Description The position and size of an image in a document are set when the document is displayed in the web browser and cannot be changed using JavaScript (the `width` and `height` properties are read-only for these objects). You can change which image is displayed by setting the `src` and `lowsrc` properties. (See the descriptions of `Image.src` and `Image.lowsrc`.)

You can use JavaScript to create an animation with an `Image` object by repeatedly setting the `src` property, as shown in Example 4 below. JavaScript animation is slower than GIF animation, because with GIF animation the entire animation is in one file; with JavaScript animation, each frame is in a separate file, and each file must be loaded across the network (host contacted and data transferred).

The primary use for an `Image` object created with the `Image` constructor is to load an image from the network (and decode it) before it is actually needed for display. Then when you need to display the image within an existing image cell, you can set the `src` property of the displayed image to the same value as that used for the previously fetched image, as follows.

```
myImage = new Image()
myImage.src = "seaotter.gif"
...
document.images[0].src = myImage.src
```

The resulting image will be obtained from cache, rather than loaded over the network, assuming that sufficient time has elapsed to load and decode the entire image. You can use this technique to create smooth animations, or you could display one of several images based on form input.

Property Summary

Property	Description
`border`	Reflects the `BORDER` attribute.
`complete`	Boolean value indicating whether the web browser has completed its attempt to load the image.
`height`	Reflects the `HEIGHT` attribute.
`hspace`	Reflects the `HSPACE` attribute.
`lowsrc`	Reflects the `LOWSRC` attribute.
`name`	Reflects the `NAME` attribute.
`prototype`	Allows the addition of properties to an `Image` object.
`src`	Reflects the `SRC` attribute.
`vspace`	Reflects the `VSPACE` attribute.
`width`	Reflects the `WIDTH` attribute.

Method Summary

Method	Description
`handleEvent`	Invokes the handler for the specified event.

Examples **Example 1: Create an image with the `IMG` tag.** The following code defines an image using the `IMG` tag:

```
<IMG NAME="aircraft" SRC="f15e.gif" ALIGN="left" VSPACE="10">
```

The following code refers to the image:

```
document.aircraft.src='f15e.gif'
```

When you refer to an image by its name, you must include the form name if the image is on a form. The following code refers to the image if it is on a form:

```
document.myForm.aircraft.src='f15e.gif'
```

Example 2: Create an image with the Image constructor. The following example creates an `Image` object, `myImage`, that is 70 pixels wide and 50 pixels high. If the source URL, `seaotter.gif`, does not have dimensions of 70x50 pixels, it is scaled to that size.

```
myImage = new Image(70, 50)
myImage.src = "seaotter.gif"
```

If you omit the width and height arguments from the `Image` constructor, `myImage` is created with dimensions equal to that of the image named in the source URL.

```
myImage = new Image()
myImage.src = "seaotter.gif"
```

Example 3: Display an image based on form input. In the following example, the user selects which image is displayed. The user orders a shirt by filling out a form. The image displayed depends on the shirt color and size that the user chooses. All possible image choices are preloaded to speed response time. When the user clicks the button to order the shirt, the `allShirts` function displays the images of all the shirts.

```
<SCRIPT>
shirts = new Array()
shirts[0] = "R-S"
shirts[1] = "R-M"
shirts[2] = "R-L"
shirts[3] = "W-S"
```

```
shirts[4] = "W-M"
shirts[5] = "W-L"
shirts[6] = "B-S"
shirts[7] = "B-M"
shirts[8] = "B-L"

doneThis = 0
shirtImg = new Array()

// Preload shirt images
for(idx=0; idx < 9; idx++) {
   shirtImg[idx] = new Image()
   shirtImg[idx].src = "shirt-" + shirts[idx] + ".gif"
}

function changeShirt(form)
{
   shirtColor = form.color.options[form.color.selectedIndex].text
   shirtSize = form.size.options[form.size.selectedIndex].text

   newSrc = "shirt-" + shirtColor.charAt(0) + "-" + shirtSize.charAt(0)
+ ".gif"
   document.shirt.src = newSrc
}

function allShirts()
{
   document.shirt.src = shirtImg[doneThis].src
   doneThis++
   if(doneThis != 9)setTimeout("allShirts()", 500)
   else doneThis = 0

   return
}

</SCRIPT>

<FONT SIZE=+2><B>Netscape Polo Shirts!</FONT></B>

<TABLE CELLSPACING=20 BORDER=0>
<TR>
<TD><IMG name="shirt" SRC="shirt-W-L.gif"></TD>

<TD>
<FORM>
<B>Color</B>
<SELECT SIZE=3 NAME="color" onChange="changeShirt(this.form)">
<OPTION> Red
<OPTION SELECTED> White
<OPTION> Blue
</SELECT>

<P>
<B>Size</B>
<SELECT SIZE=3 NAME="size" onChange="changeShirt(this.form)">
```

```
<OPTION> Small
<OPTION> Medium
<OPTION SELECTED> Large
</SELECT>

<P><INPUT type="button" name="buy" value="Buy This Shirt!"
   onClick="allShirts()">
</FORM>

</TD>
</TR>
</TABLE>
```

Example 4: JavaScript animation. The following example uses JavaScript to create an animation with an `Image` object by repeatedly changing the value the `src` property. The script begins by preloading the 10 images that make up the animation (`image1.gif`, `image2.gif`, `image3.gif`, and so on). When the `Image` object is placed on the document with the `IMG` tag, `image1.gif` is displayed and the `onLoad` event handler starts the animation by calling the `animate` function. Notice that the `animate` function does not call itself after changing the `src` property of the `Image` object. This is because when the `src` property changes, the image's `onLoad` event handler is triggered and the `animate` function is called.

```
<SCRIPT>
delay = 100
imageNum = 1

// Preload animation images
theImages = new Array()
for(i = 1; i < 11; i++) {
   theImages[i] = new Image()
   theImages[i].src = "image" + i + ".gif"
}

function animate() {
   document.animation.src = theImages[imageNum].src
   imageNum++
   if(imageNum > 10) {
      imageNum = 1
   }
}

function slower() {
   delay+=10
   if(delay > 4000) delay = 4000
}

function faster() {
   delay-=10
   if(delay < 0) delay = 0
}
</SCRIPT>

<BODY BGCOLOR="white">

<IMG NAME="animation" SRC="image1.gif" ALT="[Animation]"
   onLoad="setTimeout('animate()', delay)">

<FORM>
   <INPUT TYPE="button" Value="Slower" onClick="slower()">
   <INPUT TYPE="button" Value="Faster" onClick="faster()">
</FORM>
</BODY>
```

See also the examples for the `onAbort`, `onError`, and `onLoad` event handlers.

See also `Link`, `onClick`, `onMouseOut`, `onMouseOver`

Properties

border

A string specifying the width, in pixels, of an image border.

Property of	`Image`
Read-only	
Implemented in	Navigator 3.0:

Description The `border` property reflects the `BORDER` attribute of the `IMG` tag. For images created with the `Image` constructor, the value of the `border` property is 0.

Examples The following function displays the value of an image's `border` property if the value is not 0.

```
function checkBorder(theImage) {
   if (theImage.border==0) {
      alert('The image has no border!')
   }
   else alert('The image's border is ' + theImage.border)
}
```

See also `Image.height`, `Image.hspace`, `Image.vspace`, `Image.width`

complete

A boolean value that indicates whether the web browser has completed its attempt to load an image.

Property of	`Image`
Read-only	
Implemented in	Navigator 3.0:

Examples The following example displays an image and three radio buttons. The user can click the radio buttons to choose which image is displayed. Clicking another button lets the user see the current value of the `complete` property.

```
<B>Choose an image:</B>
<BR><INPUT TYPE="radio" NAME="imageChoice" VALUE="image1" CHECKED
   onClick="document.images[0].src='f15e.gif'">F-15 Eagle
<BR><INPUT TYPE="radio" NAME="imageChoice" VALUE="image2"
   onClick="document.images[0].src='f15e2.gif'">F-15 Eagle 2
<BR><INPUT TYPE="radio" NAME="imageChoice" VALUE="image3"
   onClick="document.images[0].src='ah64.gif'">AH-64 Apache

<BR><INPUT TYPE="button" VALUE="Is the image completely loaded?"
   onClick="alert('The value of the complete property is '
      + document.images[0].complete)">
<BR>
<IMG NAME="aircraft" SRC="f15e.gif" ALIGN="left" VSPACE="10"><BR>
```

See also `Image.lowsrc`, `Image.src`

height

A string specifying the height of an image in pixels.

Property of	`Image`
Read-only	
Implemented in	Navigator 3.0:

Description The `height` property reflects the `HEIGHT` attribute of the `IMG` tag. For images created with the `Image` constructor, the value of the `height` property is the actual, not the displayed, height of the image.

Examples The following function displays the values of an image's `height`, `width`, `hspace`, and `vspace` properties.

```
function showImageSize(theImage) {
   alert('height=' + theImage.height+
      '; width=' + theImage.width +
      '; hspace=' + theImage.hspace +
      '; vspace=' + theImage.vspace)
}
```

See also `Image.border`, `Image.hspace`, `Image.vspace`, `Image.width`

hspace

A string specifying a margin in pixels between the left and right edges of an image and the surrounding text.

Property of	`Image`
Read-only	
Implemented in	Navigator 3.0:

Description The `hspace` property reflects the `HSPACE` attribute of the `IMG` tag. For images created with the `Image` constructor, the value of the `hspace` property is 0.

Examples See the examples for the `height` property.

See also `Image.border`, `Image.height`, `Image.vspace`, `Image.width`

lowsrc

A string specifying the URL of a low-resolution version of an image to be displayed in a document.

Property of	`Image`
Implemented in	Navigator 3.0:

Description The `lowsrc` property initially reflects the `LOWSRC` attribute of the `IMG` tag. The web browser loads the smaller image specified by `lowsrc` and then replaces it with the larger image specified by the `src` property. You can change the `lowsrc` property at any time.

Examples See the examples for the `src` property.

See also `Image.complete`, `Image.src`

name

A string specifying the name of an object.

Property of	`Image`
Read-only	
Implemented in	Navigator 3.0:

Security Navigator 3.0: This property is tainted by default. For information on data tainting, see "Security" on page 55.

Description Represents the value of the `NAME` attribute. For images created with the `Image` constructor, the value of the `name` property is null.

Examples In the following example, the `valueGetter` function uses a `for` loop to iterate over the array of elements on the `valueTest` form. The `msgWindow` window displays the names of all the elements on the form:

```
newWindow=window.open("http://home.netscape.com")

function valueGetter() {
   var msgWindow=window.open("")
   for (var i = 0; i < newWindow.document.valueTest.elements.length; i++) {
      msgWindow.document.write(newWindow.document.valueTest.elements[i].name + "<BR>")
   }
}
```

In the following example, the first statement creates a window called `netscapeWin`. The second statement displays the value `"netscapeHomePage"` in the Alert dialog box, because `"netscapeHomePage"` is the value of the `windowName` argument of `netscapeWin`.

```
netscapeWin=window.open("http://home.netscape.com","netscapeHomePage")
alert(netscapeWin.name)
```

prototype

Represents the prototype for this class. You can use the prototype to add properties or methods to all instances of a class. For more information, see `Function.prototype`.

Property of	`Image`
Implemented in	Navigator 3.0

src

A string specifying the URL of an image to be displayed in a document.

Property of	`Image`
Implemented in	Navigator 3.0:

Description The `src` property initially reflects the `SRC` attribute of the `IMG` tag. Setting the `src` property begins loading the new URL into the image area (and aborts the transfer of any image data that is already loading into the same area). Therefore, if you plan to alter the `lowsrc` property, you should do so before setting the `src` property.

If the URL in the `src` property refers to an image that is not the same size as the image cell it is loaded into, the source image is scaled to fit.

When you change the `src` property of a displayed image, the new image you specify is displayed in the area defined for the original image. For example, suppose an `Image` object originally displays the file `beluga.gif`:

```
<IMG NAME="myImage" SRC="beluga.gif" ALIGN="left">
```

If you set `myImage.src='seaotter.gif'`, the image `seaotter.gif` is scaled to fit in the same space originally used by `beluga.gif`, even if `seaotter.gif` is not the same size as `beluga.gif`.

You can change the `src` property at any time.

Examples The following example displays an image and three radio buttons. The user can click the radio buttons to choose which image is displayed. Each image also uses the `lowsrc` property to display a low-resolution image.

```
<SCRIPT>
function displayImage(lowRes,highRes) {
   document.images[0].lowsrc=lowRes
   document.images[0].src=highRes
}
</SCRIPT>

<FORM NAME="imageForm">
<B>Choose an image:</B>
<BR><INPUT TYPE="radio" NAME="imageChoice" VALUE="image1" CHECKED
   onClick="displayImage('f15el.gif','f15e.gif')">F-15 Eagle
<BR><INPUT TYPE="radio" NAME="imageChoice" VALUE="image2"
   onClick="displayImage('f15e2l.gif','f15e2.gif')">F-15 Eagle 2
<BR><INPUT TYPE="radio" NAME="imageChoice" VALUE="image3"
   onClick="displayImage('ah64l.gif','ah64.gif')">AH-64 Apache
```

```
<BR>
<IMG NAME="aircraft" SRC="f15e.gif" LOWSRC="f15el.gif" ALIGN="left" VSPACE="10"><BR>
</FORM>
```

See also `Image.complete`, `Image.lowsrc`

vspace

A string specifying a margin in pixels between the top and bottom edges of an image and the surrounding text.

Property of	`Image`
Read-only	
Implemented in	Navigator 3.0:

Description The `vspace` property reflects the `VSPACE` attribute of the `IMG` tag. For images created with the `Image` constructor, the value of the `vspace` property is 0.

Examples See the examples for the `height` property.

See also `Image.border`, `Image.height`, `Image.hspace`, `Image.width`

width

A string specifying the width of an image in pixels.

Property of	`Image`
Read-only	
Implemented in	Navigator 3.0:

Description The `width` property reflects the `WIDTH` attribute of the `IMG` tag. For images created with the `Image` constructor, the value of the `width` property is the actual, not the displayed, width of the image.

Examples See the examples for the `height` property.

See also `Image.border`, `Image.height`, `Image.hspace`, `Image.vspace`

Methods

handleEvent

Invokes the handler for the specified event.

Method of	`Image`
Implemented in	Navigator 4.0:

Syntax `handleEvent(event)`

Parameters

`event`	The name of an event for which the specified object has an event handler.

Description For information on handling events, see "General Information about Events" on page 481.

Applet

Includes a Java applet in a web page.

Client-side object	
Implemented in	Navigator 3.0:

Created by The HTML `APPLET` tag. The JavaScript runtime engine creates an `Applet` object corresponding to each applet in your document. It puts these objects in an array in the `document.applets` property. You access an `Applet` object by indexing this array.

To define an applet, use standard HTML syntax. If you specify the `NAME` attribute, you can use the value of that attribute to index into the `applets` array. To refer to an applet in JavaScript, you must supply the `MAYSCRIPT` attribute in its definition.

Description The author of an HTML page must permit an applet to access JavaScript by specifying the `MAYSCRIPT` attribute of the `APPLET` tag. This prevents an applet from accessing JavaScript on a page without the knowledge of the page author. For example, to allow the `musicPicker.class` applet access to JavaScript on your page, specify the following:

```
<APPLET CODE="musicPicker.class" WIDTH=200 HEIGHT=35
   NAME="musicApp" MAYSCRIPT>
```

Accessing JavaScript when the `MAYSCRIPT` attribute is not specified results in an exception.

For more information on using applets, see the *JavaScript Guide*.

Property Summary All public properties of the applet are available for JavaScript access to the `Applet` object.

Method Summary All public methods of the applet

Examples The following code launches an applet called `musicApp`:

```
<APPLET CODE="musicSelect.class" WIDTH=200 HEIGHT=35
   NAME="musicApp" MAYSCRIPT>
</APPLET>
```

For more examples, see the *JavaScript Guide*.

See also `MimeType`, `Plugin`

Layer

Corresponds to a layer in an HTML page and provides a means for manipulating that layer.

Client-side object	
Implemented in	Navigator 4.0

Created by The HTML `LAYER` or `ILAYER` tag, or using cascading style sheet syntax. The JavaScript runtime engine creates a `Layer` object corresponding to each layer in your document. It puts these objects in an array in the `document.layers` property. You access a `Layer` object by indexing this array.

To define a layer, use standard HTML syntax. If you specify the `ID` attribute, you can use the value of that attribute to index into the `layers` array.

For a complete description of layers, see *Dynamic HTML in Netscape Communicator*[1].

Some `layer` properties can be directly modified by assignment; for example, `"mylayer.visibility = hide"`. A layer object also has methods that can affect these properties.

Event handlers

- `onMouseOver`
- `onMouseOut`
- `onLoad`
- `onFocus`
- `onBlur`

Property Summary

Property	Description
`above`	The `layer` object above this one in z-order, among all layers in the document or the enclosing window object if this layer is topmost.
`background`	The image to use as the background for the layer's canvas.
`bgColor`	The color to use as a solid background color for the layer's canvas.
`below`	The `layer` object below this one in z-order, among all layers in the document or null if this layer is at the bottom.
`clip.bottom`	The bottom edge of the clipping rectangle (the part of the layer that is visible.)
`clip.height`	The height of the clipping rectangle (the part of the layer that is visible.)
`clip.left`	The left edge of the clipping rectangle (the part of the layer that is visible.)
`clip.right`	The right edge of the clipping rectangle (the part of the layer that is visible.)
`clip.top`	The top edge of the clipping rectangle (the part of the layer that is visible.)

1. http://developer.netscape.com/library/documentation/communicator/dynhtml/index.htm

Property	Description
`clip.width`	The width of the clipping rectangle (the part of the layer that is visible.)
`document`	The layer’s associated document.
`left`	The horizontal position of the layer's left edge, in pixels, relative to the origin of its parent layer.
`name`	A string specifying the name assigned to the layer through the `ID` attribute in the `LAYER` tag.
`pageX`	The horizontal position of the layer, in pixels, relative to the page.
`page y`	The vertical position of the layer, in pixels, relative to the page.
`parentLayer`	The `layer` object that contains this layer, or the enclosing `window` object if this layer is not nested in another layer.
`siblingAbove`	The `layer` object above this one in z-order, among all layers that share the same parent layer, or null if the layer has no sibling above.
`siblingBelow`	The `layer` object below this one in z-order, among all layers that share the same parent layer, or null if layer is at the bottom.
`src`	A string specifying the URL of the layer’s content.
`top`	The vertical position of the layer's top edge, in pixels, relative to the origin of its parent layer.
`visibility`	Whether or not the layer is visible.
`zIndex`	The relative z-order of this layer with respect to its siblings.

Method Summary

Method	Description
`captureEvents`	Sets the window or document to capture all events of the specified type.
`handleEvent`	Invokes the handler for the specified event.
`load`	Changes the source of a layer to the contents of the specified file, and simultaneously changes the width at which the layer's HTML contents will be wrapped.
`moveAbove`	Stacks this layer above the layer specified in the argument, without changing either layer's horizontal or vertical position.

Method	Description
`moveBelow`	Stacks this layer below the specified layer, without changing either layer's horizontal or vertical position.
`moveBy`	Changes the layer position by applying the specified deltas, measured in pixels.
`moveTo`	Moves the top-left corner of the window to the specified screen coordinates.
`moveToAbsolute`	Changes the layer position to the specified pixel coordinates within the page (instead of the containing layer.)
`releaseEvents`	Sets the layer to release captured events of the specified type, sending the event to objects further along the event hierarchy.
`resizeBy`	Resizes the layer by the specified height and width values (in pixels).
`resizeTo`	Resizes the layer to have the specified height and width values (in pixels).
`routeEvent`	Passes a captured event along the normal event hierarchy.

Note Just as in the case of a document, if you want to define mouse click response for a layer, you must capture `onMouseDown` and `onMouseUp` events at the level of the layer and process them as you want.

See “Events in Navigator 4.0” on page 482 for more details about capturing events.

If an event occurs in a point where multiple layers overlap, the topmost layer gets the event, even if it is transparent. However, if a layer is hidden, it does not get events.

Properties

above

The `layer` object above this one in z-order, among all layers in the document or the enclosing window object if this layer is topmost.

Property of `Layer`

Read-only

Implemented in	Navigator 4.0

background

The image to use as the background for the layer's canvas (which is the part of the layer within the clip rectangle).

Property of	`Layer`
Implemented in	Navigator 4.0

Description Each layer has a background property, whose value is an image object, whose `src` attribute is a URL that indicates the image to use to provide a tiled backdrop. The value is null if the layer has no backdrop. For example:

```
layer.background.src = "fishbg.gif";
```

bgColor

A string specifying the color to use as a solid background color for the layer's canvas (the part of the layer within the clip rectangle).

Property of	`Layer`
Implemented in	Navigator 4.0

Description The `bgColor` property is expressed as a hexadecimal RGB triplet or as one of the string literals listed in the *JavaScript Guide*. This property is the JavaScript reflection of the `BGCOLOR` attribute of the `BODY` tag.

You can set the `bgColor` property at any time.

If you express the color as a hexadecimal RGB triplet, you must use the format `rrggbb`. For example, the hexadecimal RGB values for salmon are `red=FA`, `green=80`, and `blue=72`, so the RGB triplet for salmon is `"FA8072"`.

Examples The following example sets the background color of the `myLayer` layer's canvas to aqua using a string literal:

```
myLayer.bgColor="aqua"
```

The following example sets the background color of the `myLayer` layer's canvas to aqua using a hexadecimal triplet:

```
myLayer.bgColor="00FFFF"
```

See also `Layer.bgColor`

below

The `layer` object below this one in z-order, among all layers in the document or null if this layer is at the bottom.

Property of	`Layer`
Read-only	
Implemented in	Navigator 4.0

clip.bottom

The bottom edge of the clipping rectangle (the part of the layer that is visible.) Any part of a layer that is outside the clipping rectangle is not displayed.

Property of	`Layer`
Implemented in	Navigator 4.0

clip.height

The height of the clipping rectangle (the part of the layer that is visible.) Any part of a layer that is outside the clipping rectangle is not displayed.

Property of	`Layer`
Implemented in	Navigator 4.0

clip.left

The left edge of the clipping rectangle (the part of the layer that is visible.) Any part of a layer that is outside the clipping rectangle is not displayed.

Property of	`Layer`
Implemented in	Navigator 4.0

clip.right

The right edge of the clipping rectangle (the part of the layer that is visible.) Any part of a layer that is outside the clipping rectangle is not displayed.

Property of	`Layer`
Implemented in	Navigator 4.0

clip.top

The top edge of the clipping rectangle (the part of the layer that is visible.) Any part of a layer that is outside the clipping rectangle is not displayed.

Property of	`Layer`
Implemented in	Navigator 4.0

clip.width

The width of the clipping rectangle (the part of the layer that is visible.) Any part of a layer that is outside the clipping rectangle is not displayed.

Property of	`Layer`
Implemented in	Navigator 4.0

document

The layer's associated document.

Property of	`Layer`
Read-only	
Implemented in	Navigator 4.0

Description Each `layer` object contains its own `document` object. This object can be used to access the images, applets, embeds, links, anchors and layers that are contained within the layer. Methods of the `document` object can also be invoked to change the contents of the layer.

left

The horizontal position of the layer's left edge, in pixels, relative to the origin of its parent layer.

Property of	`Layer`
Implemented in	Navigator 4.0

name

A string specifying the name assigned to the layer through the `ID` attribute in the `LAYER` tag.

Property of	`Layer`
Read-only	
Implemented in	Navigator 4.0

pageX

The horizontal position of the layer, in pixels, relative to the page.

Property of	`Layer`
Implemented in	Navigator 4.0

pageY

The vertical position of the layer, in pixels, relative to the page.

Property of	`Layer`
Implemented in	Navigator 4.0

parentLayer

The `layer` object that contains this layer, or the enclosing `window` object if this layer is not nested in another layer.

Property of	`Layer`
Read-only	

Implemented in	Navigator 4.0

siblingAbove

The layer object above this one in z-order, among all layers that share the same parent layer or null if the layer has no sibling above.

Property of	`Layer`
Read-only	
Implemented in	Navigator 4.0

siblingBelow

The `layer` object below this one in z-order, among all layers that share the same parent layer or null if layer is at the bottom.

Property of	`Layer`
Read-only	
Implemented in	Navigator 4.0

src

A URL string specifying the source of the layer's content. Corresponds to the `SRC` attribute.

Property of	`Layer`
Implemented in	Navigator 4.0

top

The `top` property is a synonym for the topmost Navigator window, which is a document window or web browser window.

Property of	`Layer`
Read-only	
Implemented in	Navigator 4.0

Description The `top` property refers to the topmost window that contains frames or nested framesets. Use the `top` property to refer to this ancestor window.

The value of the `top` property is

`<object` *`objectReference>`*

where `objectReference` is an internal reference.

Examples The statement `top.close()` closes the topmost ancestor window.

The statement `top.length` specifies the number of frames contained within the topmost ancestor window. When the topmost ancestor is defined as follows, `top.length` returns three:

```
<FRAMESET COLS="30%,40%,30%">
<FRAME SRC=child1.htm NAME="childFrame1">
<FRAME SRC=child2.htm NAME="childFrame2">
<FRAME SRC=child3.htm NAME="childFrame3">
</FRAMESET>
```

visibility

Whether or not the layer is visible.

Property of	`Layer`
Implemented in	Navigator 4.0

Description A value of `show` means show the layer; `hide` means hide the layer; `inherit` means inherit the visibility of the parent layer.

zIndex

The relative z-order of this layer with respect to its siblings.

Method of	`Layer`
Implemented in	Navigator 4.0

Description Sibling layers with lower numbered z-indexes are stacked underneath this layer. The value of `zIndex` must be 0 or a positive integer.

Methods

captureEvents

Sets the window or document to capture all events of the specified type.

Method of	`Layer`
Implemented in	Navigator 4.0

Syntax `captureEvents(eventType)`

Parameters

`eventType`	Type of event to be captured. Available event types are listed with `event`.

Description When a window with frames wants to capture events in pages loaded from different locations (servers), you need to use `captureEvents` in a signed script and precede it with `enableExternalCapture`. For more information and an example, see `enableExternalCapture`.

`captureEvents` works in tandem with `releaseEvents`, `routeEvent`, and `handleEvent`. For more information, see "Events in Navigator 4.0" on page 482.

handleEvent

Invokes the handler for the specified event.

Method of	`Layer`
Implemented in	Navigator 4.0

Syntax `handleEvent(event)`

Parameters

`event`	Name of an event for which the specified object has an event handler.

Description `handleEvent` works in tandem with `captureEvents`, `releaseEvents`, and `routeEvent`. For more information, see "Events in Navigator 4.0" on page 482.

load

Changes the source of a layer to the contents of the specified file and simultaneously changes the width at which the layer's HTML contents are wrapped.

Method of	`Layer`
Implemented in	Navigator 4.0

Syntax `load(sourcestring, width)`

Parameters

`sourcestring`	A string indicating the external file name.
`width`	The width of the layer as a pixel value.

moveAbove

Stacks this layer above the layer specified in the argument, without changing either layer's horizontal or vertical position. After re-stacking, both layers will share the same parent layer.

Method of	`Layer`
Implemented in	Navigator 4.0

Syntax `moveAbove(aLayer)`

Parameters

`aLayer`	The layer above which to move the current layer.

moveBelow

Stacks this layer below the specified layer, without changing either layer's horizontal or vertical position. After re-stacking, both layers will share the same parent layer.

Method of	`Layer`
Implemented in	Navigator 4.0

Syntax `moveBelow(aLayer)`

Parameters

`aLayer`	The layer below which to move the current layer.

moveBy

Changes the layer position by applying the specified deltas, measured in pixels.

Method of	`Layer`
Implemented in	Navigator 4.0

Syntax `moveBy(horizontal, vertical)`

Parameters

`horizontal`	The number of pixels by which to move the layer horizontally.
`vertical`	The number of pixels by which to move the layer vertically.

moveTo

Moves the top-left corner of the window to the specified screen coordinates.

Method of	`Layer`
Implemented in	Navigator 4.0

Syntax `moveTo(x-coordinate, y-coordinate)`

Parameters

`x-coordinate`	An integer representing the top edge of the window in screen coordinates.
`y-coordinate`	An integer representing the left edge of the window in screen coordinates.

Security To move a window offscreen, call the `moveTo` method in a signed script. For information on security in Navigator 4.0, see Chapter 7, "JavaScript Security," in the *JavaScript Guide*.

Description Changes the layer position to the specified pixel coordinates within the containing layer. For ILayers, moves the layer relative to the natural inflow position of the layer.

See also `Layer.moveBy`

moveToAbsolute

Changes the layer position to the specified pixel coordinates within the page (instead of the containing layer.)

Method of	`Layer`
Implemented in	Navigator 4.0

Syntax `moveToAbsolute(x, y)`

Parameters

`x`	An integer representing the top edge of the window in pixel coordinates.
`y`	An integer representing the left edge of the window in pixel coordinates.

Description This method is equivalent to setting both the `pageX` and `pageY` properties of the `layer` object.

releaseEvents

Sets the window or document to release captured events of the specified type, sending the event to objects further along the event hierarchy.

Method of	`Layer`

Implemented in Navigator 4.0

Syntax `releaseEvents(eventType)`

Parameters

`eventType`	Type of event to be captured.

Description If the original target of the event is a window, the window receives the event even if it is set to release that type of event. `releaseEvents` works in tandem with `captureEvents`, `routeEvent`, and `handleEvent`. For more information, see "Events in Navigator 4.0" on page 482.

resizeBy

Resizes the layer by the specified height and width values (in pixels).

Method of `Layer`

Implemented in Navigator 4.0

Syntax `resizeBy(width, height)`

Parameters

`width`	The number of pixels by which to resize the layer horizontally.
`height`	The number of pixels by which to resize the layer vertically.

Description This does not layout any HTML contained in the layer again. Instead, the layer contents may be clipped by the new boundaries of the layer. This method has the same effect as adding `width` and `height` to `clip.width` and `clip.height`.

resizeTo

Resizes the layer to have the specified height and width values (in pixels).

Method of `Layer`

Implemented in Navigator 4.0

Description This does not layout any HTML contained in the layer again. Instead, the layer

Parameters

`width`	The number of pixels by which to resize the layer horizontally.
`height`	The number of pixels by which to resize the layer vertically.

Description This does not layout any HTML contained in the layer again. Instead, the layer contents may be clipped by the new boundaries of the layer. This method has the same effect as adding `width` and `height` to `clip.width` and `clip.height`.

resizeTo

Resizes the layer to have the specified height and width values (in pixels).

Method of `Layer`

Implemented in Navigator 4.0

Description This does not layout any HTML contained in the layer again. Instead, the layer contents may be clipped by the new boundaries of the layer.

Syntax `resizeBy(width, height)`

Parameters

`width`	An integer representing the layer's width in pixels.
`height`	An integer representing the layer's height in pixels.

Description This method has the same effect setting `clip.width` and `clip.height`.

routeEvent

Passes a captured event along the normal event hierarchy.

Method of `Layer`

Implemented in Navigator 4.0

Syntax `routeEvent(event)`

Parameters

`event`	The event to route.

Description

Chapter 6

Window

This chapter deals with the `Window` object and the client-side objects associated with it: `Frame`, `Location`, and `History`.

Table 6.1 summarizes the objects in this chapter.

Table 6.1 Window objects

Object	Description
`Frame`	A window that can display multiple, independently scrollable frames on a single screen, each with its own distinct URL.
`History`	Contains an array of information on the URLs that the client has visited within a window.
`Location`	Contains information on the current URL.
`screen`	Contains properties describing the display screen and colors.
`Window`	Represents a browser window or frame. This is the top-level object for each `document`, `Location`, and `History` object group.

Window

Represents a browser window or frame. This is the top-level object for each `document`, `Location`, and `History` object group.

Client-side object.

Implemented in Navigator 2.0
Navigator 3.0: added `closed`, `history`, and `opener` properties; added `blur`, `focus`, and `scroll` methods; added `onBlur`, `onError`, and `onFocus` event handlers
Navigator 4.0: added `innerHeight`, `innerWidth`, `locationbar`, `menubar`, `outerHeight`, `outerWidth`, `pageXOffset`, `pageYOffset`, `personalbar`, `scrollbars`, `statusbar`, and `toolbar` properties; added `back`, `captureEvents`, `clearInterval`, `disableExternalCapture`, `enableExternalCapture`, `find`, `forward`, `handleEvent`, `home`, `moveBy`, `moveTo`, `releaseEvents`, `resizeBy`, `resizeTo`, `routeEvent`, `scrollBy`, `scrollTo`, `setInterval`, and `stop` methods; deprecated `scroll` method.

Created by The JavaScript runtime engine creates a `Window` object for each `BODY` or `FRAMESET` tag. It also creates a `Window` object to represent each frame defined in a `FRAME` tag. In addition, you can create other windows by calling the `Window.open` method. For details on defining a window, see `open`.

Event handlers

- `onBlur`
- `onDragDrop`
- `onError`
- `onFocus`
- `onLoad`
- `onMove`
- `onResize`
- `onUnload`

In Navigator 3.0, on some platforms, placing an `onBlur` or `onFocus` event handler in a `FRAMESET` tag has no effect.

Description The `Window` object is the top-level object in the JavaScript client hierarchy. A `Window` object can represent either a top-level window or a frame inside a frameset. As a matter of convenience, you can think about a `Frame` object as a `Window` object that isn't a top-level window. However, there is not really a separate `Frame` class; these objects really are `Window` objects, with a very few minor differences:

- For a top-level window, the `parent` and `top` properties are references to the window itself. For a frame, the `top` refers to the topmost browser window, and `parent` refers to the parent window of the current window.
- For a top-level window, setting the `defaultStatus` or `status` property sets the text appearing in the browser status line. For a frame, setting these properties only sets the status line text when the cursor is over the frame.
- The `close` method is not useful for windows that are frames.
- To create an `onBlur` or `onFocus` event handler for a frame, you must set the `onblur` or `onfocus` property and specify it in all lowercase (you cannot specify it in HTML).
- If a `FRAME` tag contains `SRC` and `NAME` attributes, you can refer to that frame from a sibling frame by using `parent.frameName` or `parent.frames[index]`. For example, if the fourth frame in a set has `NAME="homeFrame"`, sibling frames can refer to that frame using `parent.homeFrame` or `parent.frames[3]`.

For all windows, the `self` and `window` properties of a `Window` object are synonyms for the current window, and you can optionally use them to refer to the current window. For example, you can close the current window by calling the `close` method of either `window` or `self`. You can use these properties to make your code more readable or to disambiguate the property reference `self.status` from a form called `status`. See the properties and methods listed below for more examples.

Because the existence of the current window is assumed, you do not have to refer to the name of the window when you call its methods and assign its properties. For example, `status="Jump to a new location"` is a valid property assignment, and `close()` is a valid method call.

However, when you open or close a window within an event handler, you must specify `window.open()` or `window.close()` instead of simply using `open()` or `close()`. Due to the scoping of static objects in JavaScript, a call to `close()` without specifying an object name is equivalent to `document.close()`.

For the same reason, when you refer to the `location` object within an event handler, you must specify `window.location` instead of simply using `location`. A call to `location` without specifying an object name is equivalent to `document.location`, which is a synonym for `document.URL`.

You can refer to a window’s `Frame` objects in your code by using the `frames` array. In a window with a `FRAMESET` tag, the `frames` array contains an entry for each frame.

A windows lacks event handlers until HTML that contains a `BODY` or `FRAMESET` tag is loaded into it.

Property Summary

Property	Description
`closed`	Specifies whether a window has been closed.
`defaultStatus`	Reflects the default message displayed in the window’s status bar.
`document`	Contains information on the current document, and provides methods for displaying HTML output to the user.
`frames`	An array reflecting all the frames in a window.
`history`	Contains information on the URLs that the client has visited within a window.
`innerHeight`	Specifies the vertical dimension, in pixels, of the window's content area.
`innerWidth`	Specifies the horizontal dimension, in pixels, of the window's content area.
`length`	The number of frames in the window.
`location`	Contains information on the current URL.
`locationbar`	Represents the browser window's location bar.
`menubar`	Represents the browser window's menu bar.
`name`	A unique name used to refer to this window.
`opener`	Specifies the window name of the calling document when a window is opened using the `open` method
`outerHeight`	Specifies the vertical dimension, in pixels, of the window's outside boundary.
`outerWidth`	Specifies the horizontal dimension, in pixels, of the window's outside boundary.
`pageXOffset`	Provides the current x-position, in pixels, of a window's viewed page.

Property	Description
`pageYOffset`	Provides the current y-position, in pixels, of a window's viewed page.
`parent`	A synonym for a window or frame whose frameset contains the current frame.
`personalbar`	Represents the browser window's personal bar (also called the directories bar).
`scrollbars`	Represents the browser window's scroll bars.
`self`	A synonym for the current window.
`status`	Specifies a priority or transient message in the window's status bar.
`statusbar`	Represents the browser window's status bar.
`toolbar`	Represents the browser window's tool bar.
`top`	A synonym for the topmost browser window.
`window`	A synonym for the current window.

Method Summary

Method	Description
`alert`	Displays an Alert dialog box with a message and an OK button.
`back`	Undoes the last history step in any frame within the top-level window.
`blur`	Removes focus from the specified object.
`captureEvents`	Sets the window or document to capture all events of the specified type.
`clearInterval`	Cancels a timeout that was set with the `setInterval` method.
`clearTimeout`	Cancels a timeout that was set with the `setTimeout` method.
`close`	Closes the specified window.
`confirm`	Displays a Confirm dialog box with the specified message and OK and Cancel buttons.

Method	Description
`disableExternalCapture`	Disables external event capturing set by the `enableExternalCapture` method.
`enableExternalCapture`	Allows a window with frames to capture events in pages loaded from different locations (servers).
`find`	Finds the specified text string in the contents of the specified window.
`focus`	Gives focus to the specified object.
`forward`	Loads the next URL in the history list.
`handleEvent`	Invokes the handler for the specified event.
`home`	Points the browser to the URL specified in preferences as the user's home page.
`moveBy`	Moves the window by the specified amounts.
`moveTo`	Moves the top-left corner of the window to the specified screen coordinates.
`open`	Opens a new web browser window.
`print`	Prints the contents of the window or frame.
`prompt`	Displays a Prompt dialog box with a message and an input field.
`releaseEvents`	Sets the window to release captured events of the specified type, sending the event to objects further along the event hierarchy.
`resizeBy`	Resizes an entire window by moving the window's bottom-right corner by the specified amount.
`resizeTo`	Resizes an entire window to the specified outer height and width.
`routeEvent`	Passes a captured event along the normal event hierarchy.
`scroll`	Scrolls a window to a specified coordinate.
`scrollBy`	Scrolls the viewing area of a window by the specified amount.
`scrollTo`	Scrolls the viewing area of the window to the specified coordinates, such that the specified point becomes the top-left corner.

Method	Description
`setInterval`	Evaluates an expression or calls a function every time a specified number of milliseconds elapses.
`setTimeout`	Evaluates an expression or calls a function once after a specified number of milliseconds has elapsed.
`stop`	Stops the current download.

Examples **Example 1. Windows opening other windows.** In the following example, the document in the top window opens a second window, `window2`, and defines push buttons that open a message window, write to the message window, close the message window, and close `window2`. The `onLoad` and `onUnload` event handlers of the document loaded into `window2` display alerts when the window opens and closes.

`win1.html`, which defines the frames for the first window, contains the following code:

```
<HTML>
<HEAD>
<TITLE>Window object example: Window 1</TITLE>
</HEAD>
<BODY BGCOLOR="antiquewhite">
<SCRIPT>
window2=open("win2.html","secondWindow",
   "scrollbars=yes,width=250, height=400")
document.writeln("<B>The first window has no name: "
   + window.name + "</B>")
document.writeln("<BR><B>The second window is named: "
   + window2.name + "</B>")
</SCRIPT>
<FORM NAME="form1">
<P><INPUT TYPE="button" VALUE="Open a message window"
   onClick = "window3=window.open('','messageWindow',
   'scrollbars=yes,width=175, height=300')">
<P><INPUT TYPE="button" VALUE="Write to the message window"
   onClick="window3.document.writeln('Hey there');
   window3.document.close()">
<P><INPUT TYPE="button" VALUE="Close the message window"
   onClick="window3.close()">
<P><INPUT TYPE="button" VALUE="Close window2"
   onClick="window2.close()">
</FORM>
</BODY>
</HTML>
```

`win2.html`, which defines the content for `window2`, contains the following code:

```
<HTML>
<HEAD>
<TITLE>Window object example: Window 2</TITLE>
</HEAD>
<BODY BGCOLOR="oldlace"
   onLoad="alert('Message from ' + window.name + ': Hello, World.')"
   onUnload="alert('Message from ' + window.name + ': I\'m closing')">
<B>Some numbers</B>
<UL><LI>one
<LI>two
<LI>three
<LI>four</UL>
</BODY>
</HTML>
```

Example 2. Creating frames. The following example creates two windows, each with four frames. In the first window, the first frame contains push buttons that change the background colors of the frames in both windows. `framset1.html`, which defines the frames for the first window, contains the following code:

```
<HTML>
<HEAD>
<TITLE>Frames and Framesets: Window 1</TITLE>
</HEAD>
<FRAMESET ROWS="50%,50%" COLS="40%,60%"
   onLoad="alert('Hello, World.')">
<FRAME SRC=framcon1.html NAME="frame1">
<FRAME SRC=framcon2.html NAME="frame2">
<FRAME SRC=framcon2.html NAME="frame3">
<FRAME SRC=framcon2.html NAME="frame4">
</FRAMESET>
</HTML>
```

`framset2.html`, which defines the frames for the second window, contains the following code:

```
<HTML>
<HEAD>
<TITLE>Frames and Framesets: Window 2</TITLE>
</HEAD>
<FRAMESET ROWS="50%,50%" COLS="40%,60%">
<FRAME SRC=framcon2.html NAME="frame1">
<FRAME SRC=framcon2.html NAME="frame2">
<FRAME SRC=framcon2.html NAME="frame3">
<FRAME SRC=framcon2.html NAME="frame4">
```

```
</FRAMESET>
</HTML>
```

`framcon1.html`, which defines the content for the first frame in the first window, contains the following code:

```
<HTML>
<BODY>
<A NAME="frame1"><H1>Frame1</H1></A>
<P><A HREF="framcon3.htm" target=frame2>Click here</A>
   to load a different file into frame 2.
<SCRIPT>
window2=open("framset2.htm","secondFrameset")
</SCRIPT>
<FORM>
<P><INPUT TYPE="button" VALUE="Change frame2 to teal"
   onClick="parent.frame2.document.bgColor='teal'">
<P><INPUT TYPE="button" VALUE="Change frame3 to slateblue"
   onClick="parent.frames[2].document.bgColor='slateblue'">
<P><INPUT TYPE="button" VALUE="Change frame4 to darkturquoise"
   onClick="top.frames[3].document.bgColor='darkturquoise'">

<P><INPUT TYPE="button" VALUE="window2.frame2 to violet"
   onClick="window2.frame2.document.bgColor='violet'">
<P><INPUT TYPE="button" VALUE="window2.frame3 to fuchsia"
   onClick="window2.frames[2].document.bgColor='fuchsia'">
<P><INPUT TYPE="button" VALUE="window2.frame4 to deeppink"
   onClick="window2.frames[3].document.bgColor='deeppink'">
</FORM>
</BODY>
</HTML>
```

`framcon2.html`, which defines the content for the remaining frames, contains the following code:

```
<HTML>
<BODY>
<P>This is a frame.
</BODY>
</HTML>
```

`framcon3.html`, which is referenced in a `Link` object in `framcon1.html`, contains the following code:

```
<HTML>
<BODY>
<P>This is a frame. What do you think?
</BODY>
</HTML>
```

See also `document`, `Frame`

Properties

closed

Specifies whether a window is closed.

Property of	`Window`
Read-only	
Implemented in	Navigator 3.0

Description The `closed` property is a boolean value that specifies whether a window has been closed. When a window closes, the `window` object that represents it continues to exist, and its `closed` property is set to true.

Use `closed` to determine whether a window that you opened, and to which you still hold a reference (from the return value of `window.open`), is still open. Once a window is closed, you should not attempt to manipulate it.

Examples **Example 1.** The following code opens a window, `win1`, then later checks to see if that window has been closed. A function is called depending on whether `win1` is closed.

```
win1=window.open('opener1.html','window1','width=300,height=300')
...
if (win1.closed)
   function1()
   else
   function2()
```

Example 2. The following code determines if the current window's opener window is still closed, and calls the appropriate function.

```
if (window.opener.closed)
   function1()
   else
   function2()
```

See also `Window.close`, `Window.open`

defaultStatus

The default message displayed in the status bar at the bottom of the window.

Property of	`Window`
Implemented in	Navigator 2.0

Security Navigator 3.0: This property is tainted by default. For information on data tainting, see “Security” on page 55.

Description The `defaultStatus` message appears when nothing else is in the status bar. Do not confuse the `defaultStatus` property with the `status` property. The `status` property reflects a priority or transient message in the status bar, such as the message that appears when a `mouseOver` event occurs over an anchor.

You can set the `defaultStatus` property at any time. You must return true if you want to set the `defaultStatus` property in the `onMouseOut` or `onMouseOver` event handlers.

Examples In the following example, the `statusSetter` function sets both the `status` and `defaultStatus` properties in an `onMouseOver` event handler:

```
function statusSetter() {
   window.defaultStatus = "Click the link for the Netscape home page"
   window.status = "Netscape home page"
}

<A HREF="http://home.netscape.com"
   onMouseOver = "statusSetter(); return true">Netscape</A>
```

In the previous example, notice that the `onMouseOver` event handler returns a value of true. You must return true to set `status` or `defaultStatus` in an event handler.

See also `Window.status`

document

Contains information on the current document, and provides methods for displaying HTML output to the user.

Property of	`Window`
Implemented in	Navigator 2.0

Description The value of this property is the window's associated `document` object.

frames

An array of objects corresponding to child frames (created with the `FRAME` tag) in source order.

Property of	`Window`
Read-only	
Implemented in	Navigator 2.0

You can refer to the child frames of a window by using the `frames` array. This array contains an entry for each child frame (created with the `FRAME` tag) in a window containing a `FRAMESET` tag; the entries are in source order. For example, if a window contains three child frames whose `NAME` attributes are `fr1`, `fr2`, and `fr3`, you can refer to the objects in the `images` array either as:

```
parent.frames["fr1"]
parent.frames["fr2"]
parent.frames["fr3"]
```

or as:

```
parent.frames[0]
parent.frames[1]
parent.frames[2]
```

You can find out how many child frames the window has by using the `length` property of the `Window` itself or of the `frames` array.

The value of each element in the `frames` array is `<object nameAttribute>`, where `nameAttribute` is the `NAME` attribute of the frame.

history

Contains information on the URLs that the client has visited within a window.

Property of	`Window`
Implemented in	Navigator 3.0

Description The value of this property is the window's associated `History` object.

innerHeight

Specifies the vertical dimension, in pixels, of the window's content area.

Property of	`Window`
Implemented in	Navigator 4.0

Description To create a window smaller than 100 x 100 pixels, set this property in a signed script.

Security To set the inner height of a window to a size smaller than 100 x 100 or larger than the screen can accommodate, you need the `UniversalBrowserWrite` privilege. For information on security in Navigator 4.0, see Chapter 7, "JavaScript Security," in the *JavaScript Guide*.

See also `Window.innerWidth`, `Window.outerHeight`, `Window.outerWidth`

innerWidth

Specifies the horizontal dimension, in pixels, of the window's content area.

Property of	`Window`
Implemented in	Navigator 4.0

Description To create a window smaller than 100 x 100 pixels, set this property in a signed script.

Security To set the inner width of a window to a size smaller than 100 x 100 or larger than the screen can accommodate, you need the `UniversalBrowserWrite` privilege. For information on security in Navigator 4.0, see Chapter 7, "JavaScript Security," in the *JavaScript Guide*.

See also `Window.innerHeight`, `Window.outerHeight`, `Window.outerWidth`

length

The number of child frames in the window.

Property of	`Window`
Read-only	
Implemented in	Navigator 2.0

Description This property gives you the same result as using the `length` property of the `frames` array.

location

Contains information on the current URL.

Property of	`Window`
Implemented in	Navigator 2.0

Description The value of this property is the window's associated `Location` object.

locationbar

Represents the browser window's location bar (the region containing the bookmark and URL areas).

Property of	`Window`
Implemented in	Navigator 4.0

Description The value of the `locationbar` property itself has one property, `visible`. If true, the location bar is visible; if false, it is hidden.

Security Setting the value of the location bar's `visible` property requires the `UniversalBrowserWrite` privilege. For information on security in Navigator 4.0, see Chapter 7, "JavaScript Security," in the *JavaScript Guide*.

Examples The following example would make the referenced window "chromeless" (chromeless windows lack toolbars, scrollbars, status areas, and so on, much like a dialog box) by hiding most of the user interface toolbars:

```
self.menubar.visible=false;
self.toolbar.visible=false;
self.locationbar.visible=false;
self.personalbar.visible=false;
self.scrollbars.visible=false;
self.statusbar.visible=false;
```

menubar

Represents the browser window's menu bar. This region contains browser's drop-down menus such as File, Edit, View, Go, Communicator, and so on.

Property of	`Window`
Implemented in	Navigator 4.0

Description The value of the `menubar` property itself one property, `visible`. If true, the menu bar is visible; if false, it is hidden.

Security Setting the value of the menu bar's `visible` property requires the `UniversalBrowserWrite` privilege. For information on security in Navigator 4.0, see Chapter 7, "JavaScript Security," in the *JavaScript Guide*.

Examples The following example would make the referenced window "chromeless" (chromeless windows lack toolbars, scrollbars, status areas, and so on, much like a dialog box) by hiding most of the user interface toolbars:

```
self.menubar.visible=false;
self.toolbar.visible=false;
self.locationbar.visible=false;
self.personalbar.visible=false;
self.scrollbars.visible=false;
self.statusbar.visible=false;
```

name

A string specifying the window's name.

Property of	`Window`
Read-only (2.0); Modifiable (later versions)	
Implemented in	Navigator 2.0

Security Navigator 3.0: This property is tainted by default. For information on data tainting, see "Security" on page 55.

Description In Navigator 2.0, `NAME` was a read-only property. In later versions, this property is modifiable by your code. This allows you to assign a name to a top-level window.

Examples In the following example, the first statement creates a window called `netscapeWin`. The second statement displays the value `"netscapeHomePage"` in the Alert dialog box, because `"netscapeHomePage"` is the value of the `windowName` argument of `netscapeWin`.

```
netscapeWin=window.open("http://home.netscape.com","netscapeHomePage")
alert(netscapeWin.name)
```

opener

Specifies the window of the calling document when a window is opened using the `open` method.

Property of	`Window`
Implemented in	Navigator 3.0

Description When a source document opens a destination window by calling the `open` method, the `opener` property specifies the window of the source document. Evaluate the `opener` property from the destination window.

This property persists across document unload in the opened window.

You can change the `opener` property at any time.

You may use `Window.open` to open a new window and then use `Window.open` on that window to open another window, and so on. In this way, you can end up with a chain of opened windows, each of which has an `opener` property pointing to the window that opened it.

Communicator allows a maximum of 100 windows to be around at once. If you open `window2` from `window1` and then are done with `window1`, be sure to set the `opener` property of `window2` to `null`. This allows JavaScript to garbage collect `window1`. If you do not set the `opener` property to `null`, the `window1` object remains, even though it's no longer really needed.

Examples **Example 1: Close the opener.** The following code closes the window that opened the current window. When the opener window closes, `opener` is unchanged. However, `window.opener.name` then evaluates to undefined.

```
window.opener.close()
```

Example 2: Close the main browser window.

```
top.opener.close()
```

Example 3: Evaluate the name of the opener. A window can determine the name of its opener as follows:

```
document.write("<BR>opener property is " + window.opener.name)
```

Example 4: Change the value of opener. The following code changes the value of the `opener` property to null. After this code executes, you cannot close the opener window as shown in Example 1.

```
window.opener=null
```

Example 5: Change a property of the opener. The following code changes the background color of the window specified by the `opener` property.

```
window.opener.document.bgColor='bisque'
```

See also `Window.close`, `Window.open`

outerHeight

Specifies the vertical dimension, in pixels, of the window's outside boundary.

Property of	`Window`
Implemented in	Navigator 4.0

Description The outer boundary includes the scroll bars, the status bar, the tool bars, and other “chrome” (window border user interface elements). To create a window smaller than 100 x 100 pixels, set this property in a signed script.

See also `Window.innerWidth`, `Window.innerHeight`, `Window.outerWidth`

outerWidth

Specifies the horizontal dimension, in pixels, of the window's outside boundary.

Property of	`Window`
Implemented in	Navigator 4.0

Description The outer boundary includes the scroll bars, the status bar, the tool bars, and other “chrome” (window border user interface elements). To create a window smaller than 100 x 100 pixels, set this property in a signed script.

See also `Window.innerWidth`, `Window.innerHeight`, `Window.outerHeight`

pageXOffset

Provides the current x-position, in pixels, of a window's viewed page.

Property of	`Window`
Read-only	
Implemented in	Navigator 4.0

Description The `pageXOffset` property provides the current x-position of a page as it relates to the upper-left corner of the window's content area. This property is useful when you need to find the current location of the scrolled page before using `scrollTo` or `scrollBy`.

Example The following example returns the x-position of the viewed page.

```
x = myWindow.pageXOffset
```

See Also `Window.pageYOffset`

pageYOffset

Provides the current y-position, in pixels, of a window's viewed page.

Property of	`Window`
Read-only	
Implemented in	Navigator 4.0

Description The `pageYOffset` property provides the current y-position of a page as it relates to the upper-left corner of the window's content area. This property is useful when you need to find the current location of the scrolled page before using `scrollTo` or `scrollBy`.

Example The following example returns the y-position of the viewed page.

```
x = myWindow.pageYOffset
```

See also `Window.pageXOffset`

parent

The `parent` property is the window or frame whose frameset contains the current frame.

Property of	`Window`
Read-only	
Implemented in	Navigator 2.0

Description This property is only meaningful for frames; that is, windows that are not top-level windows.

The `parent` property refers to the `FRAMESET` window of a frame. Child frames within a frameset refer to sibling frames by using `parent` in place of the window name in one of the following ways:

```
parent.frameName
parent.frames[index]
```

For example, if the fourth frame in a set has `NAME="homeFrame"`, sibling frames can refer to that frame using `parent.homeFrame` or `parent.frames[3]`.

You can use `parent.parent` to refer to the “grandparent” frame or window when a `FRAMESET` tag is nested within a child frame.

The value of the `parent` property is

```
<object nameAttribute>
```

where `nameAttribute` is the `NAME` attribute if the parent is a frame, or an internal reference if the parent is a window.

Examples See examples for `Frame`.

personalbar

Represents the browser window’s personal bar (also called the directories bar). This is the region the user can use for easy access to certain bookmarks.

Property of	`Window`
Implemented in	Navigator 4.0

Description The value of the `personalbar` property itself one property, `visible`. If true, the personal bar is visible; if false, it is hidden.

Security Setting the value of the personal bar's `visible` property requires the `UniversalBrowserWrite` privilege. For information on security in Navigator 4.0, see Chapter 7, "JavaScript Security," in the *JavaScript Guide*.

Examples The following example would make the referenced window "chromeless" (chromeless windows lack toolbars, scrollbars, status areas, and so on, much like a dialog box) by hiding most of the user interface toolbars:

```
self.menubar.visible=false;
self.toolbar.visible=false;
self.locationbar.visible=false;
self.personalbar.visible=false;
self.scrollbars.visible=false;
self.statusbar.visible=false;
```

scrollbars

Represents the browser window's vertical and horizontal scroll bars for the document area.

Property of	`Window`
Implemented in	Navigator 4.0

Description The value of the `scrollbars` property itself has one property, `visible`. If true, both scrollbars are visible; if false, they are hidden.

Security Setting the value of the scrollbars' `visible` property requires the `UniversalBrowserWrite` privilege. For information on security in Navigator 4.0, see Chapter 7, "JavaScript Security," in the *JavaScript Guide*.

Examples The following example would make the referenced window "chromeless" (chromeless windows lack toolbars, scrollbars, status areas, and so on, much like a dialog box) by hiding most of the user interface toolbars:

```
self.menubar.visible=false;
self.toolbar.visible=false;
self.locationbar.visible=false;
self.personalbar.visible=false;
self.scrollbars.visible=false;
self.statusbar.visible=false;
```

self

The `self` property is a synonym for the current window.

Property of	`Window`
Read-only	
Implemented in	Navigator 2.0

Description The `self` property refers to the current window. That is, the value of this property is a synonym for the object itself.

Use the `self` property to disambiguate a `window` property from a form or form element of the same name. You can also use the `self` property to make your code more readable.

The value of the `self` property is

```
<object nameAttribute>
```

where `nameAttribute` is the `NAME` attribute if `self` refers to a frame, or an internal reference if `self` refers to a window.

Examples In the following example, `self.status` is used to set the `status` property of the current window. This usage disambiguates the `status` property of the current window from a form or form element called `status` within the current window.

```
<A HREF=""
   onClick="this.href=pickRandomURL()"
   onMouseOver="self.status='Pick a random URL' ; return true">
Go!</A>
```

status

Specifies a priority or transient message in the status bar at the bottom of the window, such as the message that appears when a `mouseOver` event occurs over an anchor.

Property of	`Window`
Implemented in	Navigator 2.0

Security Navigator 3.0: This property is tainted by default. For information on data tainting, see “Security” on page 55.

Description Do not confuse the `status` property with the `defaultStatus` property. The `defaultStatus` property reflects the default message displayed in the status bar.

You can set the `status` property at any time. You must return true if you want to set the `status` property in the `onMouseOver` event handler.

Examples Suppose you have created a JavaScript function called `pickRandomURL` that lets you select a URL at random. You can use the `onClick` event handler of an anchor to specify a value for the `HREF` attribute of the anchor dynamically, and the `onMouseOver` event handler to specify a custom message for the window in the `status` property:

```
<A HREF=""
   onClick="this.href=pickRandomURL()"
   onMouseOver="self.status='Pick a random URL'; return true">
Go!</A>
```

In the preceding example, the `status` property of the window is assigned to the window's `self` property, as `self.status`.

See also `Window.defaultStatus`

statusbar

Represents the browser window's status bar. This is the region containing the security indicator, browser status, and so on.

Property of	`Window`
Implemented in	Navigator 4.0

Description The value of the `statusbar` property itself one property, `visible`. If true, the status bar is visible; if false, it is hidden.

Security Setting the value of the status bar's `visible` property requires the `UniversalBrowserWrite` privilege. For information on security in Navigator 4.0, see Chapter 7, "JavaScript Security," in the *JavaScript Guide*.

Examples The following example would make the referenced window "chromeless" (chromeless windows lack toolbars, scrollbars, status areas, and so on, much like a dialog box) by hiding most of the user interface toolbars:

```
self.menubar.visible=false;
self.toolbar.visible=false;
self.locationbar.visible=false;
self.personalbar.visible=false;
self.scrollbars.visible=false;
self.statusbar.visible=false;
```

toolbar

Represents the browser window's tool bar, containing the navigation buttons, such as Back, Forward, Reload, Home, and so on.

Property of	`Window`
Implemented in	Navigator 4.0

Description The value of the `toolbar` property itself one property, `visible`. If true, the tool bar is visible; if false, it is hidden.

Security Setting the value of the tool bar's `visible` property requires the `UniversalBrowserWrite` privilege. For information on security in Navigator 4.0, see Chapter 7, "JavaScript Security," in the *JavaScript Guide*.

Examples The following example would make the referenced window "chromeless" (chromeless windows lack toolbars, scrollbars, status areas, and so on, much like a dialog box) by hiding most of the user interface toolbars:

```
self.menubar.visible=false;
self.toolbar.visible=false;
self.locationbar.visible=false;
self.personalbar.visible=false;
self.scrollbars.visible=false;
self.statusbar.visible=false;
```

top

The `top` property is a synonym for the topmost browser window, which is a document window or web browser window.

Property of	`Window`
Read-only	
Implemented in	Navigator 2.0

Description The `top` property refers to the topmost window that contains frames or nested framesets. Use the `top` property to refer to this ancestor window.

The value of the `top` property is

```
<object objectReference>
```

where `objectReference` is an internal reference.

Examples The statement `top.close()` closes the topmost ancestor window.

The statement `top.length` specifies the number of frames contained within the topmost ancestor window. When the topmost ancestor is defined as follows, `top.length` returns three:

```
<FRAMESET COLS="30%,40%,30%">
<FRAME SRC=child1.htm NAME="childFrame1">
<FRAME SRC=child2.htm NAME="childFrame2">
<FRAME SRC=child3.htm NAME="childFrame3">
</FRAMESET>
```

The following example sets the background color of a frame called `myFrame` to red. `myFrame` is a child of the topmost ancestor window.

```
top.myFrame.document.bgColor="red"
```

window

The `window` property is a synonym for the current window or frame.

Property of	`Window`
Read-only	
Implemented in	Navigator 2.0

Description The `window` property refers to the current window or frame. That is, the value of this property is a synonym for the object itself.

Although you can use the `window` property as a synonym for the current frame, your code may be more readable if you use the `self` property. For example, `window.name` and `self.name` both specify the name of the current frame, but `self.name` may be easier to understand (because a frame is not displayed as a separate window).

Use the `window` property to disambiguate a property of the `window` object from a form or form element of the same name. You can also use the `window` property to make your code more readable.

The value of the `window` property is

```
<object nameAttribute>
```

where `nameAttribute` is the `NAME` attribute if `window` refers to a frame, or an internal reference if `window` refers to a window.

Examples In the following example, `window.status` is used to set the `status` property of the current window. This usage disambiguates the `status` property of the current window from a form called "status" within the current window.

```
<A HREF=""
   onClick="this.href=pickRandomURL()"
   onMouseOver="window.status='Pick a random URL' ; return true">
Go!</A>
```

See also `Window.self`

Methods

alert

Displays an Alert dialog box with a message and an OK button.

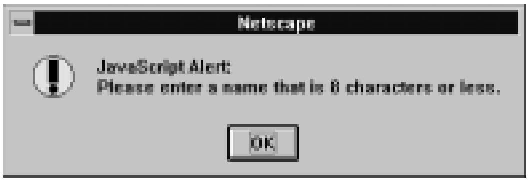
Method of	`Window`
Implemented in	Navigator 2.0

Syntax `alert("message")`

Parameters

`message`	A string.

Description An alert dialog box looks as follows:



Netscape

JavaScript Alert:
Please enter a name that is 8 characters or less.

Use the `alert` method to display a message that does not require a user decision. The `message` argument specifies a message that the dialog box contains.

You cannot specify a title for an alert dialog box, but you can use the `open` method to create your own alert dialog box. See `open`.

Examples In the following example, the `testValue` function checks the name entered by a user in the `Text` object of a form to make sure that it is no more than eight characters in length. This example uses the `alert` method to prompt the user to enter a valid value.

```
function testValue(textElement) {
   if (textElement.length > 8) {
      alert("Please enter a name that is 8 characters or less")
   }
}
```

You can call the `testValue` function in the `onBlur` event handler of a form's `Text` object, as shown in the following example:

```
Name: <INPUT TYPE="text" NAME="userName"
   onBlur="testValue(userName.value)">
```

See also `Window.confirm`, `Window.prompt`

back

Undoes the last history step in any frame within the top-level window; equivalent to the user pressing the browser's Back button.

Method of	`Window`
Implemented in	Navigator 4.0

Syntax `back()`

Parameters None

Description Calling the `back` method is equivalent to the user pressing the browser's Back button. That is, `back` undoes the last step anywhere within the top-level window, whether it occurred in the same frame or in another frame in the tree of frames loaded from the top-level window. In contrast, the `history` object's `back` method backs up the *current* window or frame history one step.

For example, consider the following scenario. While in Frame A, you click the Forward button to change Frame A's content. You then move to Frame B and click the Forward button to change Frame B's content. If you move back to Frame A and call `FrameA.back()`, the content of Frame B changes (clicking the Back button behaves the same).

If you want to navigate Frame A separately, use `FrameA.history.back()`.

Examples The following custom buttons perform the same operation as the browser's Back button:

```
<P><INPUT TYPE="button" VALUE="< Go Back"
   onClick="history.back()">
<P><INPUT TYPE="button" VALUE="> Go Back"
   onClick="myWindow.back()">
```

See also `Window.forward`, `History.back`

blur

Removes focus from the specified object.

Method of	`Window`
Implemented in	Navigator 2.0

Syntax `blur()`

Parameters None

Description Use the `blur` method to remove focus from a specific window or frame. Removing focus from a window sends the window to the background in most windowing systems.

See also `Window.focus`

captureEvents

Sets the window to capture all events of the specified type.

Method of	`Window`
Implemented in	Navigator 4.0

Syntax `captureEvents(eventType)`

Parameters

`eventType`	The type of event to be captured. The available event types are listed with the `event` object.

Security When a window with frames wants to capture events in pages loaded from different locations (servers), you need to use `captureEvents` in a signed script and precede it with `enableExternalCapture`. You must have the `UniversalBrowserWrite` privilege. For more information and an example, see `enableExternalCapture`. For information on security in Navigator 4.0, see Chapter 7, "JavaScript Security," in the *JavaScript Guide*.

See also `captureEvents` works in tandem with `releaseEvents`, `routeEvent`, and `handleEvent`. For more information, see "Events in Navigator 4.0" on page 482.

clearInterval

Cancels a timeout that was set with the `setInterval` method.

Method of	`Window`
Implemented in	Navigator 4.0

Syntax `clearInterval(intervalID)`

Parameters

`intervalID`	Timeout setting that was returned by a previous call to the `setInterval` method.

Description See `setInterval`.

Examples See `setInterval`.

See also `Window.setInterval`

clearTimeout

Cancels a timeout that was set with the `setTimeout` method.

Method of	`Window`
Implemented in	Navigator 2.0

Syntax `clearTimeout(timeoutID)`

Parameters

`timeoutID`	A timeout setting that was returned by a previous call to the `setTimeout` method.

Description See `setTimeout`.

Examples See `setTimeout`.

See also `Window.clearInterval`, `Window.setTimeout`

close

Closes the specified window.

Method of	`Window`
Implemented in	Navigator 2.0: closes any window. Navigator 3.0: closes only windows opened by JavaScript. Navigator 4.0: must use signed scripts to unconditionally close a window.

Syntax `close()`

Parameters None

Security Navigator 4.0: To unconditionally close a window, you need the `UniversalBrowserWrite` privilege. For information on security in Navigator 4.0, see Chapter 7, "JavaScript Security," in the *JavaScript Guide*.

Description The `close` method closes the specified window. If you call `close` without specifying a `windowReference`, JavaScript closes the current window.

The `close` method closes only windows opened by JavaScript using the `open` method. If you attempt to close any other window, a confirm is generated, which lets the user choose whether the window closes. This is a security feature to prevent "mail bombs" containing `self.close()`. However, if the window has only one document (the current one) in its session history, the close is allowed without any confirm. This is a special case for one-off windows that need to open other windows and then dispose of themselves.

In event handlers, you must specify `window.close()` instead of simply using `close()`. Due to the scoping of static objects in JavaScript, a call to `close()` without specifying an object name is equivalent to `document.close()`.

Examples **Example 1.** Any of the following examples closes the current window:

```
window.close()
self.close()
close()
```

Example 2: Close the main browser window. The following code closes the main browser window.

```
top.opener.close()
```

Example 3. The following example closes the `messageWin` window:

```
messageWin.close()
```

This example assumes that the window was opened in a manner similar to the following:

```
messageWin=window.open("")
```

See also `Window.closed`, `Window.open`

confirm

Displays a Confirm dialog box with the specified message and OK and Cancel buttons.

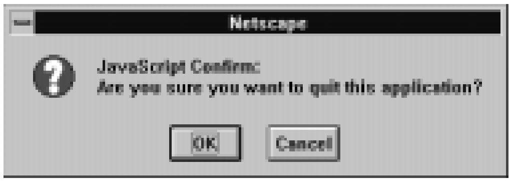
Method of	`Window`
Implemented in	Navigator 2.0

Syntax `confirm("message")`

Parameters

`message`	A string.

Description A confirm dialog box looks as follows:


Netscape

JavaScript Confirm:
Are you sure you want to quit this application?

Use the `confirm` method to ask the user to make a decision that requires either an OK or a Cancel. The `message` argument specifies a message that prompts the user for the decision. The `confirm` method returns true if the user chooses OK and false if the user chooses Cancel.

You cannot specify a title for a confirm dialog box, but you can use the `open` method to create your own confirm dialog. See `open`.

Examples This example uses the `confirm` method in the `confirmCleanUp` function to confirm that the user of an application really wants to quit. If the user chooses OK, the custom `cleanUp` function closes the application.

```
function confirmCleanUp() {
   if (confirm("Are you sure you want to quit this application?")) {
      cleanUp()
   }
}
```

You can call the `confirmCleanUp` function in the `onClick` event handler of a form's push button, as shown in the following example:

```
<INPUT TYPE="button" VALUE="Quit" onClick="confirmCleanUp()">
```

See also `Window.alert`, `Window.prompt`

disableExternalCapture

Disables external event capturing set by the `enableExternalCapture` method.

Method of	`Window`
Implemented in	Navigator 4.0

Syntax `disableExternalCapture()`

Parameters None

Description See `enableExternalCapture`.

enableExternalCapture

Allows a window with frames to capture events in pages loaded from different locations (servers).

Method of	`Window`
Implemented in	Navigator 4.0

Syntax `enableExternalCapture()`

Parameters None

Description Use this method in a signed script requesting `UniversalBrowserWrite` privileges, and use it before calling the `captureEvents` method.

If Communicator sees additional scripts that cause the set of principals in effect for the container to be downgraded, it disables external capture of events. Additional calls to `enableExternalCapture` (after acquiring the `UniversalBrowserWrite` privilege under the reduced set of principals) can be made to enable external capture again.

Example In the following example, the window is able to capture all `Click` events that occur across its frames.

```
<SCRIPT ARCHIVE="myArchive.jar" ID="2">
function captureClicks() {
   netscape.security.PrivilegeManager.enablePrivilege(
      "UniversalBrowserWrite");
   enableExternalCapture();
   captureEvents(Event.CLICK);
   ...
}
</SCRIPT>
```

See also `Window.disableExternalCapture`, `Window.captureEvents`

find

Finds the specified text string in the contents of the specified window.

Method of	`Window`
Implemented in	Navigator 4.0

Syntax `find(string, casesensitive, backward)`

Parameters

`string`	(Optional) The text string for which to search.
`casesensitive`	(Optional) Boolean value. If true, specifies a case-sensitive search. If you supply this parameter, you must also supply `backward`.
`backward`	(Optional) Boolean. If true, specifies a backward search. If you supply this parameter, you must also supply `casesensitive`.

Returns true if the string is found; otherwise, false.

Description When a string is specified, the browser performs a case-insensitive, forward search. If a string is not specified, the method displays the Find dialog box, allowing the user to enter a search string.

focus

Gives focus to the specified object.

Method of	`Window`
Implemented in	Navigator 3.0

Syntax `focus()`

Parameters None

Description Use the `focus` method to navigate to a specific window or frame, and give it focus. Giving focus to a window brings the window forward in most windowing systems.

In Navigator 3.0, on some platforms, the `focus` method gives focus to a frame but the focus is not visually apparent (for example, the frame's border is not darkened).

Examples In the following example, the `checkPassword` function confirms that a user has entered a valid password. If the password is not valid, the `focus` method returns focus to the `Password` object and the `select` method highlights it so the user can reenter the password.

```
function checkPassword(userPass) {
   if (badPassword) {
      alert("Please enter your password again.")
      userPass.focus()
      userPass.select()
   }
}
```

This example assumes that the `Password` object is defined as

```
<INPUT TYPE="password" NAME="userPass">
```

See also `Window.blur`

forward

Points the browser to the next URL in the current history list; equivalent to the user pressing the browser's Forward button

Method of	`Window`
Implemented in	Navigator 4.0

Syntax

```
history.forward()
```

```
forward()
```

Parameters None

Description This method performs the same action as a user choosing the Forward button in the browser. The `forward` method is the same as `history.go(1)`.

When used with the Frame object, `forward` behaves as follows: While in Frame A, you click the Back button to change Frame A's content. You then move to Frame B and click the Back button to change Frame B's content. If you move back to Frame A and call `FrameA.forward()`, the content of Frame B changes (clicking the Forward button behaves the same). If you want to navigate Frame A separately, use `FrameA.history.forward()`.

Examples The following custom buttons perform the same operation as the browser's Forward button:

```
<P><INPUT TYPE="button" VALUE="< Go Forth"
   onClick="history.forward()">
<P><INPUT TYPE="button" VALUE="> Go Forth"
   onClick="myWindow.forward()">
```

See also `Window.back`

handleEvent

Invokes the handler for the specified event.

Method of	`Window`
Implemented in	Navigator 4.0

Syntax `handleEvent(event)`

Parameters

`event`	The name of an event for which the specified object has an event handler.

Description `handleEvent` works in tandem with `captureEvents`, `releaseEvents`, and `routeEvent`. For more information, see "Events in Navigator 4.0" on page 482.

home

Points the browser to the URL specified in preferences as the user's home page; equivalent to the user pressing the browser's Home button.

Method of	`Window`
Implemented in	Navigator 4.0

Syntax `home()`

Parameters None

Description This method performs the same action as a user choosing the Home button in the browser.

moveBy

Moves the window relative to its current position, moving the specified number of pixels.

Method of	`Window`
Implemented in	Navigator 4.0

Syntax `moveBy(horizontal, vertical)`

Parameters

`horizontal`	The number of pixels by which to move the window horizontally.
`vertical`	The number of pixels by which to move the window vertically.

Description This method moves the window by adding or subtracting the specified number of pixels to the current location.

Security Exceeding any of the boundaries of the screen (to hide some or all of a window) requires signed JavaScript, so a window won't move past the screen boundaries. You need the `UniversalBrowserWrite` privilege for this. For information on security in Navigator 4.0, see Chapter 7, "JavaScript Security," in the *JavaScript Guide*.

Examples: To move the current window 5 pixels up towards the top of the screen (x-axis), and 10 pixels towards the right (y-axis) of the current window position, use this statement:

```
self.moveBy(-5,10); // relative positioning
```

See also `Window.moveTo`

moveTo

Moves the top-left corner of the window to the specified screen coordinates.

Method of	`Window`
Implemented in	Navigator 4.0

Syntax `moveTo(x-coordinate, y-coordinate)`

Parameters

`x-coordinate`	The left edge of the window in screen coordinates.
`y-coordinate`	The top edge of the window in screen coordinates.

Description This method moves the window to the absolute pixel location indicated by its parameters. The origin of the axes is at absolute position (0,0); this is the upper left-hand corner of the display.

Security Exceeding any of the boundaries of the screen (to hide some or all of a window) requires signed JavaScript, so a window won't move past the screen boundaries. You need the `UniversalBrowserWrite` privilege for this. For information on security in Navigator 4.0, see Chapter 7, "JavaScript Security," in the *JavaScript Guide*.

Examples: To move the current window to 25 pixels from the top boundary of the screen (x-axis), and 10 pixels from the left boundary of the screen (y-axis), use this statement:

```
self.moveTo(25,10); // absolute positioning
```

See also `Window.moveBy`

open

Opens a new web browser window.

Method of	`Window`
Implemented in	Navigator 2.0 Navigator 4.0: added several new `windowFeatures`

Syntax `open(URL, windowName, windowFeatures)`

Parameters

`URL`	A string specifying the URL to open in the new window. See the `Location` object for a description of the URL components.
`windowName`	A string specifying the window name to use in the `TARGET` attribute of a `FORM` or `A` tag. `windowName` can contain only alphanumeric or underscore (_) characters.
`windowFeatures`	(Optional) A string containing a comma-separated list determining whether or not to create various standard window features. These options are described below.

Description In event handlers, you must specify `window.open()` instead of simply using `open()`. Due to the scoping of static objects in JavaScript, a call to `open()` without specifying an object name is equivalent to `document.open()`.

The `open` method opens a new Web browser window on the client, similar to choosing New Navigator Window from the File menu of the browser. The `URL` argument specifies the URL contained by the new window. If `URL` is an empty string, a new, empty window is created.

You can use `open` on an existing window, and if you pass the empty string for the URL, you will get a reference to the existing window, but not load anything into it. You can, for example, then look for properties in the window.

`windowFeatures` is an optional string containing a comma-separated list of options for the new window (do not include any spaces in this list). After a window is open, you cannot use JavaScript to change the `windowFeatures.` The features you can specify are:

`alwaysLowered`	(Navigator 4.0) If yes, creates a new window that floats below other windows, whether it is active or not. This is a secure feature and must be set in signed scripts.
`alwaysRaised`	(Navigator 4.0) If yes, creates a new window that floats on top of other windows, whether it is active or not. This is a secure feature and must be set in signed scripts.
`dependent`	(Navigator 4.0) If yes, creates a new window as a child of the current window. A dependent window closes when its parent window closes. On Windows platforms, a dependent window does not show on the task bar.
`directories`	If yes, creates the standard browser directory buttons, such as What's New and What's Cool.
`height`	(Navigator 2.0 and 3.0) Specifies the height of the window in pixels.
`hotkeys`	(Navigator 4.0) If no (or 0), disables most hotkeys in a new window that has no menu bar. The security and quit hotkeys remain enabled.
`innerHeight`	(Navigator 4.0) Specifies the height, in pixels, of the window's content area. To create a window smaller than 100 x 100 pixels, set this feature in a signed script. This feature replaces `height`, which remains for backwards compatibility.
`innerWidth`	(Navigator 4.0) Specifies the width, in pixels, of the window's content area. To create a window smaller than 100 x 100 pixels, set this feature in a signed script. This feature replaces `width`, which remains for backwards compatibility.
`location`	If yes, creates a Location entry field.
`menubar`	If yes, creates the menu at the top of the window.
`outerHeight`	(Navigator 4.0) Specifies the vertical dimension, in pixels, of the outside boundary of the window. To create a window smaller than 100 x 100 pixels, set this feature in a signed script.

`resizable`	If yes, allows a user to resize the window.
`screenX`	(Navigator 4.0) Specifies the distance the new window is placed from the left side of the screen. To place a window offscreen, set this feature in a signed scripts.
`screenY`	(Navigator 4.0) Specifies the distance the new window is placed from the top of the screen. To place a window offscreen, set this feature in a signed scripts.
`scrollbars`	If yes, creates horizontal and vertical scrollbars when the Document grows larger than the window dimensions.
`status`	If yes, creates the status bar at the bottom of the window.
`titlebar`	(Navigator 4.0) If yes, creates a window with a title bar. To set the titlebar to no, set this feature in a signed script.
`toolbar`	If yes, creates the standard browser toolbar, with buttons such as Back and Forward.
`width`	(Navigator 2.0 and 3.0) Specifies the width of the window in pixels.
`z-lock`	(Navigator 4.0) If yes, creates a new window that does not rise above other windows when activated. This is a secure feature and must be set in signed scripts.

Many of these features (as noted above) can either be yes or no. For these features, you can use 1 instead of yes and 0 instead of no. If you want to turn a feature on, you can also simply list the feature name in the `windowFeatures` string.

If `windowName` does not specify an existing window and you do not supply the `windowFeatures` parameter, all of the features which have a yes/no choice are yes by default. However, if you do supply the `windowFeatures` parameter, then the `titlebar` and `hotkeys` are still yes by default, but the other features which have a yes/no choice are no by default.

For example, all of the following statements turn on the toolbar option and turn off all other Boolean options:

```
open("", "messageWindow", "toolbar")
open("", "messageWindow", "toolbar=yes")
open("", "messageWindow", "toolbar=1")
```

The following statement turn on the location and directories options and turns off all other Boolean options:

```
open("", "messageWindow", "toolbar,directories=yes")
```

How the `alwaysLowered`, `alwaysRaised`, and `z-lock` features behave depends on the windowing hierarchy of the platform. For example, on Windows, an `alwaysLowered` or `z-locked` browser window is below all windows in all open applications. On Macintosh, an `alwaysLowered` browser window is below all browser windows, but not necessarily below windows in other open applications. Similarly for an `alwaysRaised` window.

You may use `open` to open a new window and then use `open` on that window to open another window, and so on. In this way, you can end up with a chain of opened windows, each of which has an `opener` property pointing to the window that opened it.

Communicator allows a maximum of 100 windows to be around at once. If you open `window2` from `window1` and then are done with `window1`, be sure to set the `opener` property of `window2` to `null`. This allows JavaScript to garbage collect `window1`. If you do not set the `opener` property to `null`, the `window1` object remains, even though it's no longer really needed.

Security To perform the following operations using the specified screen features, you need the `UniversalBrowserWrite` privilege:

- To create a window smaller than 100 x 100 pixels or larger than the screen can accommodate by using `innerWidth`, `innerHeight`, `outerWidth`, and `outerHeight`.
- To place a window off screen by using `screenX` and `screenY`.
- To create a window without a titlebar by using `titlebar`.
- To use `alwaysRaised`, `alwaysLowered`, or `z-lock` for any setting.

For information on security in Navigator 4.0, see Chapter 7, "JavaScript Security," in the *JavaScript Guide*.

Examples **Example 1.** In the following example, the `windowOpener` function opens a window and uses `write` methods to display a message:

```
function windowOpener() {
   msgWindow=window.open("","displayWindow","menubar=yes")
   msgWindow.document.write
      ("<HEAD><TITLE>Message window</TITLE></HEAD>")
   msgWindow.document.write
      ("<CENTER><BIG><B>Hello, world!</B></BIG></CENTER>")
}
```

Example 2. The following is an `onClick` event handler that opens a new client window displaying the content specified in the file `sesame.html`. The window opens with the specified option settings; all other options are false because they are not specified.

```
<FORM NAME="myform">
<INPUT TYPE="button" NAME="Button1" VALUE="Open Sesame!"
   onClick="window.open ('sesame.html', 'newWin',
   'scrollbars=yes,status=yes,width=300,height=300')">
</FORM>
```

See also `Window.close`

print

Prints the contents of the window.

Method of	`Window`
Implemented in	Navigator 4.0

Syntax `print()`

Parameters None

prompt

Displays a Prompt dialog box with a message and an input field.

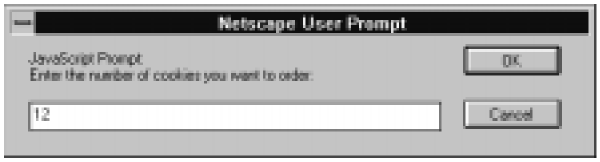
Method of	`Window`
Implemented in	Navigator 2.0

Syntax `prompt(message, inputDefault)`

Parameters

`message`	A string to be displayed as the message.
`inputDefault`	(Optional) A string or integer representing the default value of the input field.

Description A prompt dialog box looks as follows:



JavaScript Prompt:
Enter the number of cookies you want to order:

12

Use the `prompt` method to display a dialog box that receives user input. If you do not specify an initial value for `inputDefault`, the dialog box displays `<undefined>`.

You cannot specify a title for a prompt dialog box, but you can use the `open` method to create your own prompt dialog. See `open`.

Examples

```
prompt("Enter the number of cookies you want to order:", 12)
```

See also `Window.alert`, `Window.confirm`

releaseEvents

Sets the window or document to release captured events of the specified type, sending the event to objects further along the event hierarchy.

Method of	`Window`
Implemented in	Navigator 4.0

Note If the original target of the event is a window, the window receives the event even if it is set to release that type of event.

Syntax `releaseEvents(eventType)`

Parameters

`eventType`	Type of event to be captured.

Description `releaseEvents` works in tandem with `captureEvents`, `routeEvent`, and `handleEvent`. For more information, see “Events in Navigator 4.0” on page 482.

resizeBy

Resizes an entire window by moving the window's bottom-right corner by the specified amount.

Method of	`Window`
Implemented in	Navigator 4.0

Syntax `resizeBy(horizontal, vertical)`

Parameters

`horizontal`	The number of pixels by which to resize the window horizontally.
`vertical`	The number of pixels by which to resize the window vertically.

Description This method changes the window's dimensions by setting its `outerWidth` and `outerHeight` properties. The upper left-hand corner remains anchored and the lower right-hand corner moves. `resizeBy` moves the window by adding or subtracting the specified number of pixels to that corner's current location.

Security Exceeding any of the boundaries of the screen (to hide some or all of a window) requires signed JavaScript, so a window won't move past the screen boundaries. In addition, windows have an enforced minimum size of 100 x 100 pixels; resizing a window to be smaller than this minimum requires signed JavaScript. You need the `UniversalBrowserWrite` privilege for this. For information on security in Navigator 4.0, see Chapter 7, "JavaScript Security," in the *JavaScript Guide*.

Examples To make the current window 5 pixels narrower and 10 pixels taller than its current dimensions, use this statement:

```
self.resizeBy(-5,10); // relative positioning
```

See also `Window.resizeTo`

resizeTo

Resizes an entire window to the specified pixel dimensions.

Method of	`Window`
Implemented in	Navigator 4.0

Syntax `resizeTo(outerWidth, outerHeight)`

Parameters

`outerWidth`	An integer representing the window's width in pixels.
`outerHeight`	An integer representing the window's height in pixels.

Description This method changes the window's dimensions by setting its `outerWidth` and `outerHeight` properties. The upper left-hand corner remains anchored and the lower right-hand corner moves. `resizeBy` moves to the specified position. The origin of the axes is at absolute position (0,0); this is the upper left-hand corner of the display.

Security Exceeding any of the boundaries of the screen (to hide some or all of a window) requires signed JavaScript, so a window won't move past the screen boundaries. In addition, windows have an enforced minimum size of 100 x 100 pixels; resizing a window to be smaller than this minimum requires signed JavaScript. You need the `UniversalBrowserWrite` privilege for this. For information on security in Navigator 4.0, see Chapter 7, "JavaScript Security," in the *JavaScript Guide*.

Examples To make the window 225 pixels wide and 200 pixels tall, use this statement:

```
self.resizeTo(225,200); // absolute positioning
```

See also `Window.resizeBy`

routeEvent

Passes a captured event along the normal event hierarchy.

Method of	`Window`
Implemented in	Navigator 4.0

Syntax `routeEvent(event)`

Parameters

`event`	Name of the event to be routed.

Description If a subobject (document or layer) is also capturing the event, the event is sent to that object. Otherwise, it is sent to its original target.

`routeEvent` works in tandem with `captureEvents`, `releaseEvents`, and `handleEvent`. For more information, see “Events in Navigator 4.0” on page 482.

scroll

Scrolls a window to a specified coordinate.

Method of	`Window`
Implemented in	Navigator 3.0; deprecated in 4.0

Description In Navigator 4.0, `scroll` is no longer used and has been replaced by `scrollTo`. `scrollTo` extends the capabilities of `scroll`. `scroll` remains for backward compatibility.

scrollBy

Scrolls the viewing area of a window by the specified amount.

Method of	`Window`
Implemented in	Navigator 4.0

Syntax

```
scrollBy(horizontal, vertical)
```

Parameters

`horizontal`	The number of pixels by which to scroll the viewing area horizontally.
`vertical`	The number of pixels by which to scroll the viewing area vertically.

Description This method scrolls the content in the window if portions that can’t be seen exist outside of the window. `scrollBy` scrolls the window by adding or subtracting the specified number of pixels to the current scrolled location.

For this method to have an effect the `visible` property of `Window.scrollbars` must be true.

Examples To scroll the current window 5 pixels towards the left and 30 pixels down from the current position, use this statement:

```
self.scrollBy(-5,30); // relative positioning
```

See also `Window.scrollTo`

scrollTo

Scrolls the viewing area of the window so that the specified point becomes the top-left corner.

Method of	`Window`
Implemented in	Navigator 4.0

Syntax `scrollTo(x-coordinate, y-coordinate)`

Parameters

`x-coordinate`	An integer representing the x-coordinate of the viewing area in pixels.
`y-coordinate`	An integer representing the y-coordinate of the viewing area in pixels.

Description `scrollTo` replaces `scroll`. `scroll` remains for backward compatibility.

The `scrollTo` method scrolls the content in the window if portions that can't be seen exist outside of the window. For this method to have an effect the `visible` property of `Window.scrollbars` must be true.

Examples **Example 1: Scroll the current viewing area.** To scroll the current window to the leftmost boundary and 20 pixels down from the top of the window, use this statement:

```
self.scrollTo(0,20); // absolute positioning
```

Example 2: Scroll a different viewing area. The following code, which exists in one frame, scrolls the viewing area of a second frame. Two `Text` objects let the user specify the x and y coordinates. When the user clicks the Go button, the document in `frame2` scrolls to the specified coordinates.

```
<SCRIPT>
function scrollIt(form) {
   var x = parseInt(form.x.value)
   var y = parseInt(form.y.value)
   parent.frame2.scrollTo(x, y)
}
</SCRIPT>
<BODY>

<FORM NAME="myForm">
<P><B>Specify the coordinates to scroll to:</B>
<BR>Horizontal:
```

```
<INPUT TYPE="text" NAME=x VALUE="0" SIZE=4>
<BR>Vertical:
<INPUT TYPE="text" NAME=y VALUE="0" SIZE=4>
<BR><INPUT TYPE="button" VALUE="Go"
   onClick="scrollIt(document.myForm)">
</FORM>
```

See also `Window.scrollBy`

setInterval

Evaluates an expression or calls a function every time a specified number of milliseconds elapses, until canceled by a call to `clearInterval`.

Method of	`Window`
Implemented in	Navigator 4.0

Syntax

```
setInterval(expression, msec)
setInterval(function, msec, arg1, ..., argN)
```

Parameters

`function`	Any function.
`expression`	A string containing a JavaScript expression. The expression must be quoted; otherwise, `setInterval` calls it immediately. For example, `setInterval("calcnum(3, 2)", 25)`.
`msec`	A numeric value or numeric string, in millisecond units.
`arg1, ..., argn`	(Optional) The arguments, if any, passed to `function`.

Description The timeouts continue to fire until the associated window or frame is destroyed or the interval is canceled using the `clearInterval` method.

Examples <<<*Redo for setInterval*>>>

Example 1. The following example displays an alert message five seconds (5,000 milliseconds) after the user clicks a button. If the user clicks the second button before the alert message is displayed, the timeout is canceled and the alert does not display.

```
<SCRIPT LANGUAGE="JavaScript">
function displayAlert() {
   alert("5 seconds have elapsed since the button was clicked.")
}
```

```
</SCRIPT>
<BODY>
<FORM>
Click the button on the left for a reminder in 5 seconds;
click the button on the right to cancel the reminder before
it is displayed.
<P>
<INPUT TYPE="button" VALUE="5-second reminder"
   NAME="remind_button"
   onClick="timerID=setTimeout('displayAlert()',5000)">
<INPUT TYPE="button" VALUE="Clear the 5-second reminder"
   NAME="remind_disable_button"
   onClick="clearTimeout(timerID)">
</FORM>
</BODY>
```

Example 2. The following example displays the current time in a `Text` object. The `showtime` function, which is called recursively, uses the `setTimeout` method to update the time every second.

```
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
<!--
var timerID = null
var timerRunning = false
function stopclock(){
   if(timerRunning)
      clearTimeout(timerID)
   timerRunning = false
}
function startclock(){
   // Make sure the clock is stopped
   stopclock()
   showtime()
}
function showtime(){
   var now = new Date()
   var hours = now.getHours()
   var minutes = now.getMinutes()
   var seconds = now.getSeconds()
   var timeValue = "" + ((hours > 12) ? hours - 12 : hours)
   timeValue += ((minutes < 10) ? ":0" : ":") + minutes
   timeValue += ((seconds < 10) ? ":0" : ":") + seconds
   timeValue += (hours >= 12) ? " P.M." : " A.M."
   document.clock.face.value = timeValue
   timerID = setTimeout("showtime()",1000)
   timerRunning = true
}
//-->
```

```
</SCRIPT>
</HEAD>

<BODY onLoad="startclock()">
<FORM NAME="clock" onSubmit="0">
   <INPUT TYPE="text" NAME="face" SIZE=12 VALUE ="">
</FORM>
</BODY>
```

See also `Window.clearInterval`, `Window.setTimeout`

setTimeout

Evaluates an expression or calls a function once after a specified number of milliseconds elapses.

Method of	`Window`
Implemented in	Navigator 2.0: Evaluating an expression. Navigator 4.0: Calling a function.

Syntax

```
setTimeout(expression, msec)
setTimeout(function, msec, arg1, ..., argN)
```

Parameters

`expression`	A string containing a JavaScript expression. The expression must be quoted; otherwise, `setTimeout` calls it immediately. For example, `setTimeout("calcnum(3, 2)", 25)`.
`msec`	A numeric value or numeric string, in millisecond units.
`function`	Any function.
`arg1, ..., argN`	(Optional) The arguments, if any, passed to `function`.

Description The `setTimeout` method evaluates an expression or calls a function after a specified amount of time. It does not act repeatedly. For example, if a `setTimeout` method specifies five seconds, the expression is evaluated or the function is called after five seconds, not every five seconds. For repetitive timeouts, use the `setInterval` method.

`setTimeout` does not stall the script. The script continues immediately (not waiting for the timeout to expire). The call simply schedules an additional future event.

Examples **Example 1.** The following example displays an alert message five seconds (5,000 milliseconds) after the user clicks a button. If the user clicks the second button before the alert message is displayed, the timeout is canceled and the alert does not display.

```
<SCRIPT LANGUAGE="JavaScript">
function displayAlert() {
   alert("5 seconds have elapsed since the button was clicked.")
}
</SCRIPT>
<BODY>
<FORM>
Click the button on the left for a reminder in 5 seconds;
click the button on the right to cancel the reminder before
it is displayed.
<P>
<INPUT TYPE="button" VALUE="5-second reminder"
   NAME="remind_button"
   onClick="timerID=setTimeout('displayAlert()',5000)">
<INPUT TYPE="button" VALUE="Clear the 5-second reminder"
   NAME="remind_disable_button"
   onClick="clearTimeout(timerID)">
</FORM>
</BODY>
```

Example 2. The following example displays the current time in a `Text` object. The `showtime` function, which is called recursively, uses the `setTimeout` method to update the time every second.

```
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
<!--
var timerID = null
var timerRunning = false
function stopclock(){
   if(timerRunning)
      clearTimeout(timerID)
   timerRunning = false
}
function startclock(){
   // Make sure the clock is stopped
   stopclock()
   showtime()
}
function showtime(){
   var now = new Date()
   var hours = now.getHours()
   var minutes = now.getMinutes()
   var seconds = now.getSeconds()
   var timeValue = "" + ((hours > 12) ? hours - 12 : hours)
   timeValue += ((minutes < 10) ? ":0" : ":") + minutes
   timeValue += ((seconds < 10) ? ":0" : ":") + seconds
   timeValue += (hours >= 12) ? " P.M." : " A.M."
   document.clock.face.value = timeValue
   timerID = setTimeout("showtime()",1000)
   timerRunning = true
}
//-->
</SCRIPT>
</HEAD>

<BODY onLoad="startclock()">
<FORM NAME="clock" onSubmit="0">
   <INPUT TYPE="text" NAME="face" SIZE=12 VALUE ="">
</FORM>
</BODY>
```

See also `Window.clearTimeout`, `Window.setInterval`

stop

Stops the current download.

Method of	`Window`
Implemented in	Navigator 4.0

Syntax `stop()`

Parameters None

Definition This method performs the same action as a user choosing the Stop button in the browser.

Frame

A window can display multiple, independently scrollable **frames** on a single screen, each with its own distinct URL. These frames are created using the `FRAME` tag inside a `FRAMESET` tag. Frames can point to different URLs and be targeted by other URLs, all within the same screen. A series of frames makes up a page. The `Frame` object is a convenience for thinking about the objects that constitute these frames. However, JavaScript actually represents a frame using a `Window` object. Every `Frame` object is a `Window` object, and has all the methods and properties of a `Window` object. There are a small number of minor differences between a window that is a frame and a top-level window. See `Window` for complete information on frames.

Client-side object

Implemented in	Navigator 2.0 Navigator 3.0: added `blur` and `focus` methods; added `onBlur` and `onFocus` event handlers

Location

Contains information on the current URL.

Client-side object

Implemented in	Navigator 2.0 Navigator 3.0: added `reload`, `replace` methods

Created by `Location` objects are predefined JavaScript objects that you access through the `location` property of a `Window` object:

Description The `location` object represents the complete URL associated with a given `Window` object. Each property of the `location` object represents a different portion of the URL.

In general, a URL has this form:

```
protocol//host:port/pathname#hash?search
```

For example:

```
http://home.netscape.com/assist/extensions.html#topic1?x=7&y=2
```

These parts serve the following purposes:

- `protocol` represents the beginning of the URL, up to and including the first colon.
- `host` represents the host and domain name, or IP address, of a network host.
- `port` represents the communications port that the server uses for communications.
- `pathname` represents the URL-path portion of the URL.
- `hash` represents an anchor name fragment in the URL, including the hash mark (#). This property applies to HTTP URLs only.
- `search` represents any query information in the URL, including the question mark (?). This property applies to HTTP URLs only. The search string contains variable and value pairs; each pair is separated by an ampersand (&).

A `Location` object has a property for each of these parts of the URL. See the individual properties for more information. A `Location` object has two other properties not shown here:

- `href` represents a complete URL.
- `hostname` represents the concatenation `host`:`port`.

If you assign a string to the `location` property of an object, JavaScript creates a `location` object and assigns that string to its `href` property. For example, the following two statements are equivalent and set the URL of the current window to the Netscape home page:

```
window.location.href="http://home.netscape.com/"
window.location="http://home.netscape.com/"
```

The `location` object is contained by the `window` object and is within its scope. If you refer to a `location` object without specifying a window, the `location` object represents the current location. If you refer to a `location` object and specify a window name, as in `windowReference.location`, the `location` object represents the location of the specified window.

In event handlers, you must specify `window.location` instead of simply using `location`. Due to the scoping of static objects in JavaScript, a call to `location` without specifying an object name is equivalent to `document.location`, which is a synonym for `document.URL`.

`Location` is not a property of the `document` object; its equivalent is the `document.URL` property. The `document.location` property, which is a synonym for `document.URL`, will be removed in a future release.

How documents are loaded when location is set

When you set the `location` object or any of its properties except `hash,` whether a new document is loaded depends on which version of the browser you are running:

- In Navigator 2.0, setting `location` does a conditional ("If-modified-since") HTTP GET operation, which returns no data from the server unless the document has been modified since the last version downloaded.
- In Navigator 3.0 and later, the effect of setting `location` depends on the user's setting for comparing a document to the original over the network. The user interface option for setting this preference differs in browser versions. The user decides whether to check a document in cache every time it is accessed, once per session, or never. The document is reloaded from cache if the user sets never or once per session; the document is reloaded from the server only if the user chooses every time.

Syntax for common URL types

When you specify a URL, you can use standard URL formats and JavaScript statements. Table 6.2 shows the syntax for specifying some of the most common types of URLs.

Table 6.2 URL syntax.

URL type	Protocol	Example
JavaScript code	javascript:	javascript:history.go(-1)
Navigator source viewer	view-source:	`view-source:wysiwyg://0/file:/c|/temp/genhtml.html`
Navigator info	about:	about:cache
World Wide Web	http:	`http://home.netscape.com/`

Table 6.2 URL syntax. (Continued)

URL type	Protocol	Example
File	file:/	`file:///javascript/methods.html`
FTP	ftp:	`ftp://ftp.mine.com/home/mine`
MailTo	mailto:	mailto:info@netscape.com
Usenet	news:	`news://news.scruznet.com/comp.lang.javascript`
Gopher	gopher:	gopher.myhost.com

The `javascript:` protocol evaluates the expression after the colon (:), if there is one, and loads a page containing the string value of the expression, unless it is undefined. If the expression evaluates to undefined (by calling a void function, for example `javascript:void(0)`), no new page loads. Note that loading a new page over your script's page clears the page's variables, functions, and so on.

The `view-source:` protocol displays HTML code that was generated with JavaScript `document.write` and `document.writeln` methods. For information on printing and saving generated HTML, see `write`.

The `about:` protocol provides information on Navigator and has the following syntax:

```
about:
about:cache
about:plugins
```

- `about:` by itself is the same as choosing About Communicator from the Navigator Help menu.
- `about:cache` displays disk-cache statistics.
- `about:plugins` displays information about plug-ins you have configured. This is the same as choosing About Plug-ins from the Navigator Help menu.

Property Summary

Property	Description
`hash`	Specifies an anchor name in the URL.
`host`	Specifies the host and domain name, or IP address, of a network host.
`hostname`	Specifies the `host:port` portion of the URL.
`href`	Specifies the entire URL.
`pathname`	Specifies the URL-path portion of the URL.
`port`	Specifies the communications port that the server uses.
`protocol`	Specifies the beginning of the URL, including the colon.
`search`	Specifies a query.

Method Summary

Method	Description
`reload`	Forces a reload of the window's current document.
`replace`	Loads the specified URL over the current history entry.

Examples **Example 1.** The following two statements are equivalent and set the URL of the current window to the Netscape home page:

```
window.location.href="http://home.netscape.com/"
window.location="http://home.netscape.com/"
```

Example 2. The following statement sets the URL of a frame named `frame2` to the Sun home page:

```
parent.frame2.location.href="http://www.sun.com/"
```

See also the examples for `Anchor`.

See also `History`, `document.URL`

Properties

hash

A string beginning with a hash mark (#) that specifies an anchor name in the URL.

Property of	`Location`
Implemented in	Navigator 2.0

Security Navigator 3.0: This property is tainted by default. For information on data tainting, see "Security" on page 55.

Description The `hash` property specifies a portion of the URL. This property applies to HTTP URLs only.

You can set the `hash` property at any time, although it is safer to set the `href` property to change a location. If the hash that you specify cannot be found in the current location, you get an error.

Setting the `hash` property navigates to the named anchor without reloading the document. This differs from the way a document is loaded when other `location` properties are set (see "How documents are loaded when location is set" on page 346).

See RFC 1738 (`http://www.cis.ohio-state.edu/htbin/rfc/rfc1738.html`) for complete information about the hash.

Examples In the following example, the `window.open` statement creates a window called `newWindow` and loads the specified URL into it. The `document.write` statements display properties of `newWindow.location` in a window called `msgWindow`.

```
newWindow=window.open
   ("http://home.netscape.com/comprod/products/navigator/
   version_2.0/script/script_info/objects.html#checkbox_object")

msgWindow.document.write("newWindow.location.href = " +
   newWindow.location.href + "<P>")
msgWindow.document.write("newWindow.location.hash = " +
   newWindow.location.hash + "<P>")
msgWindow.document.close()
```

The previous example displays output such as the following:

```
newWindow.location.href =
   http://home.netscape.com/comprod/products/navigator/
   version_2.0/script/script_info/objects.html#checkbox_object
newWindow.location.hash = #checkbox_object
```

See also `Location.host`, `Location.hostname`, `Location.href`, `Location.pathname`, `Location.port`, `Location.protocol`, `Location.search`

host

A string specifying the server name, subdomain, and domain name.

Property of	`Location`
Implemented in	Navigator 2.0

Security Navigator 3.0: This property is tainted by default. For information on data tainting, see "Security" on page 55.

Description The `host` property specifies a portion of a URL. The `host` property is a substring of the `hostname` property. The `hostname` property is the concatenation of the `host` and `port` properties, separated by a colon. When the `port` property is null, the `host` property is the same as the `hostname` property.

You can set the `host` property at any time, although it is safer to set the `href` property to change a location. If the host that you specify cannot be found in the current location, you get an error.

See Section 3.1 of RFC 1738 (`http://www.cis.ohio-state.edu/htbin/rfc/rfc1738.html`) for complete information about the hostname and port.

Examples In the following example, the `window.open` statement creates a window called `newWindow` and loads the specified URL into it. The `document.write` statements display properties of `newWindow.location` in a window called `msgWindow`.

```
newWindow=window.open
   ("http://home.netscape.com/comprod/products/navigator/
   version_2.0/script/script_info/objects.html#checkbox_object")

msgWindow.document.write("newWindow.location.href = " +
   newWindow.location.href + "<P>")
msgWindow.document.write("newWindow.location.host = " +
   newWindow.location.host + "<P>")
msgWindow.document.close()
```

The previous example displays output such as the following:

```
newWindow.location.href =
   http://home.netscape.com/comprod/products/navigator/
   version_2.0/script/script_info/objects.html#checkbox_object
newWindow.location.host = home.netscape.com
```

See also `Location.hash`, `Location.hostname`, `Location.href`, `Location.pathname`, `Location.port`, `Location.protocol`, `Location.search`

hostname

A string containing the full hostname of the server, including the server name, subdomain, domain, and port number.

Property of	`Location`
Implemented in	Navigator 2.0

Security Navigator 3.0: This property is tainted by default. For information on data tainting, see "Security" on page 55.

Description The `hostname` property specifies a portion of a URL. The `hostname` property is the concatenation of the `host` and `port` properties, separated by a colon. When the `port` property is 80 (the default), the `host` property is the same as the `hostname` property.

You can set the `hostname` property at any time, although it is safer to set the `href` property to change a location. If the hostname that you specify cannot be found in the current location, you get an error.

See Section 3.1 of RFC 1738 (`http://www.cis.ohio-state.edu/htbin/rfc/rfc1738.html`) for complete information about the hostname.

Examples In the following example, the `window.open` statement creates a window called `newWindow` and loads the specified URL into it. The `document.write` statements display properties of `newWindow.location` in a window called `msgWindow`.

```
newWindow=window.open
   ("http://home.netscape.com/comprod/products/navigator/
   version_2.0/script/script_info/objects.html#checkbox_object")

msgWindow.document.write("newWindow.location.href = " +
   newWindow.location.href + "<P>")
msgWindow.document.write("newWindow.location.hostName = " +
   newWindow.location.hostName + "<P>")
msgWindow.document.close()
```

The previous example displays output such as the following:

```
newWindow.location.href =
   http://home.netscape.com/comprod/products/navigator/
   version_2.0/script/script_info/objects.html#checkbox_object
newWindow.location.hostName = home.netscape.com
```

See also `Location.hash`, `Location.host`, `Location.href`, `Location.pathname`, `Location.port`, `Location.protocol`, `Location.search`

href

A string specifying the entire URL.

Property of	`Location`
Implemented in	Navigator 2.0

Security Navigator 3.0: This property is tainted by default. For information on data tainting, see “Security” on page 55.

Description The `href` property specifies the entire URL. Other `location` object properties are substrings of the `href` property. If you want to change the URL associated with a window, you should do so by changing the `href` property; this correctly updates all of the other properties.

You can set the `href` property at any time.

Omitting a property name from the `location` object is equivalent to specifying `location.href`. For example, the following two statements are equivalent and set the URL of the current window to the Netscape home page:

```
window.location.href="http://home.netscape.com/"
window.location="http://home.netscape.com/"
```

See RFC 1738 (`http://www.cis.ohio-state.edu/htbin/rfc/rfc1738.html`) for complete information about the URL.

Examples In the following example, the `window.open` statement creates a window called `newWindow` and loads the specified URL into it. The `document.write` statements display all the properties of `newWindow.location` in a window called `msgWindow`.

```
newWindow=window.open
   ("http://home.netscape.com/comprod/products/navigator/
   version_2.0/script/script_info/objects.html#checkbox_object")

msgWindow.document.write("newWindow.location.href = " +
   newWindow.location.href + "<P>")
msgWindow.document.write("newWindow.location.protocol = " +
   newWindow.location.protocol + "<P>")
msgWindow.document.write("newWindow.location.host = " +
   newWindow.location.host + "<P>")
msgWindow.document.write("newWindow.location.hostName = " +
   newWindow.location.hostName + "<P>")
msgWindow.document.write("newWindow.location.port = " +
   newWindow.location.port + "<P>")
msgWindow.document.write("newWindow.location.pathname = " +
   newWindow.location.pathname + "<P>")
msgWindow.document.write("newWindow.location.hash = " +
   newWindow.location.hash + "<P>")
msgWindow.document.write("newWindow.location.search = " +
   newWindow.location.search + "<P>")
msgWindow.document.close()
```

The previous example displays output such as the following:

```
newWindow.location.href =
   http://home.netscape.com/comprod/products/navigator/
   version_2.0/script/script_info/objects.html#checkbox_object
newWindow.location.protocol = http:
newWindow.location.host = home.netscape.com
newWindow.location.hostName = home.netscape.com
newWindow.location.port =
newWindow.location.pathname =
   /comprod/products/navigator/version_2.0/script/
   script_info/objects.html
newWindow.location.hash = #checkbox_object
newWindow.location.search =
```

See also `Location.hash`, `Location.host`, `Location.hostname`, `Location.pathname`, `Location.port`, `Location.protocol`, `Location.search`

pathname

A string specifying the URL-path portion of the URL.

Property of	`Location`
Implemented in	Navigator 2.0

Security Navigator 3.0: This property is tainted by default. For information on data tainting, see "Security" on page 55.

Description The `pathname` property specifies a portion of the URL. The pathname supplies the details of how the specified resource can be accessed.

You can set the `pathname` property at any time, although it is safer to set the `href` property to change a location. If the pathname that you specify cannot be found in the current location, you get an error.

See Section 3.1 of RFC 1738 (`http://www.cis.ohio-state.edu/htbin/rfc/rfc1738.html`) for complete information about the pathname.

Examples In the following example, the `window.open` statement creates a window called `newWindow` and loads the specified URL into it. The `document.write` statements display properties of `newWindow.location` in a window called `msgWindow`.

```
newWindow=window.open
   ("http://home.netscape.com/comprod/products/navigator/
   version_2.0/script/script_info/objects.html#checkbox_object")

msgWindow.document.write("newWindow.location.href = " +
   newWindow.location.href + "<P>")
msgWindow.document.write("newWindow.location.pathname = " +
   newWindow.location.pathname + "<P>")
msgWindow.document.close()
```

The previous example displays output such as the following:

```
newWindow.location.href =
   http://home.netscape.com/comprod/products/navigator/
   version_2.0/script/script_info/objects.html#checkbox_object
newWindow.location.pathname =
   /comprod/products/navigator/version_2.0/script/
   script_info/objects.html
```

See also `Location.hash`, `Location.host`, `Location.hostname`, `Location.href`, `Location.port`, `Location.protocol`, `Location.search`

port

A string specifying the communications port that the server uses.

Property of	`Location`
Implemented in	Navigator 2.0

Security Navigator 3.0: This property is tainted by default. For information on data tainting, see "Security" on page 55.

Description The `port` property specifies a portion of the URL. The `port` property is a substring of the `hostname` property. The `hostname` property is the concatenation of the `host` and `port` properties, separated by a colon.

You can set the `port` property at any time, although it is safer to set the `href` property to change a location. If the port that you specify cannot be found in the current location, you get an error. If the `port` property is not specified, it defaults to 80.

See Section 3.1 of RFC 1738 (`http://www.cis.ohio-state.edu/htbin/rfc/rfc1738.html`) for complete information about the port.

Examples In the following example, the `window.open` statement creates a window called `newWindow` and loads the specified URL into it. The `document.write` statements display properties of `newWindow.location` in a window called `msgWindow`.

```
newWindow=window.open
   ("http://home.netscape.com/comprod/products/navigator/
   version_2.0/script/script_info/objects.html#checkbox_object")

msgWindow.document.write("newWindow.location.href = " +
   newWindow.location.href + "<P>")
msgWindow.document.write("newWindow.location.port = " +
   newWindow.location.port + "<P>")
msgWindow.document.close()
```

The previous example displays output such as the following:

```
newWindow.location.href =
   http://home.netscape.com/comprod/products/navigator/
   version_2.0/script/script_info/objects.html#checkbox_object
newWindow.location.port =
```

See also `Location.hash`, `Location.host`, `Location.hostname`, `Location.href`, `Location.pathname`, `Location.protocol`, `Location.search`

protocol

A string specifying the beginning of the URL, up to and including the first colon.

Property of	`Location`

Implemented in Navigator 2.0

Security Navigator 3.0: This property is tainted by default. For information on data tainting, see "Security" on page 55.

Description The `protocol` property specifies a portion of the URL. The protocol indicates the access method of the URL. For example, the value `"http:"` specifies HyperText Transfer Protocol, and the value `"javascript:"` specifies JavaScript code.

You can set the `protocol` property at any time, although it is safer to set the `href` property to change a location. If the protocol that you specify cannot be found in the current location, you get an error.

The `protocol` property represents the scheme name of the URL. See Section 2.1 of RFC 1738 (`http://www.cis.ohio-state.edu/htbin/rfc/rfc1738.html`) for complete information about the protocol.

Examples In the following example, the `window.open` statement creates a window called `newWindow` and loads the specified URL into it. The `document.write` statements display properties of `newWindow.location` in a window called `msgWindow`.

```
newWindow=window.open
   ("http://home.netscape.com/comprod/products/navigator/
   version_2.0/script/script_info/objects.html#checkbox_object")

msgWindow.document.write("newWindow.location.href = " +
   newWindow.location.href + "<P>")
msgWindow.document.write("newWindow.location.protocol = " +
   newWindow.location.protocol + "<P>")
msgWindow.document.close()
```

The previous example displays output such as the following:

```
newWindow.location.href =
   http://home.netscape.com/comprod/products/navigator/
   version_2.0/script/script_info/objects.html#checkbox_object
newWindow.location.protocol = http:
```

See also `Location.hash`, `Location.host`, `Location.hostname`, `Location.href`, `Location.pathname`, `Location.port`, `Location.search`

search

A string beginning with a question mark that specifies any query information in the URL.

Property of	`Location`
Implemented in	Navigator 2.0

Security Navigator 3.0: This property is tainted by default. For information on data tainting, see "Security" on page 55.

Description The `search` property specifies a portion of the URL. This property applies to HTTP URLs only.

The `search` property contains variable and value pairs; each pair is separated by an ampersand. For example, two pairs in a search string could look as follows:

```
?x=7&y=5
```

You can set the `search` property at any time, although it is safer to set the `href` property to change a location. If the search that you specify cannot be found in the current location, you get an error.

See Section 3.3 of RFC 1738 (`http://www.cis.ohio-state.edu/htbin/rfc/rfc1738.html`) for complete information about the search.

Examples In the following example, the `window.open` statement creates a window called `newWindow` and loads the specified URL into it. The `document.write` statements display properties of `newWindow.location` in a window called `msgWindow`.

```
newWindow=window.open
   ("http://guide-p.infoseek.com/WW/NS/Titles?qt=RFC+1738+&col=WW")

msgWindow.document.write("newWindow.location.href = " +
   newWindow.location.href + "<P>")
msgWindow.document.close()
msgWindow.document.write("newWindow.location.search = " +
   newWindow.location.search + "<P>")
msgWindow.document.close()
```

The previous example displays the following output:

```
newWindow.location.href =
   http://guide-p.infoseek.com/WW/NS/Titles?qt=RFC+1738+&col=WW
newWindow.location.search = ?qt=RFC+1738+&col=WW
```

See also `Location.hash`, `Location.host`, `Location.hostname`, `Location.href`, `Location.pathname`, `Location.port`, `Location.protocol`

Methods

reload

Forces a reload of the window's current document (the document specified by the `Location.href` property).

Method of	`Location`
Implemented in	Navigator 3.0

Syntax `reload(forceGet)`

Parameters

`forceGet`	(Optional) If you supply `true`, forces an unconditional HTTP GET of the document from the server. This should not be used unless you have reason to believe that disk and memory caches are off or broken, or the server has a new version of the document (for example, if it is generated by a CGI on each request).

Description This method uses the same policy that the browser's Reload button uses. The user interface for setting the default value of this policy varies for different browser versions.

By default, the `reload` method does not force a transaction with the server. However, if the user has set the preference to check every time, the method does a "conditional GET" request using an If-modified-since HTTP header, to ask the server to return the document only if its last-modified time is newer than the time the client keeps in its cache. In other words, `reload` reloads from the cache, unless the user has specified to check every time *and* the document has changed on the server since it was last loaded and saved in the cache.

Examples The following example displays an image and three radio buttons. The user can click the radio buttons to choose which image is displayed. Clicking another button lets the user reload the document.

```
<SCRIPT>
function displayImage(theImage) {
   document.images[0].src=theImage
}
</SCRIPT>

<FORM NAME="imageForm">
<B>Choose an image:</B>
<BR><INPUT TYPE="radio" NAME="imageChoice" VALUE="image1" CHECKED
   onClick="displayImage('seaotter.gif')">Sea otter
<BR><INPUT TYPE="radio" NAME="imageChoice" VALUE="image2"
   onClick="displayImage('orca.gif')">Killer whale
<BR><INPUT TYPE="radio" NAME="imageChoice" VALUE="image3"
   onClick="displayImage('humpback.gif')">Humpback whale

<BR>
<IMG NAME="marineMammal" SRC="seaotter.gif" ALIGN="left" VSPACE="10">

<P><INPUT TYPE="button" VALUE="Click here to reload"
   onClick="window.location.reload()">
</FORM>
```

See also `Location.replace`

replace

Loads the specified URL over the current history entry.

Method of	`Location`
Implemented in	Navigator 3.0

Syntax `replace("URL")`

Parameters

`URL`	Specifies the URL to load.

Description The `replace` method loads the specified URL over the current history entry. After calling the `replace` method, the user cannot navigate to the previous URL by using browser's Back button.

If your program will be run with JavaScript in Navigator 2.0, you could put the following line in a `SCRIPT` tag early in your program. This emulates `replace`, which was introduced in Navigator 3.0:

```
if (location.replace == null)
   location.replace = location.assign
```

The `replace` method does not create a new entry in the history list. To create an entry in the history list while loading a URL, use the `History.go` method.

Examples The following example lets the user choose among several catalogs to display. The example displays two sets of radio buttons which let the user choose a season and a category, for example the Spring/Summer Clothing catalog or the Fall/Winter Home & Garden catalog. When the user clicks the Go button, the `displayCatalog` function executes the `replace` method, replacing the current URL with the URL appropriate for the catalog the user has chosen. After invoking `displayCatalog`, the user cannot navigate to the previous URL (the list of catalogs) by using browser's Back button.

```
<SCRIPT>
function displayCatalog() {
   var seaName=""
   var catName=""

   for (var i=0; i < document.catalogForm.season.length; i++) {
      if (document.catalogForm.season[i].checked) {
         seaName=document.catalogForm.season[i].value
         i=document.catalogForm.season.length
      }
   }

   for (var i in document.catalogForm.category) {
      if (document.catalogForm.category[i].checked) {
         catName=document.catalogForm.category[i].value
         i=document.catalogForm.category.length
      }
   }
   fileName=seaName + catName + ".html"
   location.replace(fileName)
}
</SCRIPT>

<FORM NAME="catalogForm">
<B>Which catalog do you want to see?</B>

<P><B>Season</B>
<BR><INPUT TYPE="radio" NAME="season" VALUE="q1" CHECKED>Spring/Summer
<BR><INPUT TYPE="radio" NAME="season" VALUE="q3">Fall/Winter

<P><B>Category</B>
<BR><INPUT TYPE="radio" NAME="category" VALUE="clo" CHECKED>Clothing
<BR><INPUT TYPE="radio" NAME="category" VALUE="lin">Linens
<BR><INPUT TYPE="radio" NAME="category" VALUE="hom">Home & Garden

<P><INPUT TYPE="button" VALUE="Go" onClick="displayCatalog()">
</FORM>
```

See also `History`, `Window.open`, `History.go`, `Location.reload`

History

Contains an array of information on the URLs that the client has visited within a window. This information is stored in a history list and is accessible through the browser's Go menu.

Client-side object

Implemented in	Navigator 2.0 Navigator 3.0: added `current, next,` and `previous` properties.

Created by `History` objects are predefined JavaScript objects that you access through the `history` property of a `Window` object.

Description To change a window's current URL without generating a history entry, you can use the `Location.replace` method. This replaces the current page with a new one without generating a history entry. See `Location.replace`.

You can refer to the history entries by using the `Window.history` array. This array contains an entry for each history entry in source order. Each array entry is a string containing a URL. For example, if the history list contains three named entries, these entries are reflected as `history[0]`, `history[1]`, and `history[2]`.

If you access the `history` array without specifying an array element, the browser returns a string of HTML which displays a table of URLs, each of which is a link.

Property Summary

Property	Description
`current`	Specifies the URL of the current history entry.
`length`	Reflects the number of entries in the history list.
`next`	Specifies the URL of the next history entry.
`previous`	Specifies the URL of the previous history entry.

Method Summary

Method	Description
`back`	Loads the previous URL in the history list.
`forward`	Loads the next URL in the history list.
`go`	Loads a URL from the history list.

Examples

Example 1. The following example goes to the URL the user visited three clicks ago in the current window.

```
history.go(-3)
```

Example 2. You can use the `history` object with a specific window or frame. The following example causes `window2` to go back one item in its window (or session) history:

```
window2.history.back()
```

Example 3. The following example causes the second frame in a frameset to go back one item:

```
parent.frames[1].history.back()
```

Example 4. The following example causes the frame named `frame1` in a frameset to go back one item:

```
parent.frame1.history.back()
```

Example 5. The following example causes the frame named `frame2` in `window2` to go back one item:

```
window2.frame2.history.back()
```

Example 6. The following code determines whether the first entry in the `history` array contains the string `"NETSCAPE"`. If it does, the function `myFunction` is called.

```
if (history[0].indexOf("NETSCAPE") != -1) {
   myFunction(history[0])
}
```

Example 7. The following example displays the entire history list:

```
document.writeln("<B>history is</B> " + history)
```

This code displays output similar to the following:

```
history is
Welcome to Netscape http://home.netscape.com/
Sun Microsystems http://www.sun.com/
Royal Airways http://www.supernet.net/~dugbrown/
```

See also `Location`, `Location.replace`

Properties

current

A string specifying the complete URL of the current history entry.

Property of	`History`
Read-only	
Implemented in	Navigator 3.0

Security Navigator 3.0: This property is tainted by default. It has no value of data tainting is disabled. For information on data tainting, see "Security" on page 55.

Navigator 4.0: Getting the value of this property requires the `UniversalBrowserRead` privilege. It has no value if you do not have this privilege. For information on security in Navigator 4.0, see Chapter 7, "JavaScript Security," in the *JavaScript Guide*.

Examples The following example determines whether `history.current` contains the string `"netscape.com"`. If it does, the function `myFunction` is called.

```
if (history.current.indexOf("netscape.com") != -1) {
   myFunction(history.current)
}
```

See also `History.next`, `History.previous`

length

The number of elements in the `history` array.

Property of	`History`
Read-only	
Implemented in	Navigator 2.0

Security Navigator 4.0: Getting the value of this property requires the `UniversalBrowserRead` privilege. For information on security in Navigator 4.0, see Chapter 7, "JavaScript Security," in the *JavaScript Guide*.

next

A string specifying the complete URL of the next history entry.

Property of	`History`
Read-only	
Implemented in	Navigator 3.0

Security Navigator 3.0: This property is tainted by default. It has no value of data tainting is disabled. For information on data tainting, see "Security" on page 55.

Navigator 4.0: Getting the value of this property requires the `UniversalBrowserRead` privilege. It has no value if you do not have this privilege. For information on security in Navigator 4.0, see Chapter 7, "JavaScript Security," in the *JavaScript Guide*.

Description The `next` property reflects the URL that would be used if the user chose Forward from the Go menu.

Examples The following example determines whether `history.next` contains the string `"NETSCAPE.COM"`. If it does, the function `myFunction` is called.

```
if (history.next.indexOf("NETSCAPE.COM") != -1) {
   myFunction(history.next)
}
```

See also `History.current`, `History.previous`

previous

A string specifying the complete URL of the previous history entry.

Property of	`History`
Read-only	
Implemented in	Navigator 3.0

Security Navigator 3.0: This property is tainted by default. It has no value of data tainting is disabled. For information on data tainting, see "Security" on page 55.

Navigator 4.0: Getting the value of this property requires the `UniversalBrowserRead` privilege. It has no value if you do not have this privilege. For information on security in Navigator 4.0, see Chapter 7, "JavaScript Security," in the *JavaScript Guide*.

Description The `previous` property reflects the URL that would be used if the user chose Back from the Go menu.

Examples The following example determines whether `history.previous` contains the string `"NETSCAPE.COM"`. If it does, the function `myFunction` is called.

```
if (history.previous.indexOf("NETSCAPE.COM") != -1) {
   myFunction(history.previous)
}
```

See also `History.current`, `History.next`

Methods

back

Loads the previous URL in the history list.

Method of	`History`
Implemented in	Navigator 2.0

Syntax `back()`

Parameters None

Description This method performs the same action as a user choosing the Back button in the browser. The `back` method is the same as `history.go(-1)`.

Examples The following custom buttons perform the same operation as the browser's Back button:

```
<P><INPUT TYPE="button" VALUE="< Go Back"
   onClick="history.back()">
<P><INPUT TYPE="button" VALUE="> Go Back"
   onClick="myWindow.back()">
```

See also `History.forward`, `History.go`

forward

Loads the next URL in the history list.

Method of	`History`
Implemented in	Navigator 2.0

Syntax `forward()`

Parameters None

Description This method performs the same action as a user choosing the Forward button in the browser. The `forward` method is the same as `history.go(1)`.

Examples The following custom buttons perform the same operation as the browser's Forward button:

```
<P><INPUT TYPE="button" VALUE="< Forward"
   onClick="history.forward()">
<P><INPUT TYPE="button" VALUE="> Forward"
   onClick="myWindow.forward()">
```

See also `History.back`, `History.go`

go

Loads a URL from the history list.

Method of	`History`
Implemented in	Navigator 2.0

Syntax

```
go(delta)
go(location)
```

Parameters

`delta`	An integer representing a relative position in the history list.
`location`	A string representing all or part of a URL in the history list.

Description The `go` method navigates to the location in the history list determined by the specified parameter.

If the `delta` argument is 0, the browser reloads the current page. If it is an integer greater than 0, the `go` method loads the URL that is that number of

Chapter 7 Form

This chapter deals with the use of forms, which appear within a document to obtain input from the user.

Table 7.1 summarizes the objects in this chapter.

Table 7.1 Form objects

Object	Description
`Button`	A push button on an HTML form.
`Checkbox`	A checkbox on an HTML form.
`FileUpload`	A file upload element on an HTML form.
`Form`	Lets users input text and make choices from Form elements such as checkboxes, radio buttons, and selection lists.
`Hidden`	A `Text` object that is suppressed from form display on an HTML form.
`Option`	A `Select` object option.
`Password`	A text field on an HTML form that conceals its value by displaying asterisks (*).
`Radio`	A set of radio buttons on an HTML form.
`Reset`	A reset button on an HTML form.
`Select`	A selection list on an HTML form.

Table 7.1 Form objects

Object	Description
`Submit`	A submit button on an HTML form.
`Text`	A text input field on an HTML form.
`Textarea`	A multiline input field on an HTML form.

Form

Lets users input text and make choices from `Form` elements such as checkboxes, radio buttons, and selection lists. You can also use a form to post data to a server.

Client-side object

Implemented in	Navigator 2.0 Navigator 3.0: added `reset` method. Navigator 4.0: added `handleEvent` method.

Created by The HTML `FORM` tag. The JavaScript runtime engine creates a `Form` object for each `FORM` tag in the document. You access `FORM` objects through the `document.forms` property and through named properties of that object.

To define a form, use standard HTML syntax with the addition of JavaScript event handlers. If you supply a value for the `NAME` attribute, you can use that value to index into the `forms` array. In addition, the associated `document` object has a named property for each named form.

Event handlers

- `onReset`
- `onSubmit`

Description Each form in a document is a distinct object. You can refer to a form's elements in your code by using the element's name (from the `NAME` attribute) or the `Form.elements` array. The `elements` array contains an entry for each element (such as a `Checkbox`, `Radio`, or `Text` object) in a form.

If multiple objects on the same form have the same `NAME` attribute, an array of the given name is created automatically. Each element in the array represents an individual `Form` object. Elements are indexed in source order starting at 0. For example, if two `Text` elements and a `Textarea` element on the same form have their NAME attribute set to "myField", an array with the elements myField[0], myField[1], and myField[2] is created. You need to be aware of this situation in your code and know whether myField refers to a single element or to an array of elements.

Property Summary

Property	Description
action	Reflects the ACTION attribute.
elements	An array reflecting all the elements in a form.
encoding	Reflects the ENCTYPE attribute.
length	Reflects the number of elements on a form.
method	Reflects the METHOD attribute.
name	Reflects the NAME attribute.
target	Reflects the TARGET attribute.

Method Summary

Method	Description
handleEvent	Invokes the handler for the specified event.
reset	Simulates a mouse click on a reset button for the calling form.
submit	Submits a form.

Examples

Example 1: Named form. The following example creates a form called myForm that contains text fields for first name and last name. The form also contains two buttons that change the names to all uppercase or all lowercase. The function setCase shows how to refer to the form by its name.

```
<HTML>
<HEAD>
<TITLE>Form object example</TITLE>
</HEAD>
<SCRIPT>
function setCase (caseSpec){
if (caseSpec == "upper") {
   document.myForm.firstName.value=document.myForm.firstName.value.toUpperCase()
   document.myForm.lastName.value=document.myForm.lastName.value.toUpperCase()}
else {
   document.myForm.firstName.value=document.myForm.firstName.value.toLowerCase()
   document.myForm.lastName.value=document.myForm.lastName.value.toLowerCase()}
}
</SCRIPT>

<BODY>
<FORM NAME="myForm">
<B>First name:</B>
<INPUT TYPE="text" NAME="firstName" SIZE=20>
<BR><B>Last name:</B>
<INPUT TYPE="text" NAME="lastName" SIZE=20>
<P><INPUT TYPE="button" VALUE="Names to uppercase" NAME="upperButton"
   onClick="setCase('upper')">
<INPUT TYPE="button" VALUE="Names to lowercase" NAME="lowerButton"
   onClick="setCase('lower')">
</FORM>
</BODY>
</HTML>
```

Example 2: forms array. The `onLoad` event handler in the following example displays the name of the first form in an Alert dialog box.

```
<BODY onLoad="alert('You are looking at the ' + document.forms[0] + '
form!')">
```

If the form name is `musicType`, the alert displays the following message:

```
You are looking at the <object musicType> form!
```

Example 3: onSubmit event handler. The following example shows an `onSubmit` event handler that determines whether to submit a form. The form contains one `Text` object where the user enters three characters. `onSubmit` calls a function, `checkData`, that returns true if there are 3 characters; otherwise, it returns false. Notice that the form's `onSubmit` event handler, not the submit button's `onClick` event handler, calls the `checkData` function. Also, `onSubmit` contains a `return` statement that returns the value obtained with the function `call`.

```
<HTML>
<HEAD>
<TITLE>Form object/onSubmit event handler example</TITLE>
<TITLE>Form object example</TITLE>
</HEAD>
<SCRIPT>
var dataOK=false
function checkData (){
if (document.myForm.threeChar.value.length == 3) {
   return true}
   else {
      alert("Enter exactly three characters. " + document.myForm.threeChar.value +
         " is not valid.")
      return false}
}
</SCRIPT>
<BODY>
<FORM NAME="myForm" onSubmit="return checkData()">
<B>Enter 3 characters:</B>
<INPUT TYPE="text" NAME="threeChar" SIZE=3>
<P><INPUT TYPE="submit" VALUE="Done" NAME="submit1"

onClick="document.myForm.threeChar.value=document.myForm.threeChar.value.toUpperCase()">
</FORM>
</BODY>
</HTML>
```

Example 4: submit method. The following example is similar to the previous one, except it submits the form using the `submit` method instead of a `Submit` object. The form's `onSubmit` event handler does not prevent the form from being submitted. The form uses a button's `onClick` event handler to call the `checkData` function. If the value is valid, the `checkData` function submits the form by calling the form's `submit` method.

```
<HTML>
<HEAD>
<TITLE>Form object/submit method example</TITLE>
</HEAD>
<SCRIPT>
var dataOK=false
function checkData (){
if (document.myForm.threeChar.value.length == 3) {
   document.myForm.submit()}
   else {
      alert("Enter exactly three characters. " +
document.myForm.threeChar.value +
         " is not valid.")
      return false}
}
</SCRIPT>
<BODY>
<FORM NAME="myForm" onSubmit="alert('Form is being submitted.')">
<B>Enter 3 characters:</B>
<INPUT TYPE="text" NAME="threeChar" SIZE=3>
<P><INPUT TYPE="button" VALUE="Done" NAME="button1"
   onClick="checkData()">
</FORM>
</BODY>
</HTML>
```

See also `Button`, `Checkbox`, `FileUpload`, `Hidden`, `Password`, `Radio`, `Reset`, `Select`, `Submit`, `Text`, `Textarea`.

Properties

action

A string specifying a destination URL for form data that is submitted

Property of	`Form`
Implemented in	Navigator 2.0

Security Navigator 3.0: This property is tainted by default. For information on data tainting, see "Security" on page 55.

Navigator 4.0: Submitting a form to a `mailto:` or `news:` URL requires the `UniversalSendMail` privilege. For information on security in Navigator 4.0, see Chapter 7, "JavaScript Security," in the *JavaScript Guide*.

Description The `action` property is a reflection of the `ACTION` attribute of the `FORM` tag. Each section of a URL contains different information. See `Location` for a description of the URL components.

Examples The following example sets the `action` property of the `musicForm` form to the value of the variable `urlName:`

```
document.musicForm.action=urlName
```

See also `Form.encoding`, `Form.method`, `Form.target`

elements

An array of objects corresponding to form elements (such as `checkbox`, `radio`, and `Text` objects) in source order.

Property of	`Form`
Read-only	
Implemented in	Navigator 2.0

Description You can refer to a form's elements in your code by using the `elements` array. This array contains an entry for each object (`Button`, `Checkbox`, `FileUpload`, `Hidden`, `Password`, `Radio`, `Reset`, `Select`, `Submit`, `Text`, or `Textarea` object) in a form in source order. Each radio button in a `Radio` object appears as a separate element in the `elements` array. For example, if a form called `myForm` has a text field and two checkboxes, you can refer to these elements `myForm.elements[0]`, `myForm.elements[1]`, and `myForm.elements[2]`.

Although you can also refer to a form's elements by using the element's name (from the `NAME` attribute), the `elements` array provides a way to refer to `Form` objects programmatically without using their names. For example, if the first object on the `userInfo` form is the `userName` `Text` object, you can evaluate it in either of the following ways:

```
userInfo.userName.value
userInfo.elements[0].value
```

The value of each element in the `elements` array is the full HTML statement for the object.

Examples See the examples for `Frame`.

encoding

A string specifying the MIME encoding of the form.

Property of	`Form`
Implemented in	Navigator 2.0

Description The `encoding` property initially reflects the `ENCTYPE` attribute of the `FORM` tag; however, setting `encoding` overrides the `ENCTYPE` attribute.

Examples The following function returns the value of the `encoding` property of `musicForm`:

```
function getEncoding() {
   return document.musicForm.encoding
}
```

See also `Form.action`, `Form.method`, `Form.target`

length

The number of elements in the form.

Property of	`Form`
Read-only	
Implemented in	Navigator 2.0

Description The `form.length` property tells you how many elements are in the form. You can get the same information using `form.elements.length`.

method

A string specifying how form field input information is sent to the server.

Property of	`Form`
Implemented in	Navigator 2.0

Description The `method` property is a reflection of the `METHOD` attribute of the `FORM` tag. The `method` property should evaluate to either `"get"` or `"post"`.

Examples The following function returns the value of the `musicForm` `method` property:

```
function getMethod() {
   return document.musicForm.method
}
```

See also `Form.action`, `Form.encoding`, `Form.target`

name

A string specifying the name of the form.

Property of	`Form`
Implemented in	Navigator 2.0

Security Navigator 3.0: This property is tainted by default. For information on data tainting, see "Security" on page 55.

Description The `name` property initially reflects the value of the `NAME` attribute. Changing the `name` property overrides this setting.

Examples In the following example, the `valueGetter` function uses a `for` loop to iterate over the array of elements on the `valueTest` form. The `msgWindow` window displays the names of all the elements on the form:

```
newWindow=window.open("http://home.netscape.com")

function valueGetter() {
   var msgWindow=window.open("")
   for (var i = 0; i < newWindow.document.valueTest.elements.length; i++) {
      msgWindow.document.write(newWindow.document.valueTest.elements[i].name + "<BR>")
   }
}
```

target

A string specifying the name of the window that responses go to after a form has been submitted.

Property of	`Form`
Implemented in	Navigator 2.0

Description The `target` property initially reflects the `TARGET` attribute of the `A`, `AREA`, and `FORM` tags; however, setting `target` overrides these attributes.

You can set `target` using a string, if the string represents a window name. The `target` property cannot be assigned the value of a JavaScript expression or variable.

Examples The following example specifies that responses to the `musicInfo` form are displayed in the `msgWindow` window:

```
document.musicInfo.target="msgWindow"
```

See also `Form.action`, `Form.encoding`, `Form.method`

Methods

handleEvent

Invokes the handler for the specified event.

Method of	`Form`

Implemented in Navigator 4.0

Syntax `handleEvent(event)`

Parameters

`event`	The name of an event for which the specified object has an event handler.

Description For information on handling events, see “General Information about Events” on page 481.

reset

Simulates a mouse click on a reset button for the calling form.

Method of	`Form`
Implemented in	Navigator 3.0

Syntax `reset()`

Parameters None

Description The `reset` method restores a form element’s default values. A reset button does not need to be defined for the form.

Examples The following example displays a `Text` object in which the user is to type “CA” or “AZ”. The `Text` object’s `onChange` event handler calls a function that executes the form’s `reset` method if the user provides incorrect input. When the `reset` method executes, defaults are restored and the form’s `onReset` event handler displays a message.

```
<SCRIPT>
function verifyInput(textObject) {
   if (textObject.value == 'CA' || textObject.value == 'AZ') {
      alert('Nice input')
   }
   else { document.myForm.reset() }
}
</SCRIPT>

<FORM NAME="myForm" onReset="alert('Please enter CA or AZ.')">
Enter CA or AZ:
<INPUT TYPE="text" NAME="state" SIZE="2" onChange=verifyInput(this)><P>
</FORM>
```

See also `onReset`, `Reset`

submit

Submits a form.

Method of	`Form`
Implemented in	Navigator 2.0

Syntax `submit()`

Parameters None

Security Navigator 3.0: The `submit` method fails without notice if the form's `action` is a `mailto:`, `news:`, or `snews:` URL. Users can submit forms with such URLs by clicking a submit button, but a confirming dialog will tell them that they are about to give away private or sensitive information.

Navigator 4.0: Submitting a form to a `mailto:` or `news:` URL requires the `UniversalSendMail` privilege. For information on security in Navigator 4.0, see Chapter 7, "JavaScript Security," in the *JavaScript Guide*.

Description The `submit` method submits the specified form. It performs the same action as a submit button.

Use the submit method to send data back to an HTTP server. The `submit` method returns the data using either "get" or "post," as specified in `Form.method`.

Examples The following example submits a form called `musicChoice`:

```
document.musicChoice.submit()
```

If `musicChoice` is the first form created, you also can submit it as follows:

```
document.forms[0].submit()
```

See also the example for `Form`.

See also `Submit`, `onSubmit`

Hidden

A `Text` object that is suppressed from form display on an HTML form. A `Hidden` object is used for passing name/value pairs when a form submits.

Client-side object

Implemented in	Navigator 2.0 Navigator 3.0: added `type` property

Created by The HTML `INPUT` tag, with `"hidden"` as the value of the `TYPE` attribute. For a given form, the JavaScript runtime engine creates appropriate `Hidden` objects and puts these objects in the `elements` array of the corresponding `Form` object. You access a `Hidden` object by indexing this array. You can index the array either by number or, if supplied, by using the value of the `NAME` attribute.

Description A `Hidden` object is a form element and must be defined within a `FORM` tag.

A `Hidden` object cannot be seen or modified by an end user, but you can programmatically change the value of the object by changing its `value` property. You can use `Hidden` objects for client/server communication.

Property Summary

Property	Description
`form`	Specifies the form containing the `Hidden` object.
`name`	Reflects the `NAME` attribute.
`type`	Reflects the `TYPE` attribute.
`value`	Reflects the current value of the `Hidden` object.

Examples The following example uses a `Hidden` object to store the value of the last object the user clicked. The form contains a "Display hidden value" button that the user can click to display the value of the `Hidden` object in an Alert dialog box.

```
<HTML>
<HEAD>
<TITLE>Hidden object example</TITLE>
</HEAD>
<BODY>
<B>Click some of these objects, then click the "Display value" button
<BR>to see the value of the last object clicked.</B>
<FORM NAME="myForm">
```

```
<INPUT TYPE="hidden" NAME="hiddenObject" VALUE="None">
<P>
<INPUT TYPE="button" VALUE="Click me" NAME="button1"
   onClick="document.myForm.hiddenObject.value=this.value">
<P>
<INPUT TYPE="radio" NAME="musicChoice" VALUE="soul-and-r&b"
   onClick="document.myForm.hiddenObject.value=this.value"> Soul and
R&B
<INPUT TYPE="radio" NAME="musicChoice" VALUE="jazz"
   onClick="document.myForm.hiddenObject.value=this.value"> Jazz
<INPUT TYPE="radio" NAME="musicChoice" VALUE="classical"
   onClick="document.myForm.hiddenObject.value=this.value"> Classical
<P>
<SELECT NAME="music_type_single"

onFocus="document.myForm.hiddenObject.value=this.options[this.selectedI
ndex].text">
   <OPTION SELECTED> Red <OPTION> Orange <OPTION> Yellow
</SELECT>
<P><INPUT TYPE="button" VALUE="Display hidden value" NAME="button2"
   onClick="alert('Last object clicked: ' +
document.myForm.hiddenObject.value)">
</FORM>
</BODY>
</HTML>
```

See also `document.cookie`

Properties

form

An object reference specifying the form containing this object.

Method of	`Hidden`
Read-only	
Implemented in	Navigator 2.0

Description Each form element has a `form` property that is a reference to the element's parent form. This property is especially useful in event handlers, where you might need to refer to another element on the current form.

Examples **Example 1.** In the following example, the form `myForm` contains a `Hidden` object and a button. When the user clicks the button, the value of the `Hidden` object is set to the form's name. The button's `onClick` event handler uses `this.form` to refer to the parent form, `myForm`.

```
<FORM NAME="myForm">
Form name:<INPUT TYPE="hidden" NAME="h1" VALUE="Beluga">
<P>
<INPUT NAME="button1" TYPE="button" VALUE="Store Form Name"
   onClick="this.form.h1.value=this.form.name">
</FORM>
```

Example 2. The following example uses an object reference, rather than the `this` keyword, to refer to a form. The code returns a reference to `myForm`, which is a form containing `myHiddenObject`.

```
document.myForm.myHiddenObject.form
```

See also `Form`

name

A string specifying the name of this object.

Method of	`Hidden`
Implemented in	Navigator 2.0

Security Navigator 3.0: This property is tainted by default. For information on data tainting, see "Security" on page 55.

type

For all `Hidden` objects, the value of the `type` property is `"hidden"`. This property specifies the form element's type.

Method of	`Hidden`
Read-only	
Implemented in	Navigator 3.0

Examples The following example writes the value of the `type` property for every element on a form.

```
for (var i = 0; i < document.myForm.elements.length; i++) {
   document.writeln("<BR>type is " + document.myForm.elements[i].type)
}
```

value

A string that reflects the `VALUE` attribute of the object.

Method of	`Hidden`
Implemented in	Navigator 2.0

Security Navigator 3.0: This property is tainted by default. For information on data tainting, see "Security" on page 55.

Examples The following function evaluates the `value` property of a group of buttons and displays it in the `msgWindow` window:

```
function valueGetter() {
   var msgWindow=window.open("")
   msgWindow.document.write("The submit button says " +
      document.valueTest.submitButton.value + "<BR>")
   msgWindow.document.write("The reset button says " +
      document.valueTest.resetButton.value + "<BR>")
   msgWindow.document.write("The hidden field says " +
      document.valueTest.hiddenField.value + "<BR>")
   msgWindow.document.close()
}
```

This example displays the following values:

```
The submit button says Query Submit
The reset button says Reset
The hidden field says pipefish are cute.
```

The previous example assumes the buttons have been defined as follows:

```
<INPUT TYPE="submit" NAME="submitButton">
<INPUT TYPE="reset" NAME="resetButton">
<INPUT TYPE="hidden" NAME="hiddenField" VALUE="pipefish are cute.">
```

Text

A text input field on an HTML form. The user can enter a word, phrase, or series of numbers in a text field.

Client-side object	
Implemented in	Navigator 2.0 Navigator 3.0: added `type` property. Navigator 4.0: added `handleEvent` method.

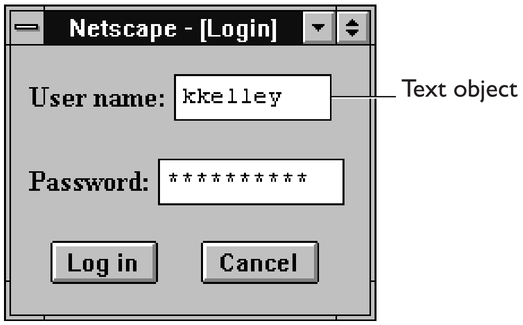
Created by The HTML `INPUT` tag, with `"text"` as the value of the `TYPE` attribute. For a given form, the JavaScript runtime engine creates appropriate `Text` objects and puts these objects in the `elements` array of the corresponding `Form` object. You access a `Text` object by indexing this array. You can index the array either by number or, if supplied, by using the value of the `NAME` attribute.

To define a `Text` object, use standard HTML syntax with the addition of JavaScript event handlers.

Event handlers

- `onBlur`
- `onChange`
- `onFocus`
- `onSelect`

Description A `Text` object on a form looks as follows:



User name: kkelley

Password: **********

A `Text` object is a form element and must be defined within a `FORM` tag.

`Text` objects can be updated (redrawn) dynamically by setting the `value` property (`this.value`).

Property Summary

Property	Descriptiohn
`defaultValue`	Reflects the `VALUE` attribute.
`form`	Specifies the form containing the `Text` object.
`name`	Reflects the `NAME` attribute.
`type`	Reflects the `TYPE` attribute.
`value`	Reflects the current value of the `Text` object's field.

Method Summary

Method	Descriptiohn
`blur`	Removes focus from the object.
`focus`	Gives focus to the object.
`handleEvent`	Invokes the handler for the specified event.
`select`	Selects the input area of the object.

Examples **Example 1.** The following example creates a `Text` object that is 25 characters long. The text field appears immediately to the right of the words "Last name:". The text field is blank when the form loads.

```
<B>Last name:</B> <INPUT TYPE="text" NAME="last_name" VALUE="" SIZE=25>
```

Example 2. The following example creates two `Text` objects on a form. Each object has a default value. The `city` object has an `onFocus` event handler that selects all the text in the field when the user tabs to that field. The `state` object has an `onChange` event handler that forces the value to uppercase.

```
<FORM NAME="form1">
<BR><B>City: </B><INPUT TYPE="text" NAME="city" VALUE="Anchorage"
   SIZE="20" onFocus="this.select()">
<B>State: </B><INPUT TYPE="text" NAME="state" VALUE="AK" SIZE="2"
   onChange="this.value=this.value.toUpperCase()">
</FORM>
```

See also the examples for the `onBlur`, `onChange`, `onFocus`, and `onSelect`.

See also `Text`, `Form`, `Password`, `String`, `Textarea`

Properties

defaultValue

A string indicating the default value of a `Text` object.

Property of	`Text`
Implemented in	Navigator 2.0

Security Navigator 3.0: This property is tainted by default. For information on data tainting, see "Security" on page 55.

Description The initial value of `defaultValue` reflects the value of the `VALUE` attribute. Setting `defaultValue` programmatically overrides the initial setting.

You can set the `defaultValue` property at any time. The display of the related object does not update when you set the `defaultValue` property, only when you set the `value` property.

Examples The following function evaluates the `defaultValue` property of objects on the `surfCity` form and displays the values in the `msgWindow` window:

```
function defaultGetter() {
   msgWindow=window.open("")
   msgWindow.document.write("hidden.defaultValue is " +
      document.surfCity.hiddenObj.defaultValue + "<BR>")
   msgWindow.document.write("password.defaultValue is " +
      document.surfCity.passwordObj.defaultValue + "<BR>")
   msgWindow.document.write("text.defaultValue is " +
      document.surfCity.textObj.defaultValue + "<BR>")
   msgWindow.document.write("textarea.defaultValue is " +
      document.surfCity.textareaObj.defaultValue + "<BR>")
   msgWindow.document.close()
}
```

See also `Text.value`

form

An object reference specifying the form containing this object.

Property of	`Text`
Read-only	

Implemented in Navigator 2.0

Description Each form element has a `form` property that is a reference to the element's parent form. This property is especially useful in event handlers, where you might need to refer to another element on the current form.

Examples **Example 1.** In the following example, the form `myForm` contains a `Text` object and a button. When the user clicks the button, the value of the `Text` object is set to the form's name. The button's `onClick` event handler uses `this.form` to refer to the parent form, `myForm`.

```
<FORM NAME="myForm">
Form name:<INPUT TYPE="text" NAME="text1" VALUE="Beluga">
<P>
<INPUT NAME="button1" TYPE="button" VALUE="Show Form Name"
   onClick="this.form.text1.value=this.form.name">
</FORM>
```

Example 2. The following example shows a form with several elements. When the user clicks `button2`, the function `showElements` displays an alert dialog box containing the names of each element on the form `myForm`.

```
function showElements(theForm) {
   str = "Form Elements of form " + theForm.name + ": \n "
   for (i = 0; i < theForm.length; i++)
      str += theForm.elements[i].name + "\n"
   alert(str)
}
</script>
<FORM NAME="myForm">
Form name:<INPUT TYPE="text" NAME="text1" VALUE="Beluga">
<P>
<INPUT NAME="button1" TYPE="button" VALUE="Show Form Name"
   onClick="this.form.text1.value=this.form.name">
<INPUT NAME="button2" TYPE="button" VALUE="Show Form Elements"
   onClick="showElements(this.form)">
</FORM>
```

The alert dialog box displays the following text:

```
JavaScript Alert:
Form Elements of form myForm:
text1
button1
button2
```

Example 3. The following example uses an object reference, rather than the `this` keyword, to refer to a form. The code returns a reference to `myForm`, which is a form containing `myTextObject`.

```
document.myForm.myTextObject.form
```

See also `Form`

name

A string specifying the name of this object.

Property of	`Text`
Implemented in	Navigator 2.0

Security Navigator 3.0: This property is tainted by default. For information on data tainting, see "Security" on page 55.

Description The `name` property initially reflects the value of the `NAME` attribute. Changing the `name` property overrides this setting. The `name` property is not displayed on-screen; it is used to refer to the objects programmatically.

If multiple objects on the same form have the same `NAME` attribute, an array of the given name is created automatically. Each element in the array represents an individual `Form` object. Elements are indexed in source order starting at 0. For example, if two `Text` elements and a `Textarea` element on the same form have their `NAME` attribute set to `"myField"`, an array with the elements `myField[0]`, `myField[1]`, and `myField[2]` is created. You need to be aware of this situation in your code and know whether `myField` refers to a single element or to an array of elements.

Examples In the following example, the `valueGetter` function uses a `for` loop to iterate over the array of elements on the `valueTest` form. The `msgWindow` window displays the names of all the elements on the form:

```
newWindow=window.open("http://home.netscape.com")

function valueGetter() {
   var msgWindow=window.open("")
   for (var i = 0; i < newWindow.document.valueTest.elements.length; i++) {
      msgWindow.document.write(newWindow.document.valueTest.elements[i].name + "<BR>")
   }
}
```

type

For all `Text` objects, the value of the `type` property is `"text"`. This property specifies the form element's type.

Property of	`Text`
Read-only	
Implemented in	Navigator 3.0

Examples The following example writes the value of the `type` property for every element on a form.

```
for (var i = 0; i < document.form1.elements.length; i++) {
   document.writeln("<BR>type is " + document.form1.elements[i].type)
}
```

value

A string that reflects the `VALUE` attribute of the object.

Property of	`Text`
Implemented in	Navigator 2.0

Security Navigator 3.0: This property is tainted by default. For information on data tainting, see "Security" on page 55.

Description The `value` property is a string that initially reflects the `VALUE` attribute. This string is displayed in the text field. The value of this property changes when a user or a program modifies the field.

You can set the `value` property at any time. The display of the `Text` object updates immediately when you set the `value` property.

Examples The following function evaluates the `value` property of a group of buttons and displays it in the `msgWindow` window:

```
function valueGetter() {
   var msgWindow=window.open("")
   msgWindow.document.write("submitButton.value is " +
      document.valueTest.submitButton.value + "<BR>")
   msgWindow.document.write("resetButton.value is " +
      document.valueTest.resetButton.value + "<BR>")
   msgWindow.document.write("myText.value is " +
      document.valueTest.myText.value + "<BR>")
   msgWindow.document.close()
}
```

This example displays the following:

```
submitButton.value is Query Submit
resetButton.value is Reset
myText.value is Stonefish are dangerous.
```

The previous example assumes the buttons have been defined as follows:

```
<INPUT TYPE="submit" NAME="submitButton">
<INPUT TYPE="reset" NAME="resetButton">
<INPUT TYPE="text" NAME="myText" VALUE="Stonefish are dangerous.">
```

See also `Text.defaultValue`

Methods

blur

Removes focus from the text field.

Method of	`Text`
Implemented in	Navigator 2.0

Syntax `blur()`

Parameters None

Examples The following example removes focus from the text element `userText:`

```
userText.blur()
```

This example assumes that the text element is defined as

```
<INPUT TYPE="text" NAME="userText">
```

See also `Text.focus`, `Text.select`

focus

Navigates to the text field and gives it focus.

Method of	`Text`
Implemented in	Navigator 2.0

Syntax `focus()`

Parameters None

Description Use the `focus` method to navigate to a text field and give it focus. You can then either programmatically enter a value in the field or let the user enter a value. If you use this method without the `select` method, the cursor is positioned at the beginning of the field.

Example See example for `select`.

See also `Text.blur`, `Text.select`

handleEvent

Invokes the handler for the specified event.

Method of	`Text`
Implemented in	Navigator 4.0

Syntax `handleEvent(event)`

Parameters

`event`	The name of an event for which the specified object has an event handler.

select

Selects the input area of the text field.

Method of	`Text`
Implemented in	Navigator 2.0

Syntax `select()`

Parameters None

Description Use the `select` method to highlight the input area of a text field. You can use the `select` method with the `focus` method to highlight a field and position the cursor for a user response. This makes it easy for the user to replace all the text in the field.

Example The following example uses an `onClick` event handler to move the focus to a text field and select that field for changing:

```
<FORM NAME="myForm">
<B>Last name: </B><INPUT TYPE="text" NAME="lastName" SIZE=20 VALUE="Pigman">
<BR><B>First name: </B><INPUT TYPE="text" NAME="firstName" SIZE=20 VALUE="Victoria">
<BR><BR>
<INPUT TYPE="button" VALUE="Change last name"
   onClick="this.form.lastName.select();this.form.lastName.focus();">
</FORM>
```

See also `Text.blur`, `Text.focus`

Textarea

A multiline input field on an HTML form. The user can use a textarea field to enter words, phrases, or numbers.

Client-side object

Implemented in	Navigator 2.0 Navigator 3.0: added `type` property. Navigator 4.0: added `handleEvent` method.

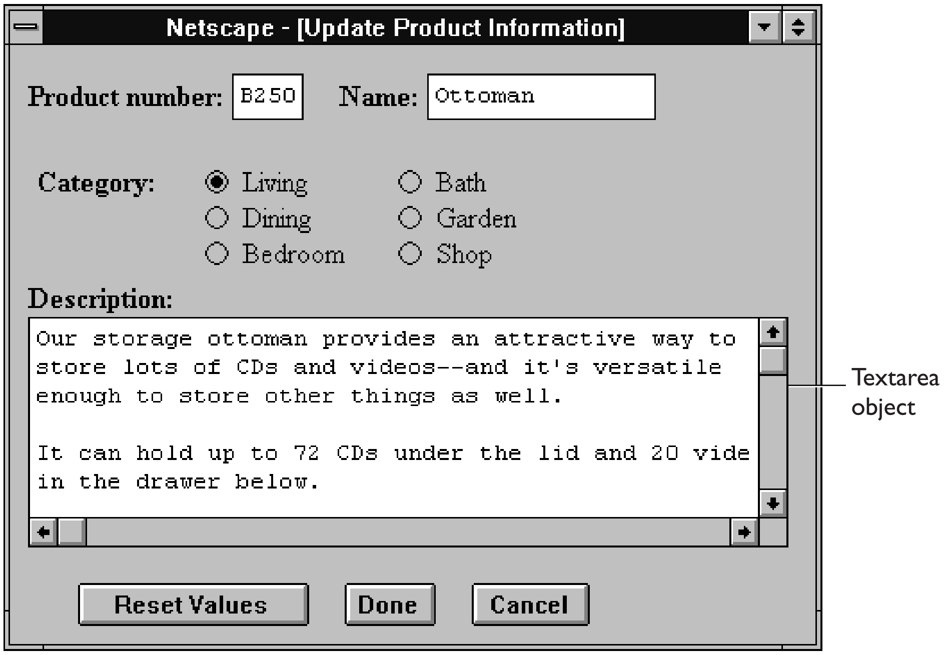
Created by The HTML `TEXTAREA` tag. For a given form, the JavaScript runtime engine creates appropriate `Textarea` objects and puts these objects in the `elements` array of the corresponding `Form` object. You access a `Textarea` object by indexing this array. You can index the array either by number or, if supplied, by using the value of the `NAME` attribute.

To define a text area, use standard HTML syntax with the addition of JavaScript event handlers.

Event handlers
- `onBlur`
- `onChange`

- `onFocus`
- `onKeyDown`
- `onKeyPress`
- `onKeyUp`
- `onSelect`

Description A `Textarea` object on a form looks as follows:



```
Our storage ottoman provides an attractive way to
store lots of CDs and videos--and it's versatile
enough to store other things as well.

It can hold up to 72 CDs under the lid and 20 vide
in the drawer below.
```

A `Textarea` object is a form element and must be defined within a `FORM` tag.

`Textarea` objects can be updated (redrawn) dynamically by setting the `value` property (`this.value`).

To begin a new line in a `Textarea` object, you can use a newline character. Although this character varies from platform to platform (Unix is `\n`, Windows is `\r`, and Macintosh is `\n`), JavaScript checks for all newline characters before setting a string-valued property and translates them as needed for the user's platform. You could also enter a newline character programmatically—one way is to test the `navigator.appVersion` property to determine the current platform, then set the newline character accordingly. See `navigator.appVersion` for an example.

Property Summary

Property	Descriptiohn
`defaultValue`	Reflects the `VALUE` attribute.
`form`	Specifies the form containing the `Textarea` object.
`name`	Reflects the `NAME` attribute.
`type`	Specifies that the object is a `Textarea` object.
`value`	Reflects the current value of the `Textarea` object.

Method Summary

Method	Descriptiohn
`blur`	Removes focus from the object.
`focus`	Gives focus to the object.
`handleEvent`	Invokes the handler for the specified event.
`select`	Selects the input area of the object.

Examples

Example 1. The following example creates a `Textarea` object that is six rows long and 55 columns wide. The textarea field appears immediately below the word "Description:". When the form loads, the `Textarea` object contains several lines of data, including one blank line.

```
<B>Description:</B>
<BR><TEXTAREA NAME="item_description" ROWS=6 COLS=55>
Our storage ottoman provides an attractive way to
store lots of CDs and videos--and it's versatile
enough to store other things as well.

It can hold up to 72 CDs under the lid and 20 videos
in the drawer below.
</TEXTAREA>
```

Example 2. The following example creates a string variable containing newline characters for different platforms. When the user clicks the button, the `Textarea` object is populated with the value from the string variable. The result is three lines of text in the `Textarea` object.

```
<SCRIPT>
myString="This is line one.\nThis is line two.\rThis is line three."
</SCRIPT>
```

```
<FORM NAME="form1">
<INPUT TYPE="button" Value="Populate the textarea"
onClick="document.form1.textarea1.value=myString">
   <P>
<TEXTAREA NAME="textarea1" ROWS=6 COLS=55></TEXTAREA>
```

See also `Form`, `Password`, `String`, `Text`

Properties

defaultValue

A string indicating the default value of a `Textarea` object.

Property of	`Textarea`
Implemented in	Navigator 2.0

Security Navigator 3.0: This property is tainted by default. For information on data tainting, see "Security" on page 55.

Description The initial value of `defaultValue` reflects the value specified between the `TEXTAREA` start and end tags. Setting `defaultValue` programmatically overrides the initial setting.

You can set the `defaultValue` property at any time. The display of the related object does not update when you set the `defaultValue` property, only when you set the `value` property.

Examples The following function evaluates the `defaultValue` property of objects on the `surfCity` form and displays the values in the `msgWindow` window:

```
function defaultGetter() {
   msgWindow=window.open("")
   msgWindow.document.write("hidden.defaultValue is " +
      document.surfCity.hiddenObj.defaultValue + "<BR>")
   msgWindow.document.write("password.defaultValue is " +
      document.surfCity.passwordObj.defaultValue + "<BR>")
   msgWindow.document.write("text.defaultValue is " +
      document.surfCity.textObj.defaultValue + "<BR>")
   msgWindow.document.write("textarea.defaultValue is " +
      document.surfCity.textareaObj.defaultValue + "<BR>")
   msgWindow.document.close()
}
```

See also `Textarea.value`

form

An object reference specifying the form containing this object.

Property of	`Textarea`
Read-only	
Implemented in	Navigator 2.0

Description Each form element has a `form` property that is a reference to the element's parent form. This property is especially useful in event handlers, where you might need to refer to another element on the current form.

Examples **Example 1.** The following example shows a form with several elements. When the user clicks `button2`, the function `showElements` displays an alert dialog box containing the names of each element on the form `myForm`.

```
function showElements(theForm) {
   str = "Form Elements of form " + theForm.name + ": \n "
   for (i = 0; i < theForm.length; i++)
      str += theForm.elements[i].name + "\n"
   alert(str)
}
</script>
<FORM NAME="myForm">
Form name:<INPUT TYPE="textarea" NAME="text1" VALUE="Beluga">
<P>
<INPUT NAME="button1" TYPE="button" VALUE="Show Form Name"
   onClick="this.form.text1.value=this.form.name">
<INPUT NAME="button2" TYPE="button" VALUE="Show Form Elements"
   onClick="showElements(this.form)">
</FORM>
```

The alert dialog box displays the following text:

```
JavaScript Alert:
Form Elements of form myForm:
text1
button1
button2
```

Example 2. The following example uses an object reference, rather than the `this` keyword, to refer to a form. The code returns a reference to `myForm`, which is a form containing `myTextareaObject`.

```
document.myForm.myTextareaObject.form
```

See also `Form`

name

A string specifying the name of this object.

Property of	`Textarea`
Implemented in	Navigator 2.0

Security Navigator 3.0: This property is tainted by default. For information on data tainting, see "Security" on page 55.

Description The `name` property initially reflects the value of the `NAME` attribute. Changing the `name` property overrides this setting. The `name` property is not displayed on-screen; it is used to refer to the objects programmatically.

If multiple objects on the same form have the same `NAME` attribute, an array of the given name is created automatically. Each element in the array represents an individual `Form` object. Elements are indexed in source order starting at 0. For example, if two `Text` elements and a `Textarea` element on the same form have their `NAME` attribute set to `"myField"`, an array with the elements `myField[0]`, `myField[1]`, and `myField[2]` is created. You need to be aware of this situation in your code and know whether `myField` refers to a single element or to an array of elements.

Examples In the following example, the `valueGetter` function uses a `for` loop to iterate over the array of elements on the `valueTest` form. The `msgWindow` window displays the names of all the elements on the form:

```
newWindow=window.open("http://home.netscape.com")

function valueGetter() {
   var msgWindow=window.open("")
   for (var i = 0; i < newWindow.document.valueTest.elements.length; i++) {
      msgWindow.document.write(newWindow.document.valueTest.elements[i].name + "<BR>")
   }
}
```

type

For all `Textarea` objects, the value of the `type` property is `"textarea"`. This property specifies the form element's type.

Property of	`Textarea`

Read-only

Implemented in Navigator 3.0

Examples The following example writes the value of the `type` property for every element on a form.

```
for (var i = 0; i < document.form1.elements.length; i++) {
   document.writeln("<BR>type is " + document.form1.elements[i].type)
}
```

value

A string that initially reflects the `VALUE` attribute.

Property of	`Textarea`
Implemented in	Navigator 2.0

Security Navigator 3.0: This property is tainted by default. For information on data tainting, see "Security" on page 55.

Description This string is displayed in the textarea field. The value of this property changes when a user or a program modifies the field.

You can set the `value` property at any time. The display of the `Textarea` object updates immediately when you set the `value` property.

Examples The following function evaluates the `value` property of a group of buttons and displays it in the `msgWindow` window:

```
function valueGetter() {
   var msgWindow=window.open("")
   msgWindow.document.write("submitButton.value is " +
      document.valueTest.submitButton.value + "<BR>")
   msgWindow.document.write("resetButton.value is " +
      document.valueTest.resetButton.value + "<BR>")
   msgWindow.document.write("blurb.value is " +
      document.valueTest.blurb.value + "<BR>")
   msgWindow.document.close()
}
```

This example displays the following:

```
submitButton.value is Query Submit
resetButton.value is Reset
blurb.value is Tropical waters contain all sorts of cool fish,
such as the harlequin ghost pipefish, dragonet, and cuttlefish.
A cuttlefish looks much like a football wearing a tutu and a mop.
```

The previous example assumes the buttons have been defined as follows:

```
<INPUT TYPE="submit" NAME="submitButton">
<INPUT TYPE="reset" NAME="resetButton">
<TEXTAREA NAME="blurb" rows=3 cols=60>
Tropical waters contain all sorts of cool fish,
such as the harlequin ghost pipefish, dragonet, and cuttlefish.
A cuttlefish looks much like a football wearing a tutu and a mop.
</TEXTAREA>
```

See also `Textarea.defaultValue`

Methods

blur

Removes focus from the object.

Method of	`Textarea`
Implemented in	Navigator 2.0

Syntax `blur()`

Parameters None

Examples The following example removes focus from the textarea element `userText`:

```
userText.blur()
```

This example assumes that the textarea is defined as

```
<TEXTAREA NAME="userText">
Initial text for the text area.
</TEXTAREA>
```

See also `Textarea.focus`, `Textarea.select`

focus

Navigates to the textarea field and gives it focus.

Method of	`Textarea`
Implemented in	Navigator 2.0

Syntax `focus()`

Parameters None

Description Use the `focus` method to navigate to the textarea field and give it focus. You can then either programmatically enter a value in the field or let the user enter a value. If you use this method without the `select` method, the cursor is positioned at the beginning of the field.

See also `Textarea.blur`, `Textarea.select`

Example See example for `Textarea.select`.

handleEvent

Invokes the handler for the specified event.

Method of	`Textarea`
Implemented in	Navigator 4.0

Syntax `handleEvent(event)`

Parameters

`event`	The name of an event for which the object has an event handler.

Description For information on handling events, see "General Information about Events" on page 481.

select

Selects the input area of the object.

Method of	`Textarea`

Implemented in Navigator 2.0

Syntax `select()`

Parameters None

Description Use the `select` method to highlight the input area of a textarea field. You can use the `select` method with the `focus` method to highlight the field and position the cursor for a user response. This makes it easy for the user to replace all the text in the field.

Example The following example uses an `onClick` event handler to move the focus to a textarea field and select that field for changing:

```
<FORM NAME="myForm">
<B>Last name: </B><INPUT TYPE="text" NAME="lastName" SIZE=20 VALUE="Pigman">
<BR><B>First name: </B><INPUT TYPE="text" NAME="firstName" SIZE=20 VALUE="Victoria">
<BR><B>Description:</B>
<BR><TEXTAREA NAME="desc" ROWS=3 COLS=40>An avid scuba diver.</TEXTAREA>
<BR><BR>
<INPUT TYPE="button" VALUE="Change description"
   onClick="this.form.desc.select();this.form.desc.focus();">
</FORM>
```

See also `Textarea.blur`, `Textarea.focus`

Password

A text field on an HTML form that conceals its value by displaying asterisks (*). When the user enters text into the field, asterisks (*) hide entries from view.

Client-side object

Implemented in Navigator 2.0
Navigator 3.0: added `type` property; added onBlur and onFocus event handlers
Navigator 4.0: added `handleEvent` method.

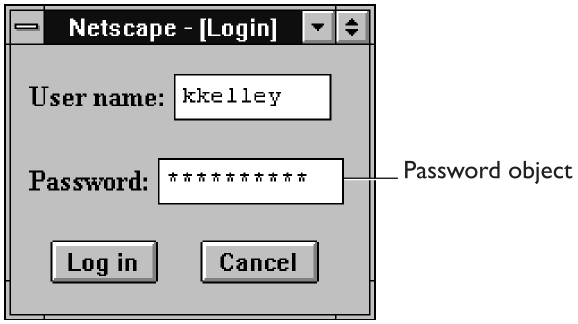
Created by The HTML `INPUT` tag, with `"password"` as the value of the `TYPE` attribute. For a given form, the JavaScript runtime engine creates appropriate `Password` objects and puts these objects in the `elements` array of the corresponding `Form` object. You access a `Password` object by indexing this array. You can index the array either by number or, if supplied, by using the value of the `NAME` attribute.

Event handlers

- `onBlur`
- `onFocus`

Description

A `Password` object on a form looks as follows:



A `Password` object is a form element and must be defined within a `FORM` tag.

Security

Navigator 3.0: If a user interactively modifies the value in a password field, you cannot evaluate it accurately unless data tainting is enabled. See the *JavaScript Guide*.

Property Summary

Property	Descriptiohn
`defaultValue`	Reflects the `VALUE` attribute.
`form`	Specifies the form containing the `Password` object.
`name`	Reflects the `NAME` attribute.
`type`	Reflects the `TYPE` attribute.
`value`	Reflects the current value of the `Password` object's field.

Method Summary

Method	Descriptiohn
`blur`	Removes focus from the object.
`focus`	Gives focus to the object.
`handleEvent`	Invokes the handler for the specified event.
`select`	Selects the input area of the object.

Examples The following example creates a `Password` object with no default value:

```
<B>Password:</B>
<INPUT TYPE="password" NAME="password" VALUE="" SIZE=25>
```

See also `Form`, `Text`

Properties

defaultValue

A string indicating the default value of a `Password` object.

Property of	`Password`
Implemented in	Navigator 2.0

Security Navigator 3.0: This property is tainted by default. For information on data tainting, see "Security" on page 55.

Description The initial value of `defaultValue` is null (for security reasons), regardless of the value of the `VALUE` attribute.

Setting `defaultValue` programmatically overrides the initial setting. If you programmatically set `defaultValue` for the `Password` object and then evaluate it, JavaScript returns the current value.

You can set the `defaultValue` property at any time. The display of the related object does not update when you set the `defaultValue` property, only when you set the `value` property.

See also `Password.value`

form

An object reference specifying the form containing this object.

Property of	`Password`
Read-only	
Implemented in	Navigator 2.0

Description Each form element has a `form` property that is a reference to the element's parent form. This property is especially useful in event handlers, where you might need to refer to another element on the current form.

name

A string specifying the name of this object.

Property of	`Password`
Implemented in	Navigator 2.0

Security Navigator 3.0: This property is tainted by default. For information on data tainting, see "Security" on page 55.

Description The `name` property initially reflects the value of the `NAME` attribute. Changing the `name` property overrides this setting. The `name` property is not displayed on-screen; it is used to refer to the objects programmatically.

If multiple objects on the same form have the same `NAME` attribute, an array of the given name is created automatically. Each element in the array represents an individual `Form` object. Elements are indexed in source order starting at 0. For example, if two `Text` elements and a `Password` element on the same form have their `NAME` attribute set to `"myField"`, an array with the elements `myField[0]`, `myField[1]`, and `myField[2]` is created. You need to be aware of this situation in your code and know whether `myField` refers to a single element or to an array of elements.

Examples In the following example, the `valueGetter` function uses a `for` loop to iterate over the array of elements on the `valueTest` form. The `msgWindow` window displays the names of all the elements on the form:

```
newWindow=window.open("http://home.netscape.com")

function valueGetter() {
   var msgWindow=window.open("")
   for (var i = 0; i < newWindow.document.valueTest.elements.length; i++) {
      msgWindow.document.write(newWindow.document.valueTest.elements[i].name + "<BR>")
   }
}
```

type

For all `Password` objects, the value of the `type` property is `"password"`. This property specifies the form element's type.

Property of	`Password`
Read-only	
Implemented in	Navigator 3.0

Examples The following example writes the value of the `type` property for every element on a form.

```
for (var i = 0; i < document.form1.elements.length; i++) {
   document.writeln("<BR>type is " + document.form1.elements[i].type)
}
```

value

A string that initially reflects the `VALUE` attribute.

Property of	`Password`
Implemented in	Navigator 2.0

Security Navigator 3.0: This property is tainted by default. For information on data tainting, see "Security" on page 55. If you programmatically set the `value` property and then evaluate it, JavaScript returns the current value. If a user interactively modifies the value in the password field, you cannot evaluate it accurately unless data tainting is enabled. See the *JavaScript Guide*.

Description This string is represented by asterisks in the `Password` object field. The value of this property changes when a user or a program modifies the field, but the value is always displayed as asterisks.

See also `Password.defaultValue`

Methods

blur

Removes focus from the object.

Method of	`Password`
Implemented in	Navigator 2.0

Syntax `blur()`

Parameters None

Examples The following example removes focus from the password element `userPass:`

```
userPass.blur()
```

This example assumes that the password is defined as

```
<INPUT TYPE="password" NAME="userPass">
```

See also `Password.focus`, `Password.select`

focus

Gives focus to the password object.

Method of	`Password`
Implemented in	Navigator 2.0

Syntax `focus()`

Parameters None

Description Use the `focus` method to navigate to the password field and give it focus. You can then either programmatically enter a value in the field or let the user enter a value.

Examples In the following example, the `checkPassword` function confirms that a user has entered a valid password. If the password is not valid, the `focus` method returns focus to the `Password` object and the `select` method highlights it so the user can reenter the password.

```
function checkPassword(userPass) {
   if (badPassword) {
      alert("Please enter your password again.")
      userPass.focus()
      userPass.select()
   }
}
```

This example assumes that the `Password` object is defined as

```
<INPUT TYPE="password" NAME="userPass">
```

See also `Password.blur`, `Password.select`

handleEvent

Invokes the handler for the specified event.

Method of	`Password`
Implemented in	Navigator 4.0

Syntax `handleEvent(event)`

Parameters

`event`	The name of an event for which the object has an event handler.

Description For information on handling events, see "General Information about Events" on page 481.

select

Selects the input area of the password field.

Method of	`Password`
Implemented in	Navigator 2.0

Syntax `select()`

Parameters None

Description Use the `select` method to highlight the input area of the password field. You can use the `select` method with the `focus` method to highlight a field and position the cursor for a user response.

Examples In the following example, the `checkPassword` function confirms that a user has entered a valid password. If the password is not valid, the `select` method highlights the password field and the `focus` method returns focus to it so the user can reenter the password.

```
function checkPassword(userPass) {
   if (badPassword) {
      alert("Please enter your password again.")
      userPass.focus()
      userPass.select()
   }
}
```

This example assumes that the password is defined as

```
<INPUT TYPE="password" NAME="userPass">
```

See also `Password.blur`, `Password.focus`

FileUpload

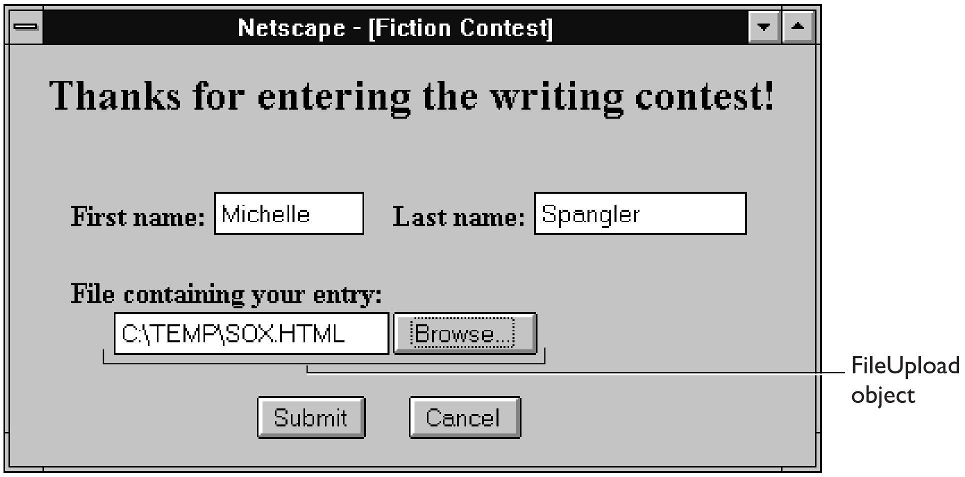
A file upload element on an HTML form. A file upload element lets the user supply a file as input.

Client-side object	
Implemented in	Navigator 2.0 Navigator 3.0: added `type` property Navigator 4.0: added `handleEvent` method.

Created by The HTML `INPUT` tag, with `"file"` as the value of the `TYPE` attribute. For a given form, the JavaScript runtime engine creates appropriate `FileUpload` objects and puts these objects in the `elements` array of the corresponding `Form` object. You access a `FileUpload` object by indexing this array. You can index the array either by number or, if supplied, by using the value of the `NAME` attribute.

Event handlers
- `onBlur`
- `onChange`
- `onFocus`

Description A `FileUpload` object on a form looks as follows:


Netscape - [Fiction Contest]

Thanks for entering the writing contest!

First name: Michelle Last name: Spangler

File containing your entry:

C:\TEMP\SOX.HTML

A `FileUpload` object is a form element and must be defined within a `FORM` tag.

Property Summary

Property	Descriptiohn
`form`	Specifies the form containing the `FileUpload` object.
`name`	Reflects the `NAME` attribute.
`type`	Reflects the `TYPE` attribute.
`value`	Reflects the current value of the file upload element's field; this corresponds to the name of the file to upload.

Method Summary

Method	Descriptiohn
`blur`	Removes focus from the object.
`focus`	Gives focus to the object.
`handleEvent`	Invokes the handler for the specified event.
`select`	Selects the input area of the file upload field.

Examples The following example places a `FileUpload` object on a form and provides two buttons that let the user display current values of the `name` and `value` properties.

```
<FORM NAME="form1">
File to send: <INPUT TYPE="file" NAME="myUploadObject">
<P>Get properties<BR>
<INPUT TYPE="button" VALUE="name"
   onClick="alert('name: ' + document.form1.myUploadObject.name)">
<INPUT TYPE="button" VALUE="value"
   onClick="alert('value: ' +
document.form1.myUploadObject.value)"><BR>
</FORM>
```

See also `Text`

Properties

form

An object reference specifying the form containing the object.

Property of	`FileUpload`
Read-only	
Implemented in	Navigator 2.0

Description Each form element has a `form` property that is a reference to the element's parent form. This property is especially useful in event handlers, where you might need to refer to another element on the current form.

name

A string specifying the name of this object.

Property of	`FileUpload`
Read-only	
Implemented in	Navigator 2.0

Security Navigator 3.0: This property is tainted by default. For information on data tainting, see "Security" on page 55.

[illegible] [illegible] input area of a file upload field. You

Button

A push button on an HTML form.

Client-side object

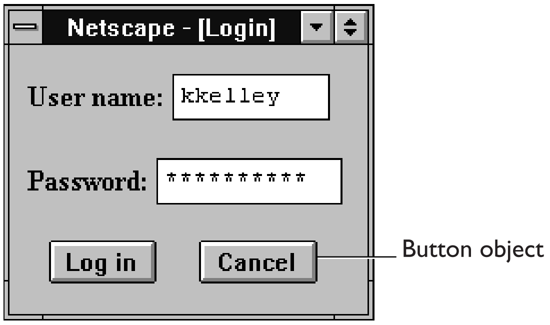
Implemented in	Navigator 2.0 Navigator 3.0: added `type` property; added `onBlur` and `onFocus` event handlers; added `blur` and `focus` methods. Navigator 4.0: added `handleEvent` method.

Created by The HTML `INPUT` tag, with `"button"` as the value of the `TYPE` attribute. For a given form, the JavaScript runtime engine creates appropriate `Button` objects and puts these objects in the `elements` array of the corresponding `Form` object. You access a `Button` object by indexing this array. You can index the array either by number or, if supplied, by using the value of the `NAME` attribute.

Event handlers

- `onBlur`
- `onClick`
- `onFocus`
- `onMouseDown`
- `onMouseUp`

Description A `Button` object on a form looks as follows:



A `Button` object is a form element and must be defined within a `FORM` tag.

The `Button` object is a custom button that you can use to perform an action you define. The button executes the script specified by its `onClick` event handler.

Property Summary

Property	Descriptiohn
`form`	Specifies the form containing the `Button` object.
`name`	Reflects the `NAME` attribute.
`type`	Reflects the `TYPE` attribute.
`value`	Reflects the `VALUE` attribute.

Method Summary

Method	Descriptiohn
`blur`	Removes focus from the button.
`click`	Simulates a mouse-click on the button.
`focus`	Gives focus to the button.
`handleEvent`	Invokes the handler for the specified event.

Examples The following example creates a button named `calcButton`. The text "Calculate" is displayed on the face of the button. When the button is clicked, the function `calcFunction` is called.

```
<INPUT TYPE="button" VALUE="Calculate" NAME="calcButton"
   onClick="calcFunction(this.form)">
```

See also `Form`, `Reset`, `Submit`

Properties

form

An object reference specifying the form containing the button.

Property of	`Button`
Read-only	
Implemented in	Navigator 2.0

Description Each form element has a `form` property that is a reference to the element's parent form. This property is especially useful in event handlers, where you might need to refer to another element on the current form.

Examples **Example 1.** In the following example, the form `myForm` contains a `Text` object and a button. When the user clicks the button, the value of the `Text` object is set to the form's name. The button's `onClick` event handler uses `this.form` to refer to the parent form, `myForm`.

```
<FORM NAME="myForm">
Form name:<INPUT TYPE="text" NAME="text1" VALUE="Beluga">
<P>
<INPUT NAME="button1" TYPE="button" VALUE="Show Form Name"
   onClick="this.form.text1.value=this.form.name">
</FORM>
```

Example 2. The following example shows a form with several elements. When the user clicks `button2`, the function `showElements` displays an alert dialog box containing the names of each element on the form `myForm`.

```
function showElements(theForm) {
   str = "Form Elements of form " + theForm.name + ": \n "
   for (i = 0; i < theForm.length; i++)
      str += theForm.elements[i].name + "\n"
   alert(str)
}
</script>
<FORM NAME="myForm">
Form name:<INPUT TYPE="text" NAME="text1" VALUE="Beluga">
<P>
<INPUT NAME="button1" TYPE="button" VALUE="Show Form Name"
   onClick="this.form.text1.value=this.form.name">
<INPUT NAME="button2" TYPE="button" VALUE="Show Form Elements"
   onClick="showElements(this.form)">
</FORM>
```

The alert dialog box displays the following text:

```
JavaScript Alert:
Form Elements of form myForm:
text1
button1
button2
```

Example 3. The following example uses an object reference, rather than the `this` keyword, to refer to a form. The code returns a reference to `myForm`, which is a form containing `myButton`.

```
document.myForm.myButton.form
```

See also `Form`

name

A string specifying the button's name.

Property of	`Button`
Implemented in	Navigator 2.0

Security Navigator 3.0: This property is tainted by default. For information on data tainting, see "Security" on page 55.

Description The `name` property initially reflects the value of the `NAME` attribute. Changing the `name` property overrides this setting.

Do not confuse the `name` property with the label displayed on a button. The `value` property specifies the label for the button. The `name` property is not displayed on the screen; it is used to refer programmatically to the object.

If multiple objects on the same form have the same `NAME` attribute, an array of the given name is created automatically. Each element in the array represents an individual `Form` object. Elements are indexed in source order starting at 0. For example, if two `Text` elements and a `Button` element on the same form have their `NAME` attribute set to `"myField"`, an array with the elements `myField[0]`, `myField[1]`, and `myField[2]` is created. You need to be aware of this situation in your code and know whether `myField` refers to a single element or to an array of elements.

Examples In the following example, the `valueGetter` function uses a `for` loop to iterate over the array of elements on the `valueTest` form. The `msgWindow` window displays the names of all the elements on the form:

```
newWindow=window.open("http://home.netscape.com")

function valueGetter() {
   var msgWindow=window.open("")
   for (var i = 0; i < newWindow.document.valueTest.elements.length; i++) {
      msgWindow.document.write(newWindow.document.valueTest.elements[i].name + "<BR>")
   }
}
```

In the following example, the first statement creates a window called `netscapeWin`. The second statement displays the value `"netscapeHomePage"` in the Alert dialog box, because `"netscapeHomePage"` is the value of the `windowName` argument of `netscapeWin`.

```
netscapeWin=window.open("http://home.netscape.com","netscapeHomePage")
```

```
alert(netscapeWin.name)
```

See also `Button.value`

type

For all `Button` objects, the value of the `type` property is `"button"`. This property specifies the form element's type.

Property of	`Button`
Read-only	
Implemented in	Navigator 3.0

Examples The following example writes the value of the `type` property for every element on a form.

```
for (var i = 0; i < document.form1.elements.length; i++) {
   document.writeln("<BR>type is " + document.form1.elements[i].type)
}
```

value

A string that reflects the button's `VALUE` attribute.

Property of	`Button`
Read-only on Mac and UNIX; modifiable on Windows	
Implemented in	Navigator 2.0

Security Navigator 3.0: This property is tainted by default. For information on data tainting, see "Security" on page 55.

Description This string is displayed on the face of the button.

The `value` property is read-only for Macintosh and UNIX systems. On Windows, you can change this property.

When a `VALUE` attribute is not specified in HTML, the `value` property is an empty string.

Do not confuse the `value` property with the `name` property. The `name` property is not displayed on the screen; it is used to refer programmatically to the objects.

Examples The following function evaluates the `value` property of a group of buttons and displays it in the `msgWindow` window:

```
function valueGetter() {
   var msgWindow=window.open("")
   msgWindow.document.write("submitButton.value is " +
      document.valueTest.submitButton.value + "<BR>")
   msgWindow.document.write("resetButton.value is " +
      document.valueTest.resetButton.value + "<BR>")
   msgWindow.document.write("helpButton.value is " +
      document.valueTest.helpButton.value + "<BR>")
   msgWindow.document.close()
}
```

This example displays the following values:

```
Query Submit
Reset
Help
```

The previous example assumes the buttons have been defined as follows:

```
<INPUT TYPE="submit" NAME="submitButton">
<INPUT TYPE="reset" NAME="resetButton">
<INPUT TYPE="button" NAME="helpButton" VALUE="Help">
```

See also `Button.name`

Methods

blur

Removes focus from the button.

Method of	`Button`
Implemented in	Navigator 2.0

Syntax `blur()`

Parameters None

Examples The following example removes focus from the button element `userButton:`

```
userButton.blur()
```

This example assumes that the button is defined as

```
<INPUT TYPE="button" NAME="userButton">
```

See also `Button.focus`

click

Simulates a mouse-click on the button, but does not trigger the button's `onClick` event handler.

Method of	`Button`
Implemented in	Navigator 2.0

Syntax `click()`

Parameters None.

Security Navigator 4.0: Submitting a form to a `mailto:` or `news:` URL requires the `UniversalSendMail` privilege. For information on security in Navigator 4.0, see Chapter 7, "JavaScript Security," in the *JavaScript Guide*.

focus

Navigates to the button and gives it focus.

Method of	`Button`
Implemented in	Navigator 2.0

Syntax `focus()`

Parameters None.

See also `Button.blur`

handleEvent

Invokes the handler for the specified event.

Method of	`Button`
Implemented in	Navigator 4.0

Syntax `handleEvent(event)`

Parameters

`event`	The name of an event for which the object has an event handler.

Description For information on handling events, see "General Information about Events" on page 481.

Submit

A submit button on an HTML form. A submit button causes a form to be submitted.

Client-side object	
Implemented in	Navigator 2.0 Navigator 3.0: added `type` property; added onBlur and onFocus event handlers; added `blur` and `focus` methods Navigator 4.0: added `handleEvent` method.

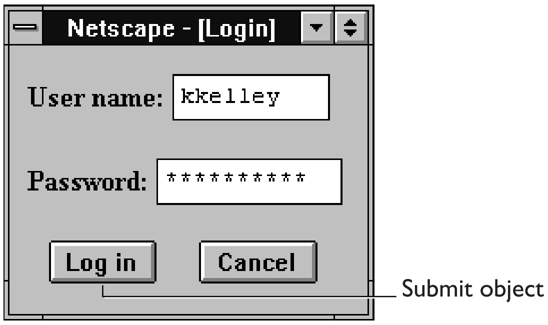
Created by The HTML `INPUT` tag, with `"submit"` as the value of the `TYPE` attribute. For a given form, the JavaScript runtime engine creates an appropriate `Submit` object and puts it in the `elements` array of the corresponding `Form` object. You access a `Submit` object by indexing this array. You can index the array either by number or, if supplied, by using the value of the `NAME` attribute.

Event handlers

- `onBlur`
- `onClick`
- `onFocus`

Security Navigator 4.0: Submitting a form to a `mailto:` or `news:` URL requires the `UniversalSendMail` privilege. For information on security in Navigator 4.0, see Chapter 7, "JavaScript Security," in the *JavaScript Guide*.

Description A `Submit` object on a form looks as follows:



A `Submit` object is a form element and must be defined within a `FORM` tag.

Clicking a submit button submits a form to the URL specified by the form's `action` property. This action always loads a new page into the client; it may be the same as the current page, if the action so specifies or is not specified.

The submit button's `onClick` event handler cannot prevent a form from being submitted; instead, use the form's `onSubmit` event handler or use the `submit` method instead of a `Submit` object. See the examples for the `Form` object.

Property Summary

Property	Descriptiohn
`form`	Specifies the form containing the `Submit` object.
`name`	Reflects the `NAME` attribute.
`type`	Reflects the `TYPE` attribute.
`value`	Reflects the `VALUE` attribute.

Method Summary

Method	Descriptiohn
`blur`	Removes focus from the submit button.
`click`	Simulates a mouse-click on the submit button.
`focus`	Gives focus to the submit button.
`handleEvent`	Invokes the handler for the specified event.

Examples The following example creates a `Submit` object called `submitButton`. The text "Done" is displayed on the face of the button.

```
<INPUT TYPE="submit" NAME="submitButton" VALUE="Done">
```

See also the examples for the `Form`.

See also `Button`, `Form`, `Reset`, `Form.submit`, `onSubmit`

Properties

form

An object reference specifying the form containing the submit button.

Property of	`Submit`
Read-only	
Implemented in	Navigator 2.0

Description Each form element has a `form` property that is a reference to the element's parent form. This property is especially useful in event handlers, where you might need to refer to another element on the current form.

Examples The following example shows a form with several elements. When the user clicks `button2`, the function `showElements` displays an alert dialog box containing the names of each element on the form `myForm`.

```
<SCRIPT>
function showElements(theForm) {
   str = "Form Elements of form " + theForm.name + ": \n "
   for (i = 0; i < theForm.length; i++)
      str += theForm.elements[i].name + "\n"
   alert(str)
}
</SCRIPT>
<FORM NAME="myForm">
Form name:<INPUT TYPE="text" NAME="text1" VALUE="Beluga">
<P>
<INPUT NAME="button1" TYPE="button" VALUE="Show Form Name"
   onClick="this.form.text1.value=this.form.name">
<INPUT NAME="button2" TYPE="submit" VALUE="Show Form Elements"
   onClick="showElements(this.form)">
</FORM>
```

The alert dialog box displays the following text:

```
Form Elements of form myForm:
text1
button1
button2
```

See also `Form`

name

A string specifying the submit button's name.

Property of	`Submit`
Implemented in	Navigator 2.0

Security Navigator 3.0: This property is tainted by default. For information on data tainting, see "Security" on page 55.

Description The `name` property initially reflects the value of the `NAME` attribute. Changing the `name` property overrides this setting.

Do not confuse the `name` property with the label displayed on the `Submit` button. The `value` property specifies the label for this button. The `name` property is not displayed on the screen; it is used to refer programmatically to the button.

If multiple objects on the same form have the same `NAME` attribute, an array of the given name is created automatically. Each element in the array represents an individual `Form` object. Elements are indexed in source order starting at 0. For example, if two `Text` elements and a `Submit` element on the same form have their `NAME` attribute set to `"myField"`, an array with the elements `myField[0]`, `myField[1]`, and `myField[2]` is created. You need to be aware of this situation in your code and know whether `myField` refers to a single element or to an array of elements.

Examples In the following example, the `valueGetter` function uses a `for` loop to iterate over the array of elements on the `valueTest` form. The `msgWindow` window displays the names of all the elements on the form:

```
newWindow=window.open("http://home.netscape.com")

function valueGetter() {
   var msgWindow=window.open("")
   for (var i = 0; i < newWindow.document.valueTest.elements.length; i++) {
      msgWindow.document.write(newWindow.document.valueTest.elements[i].name + "<BR>")
   }
}
```

See also `Submit.value`

type

For all `Submit` objects, the value of the `type` property is `"submit"`. This property specifies the form element's type.

Property of	`Submit`
Read-only	
Implemented in	Navigator 3.0

Examples The following example writes the value of the `type` property for every element on a form.

```
for (var i = 0; i < document.form1.elements.length; i++) {
   document.writeln("<BR>type is " + document.form1.elements[i].type)
}
```

value

A string that reflects the submit button's `VALUE` attribute.

Property of	`Submit`
Read-only	
Implemented in	Navigator 2.0

Security Navigator 3.0: This property is tainted by default. For information on data tainting, see "Security" on page 55.

Description When a `VALUE` attribute is specified in HTML, the `value` property is that string and is displayed on the face of the button. When a `VALUE` attribute is not specified in HTML, the `value` property for the button is the string "Submit Query."

Do not confuse the `value` property with the `name` property. The `name` property is not displayed on the screen; it is used to refer programmatically to the button.

Examples The following function evaluates the `value` property of a group of buttons and displays it in the `msgWindow` window:

```
function valueGetter() {
   var msgWindow=window.open("")
   msgWindow.document.write("submitButton.value is " +
      document.valueTest.submitButton.value + "<BR>")
   msgWindow.document.write("resetButton.value is " +
      document.valueTest.resetButton.value + "<BR>")
   msgWindow.document.write("helpButton.value is " +
      document.valueTest.helpButton.value + "<BR>")
   msgWindow.document.close()
}
```

This example displays the following values:

```
Query Submit
Reset
Help
```

The previous example assumes the buttons have been defined as follows:

```
<INPUT TYPE="submit" NAME="submitButton">
<INPUT TYPE="reset" NAME="resetButton">
<INPUT TYPE="button" NAME="helpButton" VALUE="Help">
```

See also `Submit.name`

Methods

blur

Removes focus from the submit button.

Method of	`Submit`
Implemented in	Navigator 2.0

Syntax `blur()`

Parameters None

See also `Submit.focus`

click

Simulates a mouse-click on the submit button, but does *not* trigger an object's `onClick` event handler.

Method of	`Submit`
Implemented in	Navigator 2.0

Syntax `click()`

Parameters None

focus

Navigates to the submit button and gives it focus.

Method of	`Submit`
Implemented in	Navigator 2.0

Syntax `focus()`

Parameters None

See also `Submit.blur`

handleEvent

Invokes the handler for the specified event.

Method of	`Submit`
Implemented in	Navigator 4.0

Syntax `handleEvent(event)`

Parameters

`event`	The name of an event for which the specified object has an event handler.

Description For information on handling events, see "General Information about Events" on page 481.

Reset

A reset button on an HTML form. A reset button resets all elements in a form to their defaults.

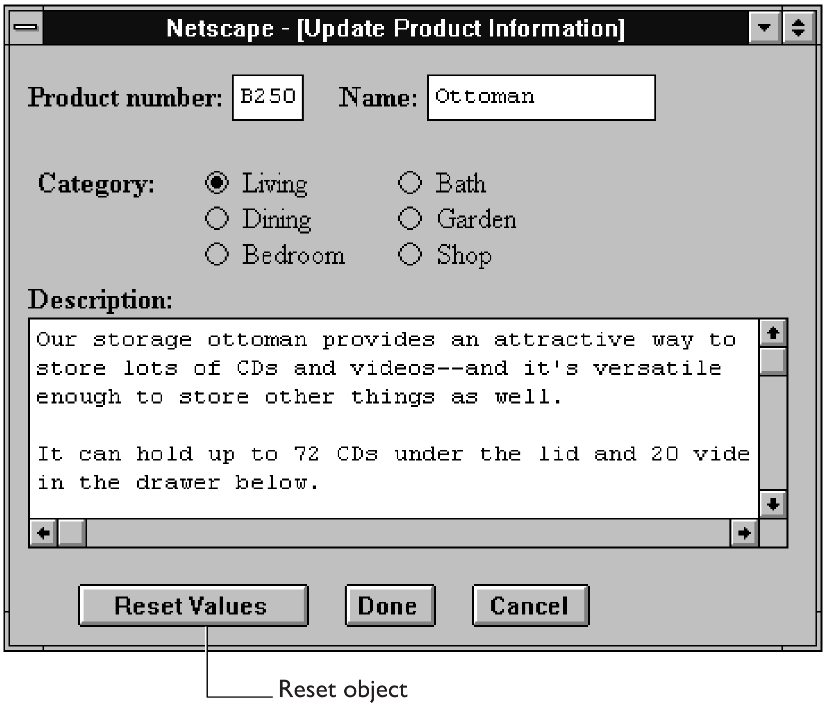
Client-side object	
Implemented in	Navigator 2.0 Navigator 3.0: added `type` property; added `onBlur` and `onFocus` event handlers; added `blur` and `focus` methods Navigator 4.0: added `handleEvent` method.

Created by The HTML `INPUT` tag, with `"reset"` as the value of the `TYPE` attribute. For a given form, the JavaScript runtime engine creates an appropriate `Reset` object and puts it in the `elements` array of the corresponding `Form` object. You access a `Reset` object by indexing this array. You can index the array either by number or, if supplied, by using the value of the `NAME` attribute.

Event handlers

- `onBlur`
- `onClick`
- `onFocus`

Description A `Reset` object on a form looks as follows:



Our storage ottoman provides an attractive way to
store lots of CDs and videos--and it's versatile
enough to store other things as well.

It can hold up to 72 CDs under the lid and 20 vide
in the drawer below.

A `Reset` object is a form element and must be defined within a `FORM` tag.

The reset button's `onClick` event handler cannot prevent a form from being reset; once the button is clicked, the reset cannot be canceled.

Property Summary

Property	Descriptiohn
`form`	Specifies the form containing the `Reset` object.
`name`	Reflects the `NAME` attribute.
`type`	Reflects the `TYPE` attribute.
`value`	Reflects the `VALUE` attribute.

Method Summary

Method	Descriptiohn
`blur`	Removes focus from the reset button.
`click`	Simulates a mouse-click on the reset button.
`focus`	Gives focus to the reset button.
`handleEvent`	Invokes the handler for the specified event.

Examples **Example 1.** The following example displays a `Text` object with the default value "CA" and a reset button with the text "Clear Form" displayed on its face. If the user types a state abbreviation in the `Text` object and then clicks the Clear Form button, the original value of "CA" is restored.

```
<B>State: </B><INPUT TYPE="text" NAME="state" VALUE="CA" SIZE="2">
<P><INPUT TYPE="reset" VALUE="Clear Form">
```

Example 2. The following example displays two `Text` objects, a `Select` object, and three radio buttons; all of these objects have default values. The form also has a reset button with the text "Defaults" on its face. If the user changes the value of any of the objects and then clicks the Defaults button, the original values are restored.

```
<HTML>
<HEAD>
<TITLE>Reset object example</TITLE>
</HEAD>
<BODY>
<FORM NAME="form1">
<BR><B>City: </B><INPUT TYPE="text" NAME="city" VALUE="Santa Cruz" SIZE="20">
<B>State: </B><INPUT TYPE="text" NAME="state" VALUE="CA" SIZE="2">
<P><SELECT NAME="colorChoice">
   <OPTION SELECTED> Blue
   <OPTION> Yellow
   <OPTION> Green
   <OPTION> Red
</SELECT>
<P><INPUT TYPE="radio" NAME="musicChoice" VALUE="soul-and-r&b"
   CHECKED> Soul and R&B
<BR><INPUT TYPE="radio" NAME="musicChoice" VALUE="jazz">
   Jazz
<BR><INPUT TYPE="radio" NAME="musicChoice" VALUE="classical">
   Classical
<P><INPUT TYPE="reset" VALUE="Defaults" NAME="reset1">
</FORM>
</BODY>
```

```
</HTML>
```

See also `Button`, `Form`, `onReset`, `Form.reset`, `Submit`

Properties

form

An object reference specifying the form containing the reset button.

Property of	`Reset`
Read-only	
Implemented in	Navigator 2.0

Description Each form element has a `form` property that is a reference to the element's parent form. This property is especially useful in event handlers, where you might need to refer to another element on the current form.

See also `Form`

name

A string specifying the name of the reset button.

Property of	`Reset`
Implemented in	Navigator 2.0

Security Navigator 3.0: This property is tainted by default. For information on data tainting, see "Security" on page 55.

Description The value of the `name` property initially reflects the value of the `NAME` attribute. Changing the `name` property overrides this setting.

Do not confuse the `name` property with the label displayed on the reset button. The `value` property specifies the label for this button. The `name` property is not displayed on the screen; it is used to refer programmatically to the button.

If multiple objects on the same form have the same `NAME` attribute, an array of the given name is created automatically. Each element in the array represents an individual `Form` object. Elements are indexed in source order starting at 0.

For example, if two `Text` elements and a `Reset` element on the same form have their `NAME` attribute set to `"myField"`, an array with the elements `myField[0]`, `myField[1]`, and `myField[2]` is created. You need to be aware of this situation in your code and know whether `myField` refers to a single element or to an array of elements.

Examples In the following example, the `valueGetter` function uses a `for` loop to iterate over the array of elements on the `valueTest` form. The `msgWindow` window displays the names of all the elements on the form:

```
newWindow=window.open("http://home.netscape.com")

function valueGetter() {
   var msgWindow=window.open("")
   for (var i = 0; i < newWindow.document.valueTest.elements.length; i++) {
      msgWindow.document.write(newWindow.document.valueTest.elements[i].name + "<BR>")
   }
}
```

See also `Reset.value`

type

For all `Reset` objects, the value of the `type` property is `"reset"`. This property specifies the form element's type.

Property of	`Reset`
Read-only	
Implemented in	Navigator 3.0

Examples The following example writes the value of the `type` property for every element on a form.

```
for (var i = 0; i < document.form1.elements.length; i++) {
   document.writeln("<BR>type is " + document.form1.elements[i].type)
}
```

value

A string that reflects the reset button's `VALUE` attribute.

Property of	`Reset`
Read-only	
Implemented in	Navigator 2.0

ParSSyntax `Reset()blur`

Radio

An individual radio button in a set of radio buttons on an HTML form. The user can use a set of radio buttons to choose one item from a list.

Client-side object

Implemented in Navigator 2.0
Navigator 3.0: added `type` property; added `blur` and `focus` methods.
Navigator 4.0: added `handleEvent` method.

Created by The HTML `INPUT` tag, with `"radio"` as the value of the `TYPE` attribute. All the radio buttons in a single group must have the same value for the `NAME` attribute. This allows them to be accessed as a single group.

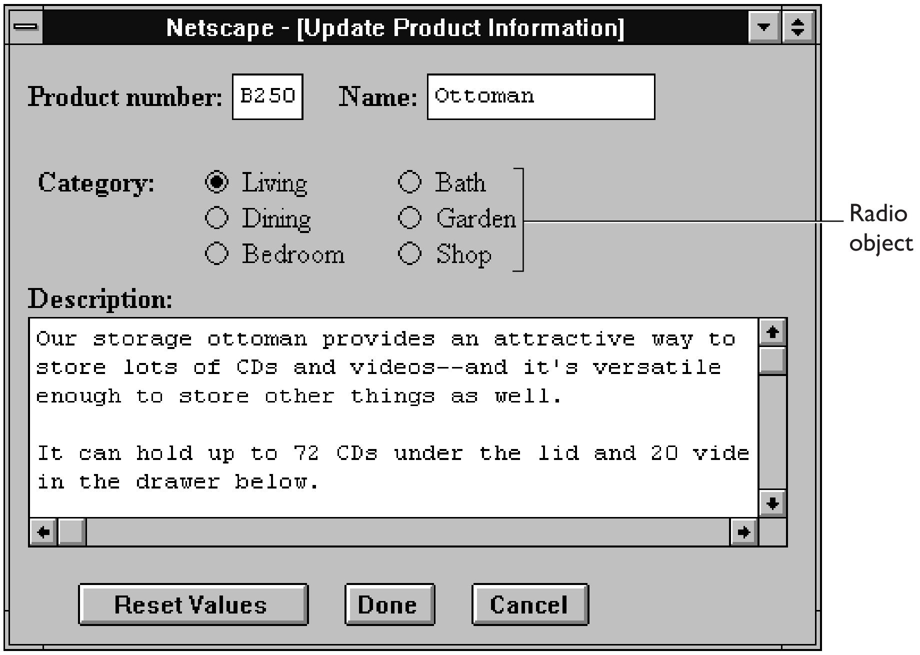
For a given form, the JavaScript runtime engine creates an individual `Radio` object for each radio button in that form. It puts in a single array all the `Radio` objects that have the same value for the `NAME` attribute. It puts that array in the `elements` array of the corresponding `Form` object. If a single form has multiple sets of radio buttons, the `elements` array has multiple `Radio` objects.

You access a set of buttons by accessing the `Form.elements` array (either by number or by using the value of the `NAME` attribute). To access the individual radio buttons in that set, you use the returned object array. For example, if your document has a form called `emp` with a set of radio buttons whose `NAME` attribute is `"dept"`, you would access the individual buttons as `document.emp.dept[0]`, `document.emp.dept[1]`, and so on.

Event handlers

- `onBlur`
- `onClick`
- `onFocus`

Description A `Radio` object on a form looks as follows:



Product number: B250 Name: Ottoman

Description:

Our storage ottoman provides an attractive way to store lots of CDs and videos--and it's versatile enough to store other things as well.

It can hold up to 72 CDs under the lid and 20 vide in the drawer below.

A `Radio` object is a form element and must be defined within a `FORM` tag.

Property Summary

Property	Descriptiohn
`checked`	Lets you programmatically select a radio button (property of the individual button).
`defaultChecked`	Reflects the `CHECKED` attribute (property of the individual button).
`form`	Specifies the form containing the `Radio` object (property of the array of buttons).
`name`	Reflects the `NAME` attribute (property of the array of buttons).
`type`	Reflects the `TYPE` attribute (property of the array of buttons).
`value`	Reflects the `VALUE` attribute (property of the array of buttons).

Method Summary

Method	Descriptiohn
`blur`	Removes focus from the radio button.
`click`	Simulates a mouse-click on the radio button.
`focus`	Gives focus to the radio button.
`handleEvent`	Invokes the handler for the specified event.

Examples **Example 1.** The following example defines a radio button group to choose among three music catalogs. Each radio button is given the same name, `NAME="musicChoice"`, forming a group of buttons for which only one choice can be selected. The example also defines a text field that defaults to what was chosen via the radio buttons but that allows the user to type a nonstandard catalog name as well. The `onClick` event handler sets the catalog name input field when the user clicks a radio button.

```
<INPUT TYPE="text" NAME="catalog" SIZE="20">
<INPUT TYPE="radio" NAME="musicChoice" VALUE="soul-and-r&b"
   onClick="musicForm.catalog.value = 'soul-and-r&b'"> Soul and R&B
<INPUT TYPE="radio" NAME="musicChoice" VALUE="jazz"
   onClick="musicForm.catalog.value = 'jazz'"> Jazz
<INPUT TYPE="radio" NAME="musicChoice" VALUE="classical"
   onClick="musicForm.catalog.value = 'classical'"> Classical
```

Example 2. The following example contains a form with three text boxes and three radio buttons. The radio buttons let the user choose whether the text fields are converted to uppercase or lowercase, or not converted at all. Each text field has an `onChange` event handler that converts the field value depending on which radio button is checked. The radio buttons for uppercase and lowercase have `onClick` event handlers that convert all fields when the user clicks the radio button.

```
<HTML>
<HEAD>
<TITLE>Radio object example</TITLE>
</HEAD>
<SCRIPT>
function convertField(field) {
   if (document.form1.conversion[0].checked) {
      field.value = field.value.toUpperCase()}
   else {
   if (document.form1.conversion[1].checked) {
      field.value = field.value.toLowerCase()}
   }
}
function convertAllFields(caseChange) {
   if (caseChange=="upper") {
   document.form1.lastName.value = document.form1.lastName.value.toUpperCase()
   document.form1.firstName.value = document.form1.firstName.value.toUpperCase()
   document.form1.cityName.value = document.form1.cityName.value.toUpperCase()}
   else {
   document.form1.lastName.value = document.form1.lastName.value.toLowerCase()
   document.form1.firstName.value = document.form1.firstName.value.toLowerCase()
   document.form1.cityName.value = document.form1.cityName.value.toLowerCase()
   }
}
</SCRIPT>
<BODY>
<FORM NAME="form1">
<B>Last name:</B>
<INPUT TYPE="text" NAME="lastName" SIZE=20 onChange="convertField(this)">
<BR><B>First name:</B>
<INPUT TYPE="text" NAME="firstName" SIZE=20 onChange="convertField(this)">
<BR><B>City:</B>
<INPUT TYPE="text" NAME="cityName" SIZE=20 onChange="convertField(this)">
<P><B>Convert values to:</B>
<BR><INPUT TYPE="radio" NAME="conversion" VALUE="upper"
   onClick="if (this.checked) {convertAllFields('upper')}"> Upper case
<BR><INPUT TYPE="radio" NAME="conversion" VALUE="lower"
   onClick="if (this.checked) {convertAllFields('lower')}"> Lower case
<BR><INPUT TYPE="radio" NAME="conversion" VALUE="noChange"> No conversion
</FORM>
</BODY>
</HTML>
```

See also the example for `Link`.

See also `Checkbox`, `Form`, `Select`

Properties

checked

A Boolean value specifying the selection state of a radio button.

Property of	`Radio`
Implemented in	Navigator 2.0

Security Navigator 3.0: This property is tainted by default. For information on data tainting, see "Security" on page 55.

Description If a radio button is selected, the value of its `checked` property is true; otherwise, it is false. You can set the `checked` property at any time. The display of the radio button updates immediately when you set the `checked` property.

At any given time, only one button in a set of radio buttons can be checked. When you set the `checked` property for one radio button in a group to true, that property for all other buttons in the group becomes false.

Examples The following example examines an array of radio buttons called `musicType` on the `musicForm` form to determine which button is selected. The `VALUE` attribute of the selected button is assigned to the `checkedButton` variable.

```
function stateChecker() {
   var checkedButton = ""
   for (var i in document.musicForm.musicType) {
      if (document.musicForm.musicType[i].checked=="1") {
         checkedButton=document.musicForm.musicType[i].value
      }
   }
}
```

See also `Radio.defaultChecked`

defaultChecked

A Boolean value indicating the default selection state of a radio button.

Property of	`Radio`
Implemented in	Navigator 2.0

Security Navigator 3.0: This property is tainted by default. For information on data tainting, see "Security" on page 55.

Description If a radio button is selected by default, the value of the `defaultChecked` property is true; otherwise, it is false. `defaultChecked` initially reflects whether the `CHECKED` attribute is used within an `INPUT` tag; however, setting `defaultChecked` overrides the `CHECKED` attribute.

Unlike for the `checked` property, changing the value of `defaultChecked` for one button in a radio group does not change its value for the other buttons in the group.

You can set the `defaultChecked` property at any time. The display of the radio button does not update when you set the `defaultChecked` property, only when you set the `checked` property.

Examples The following example resets an array of radio buttons called `musicType` on the `musicForm` form to the default selection state:

```
function radioResetter() {
   var i=""
   for (i in document.musicForm.musicType) {
      if (document.musicForm.musicType[i].defaultChecked==true) {
         document.musicForm.musicType[i].checked=true
      }
   }
}
```

See also `Radio.checked`

form

An object reference specifying the form containing the radio button.

Property of	`Radio`
Read-only	
Implemented in	Navigator 2.0

Description Each form element has a `form` property that is a reference to the element's parent form. This property is especially useful in event handlers, where you might need to refer to another element on the current form.

name

A string specifying the name of the set of radio buttons with which this button is associated.

Property of	`Radio`
Implemented in	Navigator 2.0

Security Navigator 3.0: This property is tainted by default. For information on data tainting, see "Security" on page 55.

Description The `name` property initially reflects the value of the `NAME` attribute. Changing the `name` property overrides this setting.

All radio buttons that have the same value for their `name` property are in the same group and are treated together. If you change the `name` of a single radio button, you change which group of buttons it belongs to.

Do not confuse the name property with the label displayed on a Button. The `value` property specifies the label for the button. The `name` property is not displayed onscreen; it is used to refer programmatically to the button.

Examples In the following example, the `valueGetter` function uses a `for` loop to iterate over the array of elements on the `valueTest` form. The `msgWindow` window displays the names of all the elements on the form:

```
newWindow=window.open("http://home.netscape.com")

function valueGetter() {
   var msgWindow=window.open("")
   for (var i = 0; i < newWindow.document.valueTest.elements.length; i++) {
      msgWindow.document.write(newWindow.document.valueTest.elements[i].name + "<BR>")
   }
}
```

type

For all `Radio` objects, the value of the `type` property is `"radio"`. This property specifies the form element's type.

Property of	`Radio`
Read-only	
Implemented in	Navigator 3.0

Examples The following example writes the value of the `type` property for every element on a form.

```
for (var i = 0; i < document.form1.elements.length; i++) {
   document.writeln("<BR>type is " + document.form1.elements[i].type)
}
```

value

A string that reflects the `VALUE` attribute of the radio button.

Property of	`Radio`
Read-only	
Implemented in	Navigator 2.0

Security Navigator 3.0: This property is tainted by default. For information on data tainting, see "Security" on page 55.

Description When a `VALUE` attribute is specified in HTML, the `value` property is a string that reflects it. When a `VALUE` attribute is not specified in HTML, the `value` property is a string that evaluates to `"on"`. The `value` property is not displayed on the screen but is returned to the server if the radio button or checkbox is selected.

Do not confuse the property with the selection state of the radio button or the text that is displayed next to the button. The `checked` property determines the selection state of the object, and the `defaultChecked` property determines the default selection state. The text that is displayed is specified following the `INPUT` tag.

Examples The following function evaluates the `value` property of a group of radio buttons and displays it in the `msgWindow` window:

```
function valueGetter() {
   var msgWindow=window.open("")
   for (var i = 0; i < document.valueTest.radioObj.length; i++) {
       msgWindow.document.write
          ("The value of radioObj[" + i + "] is " +
          document.valueTest.radioObj[i].value +"<BR>")
   }
   msgWindow.document.close()
}
```

This example displays the following values:

```
on
on
on
on
```

The previous example assumes the buttons have been defined as follows:

```
<BR><INPUT TYPE="radio" NAME="radioObj">R&B
<BR><INPUT TYPE="radio" NAME="radioObj" CHECKED>Soul
<BR><INPUT TYPE="radio" NAME="radioObj">Rock and Roll
<BR><INPUT TYPE="radio" NAME="radioObj">Blues
```

See also `Radio.checked`, `Radio.defaultChecked`

Methods

blur

Removes focus from the radio button.

Method of	`Radio`
Implemented in	Navigator 2.0

Syntax `blur()`

Parameters None

See also `Radio.focus`

click

Simulates a mouse-click on the radio button, but does *not* trigger the button's `onClick` event handler.

Method of	`Radio`
Implemented in	Navigator 2.0

Syntax `click()`

Parameters None

Examples The following example toggles the selection status of the first radio button in the `musicType` `Radio` object on the `musicForm` form:

```
document.musicForm.musicType[0].click()
```

The following example toggles the selection status of the `newAge` checkbox on the `musicForm` form:

```
document.musicForm.newAge.click()
```

focus

Gives focus to the radio button.

Method of	`Radio`

Implemented in	Navigator 2.0

Syntax `focus()`

Parameters None

Description Use the `focus` method to navigate to the radio button and give it focus. The user can then easily toggle that button.

See also `Radio.blur`

handleEvent

Invokes the handler for the specified event.

Method of	`Radio`
Implemented in	Navigator 4.0

Syntax `handleEvent(event)`

Parameters

`event`	The name of an event for which the specified object has an event handler.

Checkbox

A checkbox on an HTML form. A checkbox is a toggle switch that lets the user set a value on or off.

Client-side object

Implemented in	Navigator 2.0 Navigator 3.0: added `type` property; added `onBlur` and `onFocus` event handlers; added `blur` and `focus` methods. Navigator 4.0: added `handleEvent` method.

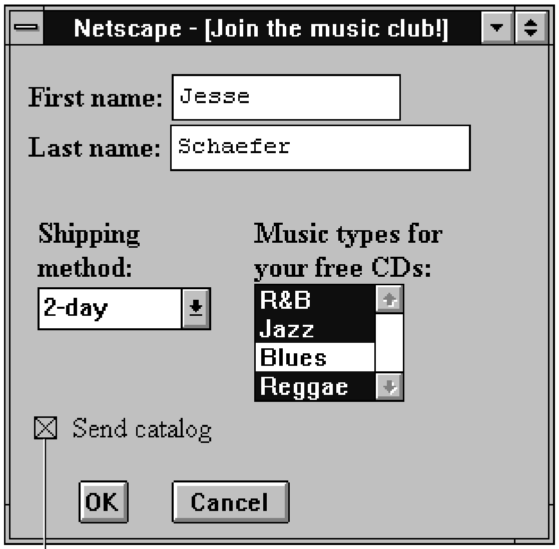
Created by The HTML `INPUT` tag, with `"checkbox"` as the value of the `TYPE` attribute. For a given form, the JavaScript runtime engine creates appropriate `Checkbox` objects and puts these objects in the `elements` array of the corresponding `Form` object. You access a `Checkbox` object by indexing this array. You can index the array either by number or, if supplied, by using the value of the `NAME` attribute.

Event handlers

- `onBlur`
- `onClick`
- `onFocus`

Description

A `Checkbox` object on a form looks as follows:



Checkbox object

A `Checkbox` object is a form element and must be defined within a `FORM` tag.

Use the `checked` property to specify whether the checkbox is currently checked. Use the `defaultChecked` property to specify whether the checkbox is checked when the form is loaded or reset.

Property Summary

Property	Descriptiohn
`checked`	Boolean property that reflects the current state of the checkbox.
`defaultChecked`	Boolean property that reflects the `CHECKED` attribute.
`form`	Specifies the form containing the `Checkbox` object.
`name`	Reflects the `NAME` attribute.
`type`	Reflects the `TYPE` attribute.
`value`	Reflects the `TYPE` attribute.

Method Summary

Method	Descriptiohn
`blur`	Removes focus from the checkbox.
`click`	Simulates a mouse-click on the checkbox.
`focus`	Gives focus to the checkbox.
`handleEvent`	Invokes the handler for the specified event.

Examples **Example 1.** The following example displays a group of four checkboxes that all appear checked by default:

```
<B>Specify your music preferences (check all that apply):</B>
<BR><INPUT TYPE="checkbox" NAME="musicpref_rnb" CHECKED> R&B
<BR><INPUT TYPE="checkbox" NAME="musicpref_jazz" CHECKED> Jazz
<BR><INPUT TYPE="checkbox" NAME="musicpref_blues" CHECKED> Blues
<BR><INPUT TYPE="checkbox" NAME="musicpref_newage" CHECKED> New Age
```

Example 2. The following example contains a form with three text boxes and one checkbox. The user can use the checkbox to choose whether the text fields are converted to uppercase. Each text field has an `onChange` event handler that converts the field value to uppercase if the checkbox is checked. The checkbox has an `onClick` event handler that converts all fields to uppercase when the user checks the checkbox.

```
<HTML>
<HEAD>
<TITLE>Checkbox object example</TITLE>
</HEAD>
<SCRIPT>
function convertField(field) {
   if (document.form1.convertUpper.checked) {
      field.value = field.value.toUpperCase()}
}
function convertAllFields() {
   document.form1.lastName.value = document.form1.lastName.value.toUpperCase()
   document.form1.firstName.value = document.form1.firstName.value.toUpperCase()
   document.form1.cityName.value = document.form1.cityName.value.toUpperCase()
}
</SCRIPT>
<BODY>
<FORM NAME="form1">
<B>Last name:</B>
<INPUT TYPE="text" NAME="lastName" SIZE=20 onChange="convertField(this)">
<BR><B>First name:</B>
<INPUT TYPE="text" NAME="firstName" SIZE=20 onChange="convertField(this)">
```

```
<BR><B>City:</B>
<INPUT TYPE="text" NAME="cityName" SIZE=20 onChange="convertField(this)">
<P><INPUT TYPE="checkBox" NAME="convertUpper"
   onClick="if (this.checked) {convertAllFields()}"
   > Convert fields to upper case
</FORM>
</BODY>
</HTML>
```

See also `Form`, `Radio`

Properties

checked

A Boolean value specifying the selection state of the checkbox.

Property of	`Checkbox`
Implemented in	Navigator 2.0

Security Navigator 3.0: This property is tainted by default. For information on data tainting, see "Security" on page 55.

Description If a checkbox button is selected, the value of its `checked` property is true; otherwise, it is false.

You can set the `checked` property at any time. The display of the checkbox button updates immediately when you set the `checked` property.

See also `Checkbox.defaultChecked`

defaultChecked

A Boolean value indicating the default selection state of a checkbox button.

Property of	`Checkbox`
Implemented in	Navigator 2.0

Security Navigator 3.0: This property is tainted by default. For information on data tainting, see "Security" on page 55.

Description If a checkbox is selected by default, the value of the `defaultChecked` property is true; otherwise, it is false. `defaultChecked` initially reflects whether the `CHECKED` attribute is used within an `INPUT` tag; however, setting `defaultChecked` overrides the `CHECKED` attribute.

You can set the `defaultChecked` property at any time. The display of the checkbox does not update when you set the `defaultChecked` property, only when you set the `checked` property.

See also `Checkbox.checked`

form

An object reference specifying the form containing the checkbox.

Property of	`Checkbox`
Read-only	
Implemented in	Navigator 2.0

Description Each form element has a `form` property that is a reference to the element's parent form. This property is especially useful in event handlers, where you might need to refer to another element on the current form.

See also `Form`

name

A string specifying the checkbox's name.

Property of	`Checkbox`
Implemented in	Navigator 2.0

Security Navigator 3.0: This property is tainted by default. For information on data tainting, see "Security" on page 55.

Description If multiple objects on the same form have the same `NAME` attribute, an array of the given name is created automatically. Each element in the array represents an individual `Form` object. Elements are indexed in source order starting at 0. For example, if two `Text` elements and a `Button` element on the same form have their `NAME` attribute set to `"myField"`, an array with the elements `myField[0]`, `myField[1]`, and `myField[2]` is created. You need to be aware of this situation in your code and know whether `myField` refers to a single element or to an array of elements.

Examples In the following example, the `valueGetter` function uses a `for` loop to iterate over the array of elements on the `valueTest` form. The `msgWindow` window displays the names of all the elements on the form:

```
newWindow=window.open("http://home.netscape.com")

function valueGetter() {
   var msgWindow=window.open("")
   for (var i = 0; i < newWindow.document.valueTest.elements.length; i++) {
      msgWindow.document.write(newWindow.document.valueTest.elements[i].name + "<BR>")
   }
}
```

type

For all `Checkbox` objects, the value of the `type` property is `"checkbox"`. This property specifies the form element's type.

Property of	`Checkbox`
Read-only	
Implemented in	Navigator 3.0

Examples The following example writes the value of the `type` property for every element on a form.

```
for (var i = 0; i < document.form1.elements.length; i++) {
   document.writeln("<BR>type is " + document.form1.elements[i].type)
}
```

value

A string that reflects the `VALUE` attribute of the checkbox.

Property of	`Checkbox`
Implemented in	Navigator 2.0

Security Navigator 3.0: This property is tainted by default. For information on data tainting, see "Security" on page 55.

See also `Checkbox.checked`, `Checkbox.defaultChecked`

Methods

blur

Removes focus from the checkbox.

Method of	`Checkbox`
Implemented in	Navigator 2.0

Syntax `blur()`

Parameters None

See also `Checkbox.focus`

click

Simulates a mouse-click on the checkbox, but does not trigger its `onClick` event handler. The method checks the checkbox and sets toggles its value.

Method of	`Checkbox`
Implemented in	Navigator 2.0

Syntax `click()`

Parameters None.

Examples The following example toggles the selection status of the `newAge` checkbox on the `musicForm` form:

```
document.musicForm.newAge.click()
```

focus

Gives focus to the checkbox.

Method of	`Checkbox`
Implemented in	Navigator 2.0

Syntax `focus()`

Parameters None

Description Use the `focus` method to navigate to a the checkbox and give it focus. The user can then toggle the state of the checkbox.

See also `Checkbox.blur`

handleEvent

Invokes the handler for the specified event.

Method of	`Checkbox`
Implemented in	Navigator 4.0

Syntax `handleEvent(event)`

Parameters

`event`	The name of an event for which the specified object has an event handler.

Select

A selection list on an HTML form. The user can choose one or more items from a selection list, depending on how the list was created.

Client-side object	
Implemented in	Navigator 2.0 Navigator 3.0: added `type` property; added the ability to add and delete options. Navigator 4.0: added `handleEvent` method.

Created by The HTML `SELECT` tag. For a given form, the JavaScript runtime engine creates appropriate `Select` objects for each selection list and puts these objects in the `elements` array of the corresponding `Form` object. You access a `Select` object by indexing this array. You can index the array either by number or, if supplied, by using the value of the `NAME` attribute.

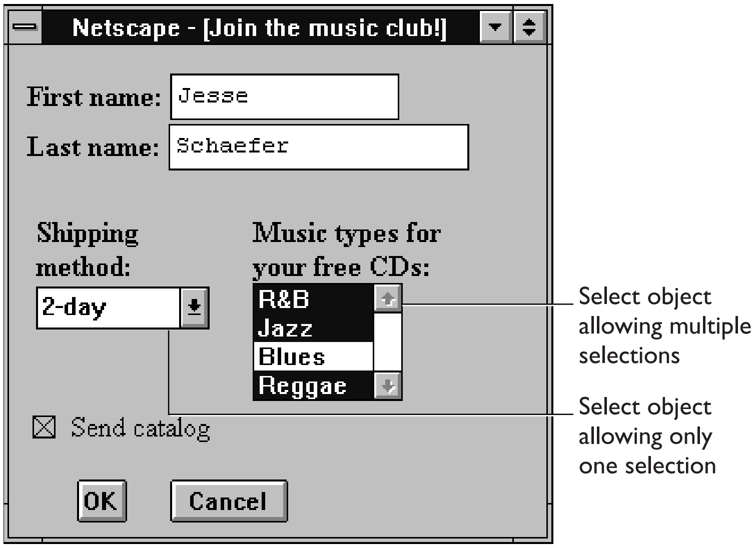
The runtime engine also creates `Option` objects for each `OPTION` tag inside the `SELECT` tag.

Event handlers

- `onBlur`
- `onChange`
- `onFocus`

Description

The following figure shows a form containing two selection lists. The user can choose one item from the list on the left and can choose multiple items from the list on the right:


Netscape - [Join the music club!]
First name:
Jesse
Last name:
Schaefer
Shipping method:
2-day
Music types for your free CDs:
R&B
Jazz
Blues
Reggae
Send catalog
OK
Cancel
Select object allowing multiple selections
Select object allowing only one selection

A `Select` object is a form element and must be defined within a `FORM` tag.

Property Summary

Property	Descriptiohn
`form`	Specifies the form containing the selection list.
`length`	Reflects the number of options in the selection list.
`name`	Reflects the `NAME` attribute.
`options`	Reflects the `OPTION` tags.
`selectedIndex`	Reflects the index of the selected option (or the first selected option, if multiple options are selected).
`type`	Specifies that the object is represents a selection list and whether it can have one or more selected options.

Method Summary

Method	Descriptiohn
`blur`	Removes focus from the selection list.
`focus`	Gives focus to the selection list.
`handleEvent`	Invokes the handler for the specified event.

Examples **Example 1.** The following example displays two selection lists. In the first list, the user can select only one item; in the second list, the user can select multiple items.

```
Choose the music type for your free CD:
<SELECT NAME="music_type_single">
   <OPTION SELECTED> R&B
   <OPTION> Jazz
   <OPTION> Blues
   <OPTION> New Age
</SELECT>
<P>Choose the music types for your free CDs:
<BR><SELECT NAME="music_type_multi" MULTIPLE>
   <OPTION SELECTED> R&B
   <OPTION> Jazz
   <OPTION> Blues
   <OPTION> New Age
</SELECT>
```

Example 2. The following example displays two selection lists that let the user choose a month and day. These selection lists are initialized to the current date. The user can change the month and day by using the selection lists or by choosing preset dates from radio buttons. Text fields on the form display the values of the `Select` object's properties and indicate the date chosen and whether it is Cinco de Mayo.

```
<HTML>
<HEAD>
<TITLE>Select object example</TITLE>
</HEAD>
<BODY>
<SCRIPT>
var today = new Date()
//---------------
function updatePropertyDisplay(monthObj,dayObj) {
   // Get date strings
   var monthInteger, dayInteger, monthString, dayString
   monthInteger=monthObj.selectedIndex
   dayInteger=dayObj.selectedIndex
   monthString=monthObj.options[monthInteger].text
   dayString=dayObj.options[dayInteger].text
   // Display property values
   document.selectForm.textFullDate.value=monthString + " " + dayString
   document.selectForm.textMonthLength.value=monthObj.length
   document.selectForm.textDayLength.value=dayObj.length
   document.selectForm.textMonthName.value=monthObj.name
   document.selectForm.textDayName.value=dayObj.name
   document.selectForm.textMonthIndex.value=monthObj.selectedIndex
   document.selectForm.textDayIndex.value=dayObj.selectedIndex
   // Is it Cinco de Mayo?
   if (monthObj.options[4].selected && dayObj.options[4].selected)
      document.selectForm.textCinco.value="Yes!"
   else
      document.selectForm.textCinco.value="No"
}
</SCRIPT>
<!--------------->
<FORM NAME="selectForm">
<P><B>Choose a month and day:</B>
Month: <SELECT NAME="monthSelection"
   onChange="updatePropertyDisplay(this,document.selectForm.daySelection)">
   <OPTION> January <OPTION> February <OPTION> March
   <OPTION> April <OPTION> May <OPTION> June
   <OPTION> July <OPTION> August <OPTION> September
   <OPTION> October <OPTION> November <OPTION> December
</SELECT>
Day: <SELECT NAME="daySelection"
   onChange="updatePropertyDisplay(document.selectForm.monthSelection,this)">
   <OPTION> 1 <OPTION> 2 <OPTION> 3 <OPTION> 4 <OPTION> 5
   <OPTION> 6 <OPTION> 7 <OPTION> 8 <OPTION> 9 <OPTION> 10
   <OPTION> 11 <OPTION> 12 <OPTION> 13 <OPTION> 14 <OPTION> 15
   <OPTION> 16 <OPTION> 17 <OPTION> 18 <OPTION> 19 <OPTION> 20
   <OPTION> 21 <OPTION> 22 <OPTION> 23 <OPTION> 24 <OPTION> 25
   <OPTION> 26 <OPTION> 27 <OPTION> 28 <OPTION> 29 <OPTION> 30
   <OPTION> 31
</SELECT>
<P><B>Set the date to: </B>
<INPUT TYPE="radio" NAME="dateChoice"
   onClick="
      monthSelection.selectedIndex=0;
      daySelection.selectedIndex=0;
      updatePropertyDisplay
         document.selectForm.monthSelection,document.selectForm.daySelection)">
   New Year's Day
<INPUT TYPE="radio" NAME="dateChoice"
   onClick="
      monthSelection.selectedIndex=4;
      daySelection.selectedIndex=4;
      updatePropertyDisplay
```

```
         (document.selectForm.monthSelection,document.selectForm.daySelection)">
   Cinco de Mayo
<INPUT TYPE="radio" NAME="dateChoice"
   onClick="
      monthSelection.selectedIndex=5;
      daySelection.selectedIndex=20;
      updatePropertyDisplay
         (document.selectForm.monthSelection,document.selectForm.daySelection)">
   Summer Solstice
<P><B>Property values:</B>
<BR>Date chosen: <INPUT TYPE="text" NAME="textFullDate" VALUE="" SIZE=20">
<BR>monthSelection.length<INPUT TYPE="text" NAME="textMonthLength" VALUE="" SIZE=20">
<BR>daySelection.length<INPUT TYPE="text" NAME="textDayLength" VALUE="" SIZE=20">
<BR>monthSelection.name<INPUT TYPE="text" NAME="textMonthName" VALUE="" SIZE=20">
<BR>daySelection.name<INPUT TYPE="text" NAME="textDayName" VALUE="" SIZE=20">
<BR>monthSelection.selectedIndex
   <INPUT TYPE="text" NAME="textMonthIndex" VALUE="" SIZE=20">
<BR>daySelection.selectedIndex<INPUT TYPE="text" NAME="textDayIndex" VALUE="" SIZE=20">
<BR>Is it Cinco de Mayo? <INPUT TYPE="text" NAME="textCinco" VALUE="" SIZE=20">
<SCRIPT>
document.selectForm.monthSelection.selectedIndex=today.getMonth()
document.selectForm.daySelection.selectedIndex=today.getDate()-1
updatePropertyDisplay(document.selectForm.monthSelection,document.selectForm.daySelection)
</SCRIPT>
</FORM>
</BODY>
</HTML>
```

Example 3. Add an option with the Option constructor. The following example creates two `Select` objects, one with and one without the `MULTIPLE` attribute. No options are initially defined for either object. When the user clicks a button associated with the `Select` object, the `populate` function creates four options for the `Select` object and selects the first option.

```
<SCRIPT>
function populate(inForm) {
   colorArray = new Array("Red", "Blue", "Yellow", "Green")

   var option0 = new Option("Red", "color_red")
   var option1 = new Option("Blue", "color_blue")
   var option2 = new Option("Yellow", "color_yellow")
   var option3 = new Option("Green", "color_green")

   for (var i=0; i < 4; i++) {
      eval("inForm.selectTest.options[i]=option" + i)
      if (i==0) {
         inForm.selectTest.options[i].selected=true
      }
   }

   history.go(0)
}
</SCRIPT>

<H3>Select Option() constructor</H3>
<FORM>
<SELECT NAME="selectTest"></SELECT><P>
<INPUT TYPE="button" VALUE="Populate Select List" onClick="populate(this.form)">
<P>
</FORM>

<HR>
<H3>Select-Multiple Option() constructor</H3>
<FORM>
<SELECT NAME="selectTest" multiple></SELECT><P>
<INPUT TYPE="button" VALUE="Populate Select List" onClick="populate(this.form)">
</FORM>
```

Example 4. Delete an option. The following function removes an option from a `Select` object.

```
function deleteAnItem(theList,itemNo) {
   theList.options[itemNo]=null
   history.go(0)
}
```

See also `Form`, `Radio`

Properties

form

An object reference specifying the form containing the selection list.

Property of	`Select`
Read-only	
Implemented in	Navigator 2.0

Description Each form element has a `form` property that is a reference to the element's parent form. This property is especially useful in event handlers, where you might need to refer to another element on the current form.

See also `Form`

length

The number of options in the selection list.

Property of	`Select`
Read-only	
Implemented in	Navigator 2.0

name

A string specifying the name of the selection list.

Property of	`Select`
Implemented in	Navigator 2.0

Security Navigator 3.0: This property is tainted by default. For information on data tainting, see "Security" on page 55.

Description The `name` property initially reflects the value of the `NAME` attribute. Changing the `name` property overrides this setting. The `name` property is not displayed on the screen; it is used to refer to the list programmatically.

If multiple objects on the same form have the same `NAME` attribute, an array of the given name is created automatically. Each element in the array represents an individual `Form` object. Elements are indexed in source order starting at 0. For example, if two `Text` elements and a `Select` element on the same form have their `NAME` attribute set to `"myField"`, an array with the elements `myField[0]`, `myField[1]`, and `myField[2]` is created. You need to be aware of this situation in your code and know whether `myField` refers to a single element or to an array of elements.

Examples In the following example, the `valueGetter` function uses a `for` loop to iterate over the array of elements on the `valueTest` form. The `msgWindow` window displays the names of all the elements on the form:

```
newWindow=window.open("http://home.netscape.com")

function valueGetter() {
   var msgWindow=window.open("")
   for (var i = 0; i < newWindow.document.valueTest.elements.length; i++) {
      msgWindow.document.write(newWindow.document.valueTest.elements[i].name + "<BR>")
   }
}
```

options

An array corresponding to options in a `Select` object in source order.

Property of	`Select`
Read-only	
Implemented in	Navigator 2.0

Description You can refer to the options of a `Select` object by using the `options` array. This array contains an entry for each option in a `Select` object (`OPTION` tag) in source order. For example, if a `Select` object named `musicStyle` contains three options, you can access these options as `musicStyle.options[0]`, `musicStyle.options[1]`, and `musicStyle.options[2]`.

To obtain the number of options in the selection list, you can use either `Select.length` or the `length` property of the `options` array. For example, you can get the number of options in the `musicStyle` selection list with either of these expressions:

```
musicStyle.length
musicStyle.options.length
```

You can add or remove options from a selection list using this array. To add or replace an option to an existing `Select` object, you assign a new `Option` object to a place in the array. For example, to create a new `Option` object called `jeans` and add it to the end of the selection list named `myList`, you could use this code:

```
jeans = new Option("Blue Jeans", "jeans", false, false);
myList.options[myList.length] = jeans;
```

To delete an option from a `Select` object, you set the appropriate index of the `options` array to null. Removing an option compresses the options array. For example, assume that `myList` has 5 elements in it, the value of the fourth element is `"foo"`, and you execute this statement:

```
myList.options[1] = null
```

Now, `myList` has 4 elements in it and the value of the *third* element is `"foo"`.

After you delete an option, you must refresh the document by using `history.go(0)`. This statement must be last. When the document reloads, variables are lost if not saved in cookies or form element values.

You can determine which option in a selection list is currently selected by using either the `selectedIndex` property of the `options` array or of the `Select` object itself. That is, the following expressions return the same value:

```
musicStyle.selectedIndex
musicStyle.options.selectedIndex
```

For more information about this property, see `Select.selectedIndex`.

For `Select` objects that can have multiple selections (that is, the `SELECT` tag has the `MULTIPLE` attribute), the `selectedIndex` property is not very useful. In this case, it returns the index of the first selection. To find all the selected options, you have to loop and test each option individually. For example, to print a list of all selected options in a selection list named `mySelect`, you could use code such as this:

```
document.write("You've selected the following options:\n")
for (var i = 0; i < mySelect.options.length; i++) {
   if (mySelect.options[i].selected)
      document.write(" mySelect.options[i].text\n")
}
```

In general, to work with individual options in a selection list, you work with the appropriate `Option` object.

selectedIndex

An integer specifying the index of the selected option in a `Select` object.

Property of	`Select`
Implemented in	Navigator 2.0

Security Navigator 3.0: This property is tainted by default. For information on data tainting, see "Security" on page 55.

Description Options in a `Select` object are indexed in the order in which they are defined, starting with an index of 0. You can set the `selectedIndex` property at any time. The display of the `Select` object updates immediately when you set the `selectedIndex` property.

If no option is selected, `selectedIndex` has a value of -1.

In general, the `selectedIndex` property is more useful for `Select` objects that are created without the `MULTIPLE` attribute. If you evaluate `selectedIndex` when multiple options are selected, the `selectedIndex` property specifies the index of the first option only. Setting `selectedIndex` clears any other options that are selected in the `Select` object.

The `Option.selected` property is more useful in conjunction with `Select` objects that are created with the `MULTIPLE` attribute. With the `Option.selected` property, you can evaluate every option in the `options` array to determine multiple selections, and you can select individual options without clearing the selection of other options.

Examples In the following example, the `getSelectedIndex` function returns the selected index in the `musicType` `Select` object:

```
function getSelectedIndex() {
   return document.musicForm.musicType.selectedIndex
}
```

The previous example assumes that the `Select` object is similar to the following:

```
<SELECT NAME="musicType">
   <OPTION SELECTED> R&B
   <OPTION> Jazz
   <OPTION> Blues
   <OPTION> New Age
</SELECT>
```

See also `Option.defaultSelected`, `Option.selected`

type

For all `Select` objects created with the `MULTIPLE` keyword, the value of the `type` property is `"select-multiple"`. For `Select` objects created without this keyword, the value of the `type` property is `"select-one"`. This property specifies the form element's type.

Property of	`Select`
Read-only	
Implemented in	Navigator 3.0

Examples The following example writes the value of the `type` property for every element on a form.

```
for (var i = 0; i < document.form1.elements.length; i++) {
   document.writeln("<BR>type is " + document.form1.elements[i].type)
}
```

Methods

blur

Removes focus from the selection list.

Method of	`Select`
Implemented in	Navigator 2.0

Syntax `blur()`

Parameters None

See also `Select.focus`

focus

Navigates to the selection list and gives it focus.

Method of	`Select`
Implemented in	Navigator 2.0

Syntax `focus()`

Parameters None

Description Use the `focus` method to navigate to a selection list and give it focus. The user can then make selections from the list.

See also `Select.blur`

handleEvent

Invokes the handler for the specified event.

Method of	`Select`

Implemented in	Navigator 4.0

Syntax `handleEvent(event)`

Parameters

`event`	The name of an event for which the object has an event handler.

Option

An option in a selection list.

Client-side object

Implemented in	Navigator 2.0 Navigator 3.0: added `defaultSelected` property; `text` property can be changed to change the text of an option

Created by The `Option` constructor or the HTML `OPTION` tag. To create an `Option` object with its constructor:

```
new Option(text, value, defaultSelected, selected)
```

Once you've created an Option object, you can add it to a selection list using the `Select.options` array.

Parameters

`text`	(Optional) Specifies the text to display in the select list.
`value`	(Optional) Specifies a value that is returned to the server when the option is selected and the form is submitted.
`defaultSelected`	(Optional) Specifies whether the option is initially selected (true or false).
`selected`	(Optional) Specifies the current selection state of the option (true or false).

Property Summary

Property	Descriptiohn
`defaultSelected`	Specifies the initial selection state of the option
`selected`	Specifies the current selection state of the option
`text`	Specifies the text for the option
`value`	Specifies the value that is returned to the server when the option is selected and the form is submitted

Description Usually you work with `Option` objects in the context of a selection list (a `Select` object). When JavaScript creates a `Select` object for each `SELECT` tag in the document, it creates `Option` objects for the `OPTION` tags inside the `SELECT` tag and puts those objects in the `options` array of the `Select` object.

In addition, you can create new options using the `Option` constructor and add those to a selection list. After you create an option and add it to the `Select` object, you must refresh the document by using `history.go(0)`. This statement must be last. When the document reloads, variables are lost if not saved in cookies or form element values.

You can use the `Option.selected` and `Select.selectedIndex` properties to change the selection state of an option.

- The `Select.selectedIndex` property is an integer specifying the index of the selected option. This is most useful for `Select` objects that are created without the `MULTIPLE` attribute. The following statement sets a `Select` object's `selectedIndex` property:

  ```
  document.myForm.musicTypes.selectedIndex = i
  ```

- The `Option.selected` property is a Boolean value specifying the current selection state of the option in a `Select` object. If an option is selected, its `selected` property is true; otherwise it is false. This is more useful for `Select` objects that are created with the `MULTIPLE` attribute. The following statement sets an option's `selected` property to true:

  ```
  document.myForm.musicTypes.options[i].selected = true
  ```

To change an option's text, use is `Option.text` property. For example, suppose a form has the following `Select` object:

```
<SELECT name="userChoice">
   <OPTION>Choice 1
   <OPTION>Choice 2
   <OPTION>Choice 3
</SELECT>
```

You can set the text of the `i`th item in the selection based on text entered in a text field named `whatsNew` as follows:

```
myform.userChoice.options[i].text = myform.whatsNew.value
```

You do not need to reload or refresh after changing an option's text.

Examples The following example creates two `Select` objects, one with and one without the `MULTIPLE` attribute. No options are initially defined for either object. When the user clicks a button associated with the `Select` object, the `populate` function creates four options for the `Select` object and selects the first option.

```
<SCRIPT>
function populate(inForm) {
   colorArray = new Array("Red", "Blue", "Yellow", "Green")

   var option0 = new Option("Red", "color_red")
   var option1 = new Option("Blue", "color_blue")
   var option2 = new Option("Yellow", "color_yellow")
   var option3 = new Option("Green", "color_green")

   for (var i=0; i < 4; i++) {
      eval("inForm.selectTest.options[i]=option" + i)
      if (i==0) {
         inForm.selectTest.options[i].selected=true
      }
   }

   history.go(0)
}
</SCRIPT>


<H3>Select Option() constructor</H3>
<FORM>
<SELECT NAME="selectTest"></SELECT><P>
<INPUT TYPE="button" VALUE="Populate Select List" onClick="populate(this.form)">
<P>
</FORM>

<HR>
<H3>Select-Multiple Option() constructor</H3>
<FORM>
<SELECT NAME="selectTest" multiple></SELECT><P>
<INPUT TYPE="button" VALUE="Populate Select List" onClick="populate(this.form)">
</FORM>
```

Properties

defaultSelected

A Boolean value indicating the default selection state of an option in a selection list.

Property of	`Option`
Implemented in	Navigator 3.0

Security Navigator 3.0: This property is tainted by default. For information on data tainting, see "Security" on page 55.

Description If an option is selected by default, the value of the `defaultSelected` property is true; otherwise, it is false. `defaultSelected` initially reflects whether the `SELECTED` attribute is used within an `OPTION` tag; however, setting `defaultSelected` overrides the `SELECTED` attribute.

You can set the `defaultSelected` property at any time. The display of the corresponding `Select` object does not update when you set the `defaultSelected` property of an option, only when you set the `Option.selected` or `Select.selectedIndex` properties.

A `Select` object created without the `MULTIPLE` attribute can have only one option selected by default. When you set `defaultSelected` in such an object, any previous default selections, including defaults set with the `SELECTED` attribute, are cleared. If you set `defaultSelected` in a `Select` object created with the `MULTIPLE` attribute, previous default selections are not affected.

Examples In the following example, the `restoreDefault` function returns the `musicType` `Select` object to its default state. The `for` loop uses the `options` array to evaluate every option in the `Select` object. The `if` statement sets the `selected` property if `defaultSelected` is true.

```
function restoreDefault() {
   for (var i = 0; i < document.musicForm.musicType.length; i++) {
      if (document.musicForm.musicType.options[i].defaultSelected == true) {
         document.musicForm.musicType.options[i].selected=true
      }
   }
}
```

The previous example assumes that the `Select` object is similar to the following:

```
<SELECT NAME="musicType">
   <OPTION SELECTED> R&B
   <OPTION> Jazz
   <OPTION> Blues
   <OPTION> New Age
</SELECT>
```

See also `Option.selected`, `Select.selectedIndex`

selected

A Boolean value indicating whether an option in a `Select` object is selected.

Property of	`Option`
Implemented in	Navigator 2.0

Security Navigator 3.0: This property is tainted by default. For information on data tainting, see "Security" on page 55.

Description If an option in a `Select` object is selected, the value of its `selected` property is true; otherwise, it is false. You can set the `selected` property at any time. The display of the associated `Select` object updates immediately when you set the `selected` property for one of its options.

In general, the `Option.selected` property is more useful than the `Select.selectedIndex` property for `Select` objects that are created with the `MULTIPLE` attribute. With the `Option.selected` property, you can evaluate every option in the `Select.options` array to determine multiple selections, and you can select individual options without clearing the selection of other options.

Examples See the examples for `defaultSelected`.

See also `Option.defaultSelected`, `Select.selectedIndex`

text

A string specifying the text of an option in a selection list.

Property of	`Option`

Implemented in	Navigator 2.0 Navigator 3.0: The `text` property can be changed to updated the selection option. In previous releases, you could set the `text` property but the new value was not reflected in the `Select` object.

Security Navigator 3.0: This property is tainted by default. For information on data tainting, see "Security" on page 55.

Description The `text` property initially reflects the text that follows an `OPTION` tag of a `SELECT` tag. You can set the `text` property at any time and the text displayed by the option in the selection list changes.

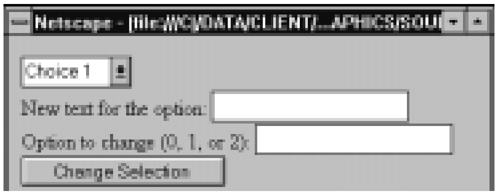
Examples **Example 1.** In the following example, the `getChoice` function returns the value of the `text` property for the selected option. The `for` loop evaluates every option in the `musicType` `Select` object. The `if` statement finds the option that is selected.

```
function getChoice() {
   for (var i = 0; i < document.musicForm.musicType.length; i++) {
      if (document.musicForm.musicType.options[i].selected == true) {
         return document.musicForm.musicType.options[i].text
      }
   }
   return null
}
```

The previous example assumes that the `Select` object is similar to the following:

```
<SELECT NAME="musicType">
   <OPTION SELECTED> R&B
   <OPTION> Jazz
   <OPTION> Blues
   <OPTION> New Age
</SELECT>
```

Example 2. In the following form, the user can enter some text in the first text field and then enter a number between 0 and 2 (inclusive) in the second text field. When the user clicks the button, the text is substituted for the indicated option number and that option is selected.


Netscape - [file:///C|/DATA/CLIENT/...APHICS/SOU
Choice 1
New text for the option:
Option to change (0, 1, or 2):
Change Selection

Chapter

8

Browser

This chapter deals with the browser and elements associated with it.

Table 8.1 summarizes the objects in this chapter.

Table 8.1 Browser-related objects

Object	Description
`navigator`	Contains information about the version of Navigator in use.
`MimeType`	Represents a MIME type (Multipart Internet Mail Extension) supported by the client.
`Plugin`	Represents a plug-in module installed on the client.

navigator

Contains information about the version of Navigator in use.

Client-side object

Implemented in	Navigator 2.0 Navigator 3.0: added `mimeTypes` and `plugins` properties; added `javaEnabled` and `taintEnabled` methods. Navigator 4.0: added `language` and `platform` properties; added `preference` method.

Created by The JavaScript runtime engine on the client automatically creates the `navigator` object.

Description Use the `navigator` object to determine which version of the Navigator your users have, what MIME types the user's Navigator can handle, and what plug-ins the user has installed. All of the properties of the `navigator` object are read-only.

Property Summary

Property	Descriptiohn
`appCodeName`	Specifies the code name of the browser.
`appName`	Specifies the name of the browser.
`appVersion`	Specifies version information for the Navigator.
`language`	Indicates the translation of the Navigator being used.
`mimeTypes`	An array of all MIME types supported by the client.
`platform`	Indicates the machine type for which the Navigator was compiled.
`plugins`	An array of all plug-ins currently installed on the client.
`userAgent`	Specifies the user-agent header.

Method Summary

Method	Descriptiohn
`javaEnabled`	Tests whether Java is enabled.
`plugins.refresh`	Makes newly installed plug-ins available and optionally reloads open documents that contain plug-ins.
`preference`	Allows a signed script to get and set certain Navigator preferences.
`taintEnabled`	Specifies whether data tainting is enabled.

Properties

appCodeName

A string specifying the code name of the browser.

Property of	`navigator`
Read-only	
Implemented in	Navigator 2.0

Examples The following example displays the value of the `appCodeName` property:

```
document.write("The value of navigator.appCodeName is " +
   navigator.appCodeName)
```

For Navigator 2.0 and 3.0, this displays the following:

```
The value of navigator.appCodeName is Mozilla
```

appName

A string specifying the name of the browser.

Property of	`navigator`
Read-only	
Implemented in	Navigator 2.0

Examples The following example displays the value of the `appName` property:

```
document.write("The value of navigator.appName is " +
   navigator.appName)
```

For Navigator 2.0 and 3.0, this displays the following:

```
The value of navigator.appName is Netscape
```

appVersion

A string specifying version information for the Navigator.

Property of	`navigator`
Read-only	

Implemented in Navigator 2.0

Description The `appVersion` property specifies version information in the following format:

`releaseNumber` (`platform`; `country`)

The values contained in this format are the following:

- `releaseNumber` is the version number of the Navigator. For example, `"2.0b4"` specifies Navigator 2.0, beta 4.
- `platform` is the platform upon which the Navigator is running. For example, `"Win16"` specifies a 16-bit version of Windows such as Windows 3.1.
- `country` is either `"I"` for the international release, or `"U"` for the domestic U.S. release. The domestic release has a stronger encryption feature than the international release.

Examples **Example 1.** The following example displays version information for the Navigator:

```
document.write("The value of navigator.appVersion is " +
   navigator.appVersion)
```

For Navigator 2.0 on Windows 95, this displays the following:

```
The value of navigator.appVersion is 2.0 (Win95, I)
```

For Navigator 3.0 on Windows NT, this displays the following:

```
The value of navigator.appVersion is 3.0 (WinNT, I)
```

Example 2. The following example populates a `Textarea` object with newline characters separating each line. Because the newline character varies from platform to platform, the example tests the `appVersion` property to determine whether the user is running Windows (`appVersion` contains `"Win"` for all versions of Windows). If the user is running Windows, the newline character is set to \r\n; otherwise, it's set to \n, which is the newline character for Unix and Macintosh.

```
<SCRIPT>
var newline=null
function populate(textareaObject){
   if (navigator.appVersion.lastIndexOf('Win') != -1)
      newline="\r\n"
      else newline="\n"
   textareaObject.value="line 1" + newline + "line 2" + newline
   + "line 3"
}
</SCRIPT>
<FORM NAME="form1">
<BR><TEXTAREA NAME="testLines" ROWS=8 COLS=55></TEXTAREA>
<P><INPUT TYPE="button" VALUE="Populate the Textarea object"
   onClick="populate(document.form1.testLines)">
</TEXTAREA>
</FORM>
```

language

Indicates the translation of the Navigator being used.

Property of	`navigator`
Read-only	
Implemented in	Navigator 4.0

Description The value for language is usually a 2-letter code, such as "en" and occasionally a five-character code to indicate a language subtype, such as "zh_CN".

Use this property to determine the language of the Navigator client software being used. For example you might want to display translated text for the user.

mimeTypes

An array of all MIME types supported by the client.

Property of	`navigator`
Read-only	
Implemented in	Navigator 3.0

The `mimeTypes` array contains an entry for each MIME type supported by the client (either internally, via helper applications, or by plug-ins). For example, if a client supports three MIME types, these MIME types are reflected as `navigator.mimeTypes[0]`, `navigator.mimeTypes[1]`, and `navigator.mimeTypes[2]`.

Each element of the `mimeTypes` array is a `MimeType` object.

See also `MimeType`

platform

Indicates the machine type for which the Navigator was compiled.

Property of	`navigator`
Read-only	
Implemented in	Navigator 4.0

Description Platform values are Win32, Win16, Mac68k, MacPPC and various Unix.

The machine type the Navigator was compiled for may differ from the actual machine type due to version differences, emulators, or other reasons.

If you use SmartUpdate to download software to a user's machine, you can use this property to ensure that the trigger downloads the appropriate JAR files. The triggering page checks the Navigator version before checking the platform property. For information on using SmartUpdate, see *Using JAR Installation Manager for SmartUpdate*[1].

plugins

An array of all plug-ins currently installed on the client.

Property of	`navigator`
Read-only	
Implemented in	Navigator 3.0

You can refer to the `Plugin` objects installed on the client by using this array. Each element of the `plugins` array is a `Plugin` object. For example, if three plug-ins are installed on the client, these plug-ins are reflected as `navigator.plugins[0]`, `navigator.plugins[1]`, and `navigator.plugins[2]`.

To use the `plugins` array:

1. http://developer.netscape.com/library/documentation/communicator/jarman/index.htm

```
1. navigator.plugins[index]
2. navigator.plugins[index][mimeTypeIndex]
```

`index` is an integer representing a plug-in installed on the client or a string containing the name of a `Plugin` object (from the `name` property). The first form returns the `Plugin` object stored at the specified location in the plugins array. The second form returns the `MimeType` object at the specified index in that `Plugin` object.

To obtain the number of plug-ins installed on the client, use the `length` property: `navigator.plugins.length`.

plugins.refresh The `plugins` array has its own method, `refresh`. This method makes newly installed plug-ins available, updates related arrays such as the `plugins` array, and optionally reloads open documents that contain plug-ins. You call this method with one of the following statements:

```
navigator.plugins.refresh(true)
navigator.plugins.refresh(false)
```

If you supply true, `refresh` refreshes the `plugins` array to make newly installed plug-ins available and reloads all open documents that contain embedded objects (`EMBED` tag). If you supply false, it refreshes the `plugins` array, but does not reload open documents.

When the user installs a plug-in, that plug-in is not available until `refresh` is called or the user closes and restarts Navigator.

Examples The following code refreshes arrays and reloads open documents containing embedded objects:

```
navigator.plugins.refresh(true)
```

See also the examples for the `Plugin` object.

userAgent

A string representing the value of the user-agent header sent in the HTTP protocol from client to server.

Property of	`navigator`
Read-only	
Implemented in	Navigator 2.0

Description Servers use the value sent in the user-agent header to identify the client.

Examples The following example displays `userAgent` information for the Navigator:

```
document.write("The value of navigator.userAgent is " +
   navigator.userAgent)
```

For Navigator 2.0, this displays the following:

```
The value of navigator.userAgent is Mozilla/2.0 (Win16; I)
```

Methods

javaEnabled

Tests whether Java is enabled.

Method of	`navigator`
Static	
Implemented in	Navigator 3.0

Syntax `javaEnabled()`

Parameters None.

Description `javaEnabled` returns true if Java is enabled; otherwise, false. The user can enable or disable Java by through user preferences.

Examples The following code executes `function1` if Java is enabled; otherwise, it executes `function2`.

```
if (navigator.javaEnabled()) {
   function1()
}
else function2()
```

See also `navigator.appCodeName`, `navigator.appName`, `navigator.userAgent`

preference

Allows a signed script to get and set certain Navigator preferences.

Method of	`navigator`

Static

Implemented in Navigator 4.0

Syntax

```
preference(prefName)
preference(prefName, setValue)
```

Parameters

`prefName`	A string representing the name of the preference you want to get or set. Allowed preferences are listed below.
`setValue`	The value you want to assign to the preference. This can be a string, number, or Boolean.

Description This method returns the value of the preference. If you use the method to set the value, it returns the new value.

Security Reading a preference with the `preference` method requires the `UniversalPreferencesRead` privilege. Setting a preference with this method requires the `UniversalPreferencesWrite` privilege.

For information on security in Navigator 4.0, see Chapter 7, "JavaScript Security," in the *JavaScript Guide*.

With permission, you can get and set the preferences shown in Table 8.2.

Table 8.2 Preferences.

To do this...	Set this preference...	To...
Automatically load images	`general.always_load_images`	true or false
Enable Java	`security.enable_java`	true or false
Enable JavaScript	`javascript.enabled`	true or false
Enable style sheets	`browser.enable_style_sheets`	true or false
Enable SmartUpdate	`autoupdate.enabled`	true or false
Accept all cookies	`network.cookie.cookieBehavior`	0
Accept only cookies that get sent back to the originating server	`network.cookie.cookieBehavior`	1
Disable cookies	`network.cookie.cookieBehavior`	2
Warn before accepting cookie	`network.cookie.warnAboutCookies`	true or false

taintEnabled

Specifies whether data tainting is enabled.

Method of	`navigator`
Static	
Implemented in	Navigator 3.0; removed in Navigator 4.0

Syntax `navigator.taintEnabled()`

Description Tainting prevents other scripts from passing information that should be secure and private, such as directory structures or user session history. JavaScript cannot pass tainted values on to any server without the end user's permission.

Use `taintEnabled` to determine if data tainting is enabled. `taintEnabled` returns true if data tainting is enabled, false otherwise. The user enables or disables data tainting by using the environment variable `NS_ENABLE_TAINT`.

Examples The following code executes function1 if data tainting is enabled; otherwise it executes function2.

```
if (navigator.taintEnabled()) {
   function1()
   }
else function2()
```

See also `taint`, `untaint`

MimeType

A MIME type (Multipart Internet Mail Extension) supported by the client.

Client-side object	
Implemented in	Navigator 3.0

Created by You do not create `MimeType` objects yourself. These objects are predefined JavaScript objects that you access through the `mimeTypes` array of the `navigator` or `Plugin` object:

```
navigator.mimeTypes[index]
```

where `index` is either an integer representing a MIME type supported by the client or a string containing the type of a `MimeType` object (from the `MimeType.type` property).

Description Each `MimeType` object is an element in a `mimeTypes` array. The `mimeTypes` array is a property of both `navigator` and `Plugin` objects. For example, the following table summarizes the values for displaying JPEG images:

Expression	Value
`navigator.mimeTypes["image/jpeg"].type`	`image/jpeg`
`navigator.mimeTypes["image/jpeg"].description`	`JPEG Image`
`navigator.mimeTypes["image/jpeg"].suffixes`	`jpeg, jpg, jpe, jfif, pjpeg, pjp`
`navigator.mimeTypes["image/jpeg"].enabledPlugins`	`null`

Property Summary

Property	Descriptiohn
`description`	A description of the MIME type.
`enabledPlugin`	Reference to the `Plugin` object configured for the MIME type.
`suffixes`	A string listing possible filename extensions for the MIME type, for example `"mpeg, mpg, mpe, mpv, vbs, mpegv"`.
`type`	The name of the MIME type, for example `"video/mpeg"` or `"audio/x-wav"`.

Methods None.

Examples The following code displays the `type`, `description`, `suffixes`, and `enabledPlugin` properties for each `MimeType` object on a client:

```
document.writeln("<TABLE BORDER=1><TR VALIGN=TOP>",
   "<TH ALIGN=left>i",
   "<TH ALIGN=left>type",
   "<TH ALIGN=left>description",
   "<TH ALIGN=left>suffixes",
   "<TH ALIGN=left>enabledPlugin.name</TR>")
for (i=0; i < navigator.mimeTypes.length; i++) {
   document.writeln("<TR VALIGN=TOP><TD>",i,
      "<TD>",navigator.mimeTypes[i].type,
      "<TD>",navigator.mimeTypes[i].description,
      "<TD>",navigator.mimeTypes[i].suffixes)
   if (navigator.mimeTypes[i].enabledPlugin==null) {
      document.writeln(
      "<TD>None",
      "</TR>")
   } else {
      document.writeln(
      "<TD>",navigator.mimeTypes[i].enabledPlugin.name,
      "</TR>")
   }
}
document.writeln("</TABLE>")
```

The preceding example displays output similar to the following:

i	type	description	suffixes	enabledPlugin.name
0	audio/aiff	AIFF	aif, aiff	LiveAudio
1	audio/wav	WAV	wav	LiveAudio
2	audio/x-midi	MIDI	mid, midi	LiveAudio
3	audio/midi	MIDI	mid, midi	LiveAudio
4	video/msvideo	Video for Windows	avi	NPAVI32 Dynamic Link Library
5	*	Netscape Default Plugin		Netscape Default Plugin
6	zz-application/zz-winassoc-TGZ		TGZ	None

See also `navigator`, `navigator.mimeTypes`, `Plugin`

Properties

description

A human-readable description of the data type described by the MIME type object.

Property of `MimeType`

Read-only	
Implemented in	Navigator 3.0

enabledPlugin

The `Plugin` object for the plug-in that is configured for the specified MIME type If the MIME type does not have a plug-in configured, `enabledPlugin` is null.

Property of	`MimeType`
Read-only	
Implemented in	Navigator 3.0

Description Use the `enabledPlugin` property to determine which plug-in is configured for a specific MIME type. Each plug-in may support multiple MIME types, and each MIME type could potentially be supported by multiple plug-ins. However, only one plug-in can be configured for a MIME type. (On Macintosh and Unix, the user can configure the handler for each MIME type; on Windows, the handler is determined at browser start-up time.)

The `enabledPlugin` property is a reference to a `Plugin` object that represents the plug-in that is configured for the specified MIME type.

You might need to know which plug-in is configured for a MIME type, for example, to dynamically emit an `EMBED` tag on the page if the user has a plug-in configured for the MIME type.

Examples The following example determines whether the Shockwave plug-in is installed. If it is, a movie is displayed.

```
// Can we display Shockwave movies?
mimetype = navigator.mimeTypes["application/x-director"]
if (mimetype) {
   // Yes, so can we display with a plug-in?
   plugin = mimetype.enabledPlugin
   if (plugin)
      // Yes, so show the data in-line
      document.writeln("Here\'s a movie: <EMBED SRC=mymovie.dir HEIGHT=100 WIDTH=100>")
      else
      // No, so provide a link to the data
      document.writeln("<A HREF='mymovie.dir'>Click here</A> to see a movie.")
   } else {
   // No, so tell them so
   document.writeln("Sorry, can't show you this cool movie.")
}
```

suffixes

A string listing possible file suffixes (also known as filename extensions) for the MIME type.

Property of	`MimeType`
Read-only	
Implemented in	Navigator 3.0

Description The `suffixes` property is a string consisting of each valid suffix (typically three letters long) separated by commas. For example, the suffixes for the `"audio/x-midi"` MIME type are `"mid, midi"`.

type

A string specifying the name of the MIME type. This string distinguishes the MIME type from all others; for example `"video/mpeg"` or `"audio/x-wav"`.

Property of	`MimeType`
Read-only	
Implemented in	Navigator 3.0

Property of `MimeType`

Plugin

A plug-in module installed on the client.

Client-side object	
Implemented in	Navigator 3.0

Created by `Plugin` objects are predefined JavaScript objects that you access through the `navigator.plugins` array.

Description A `Plugin` object is a plug-in installed on the client. A plug-in is a software module that the browser can invoke to display specialized types of embedded data within the browser. The user can obtain a list of installed plug-ins by choosing About Plug-ins from the Help menu.

Each `Plugin` object is itself array containing one element for each MIME type supported by the plug-in. Each element of the array is a `MimeType` object. For example, the following code displays the `type` and `description` properties of the first `Plugin` object's first `MimeType` object.

```
myPlugin=navigator.plugins[0]
myMimeType=myPlugin[0]
document.writeln('myMimeType.type is ',myMimeType.type,"<BR>")
document.writeln('myMimeType.description is ',myMimeType.description)
```

The preceding code displays output similar to the following:

```
myMimeType.type is video/quicktime
myMimeType.description is QuickTime for Windows
```

The `Plugin` object lets you dynamically determine which plug-ins are installed on the client. You can write scripts to display embedded plug-in data if the appropriate plug-in is installed, or display some alternative information such as images or text if not.

Plug-ins can be platform dependent and configurable, so a `Plugin` object's array of `MimeType` objects can vary from platform to platform, and from user to user.

Each `Plugin` object is an element in the `plugins` array.

When you use the `EMBED` tag to generate output from a plug-in application, you are not creating a `Plugin` object. Use the `document.embeds` array to refer to plug-in instances created with `EMBED` tags. See the `document.embeds` array.

Property Summary

Property	Descriptiohn
`description`	A description of the plug-in.
`filename`	Name of the plug-in file on disk.
`length`	Number of elements in the plug-in's array of `MimeType` objects.
`name`	Name of the plug-in.

Examples **Example 1.** The user can obtain a list of installed plug-ins by choosing About Plug-ins from the Help menu. To see the code the browser uses for this report, choose About Plug-ins from the Help menu, then choose Page Source from the View menu.

Example 2. The following code assigns shorthand variables for the predefined LiveAudio properties.

```
var myPluginName = navigator.plugins["LiveAudio"].name
var myPluginFile = navigator.plugins["LiveAudio"].filename
var myPluginDesc = navigator.plugins["LiveAudio"].description
```

Example 3. The following code displays the message "LiveAudio is configured for audio/wav" if the LiveAudio plug-in is installed and is enabled for the `"audio/wav"` MIME type:

```
var myPlugin = navigator.plugins["LiveAudio"]
var myType = myPlugin["audio/wav"]
if (myType && myType.enabledPlugin == myPlugin)
   document.writeln("LiveAudio is configured for audio/wav")
```

Example 4. The following expression represents the number of MIME types that Shockwave can display:

```
navigator.plugins["Shockwave"].length
```

Example 5. The following code displays the `name`, `filename`, `description`, and `length` properties for each `Plugin` object on a client:

```
document.writeln("<TABLE BORDER=1><TR VALIGN=TOP>",
   "<TH ALIGN=left>i",
   "<TH ALIGN=left>name",
   "<TH ALIGN=left>filename",
   "<TH ALIGN=left>description",
   "<TH ALIGN=left># of types</TR>")
for (i=0; i < navigator.plugins.length; i++) {
   document.writeln("<TR VALIGN=TOP><TD>",i,
      "<TD>",navigator.plugins[i].name,
      "<TD>",navigator.plugins[i].filename,
      "<TD>",navigator.plugins[i].description,
      "<TD>",navigator.plugins[i].length,
      "</TR>")
}
document.writeln("</TABLE>")
```

The preceding example displays output similar to the following:

i	name	filename	description	# of types
0	QuickTime Plug-In	d:\nettools\netscape\nav30\Program\plugins\NPQTW32.DLL	QuickTime Plug-In for Win32 v.1.0.0	1
1	LiveAudio	d:\nettools\netscape\nav30\Program\plugins\NPAUDIO.DLL	LiveAudio - Netscape Navigator sound playing component	7
2	NPAVI32 Dynamic Link Library	d:\nettools\netscape\nav30\Program\plugins\npavi32.dll	NPAVI32, avi plugin DLL	2
3	Netscape Default Plugin	d:\nettools\netscape\nav30\Program\plugins\npnul32.dll	Null Plugin	1

See also `MimeType`, `document.embeds`

Properties

description

A human-readable description of the plug-in. The text is provided by the plug-in developers.

Property of	`Plugin`
Read-only	
Implemented in	Navigator 3.0

filename

The name of a plug-in file on disk.

Property of	`Plugin`
Read-only	
Implemented in	Navigator 3.0

Description The `filename` property is the plug-in program's file name and is supplied by the plug-in itself. This name may vary from platform to platform.

Examples See the examples for `Plugin`.

length

The number of elements in the plug-in's array of `MimeType` objects.

Property of	`Plugin`
Read-only	
Implemented in	Navigator 3.0

name

A string specifying the plug-in's name.

Property of	`Plugin`
Read-only	
Implemented in	Navigator 3.0

Security Navigator 3.0: This property is tainted by default. For information on data tainting, see "Security" on page 55.

Description The plug-in's name, supplied by the plug-in itself. Each plug-in should have a name that uniquely identifies it.

Chapter

9

Events and Event Handlers

This chapter contains the `event` object and the event handlers that are used with client-side objects in JavaScript to evoke particular actions. In addition, it contains general information about using events and event handlers.

Table 9.1 lists the one object in this chapter.

Table 9.1 Event-related object

Object	Description
`event`	Represents a JavaScript event. Passed to every event handler.

Table 9.2 summarizes the JavaScript event handlers.

Table 9.2 Events and their corresponding event handlers.

Event	Event handler	Event occurs when...
`abort`	`onAbort`	The user aborts the loading of an image (for example by clicking a link or clicking the Stop button).
`blur`	`onBlur`	A form element loses focus or when a window or frame loses focus.
`change`	`onChange`	A select, text, or textarea field loses focus and its value has been modified.
`click`	`onClick`	An object on a form is clicked.
`dblclick`	`onDblClick`	The user double-clicks a form element or a link.

Table 9.2 Events and their corresponding event handlers.

Event	Event handler	Event occurs when...
`dragdrop`	`onDragDrop`	The user drops an object onto the browser window, such as dropping a file on the browser window.
`error`	`onError`	The loading of a document or image causes an error.
`focus`	`onFocus`	A window, frame, or frameset receives focus or when a form element receives input focus.
`keydown`	`onKeyDown`	The user depresses a key.
`keypress`	`onKeyPress`	The user presses or holds down a key.
`keyup`	`onKeyUp`	The user releases a key.
`load`	`onLoad`	The browser finishes loading a window or all of the frames within a `FRAMESET` tag.
`mousedown`	`onMouseDown`	The user depresses a mouse button.
`mousemove`	`onMouseMove`	The user moves the cursor.
`mouseout`	`onMouseOut`	The cursor leaves an area (client-side image map) or link from inside that area or link.
`mouseover`	`onMouseOver`	The cursor moves over an object or area from outside that object or area.
`mouseup`	`onMouseUp`	The user releases a mouse button.
`move`	`onMove`	The user or script moves a window or frame.
`reset`	`onReset`	The user resets a form (clicks a Reset button).
`resize`	`onResize`	The user or script resizes a window or frame.
`select`	`onSelect`	The user selects some of the text within a text or textarea field.
`submit`	`onSubmit`	The user submits a form.
`unload`	`onUnload`	The user exits a document.

General Information about Events

JavaScript applications in the browser are largely event-driven. *Events* are actions that occur usually as a result of something the user does. For example, clicking a button is an event, as is changing a text field or moving the mouse over a link. For your script to react to an event, you define *event handlers*, such as `onChange` and `onClick`.

Defining Event Handlers

If an event applies to an HTML tag, then you can define an event handler for it. The name of an event handler is the name of the event, preceded by `"on"`. For example, the event handler for the focus event is `onFocus`.

To create an event handler for an HTML tag, add an event handler attribute to the tag. Put JavaScript code in quotation marks as the attribute value. The general syntax is

```
<TAG eventHandler="JavaScript Code">
```

where `TAG` is an HTML tag and `eventHandler` is the name of the event handler. For example, suppose you have created a JavaScript function called `compute`. You can cause the browser to perform this function when the user clicks a button by assigning the function call to the button's `onClick` event handler:

```
<INPUT TYPE="button" VALUE="Calculate" onClick="compute(this.form)">
```

You can put any JavaScript statements inside the quotation marks following `onClick`. These statements are executed when the user clicks the button. If you want to include more than one statement, separate statements with a semicolon.

When you create an event handler, the corresponding JavaScript object gets a property with the name of the event handler in lower case letters. (In Navigator 4.0, you can also use the mixed case name of the event handler for the property name.) This property allows you to access the object's event handler. For example, in the preceding example, JavaScript creates a `Button` object with an `onclick` property whose value is `"compute(this.form)"`.

Chapter 7, "JavaScript Security," in *JavaScript Guide* contains more information about creating and using event handlers.

Events in Navigator 4.0

In Navigator 4.0, JavaScript includes `event` objects as well as event handlers. Each event has an `event` object associated with it. The `event` object provides information about the event, such as the type of event and the location of the cursor at the time of the event. When an event occurs, and if an event handler has been written to handle the event, the `event` object is sent as an argument to the event handler.

Typically, the object on which the event occurs handles the event. For example, when the user clicks a button, it is often the button's event handler that handles the event. Sometimes you may want the `Window` or `document` object to handle certain types of events. For example, you may want the `document` object to handle all MouseDown events no matter where they occur in the document. JavaScript's event capturing model allows you to define methods that capture and handle events before they reach their intended target.

In addition to providing the `event` object, Navigator 4.0 allows a `Window` or `document` to capture and handle an event before it reaches its intended target. To accomplish this, the `Window`, `document`, and `Layer` objects have these new methods:

- `captureEvents`
- `releaseEvents`
- `routeEvent`
- `handleEvent` (Not a method of the `Layer` object)

For example, suppose you want to capture all click events that occur in a window. First, you need to set up the window to capture click events:

```
window.captureEvents(Event.CLICK);
```

The argument to `Window.captureEvents` is a property of the `event` object and indicates the type of event to capture. To capture multiple events, the argument is a list separated by vertical slashes (|). For example:

```
window.captureEvents(Event.CLICK | Event.MOUSEDOWN | Event.MOUSEUP)
```

Next, you need to define a function that handles the event. The argument `evnt` is the `event` object for the event.

```
function clickHandler(evnt) {
   //What goes here depends on how you want to handle the event.
   //This is described below.
}
```

You have four options for handling the event:

- Return true. In the case of a link, the link is followed and no other event handler is checked. If the event cannot be canceled, this ends the event handling for that event.

  ```
  function clickHandler(evnt) { return true; }
  ```

- Return false. In the case of a link, the link is not followed. If the event is non-cancelable, this ends the event handling for that event.

  ```
  function clickHandler(evnt) { return false; }
  ```

- Call `routeEvent`. JavaScript looks for other event handlers for the event. If another object is attempting to capture the event (such as the document), JavaScript calls its event handler. If no other object is attempting to capture the event, JavaScript looks for an event handler for the event's original target (such as a button). The `routeEvent` method returns the value returned by the event handler. The capturing object can look at this return value and decide how to proceed.

  ```
  function clickHandler(evnt) {
     var retval = routeEvent(evnt);
     if (retval == false) return false;
     else return true;
  }
  ```

  Note: When `routeEvent` calls an event handler, the event handler is activated. If `routeEvent` calls an event handler whose function is to display a new page, the action takes place without returning to the capturing object.

- Call the `handleEvent` method of an event receiver. Any object that can register event handlers is an event receiver. This method explicitly calls the event handler of the event receiver and bypasses the capturing hierarchy. For example, if you wanted all click events to go to the first link on the page, you could use:

  ```
  function clickHandler(evnt) {
     window.document.links[0].handleEvent(evnt);
  }
  ```

  As long as the link has an `onClick` handler, the link handles any click event it receives.

Finally, you need to register the function as the window's event handler for that event:

```
window.onClick = clickHandler;
```

Important If a window with frames wants to capture events in pages loaded from different locations, you need to use `captureEvents` in a signed script and call `Window.enableExternalCapture`.

In the following example, the window and document capture and release events:

```
<HTML>
<SCRIPT>

function fun1(evnt) {
   alert ("The window got an event of type: " + evnt.type +
      " and will call routeEvent.");
   window.routeEvent(evnt);
   alert ("The window returned from routeEvent.");
   return true;
}

function fun2(evnt) {
   alert ("The document got an event of type: " + evnt.type);
   return false;
}

function setWindowCapture() {
   window.captureEvents(Event.CLICK);
}

function releaseWindowCapture() {
   window.releaseEvents(Event.CLICK);
}

function setDocCapture() {
   document.captureEvents(Event.CLICK);
}

function releaseDocCapture() {
   document.releaseEvents(Event.CLICK);

}

window.onclick=fun1;
document.onclick=fun2;

</SCRIPT>
...
</HTML>
```

event

The `event` object contains properties that describe a JavaScript event, and is passed as an argument to an event handler when the event occurs.

Client-side object	
Implemented in	Navigator 4.0

In the case of a mouse-down event, for example, the `event` object contains the type of event (in this case MouseDown), the x and y position of the cursor at the time of the event, a number representing the mouse button used, and a field containing the modifier keys (Control, Alt, Meta, or Shift) that were depressed at the time of the event. The properties used within the `event` object vary from one type of event to another. This variation is provided in the descriptions of individual event handlers.

For more information, see "General Information about Events" on page 481.

Created by `event` objects are created by Communicator when an event occurs. You do not create them yourself.

Security Setting any property of this object requires the `UniversalBrowserWrite` privilege. In addition, getting the `data` property of the `DragDrop` event requires the `UniversalBrowserRead` privilege. For information on security in Navigator 4.0, see Chapter 7, "JavaScript Security," in the *JavaScript Guide*.

Property Summary Not all of these properties are relevant to each event type. To learn which properties are used by an event, see the "Event object properties used" section of the individual event handler.

Property	Descriptiohn
`target`	String representing the object to which the event was originally sent. (All events)
`type`	String representing the event type. (All events)
`data`	Returns an array of strings containing the URLs of the dropped objects. Passed with the DragDrop event.
`height`	Represents the height of the window or frame.

Property	Descriptiohn
`layerX`	Number specifying either the object width when passed with the resize event, or the cursor's horizontal position in pixels relative to the layer in which the event occurred. Note that `layerX` is synonymous with x.
`layerY`	Number specifying either the object height when passed with the resize event, or the cursor's vertical position in pixels relative to the layer in which the event occurred. Note that `layerY` is synonymous with y.
`modifiers`	String specifying the modifier keys associated with a mouse or key event. Modifier key values are: ALT_MASK, CONTROL_MASK, SHIFT_MASK, and META_MASK.
`pageX`	Number specifying the cursor's horizontal position in pixels, relative to the page.
`pageY`	Number specifying the cursor's vertical position in pixels relative to the page.
`screenX`	Number specifying the cursor's horizontal position in pixels, relative to the screen.
`screenY`	Number specifying the cursor's vertical position in pixels, relative to the screen.
`which`	Number specifying either the mouse button that was pressed or the ASCII value of a pressed key. For a mouse, 1 is the left button, 2 is the middle button, and 3 is the right button.
`width`	Represents the width of the window or frame.

Example The following example uses the event object to provide the type of event to the alert message.

```
<A HREF="http://home.netscape.com" onClick='alert("Link got an event: "
+ event.type)'>Click for link event</A>
```

The following example uses the event object in an explicitly called event handler.

```
<SCRIPT>
function fun1(evnt) {
   alert ("Document got an event: " + evnt.type);
   alert ("x position is " + evnt.layerX);
   alert ("y position is " + evnt.layerY);
   if (evnt.modifiers & Event.ALT_MASK)
      alert ("Alt key was down for event.");
   return true;
   }
document.onmousedown = fun1;
</SCRIPT>
```

onAbort

Executes JavaScript code when an abort event occurs; that is, when the user aborts the loading of an image (for example by clicking a link or clicking the Stop button).

Event handler for	`Image`
Implemented in	Navigator 3.0

Syntax `onAbort="handlerText"`

Parameters

`handlerText`	JavaScript code or a call to a JavaScript function.

Event properties used

`type`	Indicates the type of event.
`target`	Indicates the object to which the event was originally sent.

Examples In the following example, an `onAbort` handler in an `Image` object displays a message when the user aborts the image load:

```
<IMG NAME="aircraft" SRC="f15e.gif"
   onAbort="alert('You didn\'t get to see the image!')">
```

See also `onError`, `onLoad`

For general information on event handlers, see "General Information about Events" on page 481.

For information about the `event` object, see `event`.

onBlur

Executes JavaScript code when a blur event occurs; that is, when a form element loses focus or when a window or frame loses focus.

Event handler for	`Button`, `Checkbox`, `FileUpload`, `Layer`, `Password`, `Radio`, `Reset`, `Select`, `Submit`, `Text`, `Textarea`, `Window`
Implemented in	Navigator 2.0 Navigator 3.0: event handler of `Button`, `Checkbox`, `FileUpload`, `Frame`, `Password`, `Radio`, `Reset`, `Submit`, and `Window`

Syntax `onBlur="handlerText"`

Parameters

`handlerText`	JavaScript code or a call to a JavaScript function.

Description The `blur` event can result from a call to the `Window.blur` method or from the user clicking the mouse on another object or window or tabbing with the keyboard.

For windows, frames, and framesets, `onBlur` specifies JavaScript code to execute when a window loses focus.

A frame's `onBlur` event handler overrides an `onBlur` event handler in the `BODY` tag of the document loaded into frame.

Note In Navigator 3.0, on some platforms placing an `onBlur` event handler in a `FRAMESET` tag has no effect.

Event properties used

`type`	Indicates the type of event.
`target`	Indicates the object to which the event was originally sent.

Examples **Example 1: Validate form input.** In the following example, `userName` is a required text field. When a user attempts to leave the field, the `onBlur` event handler calls the `required` function to confirm that `userName` has a legal value.

```
<INPUT TYPE="text" VALUE="" NAME="userName"
   onBlur="required(this.value)">
```

Example 2: Change the background color of a window. In the following example, a window's `onBlur` and `onFocus` event handlers change the window's background color depending on whether the window has focus.

```
<BODY BGCOLOR="lightgrey"
   onBlur="document.bgColor='lightgrey'"
   onFocus="document.bgColor='antiquewhite'">
```

Example 3: Change the background color of a frame. The following example creates four frames. The source for each frame, `onblur2.html` has the `BODY` tag with the onBlur and `onFocus` event handlers shown in Example 1. When the document loads, all frames are light grey. When the user clicks a frame, the `onFocus` event handler changes the frame's background color to antique white. The frame that loses focus is changed to light grey. Note that the `onBlur` and `onFocus` event handlers are within the `BODY` tag, not the `FRAME` tag.

```
<FRAMESET ROWS="50%,50%" COLS="40%,60%">
<FRAME SRC=onblur2.html NAME="frame1">
<FRAME SRC=onblur2.html NAME="frame2">
<FRAME SRC=onblur2.html NAME="frame3">
<FRAME SRC=onblur2.html NAME="frame4">
</FRAMESET>
```

The following code has the same effect as the previous code, but is implemented differently. The `onFocus` and `onBlur` event handlers are associated with the frame, not the document. The `onBlur` and `onFocus` event handlers for the frame are specified by setting the `onblur` and `onfocus` properties.

```
<SCRIPT>
function setUpHandlers() {
   for (var i = 0; i < frames.length; i++) {
      frames[i].onfocus=new Function("document.bgColor='antiquewhite'")
      frames[i].onblur=new Function("document.bgColor='lightgrey'")
   }
}
</SCRIPT>

<FRAMESET ROWS="50%,50%" COLS="40%,60%" onLoad=setUpHandlers()>
<FRAME SRC=onblur2.html NAME="frame1">
<FRAME SRC=onblur2.html NAME="frame2">
<FRAME SRC=onblur2.html NAME="frame3">
<FRAME SRC=onblur2.html NAME="frame4">
</FRAMESET>
```

Example 4: Close a window. In the following example, a window's `onBlur` event handler closes the window when the window loses focus.

```
<BODY onBlur="window.close()">
This is some text
</BODY>
```

See also `onChange`, `onFocus`

For general information on event handlers, see "General Information about Events" on page 481.

For information about the `event` object, see `event`.

onChange

Executes JavaScript code when a change event occurs; that is, when a `Select`, `Text`, or `Textarea` field loses focus and its value has been modified.

Event handler for	`FileUpload`, `Select`, `Text`, `Textarea`
Implemented in	Navigator 2.0 event handler for `Select`, `Text`, and `Textarea` Navigator 3.0: added as event handler of `FileUpload`

Syntax `onChange="handlerText"`

Parameters

`handlerText`	JavaScript code or a call to a JavaScript function.

Description Use `onChange` to validate data after it is modified by a user.

Event properties used

`type`	Indicates the type of event.
`target`	Indicates the object to which the event was originally sent.

Examples In the following example, `userName` is a text field. When a user changes the text and leaves the field, the onChange event handler calls the `checkValue` function to confirm that `userName` has a legal value.

```
<INPUT TYPE="text" VALUE="" NAME="userName"
   onChange="checkValue(this.value)">
```

See also `onBlur`, `onFocus`

For general information on event handlers, see "General Information about Events" on page 481.

For information about the `event` object, see `event`.

onClick

Executes JavaScript code when a click event occurs; that is, when an object on a form is clicked. (A `Click` event is a combination of the MouseDown and MouseUp events).

Event handler for	`Button`, `document`, `Checkbox`, `Link`, `Radio`, `Reset`, `Submit`
Implemented in	Navigator 2.0 Navigator 3.0: added the ability to return false to cancel the action associated with a click event

Syntax `onClick="handlerText"`

Parameters

`handlerText`	JavaScript code or a call to a JavaScript function.

Event properties used

`type`	Indicates the type of event.
`target`	Indicates the object to which the event was originally sent.
When a link is clicked, `layerX`, `layerY`, `pageX`, `pageY`, `screenX`, `screenY`	Represent the cursor location at the time the event occurred.
`which`	Represents 1 for a left-mouse click and 3 for a right-mouse click.
`modifiers`	Contains the list of modifier keys held down when the event occurred.

Description For checkboxes, links, radio buttons, reset buttons, and submit buttons, `onClick` can return false to cancel the action normally associated with a `click` event.

For example, the following code creates a link that, when clicked, displays a confirm dialog box. If the user clicks the link and then chooses cancel, the page specified by the link is not loaded.

```
<A HREF = "http://home.netscape.com/"
   onClick="return confirm('Load Netscape home page?')">
Netscape</A>
```

If the event handler returns false, the default action of the object is canceled as follows:

- Buttons—no default action; nothing is canceled
- Radio buttons and checkboxes—nothing is set
- Submit buttons—form is not submitted
- Reset buttons—form is not reset

Note In Navigator 3.0, on some platforms, returning false in an `onClick` event handler for a reset button has no effect.

Examples **Example 1: Call a function when a user clicks a button.** Suppose you have created a JavaScript function called `compute`. You can execute the `compute` function when the user clicks a button by calling the function in the onClick event handler, as follows:

```
<INPUT TYPE="button" VALUE="Calculate" onClick="compute(this.form)">
```

In the preceding example, the keyword `this` refers to the current object; in this case, the Calculate button. The construct `this.form` refers to the form containing the button.

For another example, suppose you have created a JavaScript function called `pickRandomURL` that lets you select a URL at random. You can use `onClick` to specify a value for the `HREF` attribute of the `A` tag dynamically, as shown in the following example:

```
<A HREF=""
   onClick="this.href=pickRandomURL()"
   onMouseOver="window.status='Pick a random URL'; return true">
Go!</A>
```

In the above example, `onMouseOver` specifies a custom message for the browser's status bar when the user places the mouse pointer over the Go! anchor. As this example shows, you must return true to set the `window.status` property in the `onMouseOver` event handler.

Example 2: Cancel the checking of a checkbox. The following example creates a checkbox with `onClick`. The event handler displays a confirm that warns the user that checking the checkbox purges all files. If the user chooses Cancel, `onClick` returns false and the checkbox is not checked.

```
<INPUT TYPE="checkbox" NAME="check1" VALUE="check1"
   onClick="return confirm('This purges all your files. Are you sure?')"> Remove files
```

See also For general information on event handlers, see "General Information about Events" on page 481.

For information about the `event` object, see `event`.

onDblClick

Executes JavaScript code when a DblClick event occurs; that is, when the user double-clicks a form element or a link.

Event handler for	`document`, `Link`
Implemented in	Navigator 4.0

Syntax `onDblClick="handlerText"`

Parameters

`handlerText`	JavaScript code or a call to a JavaScript function.

Note `DblClick` is not implemented on the Macintosh.

Event properties used

`type`	Indicates the type of event.
`target`	Indicates the object to which the event was originally sent.
`layerX, layerY, pageX, pageY, screenX, screenY`	Represent the cursor location at the time the event occurred.

`which`	Represents 1 for a left-mouse double-click and 3 for a right-mouse double-click.
`modifiers`	Contains the list of modifier keys held down when the event occurred.

See also For general information on event handlers, see "General Information about Events" on page 481.

For information about the `event` object, see `event`.

onDragDrop

Executes JavaScript code when a DragDrop event occurs; that is, when the user drops an object onto the browser window, such as dropping a file.

Event handler for	`Window`
Implemented in	Navigator 4.0

Syntax `onDragDrop="handlerText"`

Parameters

`handlerText`	JavaScript code or a call to a JavaScript function.

Event properties used

`type`	Indicates the type of event.
`target`	Indicates the object to which the event was originally sent.
`data`	Returns an Array of Strings containing the URLs of the dropped objects.
`modifiers`	Contains the list of modifier keys held down when the event occurred.
`screenX`, `screenY`	Represent the cursor location at the time the event occurred.

Security Getting the `data` property of the DragDrop event requires the `UniversalBrowserRead` privilege. For information on security in Navigator 4.0, see Chapter 7, "JavaScript Security," in the *JavaScript Guide*.

Description The `DragDrop` event is fired whenever a system item (file, shortcut, and so on) is dropped onto the browser window using the native system's drag and drop mechanism. The normal response for the browser is to attempt to load the item into the browser window. If the event handler for the `DragDrop` event returns true, the browser loads the item normally. If the event handler returns false, the drag and drop is canceled.

See also For general information on event handlers, see "General Information about Events" on page 481.

For information about the `event` object, see `event`.

onError

Executes JavaScript code when an error event occurs; that is, when the loading of a document or image causes an error.

Event handler for	`Image`, `Window`
Implemented in	Navigator 3.0

Syntax `onError="handlerText"`

Parameters

`handlerText`	JavaScript code or a call to a JavaScript function.

Description An error event occurs only when a JavaScript syntax or runtime error occurs, not when a browser error occurs. For example, if you try set `window.location.href='notThere.html'` and `notThere.html` does not exist, the resulting error message is a browser error message; therefore, `onError` would not intercept that message. However, an error event *is* triggered by a bad URL within an `IMG` tag or by corrupted image data.

`window.onerror` applies only to errors that occur in the window containing `window.onerror`, not in other windows.

`onError` can be any of the following:

- null to suppress all JavaScript error dialogs. Setting `window.onerror` to null means your users won't see JavaScript errors caused by your own code.

- The name of a function that handles errors (arguments are message text, URL, and line number of the offending line). To suppress the standard JavaScript error dialog, the function must return true. See Example 3 below.
- A variable or property that contains null or a valid function reference.

If you write an error-handling function, you have three options for reporting errors:

- Trace errors but let the standard JavaScript dialog report them (use an error handling function that returns false or does not return a value)
- Report errors yourself and disable the standard error dialog (use an error handling function that returns true)
- Turn off all error reporting (set the onError event handler to null)

Event properties used

`type`	Indicates the type of event.
`target`	Indicates the object to which the event was originally sent.

Examples

Example 1: Null event handler. In the following `IMG` tag, the code `onError="null"` suppresses error messages if errors occur when the image loads.

```
<IMG NAME="imageBad1" SRC="corrupt.gif" ALIGN="left" BORDER="2"
   onError="null">
```

Example 2: Null event handler for a window. The `onError` event handler for windows cannot be expressed in HTML. Therefore, you must spell it all lowercase and set it in a `SCRIPT` tag. The following code assigns null to the `onError` handler for the entire window, not just the `Image` object. This suppresses all JavaScript error messages, including those for the `Image` object.

```
<SCRIPT>
window.onerror=null
</SCRIPT>
<IMG NAME="imageBad1" SRC="corrupt.gif" ALIGN="left" BORDER="2">
```

However, if the `Image` object has a custom `onError` event handler, the handler would execute if the image had an error. This is because `window.onerror=null` suppresses JavaScript error messages, not `onError` event handlers.

```
<SCRIPT>
window.onerror=null
function myErrorFunc() {
   alert("The image had a nasty error.")
}
</SCRIPT>
<IMG NAME="imageBad1" SRC="corrupt.gif" ALIGN="left" BORDER="2"
   onError="myErrorFunc()">
```

In the following example, `window.onerror=null` suppresses all error reporting. Without `onerror=null`, the code would cause a stack overflow error because of infinite recursion.

```
<SCRIPT>
window.onerror = null;
function testErrorFunction() {
   testErrorFunction();
}
</SCRIPT>
<BODY onload="testErrorFunction()">
test message
</BODY>
```

Example 3: Error handling function. The following example defines a function, `myOnError`, that intercepts JavaScript errors. The function uses three arrays to store the message, URL, and line number for each error. When the user clicks the Display Error Report button, the `displayErrors` function opens a window and creates an error report in that window. Note that the function returns true to suppress the standard JavaScript error dialog.

```
<SCRIPT>
window.onerror = myOnError

msgArray = new Array()
urlArray = new Array()
lnoArray = new Array()

function myOnError(msg, url, lno) {
   msgArray[msgArray.length] = msg
   urlArray[urlArray.length] = url
   lnoArray[lnoArray.length] = lno
   return true
}

function displayErrors() {
   win2=window.open('','window2','scrollbars=yes')
   win2.document.writeln('<B>Error Report</B><P>')

   for (var i=0; i < msgArray.length; i++) {
      win2.document.writeln('<B>Error in file:</B> ' + urlArray[i] + '<BR>')
      win2.document.writeln('<B>Line number:</B> ' + lnoArray[i] + '<BR>')
      win2.document.writeln('<B>Message:</B> ' + msgArray[i] + '<P>')
   }
   win2.document.close()
}
</SCRIPT>

<BODY onload="noSuchFunction()">
<FORM>
<BR><INPUT TYPE="button" VALUE="This button has a syntax error"
   onClick="alert('unterminated string)">

<P><INPUT TYPE="button" VALUE="Display Error Report"
   onClick="displayErrors()">
</FORM>
```

This example produces the following output:

```
Error Report

Error in file: file:///c%7C/temp/onerror.html
Line number: 34
Message: unterminated string literal

Error in file: file:///c%7C/temp/onerror.html
Line number: 34
Message: missing ) after argument list

Error in file: file:///c%7C/temp/onerror.html
Line number: 30
Message: noSuchFunction is not defined
```

Example 4: Event handler calls a function. In the following `IMG` tag, `onError` calls the function `badImage` if errors occur when the image loads.

```
<SCRIPT>
function badImage(theImage) {
   alert('Error: ' + theImage.name + ' did not load properly.')
}
</SCRIPT>
<FORM>
<IMG NAME="imageBad2" SRC="orca.gif" ALIGN="left" BORDER="2"
   onError="badImage(this)">
</FORM>
```

See also `onAbort`, `onLoad`

For general information on event handlers, see “General Information about Events” on page 481.

For information about the `event` object, see `event`.

onFocus

Executes JavaScript code when a focus event occurs; that is, when a window, frame, or frameset receives focus or when a form element receives input focus.

Event handler for	`Button`, `Checkbox`, `FileUpload`, `Layer`, `Password`, `Radio`, `Reset`, `Select`, `Submit`, `Text`, `Textarea`, `Window`
Implemented in	Navigator 2.0 Navigator 3.0: event handler of `Button, Checkbox, FileUpload, Frame, Password, Radio, Reset, Submit,` and `Window` Navigator 4.0: event handler of `Layer`

Syntax `onFocus="handlerText"`

Parameters

`handlerText`	JavaScript code or a call to a JavaScript function.

Description The focus event can result from a `focus` method or from the user clicking the mouse on an object or window or tabbing with the keyboard. Selecting within a field results in a select event, not a focus event. `onFocus` executes JavaScript code when a focus event occurs.

A frame's `onFocus` event handler overrides an `onFocus` event handler in the `BODY` tag of the document loaded into frame.

Note that placing an alert in an `onFocus` event handler results in recurrent alerts: when you press OK to dismiss the alert, the underlying window gains focus again and produces another focus event.

Note In Navigator 3.0, on some platforms, placing an `onFocus` event handler in a `FRAMESET` tag has no effect.

Event properties used

`type`	Indicates the type of event.
`target`	Indicates the object to which the event was originally sent.

Examples The following example uses an `onFocus` handler in the `valueField` `Textarea` object to call the `valueCheck` function.

```
<INPUT TYPE="textarea" VALUE="" NAME="valueField"
   onFocus="valueCheck()">
```

See also examples for `onBlur`.

See also `onBlur`, `onChange`

For general information on event handlers, see "General Information about Events" on page 481.

For information about the `event` object, see `event`.

onKeyDown

Executes JavaScript code when a KeyDown event occurs; that is, when the user depresses a key.

Event handler for	`document`, `Image`, `Link`, `Textarea`
Implemented in	Navigator 4.0

Syntax `onKeyDown="handlerText"`

Parameters

`handlerText`	JavaScript code or a call to a JavaScript function.

Event properties used

`type`	Indicates the type of event.
`target`	Indicates the object to which the event was originally sent.
`layerX, layerY, pageX, pageY, screenX, screenY`	For an event over a window, these represent the cursor location at the time the event occurred. For an event over a form, they represent the position of the form element.
`which`	Represents the ASCII value of the key pressed. To get the actual letter, number, or symbol of the pressed key, use the `String.fromCharCode` method. To set this property when the ASCII value is unknown, use the `String.charCodeAt` method.
`modifiers`	Contains the list of modifier keys held down when the event occurred.

Description A `KeyDown` event always occurs before a `KeyPress` event. If `onKeyDown` returns false, no `KeyPress` events occur. This prevents `KeyPress` events occurring due to the user holding down a key.

See also `onKeyPress`, `onKeyUp`

For general information on event handlers, see "General Information about Events" on page 481.

For information about the `event` object, see `event`.

onKeyPress

Executes JavaScript code when a KeyPress event occurs; that is, when the user presses or holds down a key.

Event handler for	`document`, `Image`, `Link`, `Textarea`
Implemented in	Navigator 4.0

Syntax `onKeyPress="handlerText"`

Parameters

`handlerText`	JavaScript code or a call to a JavaScript function.

Event properties used

`type`	Indicates the type of event.
`target`	Indicates the object to which the event was originally sent.
`layerX, layerY, pageX, pageY, screenX, screenY`	For an event over a window, these represent the cursor location at the time the event occurred. For an event over a form, they represent the position of the form element.
`which`	Represents the ASCII value of the key pressed. To get the actual letter, number, or symbol of the pressed key, use the `String.fromCharCode` method. To set this property when the ASCII value is unknown, use the `String.charCodeAt` method.
`modifiers`	Contains the list of modifier keys held down when the event occurred.

Description A `KeyPress` event occurs immediately after a `KeyDown` event only if `onKeyDown` returns something other than false. A `KeyPress` event repeatedly occurs until the user releases the key. You can cancel individual `KeyPress` events.

See also `onKeyDown`, `onKeyUp`

For general information on event handlers, see "General Information about Events" on page 481.

For information about the `event` object, see `event`.

onKeyUp

Executes JavaScript code when a KeyUp event occurs; that is, when the user releases a key.

Event handler for	`document`, `Image`, `Link`, `Textarea`
Implemented in	Navigator 4.0

Syntax `onKeyUp="handlerText"`

Parameters

`handlerText`	JavaScript code or a call to a JavaScript function.

Event properties used

`type`	Indicates the type of event.
`target`	Indicates the object to which the event was originally sent.
`layerX, layerY, pageX, pageY, screenX, screenY`	For an event over a window, these represent the cursor location at the time the event occurred. For an event over a form, they represent the position of the form element.
`which`	Represents the ASCII value of the key pressed. To get the actual letter, number, or symbol of the pressed key, use the `String.fromCharCode` method. To set this property when the ASCII value is unknown, use the `String.charCodeAt` method.
`modifiers`	Contains the list of modifier keys held down when the event occurred.

See also For general information on event handlers, see "General Information about Events" on page 481.

For information about the `event` object, see `event`.

onLoad

Executes JavaScript code when a load event occurs; that is, when the browser finishes loading a window or all frames within a `FRAMESET` tag.

Event handler for	`Image`, `Layer`, `Window`
Implemented in	Navigator 2.0 Navigator 3.0: event handler of `Image`

Syntax `onLoad="handlerText"`

Parameters

`handlerText`	JavaScript code or a call to a JavaScript function.

Description Use the `onLoad` event handler within either the `BODY` or the `FRAMESET` tag, for example, `<BODY onLoad="...">`.

In a `FRAMESET` and `FRAME` relationship, an `onLoad` event within a frame (placed in the `BODY` tag) occurs before an `onLoad` event within the `FRAMESET` (placed in the `FRAMESET` tag).

For images, the `onLoad` event handler indicates the script to execute when an image is displayed. Do not confuse displaying an image with loading an image. You can load several images, then display them one by one in the same `Image` object by setting the object's `src` property. If you change the image displayed in this way, `onLoad` executes every time an image is displayed, not just when the image is loaded into memory.

If you specify an `onLoad` event handler for an `Image` object that displays a looping GIF animation (multi-image GIF), each loop of the animation triggers the `onLoad` event, and the event handler executes once for each loop.

You can use the `onLoad` event handler to create a JavaScript animation by repeatedly setting the `src` property of an `Image` object. See `Image` for information.

Event properties used

`type`	Indicates the type of event.
`target`	Indicates the object to which the event was originally sent.
`width, height`	For an event over a window, but not over a layer, these represent the width and height of the window.

Examples

Example 1: Display message when page loads. In the following example, the onLoad event handler displays a greeting message after a Web page is loaded.

```
<BODY onLoad="window.alert("Welcome to the Brave New World home page!")>
```

Example 2: Display alert when image loads. The following example creates two `Image` objects, one with the `Image` constructor and one with the `IMG` tag. Each `Image` object has an `onLoad` event handler that calls the `displayAlert` function, which displays an alert. For the image created with the `IMG` tag, the alert displays the image name. For the image created with the `Image` constructor, the alert displays a message without the image name. This is because the onLoad handler for an object created with the `Image` constructor must be the name of a function, and it cannot specify parameters for the `displayAlert` function.

```
<SCRIPT>
imageA = new Image(50,50)
imageA.onload=displayAlert
imageA.src="cyanball.gif"

function displayAlert(theImage) {
   if (theImage==null) {
      alert('An image loaded')
   }
   else alert(theImage.name + ' has been loaded.')
}
</SCRIPT>

<IMG NAME="imageB" SRC="greenball.gif" ALIGN="top"
   onLoad=displayAlert(this)><BR>
```

Example 3: Looping GIF animation. The following example displays an image, `birdie.gif`, that is a looping GIF animation. The onLoad event handler for the image increments the variable `cycles`, which keeps track of the number of times the animation has looped. To see the value of `cycles`, the user clicks the button labeled Count Loops.

```
<SCRIPT>
var cycles=0
</SCRIPT>
<IMG ALIGN="top" SRC="birdie.gif" BORDER=0
   onLoad="++cycles">
<INPUT TYPE="button" VALUE="Count Loops"
   onClick="alert('The animation has looped ' + cycles + ' times.')">
```

Example 4: Change GIF animation displayed. The following example uses an onLoad event handler to rotate the display of six GIF animations. Each animation is displayed in sequence in one `Image` object. When the document loads, `!anim0.html` is displayed. When that animation completes, the onLoad event handler causes the next file, `!anim1.html`, to load in place of the first file. After the last animation, `!anim5.html`, completes, the first file is again displayed. Notice that the `changeAnimation` function does not call itself after changing the `src` property of the `Image` object. This is because when the `src` property changes, the image's `onLoad` event handler is triggered and the `changeAnimation` function is called.

```
<SCRIPT>
var whichImage=0
var maxImages=5

function changeAnimation(theImage) {
   ++whichImage
   if (whichImage <= maxImages) {
      var imageName="!anim" + whichImage + ".gif"
      theImage.src=imageName
   } else {
      whichImage=-1
      return
   }
}
</SCRIPT>

<IMG NAME="changingAnimation" SRC="!anim0.gif" BORDER=0 ALIGN="top"
   onLoad="changeAnimation(this)">
```

See also examples for `Image`.

See also `onAbort`, `onError`, `onUnload`

For general information on event handlers, see "General Information about Events" on page 481.

For information about the `event` object, see `event`.

onMouseDown

Executes JavaScript code when a MouseDown event occurs; that is, when the user depresses a mouse button.

Event handler for	`Button`, `document`, `Link`
Implemented in	Navigator 4.0

Syntax `onMouseDown="handlerText"`

Parameters

`handlerText`	JavaScript code or a call to a JavaScript function.

Event properties used

`type`	Indicates the type of event.
`target`	Indicates the object to which the event was originally sent.
`layerX, layerY, pageX, pageY, screenX, screenY`	Represent the cursor location at the time the `MouseDown` event occurred.
`which`	Represents 1 for a left-mouse-button down and 3 for a right-mouse-button down.
`modifiers`	Contains the list of modifier keys held down when the `MouseDown` event occurred.

Description If `onMouseDown` returns false, the default action (entering drag mode, entering selection mode, or arming a link) is canceled.

Arming is caused by a `MouseDown` over a link. When a link is armed it changes color to represent its new state.

See also For general information on event handlers, see “General Information about Events” on page 481.

For information about the `event` object, see `event`.

onMouseMove

Executes JavaScript code when a MouseMove event occurs; that is, when the user moves the cursor.

Event handler for	None
Implemented in	Navigator 4.0

Syntax `onMouseMove="handlerText"`

Parameters

`handlerText`	JavaScript code or a call to a JavaScript function.

Event of Because mouse movement happens so frequently, by default, `onMouseMove` is not an event of any object. You must explicitly set it to be associated with a particular object.

Event properties used

`type`	Indicates the type of event.
`target`	Indicates the object to which the event was originally sent.
`layerX, layerY, pageX, pageY, screenX, screenY`	Represent the cursor location at the time the `MouseMove` event occurred.

Description The `MouseMove` event is sent only when a capture of the event is requested by an object (see “Events in Navigator 4.0” on page 482).

See also `document.captureEvents`

For general information on event handlers, see “General Information about Events” on page 481.

For information about the `event` object, see `event`.

onMouseOut

Executes JavaScript code when a MouseOut event occurs; that is, each time the mouse pointer leaves an area (client-side image map) or link from inside that area or link.

Event handler for	`Layer`, `Link`
Implemented in	Navigator 3.0

Syntax `onMouseOut="handlerText"`

Parameters

`handlerText`	JavaScript code or a call to a JavaScript function.

Description If the mouse moves from one area into another in a client-side image map, you'll get `onMouseOut` for the first area, then `onMouseOver` for the second.

`Area` objects that use the `onMouseOut` event handler must include the `HREF` attribute within the `AREA` tag.

You must return true within the event handler if you want to set the `status` or `defaultStatus` properties with `onMouseOver`.

Event properties used

`type`	Indicates the type of event.
`target`	Indicates the object to which the event was originally sent.
`layerX, layerY, pageX, pageY, screenX, screenY`	Represent the cursor location at the time the `MouseOut` event occurred.

Examples See the examples for `Link`.

See also `onMouseOver`

For general information on event handlers, see "General Information about Events" on page 481.

For information about the `event` object, see `event`.

onMouseOver

Executes JavaScript code when a MouseOver event occurs; that is, once each time the mouse pointer moves over an object or area from outside that object or area.

Event handler for	`Layer`, `Link`
Implemented in	Navigator 2.0 Navigator 3.0: event handler of `Area`

Syntax `onMouseOver="handlerText"`

Parameters

`handlerText`	JavaScript code or a call to a JavaScript function.

Description If the mouse moves from one area into another in a client-side image map, you'll get `onMouseOut` for the first area, then `onMouseOver` for the second.

`Area` objects that use `onMouseOver` must include the `HREF` attribute within the `AREA` tag.

You must return true within the event handler if you want to set the `status` or `defaultStatus` properties with `onMouseOver`.

Event properties used

`type`	Indicates the type of event.
`target`	Indicates the object to which the event was originally sent.
`layerX, layerY, pageX, pageY, screenX, screenY`	Represent the cursor location at the time the `MouseOver` event occurred.

Examples By default, the `HREF` value of an anchor displays in the status bar at the bottom of the browser when a user places the mouse pointer over the anchor. In the following example, `onMouseOver` provides the custom message "Click this if you dare."

```
<A HREF="http://home.netscape.com/"
   onMouseOver="window.status='Click this if you dare!'; return true">
Click me</A>
```

See `onClick` for an example of using `onMouseOver` when the `A` tag's `HREF` attribute is set dynamically.

See also examples for `Link`.

See also `onMouseOut`

For general information on event handlers, see "General Information about Events" on page 481.

For information about the `event` object, see `event`.

onMouseUp

Executes JavaScript code when a MouseUp event occurs; that is, when the user releases a mouse button.

Event handler for	`Button`, `document`, `Link`
Implemented in	Navigator 4.0

Syntax `onMouseUp="handlerText"`

Parameters

`handlerText`	JavaScript code or a call to a JavaScript function.

Event properties used

`type`	Indicates the type of event.
`target`	Indicates the object to which the event was originally sent.
`layerX, layerY, pageX, pageY, screenX, screenY`	Represent the cursor location at the time the `MouseUp` event occurred.
`which`	Represents 1 for a left-mouse-button up and 3 for a right-mouse-button up.
`modifiers`	Contains the list of modifier keys held down when the `MouseUp` event occurred.

Description If `onMouseUp` returns false, the default action is canceled. For example, if `onMouseUp` returns false over an armed link, the link is not triggered. Also, if `MouseUp` occurs over an unarmed link (possibly due to `onMouseDown` returning false), the link is not triggered.

Note Arming is caused by a `MouseDown` over a link. When a link is armed it changes color to represent its new state.

See also For general information on event handlers, see "General Information about Events" on page 481.

For information about the `event` object, see `event`.

onMove

Executes JavaScript code when a move event occurs; that is, when the user or script moves a window or frame.

Event handler for	`Window`
Implemented in	Navigator 4.0

Syntax `onMove="handlerText"`

Parameters

`handlerText`	JavaScript code or a call to a JavaScript function.

Event properties used

`type`	Indicates the type of event.
`target`	Indicates the object to which the event was originally sent.
`screenX`, `screenY`	Represent the position of the top-left corner of the window or frame.

See also For general information on event handlers, see "General Information about Events" on page 481.

For information about the `event` object, see `event`.

onReset

Executes JavaScript code when a reset event occurs; that is, when a user resets a form (clicks a Reset button).

Event handler for	`Form`
Implemented in	Navigator 3.0

Syntax `onReset="handlerText"`

Parameters

`handlerText`	JavaScript code or a call to a JavaScript function.

Examples The following example displays a `Text` object with the default value “CA” and a reset button. If the user types a state abbreviation in the `Text` object and then clicks the reset button, the original value of “CA” is restored. The form’s onReset event handler displays a message indicating that defaults have been restored.

```
<FORM NAME="form1" onReset="alert('Defaults have been restored.')">
State:
<INPUT TYPE="text" NAME="state" VALUE="CA" SIZE="2"><P>
<INPUT TYPE="reset" VALUE="Clear Form" NAME="reset1">
</FORM>
```

Event properties used

`type`	Indicates the type of event.
`target`	Indicates the object to which the event was originally sent.

See also `Form.reset`, `Reset`

For general information on event handlers, see “General Information about Events” on page 481.

For information about the `event` object, see `event`.

onResize

Executes JavaScript code when a resize event occurs; that is, when a user or script resizes a window or frame.

Event handler for	`Window`
Implemented in	Navigator 4.0

Syntax `onResize="handlerText"`

Parameters

`handlerText`	JavaScript code or a call to a JavaScript function.

Event properties used

`type`	Indicates the type of event.
`target`	Indicates the object to which the event was originally sent.
`width`, `height`	Represent the width and height of the window or frame.

Description This event is sent after HTML layout completes within the new window inner dimensions. This allows positioned elements and named anchors to have their final sizes and locations queried, image `SRC` properties can be restored dynamically, and so on.

See also For general information on event handlers, see "General Information about Events" on page 481.

For information about the `event` object, see `event`.

onSelect

Executes JavaScript code when a select event occurs; that is, when a user selects some of the text within a text or textarea field.

Event handler for	`Text`, `Textarea`
Implemented in	Navigator 2.0

Syntax `onSelect="handlerText"`

Parameters

`handlerText`	JavaScript code or a call to a JavaScript function.

Event properties used

`type`	Indicates the type of event.
`target`	Indicates the object to which the event was originally sent.

Examples The following example uses `onSelect` in the `valueField` `Text` object to call the `selectState` function.

```
<INPUT TYPE="text" VALUE="" NAME="valueField" onSelect="selectState()">
```

See also For general information on event handlers, see “General Information about Events” on page 481.

For information about the `event` object, see `event`.

onSubmit

Executes JavaScript code when a submit event occurs; that is, when a user submits a form.

Event handler for	`Form`
Implemented in	Navigator 2.0

Syntax `onSubmit="handlerText"`

Parameters

`handlerText`	JavaScript code or a call to a JavaScript function.

Security Navigator 4.0: Submitting a form to a `mailto:` or `news:` URL requires the `UniversalSendMail` privilege. For information on security in Navigator 4.0, see Chapter 7, “JavaScript Security,” in the *JavaScript Guide*.

Description You can use `onSubmit` to prevent a form from being submitted; to do so, put a `return` statement that returns false in the event handler. Any other returned value lets the form submit. If you omit the `return` statement, the form is submitted.

Event properties used

`type`	Indicates the type of event.
`target`	Indicates the object to which the event was originally sent.

Examples In the following example, `onSubmit` calls the `validate` function to evaluate the data being submitted. If the data is valid, the form is submitted; otherwise, the form is not submitted.

```
<FORM onSubmit="return validate(this)">
...
</FORM>
```

See also the examples for `Form`.

See also `Submit`, `Form.submit`

For general information on event handlers, see "General Information about Events" on page 481.

For information about the `event` object, see `event`.

onUnload

Executes JavaScript code when an unload event occurs; that is, when the user exits a document.

Event handler for	`Window`
Implemented in	Navigator 2.0

Syntax `onUnload="handlerText"`

Parameters

`handlerText`	JavaScript code or a call to a JavaScript function.

Description Use `onUnload` within either the `BODY` or the `FRAMESET` tag, for example, `<BODY onUnload="...">`.

In a frameset and frame relationship, an `onUnload` event within a frame (placed in the `BODY` tag) occurs before an `onUnload` event within the frameset (placed in the `FRAMESET` tag).

Event properties used

`type`	Indicates the type of event.
`target`	Indicates the object to which the event was originally sent.

Examples In the following example, `onUnload` calls the `cleanUp` function to perform some shutdown processing when the user exits a Web page:

```
<BODY onUnload="cleanUp()">
```

See also `onLoad`

For general information on event handlers, see "General Information about Events" on page 481.

For information about the `event` object, see `event`.

Chapter

10

LiveWire Database Service

This chapter contains the server-side objects associated with LiveWire: `database`, `DbPool`, `Connection`, `Cursor`, `Stproc`, `Resultset` and `blob`.

Table 10.1 summarizes the objects in this chapter.

Table 10.1 LiveWire objects

Object	Description
`blob`	Provides functionality for displaying and linking to BLOb data.
`Connection`	Represents a single database connection from a pool of connections.
`Cursor`	Represents a database cursor.
`database`	Represents a database connection.
`DbPool`	Represents a pool of database connections.
`Resultset`	Represents the information returned by a database stored procedure.
`Stproc`	Represents a database stored procedure.

database

Lets an application interact with a relational database.

Server-side object

Implemented in LiveWire 1.0
Netscape Server 3.0: added `storedProc` and `storedProcArgs` methods.

Created by The JavaScript runtime engine on the server automatically creates the `database` object. You indicate that you want to use this object by calling its `connect` method.

Description The JavaScript runtime engine on the server creates a `database` object when an application connects to a database server. Each application has only one `database` object. You can use the `database` object to interact with the database on the server. Alternatively, you can use the `DbPool` and `Connection` objects.

You can use the `database` object to connect to the database server and perform the following tasks:

- Display the results of a query as an HTML table
- Execute SQL statements on the database server
- Manage transactions
- Run stored procedures
- Handle errors returned by the target database

The scope of a database connection created with the database object is a single HTML page. That is, as soon as control leaves the HTML page, the runtime engine closes the database connection. You should close all open cursors, stored-procedure objects, and result sets before the end of the page.

If possible, your application should make the database connection on its initial page. Doing so prevents conflicts from multiple client requests trying to manipulate the status of the connections at once.

Internally, JavaScript creates the `database` object as an instance of the `DbBuiltin` class. In most circumstances, this is an implementation detail you do not need to be aware of, because you cannot create instances of this class. However, you can use the `prototype` property of the `DbBuiltin` class to add a property to the predefined `database` object. If you do so, that addition applies to the `database` object when used in all applications on your server, not just in the single application that made the change. This allows you to expand the capabilities of this object for your entire server.

Property Summary

Property	Descriptiohn
`prototype`	Allows the addition of properties to the `database` object.

Method Summary

Method	Descriptiohn
`beginTransaction`	Begins an SQL transaction.
`commitTransaction`	Commits the current SQL transaction.
`connect`	Connects to a particular configuration of database and user.
`connected`	Returns true if the database pool (and hence this connection) is connected to a database.
`cursor`	Creates a database cursor for the specified SQL SELECT statement.
`disconnect`	Disconnects all connections from the database.
`execute`	Performs the specified SQL statement.
`majorErrorCode`	Major error code returned by the database server or ODBC.
`majorErrorMessage`	Major error message returned by the database server or ODBC.
`minorErrorCode`	Secondary error code returned by vendor library.
`minorErrorMessage`	Secondary message returned by vendor library.
`rollbackTransaction`	Rolls back the current SQL transaction.
`SQLTable`	Displays query results. Creates an HTML table for results of an SQL SELECT statement.
`storedProc`	Creates a stored-procedure object and runs the specified stored procedure.

Method	Descriptiohn
`storedProcArgs`	Creates a prototype for a Sybase stored procedure.
`toString`	Returns a string representing the specified object.

Examples The following example creates a `database` object and opens a standard connection to the `customer` database on an Informix server. The name of the server is `blue`, the user name is `ADMIN`, and the password is `MANAGER`.

```
database.connect("INFORMIX", "blue", "ADMIN", "MANAGER", "inventory")
```

In this example, many clients can connect to the database simultaneously, but they all share the same connection, user name, and password.

See also `Cursor`, `database.connect`

Transactions

A *transaction* is a group of database actions that are performed together. Either all the actions succeed together or all fail together. When you attempt to have all of the actions make permanent changes to the database, you are said to *commit* a transaction. You can also *roll back* a transaction that you have not committed; this cancels all the actions.

You can use explicit transaction control for any set of actions, by using the `beginTransaction`, `commitTransaction`, and `rollbackTransaction` methods. If you do not control transactions explicitly, the runtime engine uses the underlying database's autocommit feature to treat each database modification as a separate transaction. Each statement is either committed or rolled back immediately, based on the success or failure of the individual statement. Explicitly managing transactions overrides this default behavior.

In some databases, such as Oracle, autocommit is an explicit feature that LiveWire turns on for individual statements. In others, such as Informix, it is the default behavior when you do not create a transaction.

Note You must use explicit transaction control any time you make changes to a database. If you do not, your database may return errors; even it does not, you cannot be guaranteed of data integrity without using transactions. In addition, any time you use cursors, you are encourage to use explicit transactions to control the consistency of your data.

For the `database` object, the scope of a transaction is limited to the current request (HTML page) in an application. If the application exits the page before calling the `commitTransaction` or `rollbackTransaction` method, then the transaction is automatically either committed or rolled back, depending on the setting for the `commitflag` parameter when the connection was established. This parameter is provided either to the pool object's constructor or to its `connect` method. For further information, see `connect`.

Properties

prototype

Represents the prototype for this class. You can use the prototype of the `DbBuiltin` class to add properties or methods to the `database` object. For information on prototypes, see `Function.prototype`.

Property of	`database`
Implemented in	LiveWire 1.0

Methods

beginTransaction

Begins a new SQL transaction.

Method of	`database`
Implemented in	LiveWire 1.0

Syntax `beginTransaction()`

Parameters None.

Returns 0 if the call was successful; otherwise, a nonzero status code based on any error message passed by the database. If the method returns a nonzero status code, use the associated `majorErrorCode` and `majorErrorMessage` methods to interpret the cause of the error.

Description All subsequent actions that modify the database are grouped with this transaction, known as the *current transaction*.

For the `database` object, the scope of a transaction is limited to the current request (HTML page) in the application. If the application exits the page before calling the `commitTransaction` or `rollbackTransaction` method, then the transaction is automatically either committed or rolled back, based on the setting of the `commitflag` parameter when the connection was established. This parameter is provided when you make the connection by calling `database.connect`.

For `Connection` objects, the scope of a transaction is limited to the lifetime of that object. If the connection is released or the pool of connections is closed before calling the `commitTransaction` or `rollbackTransaction` method, then the transaction is automatically either committed or rolled back, based on the setting of the `commitflag` parameter when the connection was established. This parameter is provided when you make the connection by calling the `connect` method or in the `DbPool` constructor.

If there is no current transaction (that is, if the application has not called `beginTransaction`), calls to `commitTransaction` and `rollbackTransaction` are ignored.

The LiveWire Database Service does not support nested transactions. If you call `beginTransaction` when a transaction is already open (that is, you've called `beginTransaction` and have yet to commit or roll back that transaction), you'll get an error message.

Examples This example updates the `rentals` table within a transaction. The values of `customerID` and `videoID` are passed into the `cursor` method as properties of the `request` object. When the `videoReturn` `Cursor` object opens, the `next` method navigates to the only record in the virtual table and updates the value in the `returnDate` field.

The variable `x` is assigned a database status code to indicate if the `updateRow` method is successful. If `updateRow` succeeds, the value of `x` is 0, and the transaction is committed; otherwise, the transaction is rolled back.

```
// Begin a transaction
database.beginTransaction();

// Create a Date object with the value of today's date
today = new Date();

// Create a cursor with the rented video in the virtual table
videoReturn = database.cursor("SELECT * FROM rentals WHERE
   customerId = " + request.customerID + " AND
   videoId = " + request.videoID, true);

// Position the pointer on the first row of the cursor
// and update the row
videoReturn.next()
videoReturn.returndate = today;
x = videoReturn.updateRow("rentals");

// End the transaction by committing or rolling back
if (x == 0) {
   database.commitTransaction() }
else {
   database.rollbackTransaction() }

// Close the cursor
videoReturn.close();
```

commitTransaction

Commits the current transaction.

Method of	`database`
Implemented in	LiveWire 1.0

Syntax `commitTransaction()`

Parameters None.

Returns 0 if the call was successful; otherwise, a nonzero status code based on any error message passed by the database. If the method returns a nonzero status code, use the associated `majorErrorCode` and `majorErrorMessage` methods to interpret the cause of the error.

Description This method attempts to commit all actions since the last call to `beginTransaction`.

For the `database` object, the scope of a transaction is limited to the current request (HTML page) in the application. If the application exits the page before calling the `commitTransaction` or `rollbackTransaction` method, then the transaction is automatically either committed or rolled back, based on the setting of the `commitflag` parameter when the connection was established. This parameter is provided when you make the connection with the `database` or `DbPool` object.

For `Connection` objects, the scope of a transaction is limited to the lifetime of that object. If the connection is released or the pool of connections is closed before calling the `commitTransaction` or `rollbackTransaction` method, then the transaction is automatically either committed or rolled back, based on the `commitFlag` value.

If there is no current transaction (that is, if the application has not called `beginTransaction`), calls to `commitTransaction` and `rollbackTransaction` are ignored.

The LiveWire Database Service does not support nested transactions. If you call `beginTransaction` when a transaction is already open (that is, you've called `beginTransaction` and have yet to commit or roll back that transaction), you'll get an error message.

connect

Connects the pool to a particular configuration of database and user.

Method of	`database`
Implemented in	LiveWire 1.0

Syntax

```
connect (dbtype, serverName, username, password, databaseName)

connect (dbtype, serverName, username, password, databaseName,
maxConnections)

connect (dbtype, serverName, username, password, databaseName,
maxConnections, commitflag)
```

Parameters

`dbtype`	Database type; one of `ORACLE`, `SYBASE`, `INFORMIX`, `DB2`, or `ODBC`.

`serverName`

Name of the database server to which to connect. The server name typically is established when the database is installed and is different for different database types:

DB2: Local database alias. On both NT and UNIX, this is set up by the client or the DB2 Command Line Processor.

Informix: Informix server. On NT, this is specified with the `setnet32` utility; on UNIX, in the `sqlhosts` file.

Oracle: Service. On both NT and UNIX, this specified in the `tnsnames.ora` file. On NT, you can use the SQL*Net easy configuration to specify it. If your Oracle database server is local, specify the empty string for this argument.

ODBC: Data source name. On NT, this is specified in the ODBC Administrator; on UNIX, in the .`odbc.ini` file. If you are using the Web Server as a user the file `.odbc.ini` must be in your home directory; if as a system, it must be in the root directory.

Sybase: Server name (the `DSQUERY` parameter). On NT, this is specified with the `sqledit` utility; on UNIX, with the `sybinit` utility.

If in doubt, see your database or system administrator. For ODBC, this is the name of the ODBC service as specified in Control Panel.

`userName`

Name of the user to connect to the database. Some relational database management systems (RDBMS) require that this be the same as your operating system login name; others maintain their own collections of valid user names. See your system administrator if you are in doubt.

`password`

User's password. If the database does not require a password, use an empty string ("").

`databaseName`

Name of the database to connect to for the given `serverName`. If your database server supports the notion of multiple databases on a single server, supply the name of the database to use. If it does not, use an empty string (""). For Oracle, ODBC, and DB2, you must always use an empty string.

For Oracle, specify this information in the `tnsnames.ora` file.

For ODBC, if you want to connect to a particular database, specify the database name specified in the datasource definition.

For DB2, there is no concept of a database name; the database name is always the server name (as specified with `serverName`).

`maxConnections`	(Optional) Number of connections to be created and cached in the pool. The runtime engine attempts to create as many connections as specified with this parameter. If successful, it stores those connections for later use. If you do not supply this parameter, its value is whatever you specify in the Application Manager when you install the application as the value for Built-in Maximum Database Connections. Remember that your database client license probably specifies a maximum number of connections. Do not set this parameter to a number higher than your license allows. For Sybase, you can have at most 100 connections. If your database client library is not multithreaded, it can only support one connection at a time. In this case, your application performs as though you specified 1 for this parameter. For a current list of which database client libraries are multithreaded, see the *Enterprise Server 3.0 Release Notes*
`commitFlag`	(Optional) A Boolean value indicating whether to commit a pending transaction when the connection is released or the object is finalized. (If the transaction is on a single page, the object is finalized at the end of the page. If the transaction spans multiple pages, the object is finalized when the connection returns to the pool.) If this parameter is false, a pending transaction is rolled back. If this parameter is true, a pending transaction if committed. For `DbPool`, the default value is false; for `database`, the default value is true. If you specify this parameter, you must also specify the `maxConnections` parameter.

Returns 0 if the call was successful; otherwise, a nonzero status code based on any error message passed by the database. If the method returns a nonzero status code, use the associated `majorErrorCode` and `majorErrorMessage` methods to interpret the cause of the error.

Description When you call this method, the runtime engine first closes and releases any currently open connections. It then reconnects the pool with the new configuration. You should be sure that all connections have been released before calling this method.

The first version of this method creates and caches one connection. When this connection goes out of scope, pending transactions are rolled back.

The second version of this method attempts to create as many connections as specified by the `maxConnections` parameter. If successful, it stores those connections for later use. If the runtime engine does not obtain the requested connections, it returns an error. When this connection goes out of scope, pending transactions are rolled back.

The third version of this method does everything the second version does. In addition, the `commitflag` parameter indicates what to do with pending transactions when this connection goes out of scope. If this parameter is false (the default), a pending transaction is rolled back. If this parameter is true, a pending transaction if committed.

If possible, your application should call this method on its initial page. Doing so prevents conflicts from multiple client requests trying to connect and disconnect.

Example The following statement creates four connections to an Informix database named mydb on a server named myserver, with user name SYSTEM and password MANAGER. Pending transactions are rolled back at the end of a client request:

connected

Tests whether the database pool and all of its connections are connected to a database.

Method of	`database`
Implemented in	LiveWire 1.0

Syntax `connected()`

Parameters None.

Returns True if the pool (and hence a particular connection in the pool) is currently connected to a database; otherwise, false.

Description The `connected` method indicates whether this object is currently connected to a database.

If this method returns false for a `Connection` object, you cannot use any other methods of that object. You must reconnect to the database, using the `DbPool` object, and then get a new `Connection` object. Similarly, if this method returns false for the `database` object, you must reconnect before using other methods of that object.

Example **Example 1:** The following code fragment checks to see if the connection is currently open. If it's not, it reconnects the pool and reassigns a new value to the `myconn` variable.

```
if (!myconn.connected()) {
   mypool.connect ("INFORMIX", "myserver", "SYSTEM", "MANAGER", "mydb",
4);
   myconn = mypool.connection;
}
```

Example 2: The following example uses an `if` condition to determine if an application is connected to a database server. If the application is connected, the `isConnectedRoutine` function runs; if the application is not connected, the `isNotConnected` routine runs.

```
if(database.connected()) {
   isConnectedRoutine() }
else {
   isNotConnectedRoutine() }
```

cursor

Creates a `Cursor` object.

Method of	`database`
Implemented in	LiveWire 1.0

Syntax `cursor("sqlStatement",updatable)`

Parameters

`sqlStatement`	A JavaScript string representing a SQL SELECT statement supported by the database server.
`updatable`	(Optional) A `Boolean` parameter indicating whether or not the cursor is updatable.

Returns A new `Cursor` object.

Description The `cursor` method creates a `Cursor` object that contains the rows returned by a SQL `SELECT` statement. The `SELECT` statement is passed to the `cursor` method as the `sqlStatement` argument. If the `SELECT` statement does not return any rows, the resulting `Cursor` object has no rows. The first time you use the `next` method on the object, it returns false.

You can perform the following tasks with the `Cursor` object:

- Modify data in a server table.
- Navigate in a server table.
- Customize the display of the virtual table returned by a database query.
- Run stored procedures.

The `cursor` method does not automatically display the returned data. To display this data, you must create custom HTML code. This HTML code may display the rows in an HTML table, as shown in Example 3. The `SQLTable` method is an easier way to display the output of a database query, but you cannot navigate, modify data, or control the format of the output.

The optional parameter `updatable` specifies whether you can modify the `Cursor` object you create with the `cursor` method. To create a `Cursor` object you can modify, specify `updatable` as true. If you do not specify a value for the `updatable` parameter, it is false by default.

If you create an updatable `Cursor` object, the virtual table returned by the `sqlStatement` parameter must be updatable. For example, the `SELECT` statement in the `sqlStatement` parameter cannot contain a `GROUP BY` clause; in addition, the query usually must retrieve key values from a table. For more information on constructing updatable queries, consult your database vendor's documentation.

Examples **Example 1.** The following example creates the updatable cursor `custs` and returns the columns `ID`, `CUST_NAME`, and `CITY` from the `customer` table:

```
custs = database.cursor("select id, cust_name, city from customer",
true)
```

Example 2. You can construct the SELECT statement with the string concatenation operator (+) and string variables such as `client` or `request` property values, as shown in the following example:

```
custs = database.cursor("select * from customer
   where customerID = " + request.customerID);
```

Example 3. The following example demonstrates how to format the virtual table returned by the `cursor` method as an HTML table. This example first creates `Cursor` object named `videoSet` and then displays two columns of its data (`videoSet.title` and `videoSet.synopsis`).

```
// Create the videoSet cursor
<SERVER>
videoSet = database.cursor("select * from videos
   where videos.numonhand > 0 order by title");
</SERVER>

// Begin creating an HTML table to contain the virtual table
// Specify titles for the two columns in the virtual table
<TABLE BORDER>
<CAPTION> Videos on Hand </CAPTION>
<TR>
   <TH>Title</TH>
   <TH>Synopsis</TH>
</TR>

// Use a while loop to iterate over each row in the cursor
<SERVER>
while(videoSet.next()) {
</SERVER>

// Use write statements to display the data in both columns
<TR>
   <TH><A HREF=`"rent.html?videoID="+videoSet.id`>
       <SERVER>write(videoSet.title)</SERVER></A></TH>
   <TD><SERVER>write(videoSet.synopsis)</SERVER></TD>
</TR>

// End the while loop
<SERVER>
}
</SERVER>

// End the HTML table
</TABLE>
```

The values in the `videoSet.title` column are displayed within the `A` tag so a user can click them as links. When a user clicks a title, the `rent.html` page opens and the column value `videoSet.id` is passed to it as the value of `request.videoID`.

See also `database.SQLTable`, `database.cursor`

disconnect

Disconnects all connections in the pool from the database.

Method of	`database`
Implemented in	LiveWire 1.0

Syntax `disconnect()`

Parameters None.

Returns 0 if the call was successful; otherwise, a nonzero status code based on any error message passed by the database. If the method returns a nonzero status code, use the associated `majorErrorCode` and `majorErrorMessage` methods to interpret the cause of the error.

Description Before calling the `disconnect` method, you must first call the `release` method for all connections in this database pool. Otherwise, the connection is still considered in use by the system, so the disconnect waits until all connections are released.

After disconnecting from a database, the only methods of this object you can use are `connect` and `connected`.

Examples The following example uses an `if` condition to determine if an application is connected to a database server. If the application is connected, the application calls the `disconnect` method; if the application is not connected, the `isNotConnected` routine runs.

```
if(database.connected()) {
   database.disconnect() }
else {
   isNotConnectedRoutine() }
```

execute

Performs the specified SQL statement. Use for SQL statements other than queries.

Method of	`database`
Implemented in	LiveWire 1.0

Syntax `execute (stmt)`

Parameters

`stmt`	A string representing the SQL statement to execute.

Returns 0 if the call was successful; otherwise, a nonzero status code based on any error message passed by the database. If the method returns a nonzero status code, use the associated `majorErrorCode` and `majorErrorMessage` methods to interpret the cause of the error.

Description This method enables an application to execute any data definition language (DDL) or data manipulation language (DML) SQL statement supported by the database server that does not return a cursor, such as `CREATE`, `ALTER`, or `DROP`.

Each database supports a standard core of DDL and DML statements. In addition, they may each also support DDL and DML statements specific to that database vendor. You can use `execute` to call any of those statements. However, each database vendor may also provide functions you can use with the database that are not DDL or DML statements. You cannot use `execute` to call those functions. For example, you cannot call the Oracle `describe` function or the Informix `load` function from the `execute` method.

Although technically you can use `execute` to perform data modification (`INSERT`, `UPDATE`, and `DELETE` statements), you should instead use `Cursor` objects. This makes your application more database-independent. Cursors also provide support for binary large object (BLOb) data.

When using the `execute` method, your SQL statement must strictly conform to the syntax requirements of the database server. For example, some servers require each SQL statement to be terminated by a semicolon. See your server documentation for more information.

If you have not explicitly started a transaction, the single statement is automatically committed.

Examples In the following example, the `execute` method is used to delete a customer from the `customer` table. `customer.ID` represents the unique ID of a customer that is in the ID column of the `customer` table. The value for `customer.ID` is passed into the DELETE statement as the value of the `ID` property of `request`.

```
if(request.ID != null) {
   database.execute("delete from customer
      where customer.ID = " + request.ID)
}
```

majorErrorCode

Major error code returned by the database server or ODBC.

Method of	`database`
Implemented in	LiveWire 1.0

Syntax `majorErrorCode()`

Parameters None.

Returns The result returned by this method depends on the database server being used:

- Informix: the Informix error code.
- Oracle: the code as reported by Oracle Call-level Interface (OCI).
- Sybase: the DB-Library error number or the SQL server message number.

Description SQL statements can fail for a variety of reasons, including referential integrity constraints, lack of user privileges, record or table locking in a multiuser database, and so on. When an action fails, the database server returns an error message indicating the reason for failure. The LiveWire™ Database Service provides two ways of getting error information: from the status code returned by various methods or from special properties containing error messages and codes.

Status codes are integers between 0 and 27, with 0 indicating a successful execution of the statement and other numbers indicating an error, as shown in Table 10.2.:

Table 10.2 Database status codes.

Status Code	Explanation	Status Code	Explanation
0	No error	14	Null reference parameter
1	Out of memory	15	Connection object not found
2	Object never initialized	16	Required information is missing
3	Type conversion error	17	Object cannot support multiple readers

Table 10.2 Database status codes. (Continued)

Status Code	Explanation	Status Code	Explanation
4	Database not registered	18	Object cannot support deletions
5	Error reported by server	19	Object cannot support insertions
6	Message from server	20	Object cannot support updates
7	Error from vendor's library	21	Object cannot support updates
8	Lost connection	22	Object cannot support indices
9	End of fetch	23	Object cannot be dropped
10	Invalid use of object	24	Incorrect connection supplied
11	Column does not exist	25	Object cannot support privileges
12	Invalid positioning within object (bounds error)	26	Object cannot support cursors
13	Unsupported feature	27	Unable to open

Examples This example updates the `rentals` table within a transaction. The `updateRow` method assigns a database status code to the `statusCode` variable to indicate whether the method is successful.

If `updateRow` succeeds, the value of `statusCode` is 0, and the transaction is committed. If `updateRow` returns a `statusCode` value of either five or seven, the values of `majorErrorCode`, `majorErrorMessage`, `minorErrorCode`, and `minorErrorMessage` are displayed. If `statusCode` is set to any other value, the `errorRoutine` function is called.

```
database.beginTransaction()
statusCode = cursor.updateRow("rentals")

if (statusCode == 0) {
   database.commitTransaction()
   }
```

```
if (statusCode == 5 || statusCode == 7) {
   write("The operation failed to complete.<BR>"
   write("Contact your system administrator with the following:<P>"
   write("The value of statusCode is " + statusCode + "<BR>")
   write("The value of majorErrorCode is " +
      database.majorErrorCode() + "<BR>")
   write("The value of majorErrorMessage is " +
      database.majorErrorMessage() + "<BR>")
   write("The value of minorErrorCode is " +
      database.minorErrorCode() + "<BR>")
   write("The value of minorErrorMessage is " +
      database.minorErrorMessage() + "<BR>")
   database.rollbackTransaction()
   }

else {
   errorRoutine()
   }
```

majorErrorMessage

Major error message returned by database server or ODBC. For server errors, this typically corresponds to the server's `SQLCODE`.

Method of	`database`
Implemented in	LiveWire 1.0

Syntax `majorErrorMessage()`

Parameters None.

Returns A string describing that depends on the database server:

- Informix: "Vendor Library Error: *string*," where *string* is the error text from Informix.
- Oracle: "Server Error: *string*," where *string* is the translation of the return code supplied by Oracle.
- Sybase: "Vendor Library Error: *string*," where *string* is the error text from DB-Library or "Server Error *string*," where *string* is text from the SQL server, unless the severity and message number are both 0, in which case it returns just the message text.

Description SQL statements can fail for a variety of reasons, including referential integrity constraints, lack of user privileges, record or table locking in a multiuser database, and so on. When an action fails, the database server returns an error message indicating the reason for failure. The LiveWire Database Service provides two ways of getting error information: from the status code returned by `connection` and `DbPool` methods or from special `connection` or `DbPool` properties containing error messages and codes.

Examples See `database.majorErrorCode`.

minorErrorCode

Secondary error code returned by database vendor library.

Method of	`database`
Implemented in	LiveWire 1.0

Syntax `minorErrorCode()`

Parameters None.

Returns The result returned by this method depends on the database server:

- Informix: the `ISAM` error code, or 0 if there is no `ISAM` error.
- Oracle: the operating system error code as reported by OCI.
- Sybase: the severity level, as reported by DB-Library or the severity level, as reported by the SQL server.

minorErrorMessage

Secondary message returned by database vendor library.

Method of	`database`
Implemented in	LiveWire 1.0

Syntax `minorErrorMessage()`

Parameters None.

Returns The string returned by this method depends on the database server:

- Informix: "ISAM Error: *string*," where *string* is the text of the ISAM error code from Informix, or an empty string if there is no ISAM error.
- Oracle: the Oracle server name.
- Sybase: the operating system error text, as reported by DB-Library or the SQL server name.

rollbackTransaction

Rolls back the current transaction.

Method of	`database`
Implemented in	LiveWire 1.0

Syntax `rollbackTransaction()`

Parameters None.

Returns 0 if the call was successful; otherwise, a nonzero status code based on any error message passed by the database. If the method returns a nonzero status code, use the associated `majorErrorCode` and `majorErrorMessage` methods to interpret the cause of the error.

Description This method will undo all modifications since the last call to `beginTransaction`.

For the `database` object, the scope of a transaction is limited to the current request (HTML page) in the application. If the application exits the page before calling the `commitTransaction` or `rollbackTransaction` method, then the transaction is automatically either committed or rolled back, based on the setting of the `commitflag` parameter when the connection was established. This parameter is provided when you make the connection with the `database` or `DbPool` object.

For `Connection` objects, the scope of a transaction is limited to the lifetime of that object. If the connection is released or the pool of connections is closed before calling the `commitTransaction` or `rollbackTransaction` method, then the transaction is automatically either committed or rolled back, based on the `commitFlag` value.

If there is no current transaction (that is, if the application has not called `beginTransaction`), calls to `commitTransaction` and `rollbackTransaction` are ignored.

The LiveWire Database Service does not support nested transactions. If you call `beginTransaction` when a transaction is already open (that is, you've called `beginTransaction` and have yet to commit or roll back that transaction), you'll get an error message.

SQLTable

Displays query results. Creates an HTML table for results of an SQL SELECT statement.

Method of	`database`
Implemented in	LiveWire 1.0

Syntax `SQLTable (stmt)`

Parameters

`stmt`	A string representing an SQL SELECT statement.

Returns A string representing an HTML table, with each row and column in the query as a row and column of the table.

Description Although `SQLTable` does not give explicit control over how the output is formatted, it is the easiest way to display query results. If you want to customize the appearance of the output, use a `Cursor` object to create your own display function.

Note Every Sybase table you use with a cursor must have a unique index.

Example If `connobj` is a `Connection` object and `request.sql` contains an SQL query, then the following JavaScript statements display the result of the query in a table:

```
write(request.sql)
connobj.SQLTable(request.sql)
```

The first line simply displays the SELECT statement, and the second line displays the results of the query. This is the first part of the HTML generated by these statements:

```
select * from videos
<TABLE BORDER>
<TR>
<TH>title</TH>
<TH>id</TH>
<TH>year</TH>
<TH>category</TH>
<TH>quantity</TH>
<TH>numonhand</TH>
<TH>synopsis</TH>
</TR>
<TR>
<TD>A Clockwork Orange</TD>
<TD>1</TD>
<TD>1975</TD>
<TD>Science Fiction</TD>
<TD>5</TD>
<TD>3</TD>
<TD> Little Alex, played by Malcolm Macdowell,
and his droogies stop by the Miloko bar for a
refreshing libation before a wild night on the town.
</TD>
</TR>
<TR>
<TD>Sleepless In Seattle</TD>
...
```

As this example illustrates, `SQLTable` generates an HTML table, with column headings for each column in the database table and a row in the table for each row in the database table.

storedProc

Creates a stored-procedure object and runs the specified stored procedure.

Method of	`database`
Implemented in	Netscape Server 3.0

Syntax `storedProc (procName, inarg1, inarg2, ..., inargN)`

Parameters

`procName`	A string specifying the name of the stored procedure to run.
`inarg1, ..., inargN`	The input parameters to be passed to the procedure, separated by commas.

Returns A new `Stproc` object.

Description The scope of the stored-procedure object is a single page of the application. In other words, all methods to be executed for any instance of `storedProc` must be invoked on the same application page as the page on which the object is created.

When you create a stored procedure, you can specify default values for any of the parameters. Then, if a parameter is not included when the stored procedure is executed, the procedure uses the default value. However, when you call a stored procedure from a server-side JavaScript application, you must indicate that you want to use the default value by typing `"/Default/"` in place of the parameter. (Remember that JavaScript is case sensitive.) For example:

```
spObj = connobj.storedProc ("newhire", "/Default/", 3)
```

storedProcArgs

Creates a prototype for a DB2, ODBC, or Sybase stored procedure.

Method of	`database`
Implemented in	Netscape Server 3.0

Syntax `storedProcArgs (procName, type1, ..., typeN)`

Parameters

`procName`	The name of the procedure.
`type1, ..., typeN`	Each `typeI` is one of: `"IN"`, `"OUT"`, or `"INOUT"` Specifies the type of each parameter: input (`"IN"`), output (`"OUT"`), or both input and output (`"INOUT"`).

Returns Nothing.

Description This method is only needed for DB2, ODBC, or Sybase stored procedures. If you call it for Oracle or Informix stored procedures, it does nothing.

This method provides the procedure name and the parameters for that stored procedure. Stored procedures can accept parameters that are only for input ("IN"), only for output ("OUT"), or for both input and output ("INOUT").

You must create one prototype for each DB2, ODBC, or Sybase stored procedure you use in your application. Additional prototypes for the same stored procedure are ignored.

You can specify an INOUT parameter either as an INOUT or as an OUT parameter. If you use an INOUT parameter of a stored procedure as an OUT parameter, the LiveWire Database Service implicitly passes a NULL value for that parameter.

Examples Assume the `inoutdemo` stored procedure takes one input parameter and one input/output parameter, as follows:

```
create procedure inoutdemo ( @inparam int, @inoutparam int output)
as
if ( @inoutparam == null)
@inoutparam = @inparam + 1
else
@inoutparam = @inoutparam + 1
```

Assume execute the following code and then call `outParameters(0)`, the result will be 101:

```
database.storedProcArgs("inoutdemo", "IN", "INOUT")
spobj= database.storedProc("inoutdemo", 6, 100);
answer = spobj.outParameters(0);
```

The value of `answer` is 101. On the other hand, assume you execute this code:

```
database.storedProcArgs("inoutdemo", "IN", "OUT")
spobj = database.storedProc("inoutdemo", 6, 100);
answer = spobj.outParameters(0);
```

In this case, the value of `answer` is 7.

toString

Returns a string representing the specified object.

Method of	`database`
Implemented in	LiveWire 1.0

Syntax `toString()`

Parameters None.

Description Every object has a `toString` method that is automatically called when it is to be represented as a text value or when an object is referred to in a string concatenation.

You can use `toString` within your own code to convert an object into a string, and you can create your own function to be called in place of the default `toString` method.

This method returns a string of the following format:

```
db "name" "userName" "dbtype" "serverName"
```

where

`name`	The name of the database.
`userName`	The name of the user connected to the database.
`dbType`	One of `ORACLE`, `SYBASE`, `INFORMIX, DB2`, or `ODBC`.
`serverName`	The name of the database server.

The method displays an empty string for any of attributes whose value is unknown.

For information on defining your own `toString` method, see the `Object.toString` method.

DbPool

Represents a pool of connections to a particular database configuration.

Server-side object

Implemented in Netscape Server 3.0

To connect to a database, you first create a pool of database connections and then access individual connections as needed. For more information on the general methodology for using `DbPool` objects, see *Writing Server-Side JavaScript Applications*.

Created by The `DbPool` constructor.

Description

The lifetime of a `DbPool` object (its scope) varies. Assuming it has been assigned to a variable, a `DbPool` object can go out of scope at different times:

- If the variable is a property of the `project` object (such as `project.engconn`), then it remains in scope until the application terminates or until you reassign the property to another value or to null.
- If it is a property of the `server` object (such as `server.engconn`), it remains in scope until the server goes down or until you reassign the property to another value or to null.
- In all other cases, the variable is a property of the `request` object. In this situation, the variable goes out of scope when control leaves the HTML page or you reassign the property to another value or to null.

It is your responsibility to release all connections and close all cursors, stored procedures, and result sets associated with a `DbPool` object before that object goes out of scope. Release connections and close the other objects as soon as you are done with them.

If you do not release a connection, it remains bound and is unavailable to the next user until the associated `DbPool` object goes out of scope. When you do call `release` to give up a connection, the runtime engine waits until all associated cursors, stored procedures, and result sets are closed before actually releasing the connection. Therefore, you must close those objects when you are done with them.

You can use the `prototype` property of the `DbPool` object to add a property to all `DbPool` instances. If you do so, that addition applies to all `DbPool` objects running in all applications on your server, not just in the single application that made the change. This allows you to expand the capabilities of this object for your entire server.

Property Summary

Property	Descriptiohn
`prototype`	Allows the addition of properties to a `DbPool` object.

Method Summary

Method	Descriptiohn
`DbPool`	Creates a pool of database Connection objects and optionally connects the objects to a particular configuration of database and user.
`connect`	Connects the pool to a particular configuration of database and user.
`connected`	Tests whether the database pool and all of its connections are connected to a database.
`connection`	Retrieves an available connection from the pool.
`disconnect`	Disconnects all connections in the pool from the database.
`majorErrorCode`	Major error code returned by the database server or ODBC.
`majorErrorMessage`	Major error message returned by database server or ODBC. For server errors, this typically corresponds to the server's SQLCODE.
`minorErrorCode`	Secondary error code returned by database vendor library.
`minorErrorMessage`	Secondary message returned by database vendor library.
`storedProcArgs`	Creates a prototype for a Sybase stored procedure.
`toString`	Returns a string representing the specified object.

Properties

prototype

Represents the prototype for this class. You can use the prototype to add properties or methods to all instances of a class. For information on prototypes, see `Function.prototype`.

Property of	`DbPool`
Implemented in	LiveWire 1.0

Methods

DbPool

Creates a pool of database Connection objects and optionally connects the objects to a particular configuration of database and user.

Method of	`DbPool`
Implemented in	Netscape Server 3.0

Syntax

```
new DbPool();

new DbPool (dbtype, serverName, username, password,
databaseName);

new DbPool (dbtype, serverName, username, password,
databaseName, maxConnections);

new DbPool (dbtype, serverName, username, password,
databaseName, maxConnections, commitflag);
```

Parameters

`dbtype`	Database type. One of `ORACLE`, `SYBASE`, `INFORMIX`, `DB2`, or `ODBC`.

`serverName`	Name of the database server to which to connect. The server name typically is established when the database is installed and is different for different database types: DB2: Local database alias. On both NT and UNIX, this is set up by the client or the DB2 Command Line Processor. Informix: Informix server. On NT, this is specified with the `setnet32` utility; on UNIX, in the `sqlhosts` file. Oracle: Service. On both NT and UNIX, this specified in the `tnsnames.ora` file. On NT, you can use the SQL*Net easy configuration to specify it. When your Oracle database server is local, specify the empty string for this argument. ODBC: Data source name. On NT, this is specified in the ODBC Administrator; on UNIX, in the `.odbc.ini` file. If you are using the Web Server as a user the file `.odbc.ini` must be in your home directory; if as a system, it must be in the root directory. Sybase: Server name (the DSQUERY parameter). On NT, this is specified with the `sqledit` utility; on UNIX, with the `sybinit` utility. If in doubt, see your database or system administrator. For ODBC, this is the name of the ODBC service as specified in Control Panel.
`userName`	Name of the user to connect to the database. Some relational database management systems (RDBMS) require that this be the same as your operating system login name; others maintain their own collections of valid user names. See your system administrator if you are in doubt.
`password`	User's password. If the database does not require a password, use an empty string ("").
`databaseName`	Name of the database to connect to for the given `serverName`. If your database server supports the notion of multiple databases on a single server, supply the name of the database to use. If it does not, use an empty string (""). For Oracle, ODBC, and DB2, you must always use an empty string. For Oracle, specify this information in the `tnsnames.ora` file. For ODBC, if you want to connect to a particular database, specify the database name specified in the datasource definition. For DB2, there is no concept of a database name; the database name is always the server name (as specified with `serverName`).

`maxConnections`	(Optional) Number of connections to be created and cached in the pool. The runtime engine attempts to create as many connections as specified with this parameter. If successful, it stores those connections for later use. If you do not supply this parameter, its value is 1. Remember that your database client license probably specifies a maximum number of connections. Do not set this parameter to a number higher than your license allows. For Sybase, you can have at most 100 connections. If your database client library is not multithreaded, it can only support one connection at a time. In this case, your application performs as though you specified 1 for this parameter. For a current list of which database client libraries are multithreaded, see the *Enterprise Server 3.0 Release Notes*.
`commitFlag`	(Optional) A Boolean value indicating whether to commit a pending transaction when the connection is released or the object is finalized. (If the transaction is on a single page, the object is finalized at the end of the page. If the transaction spans multiple pages, the object is finalized when the connection returns to the pool.) If this parameter is false, a pending transaction is rolled back. If this parameter is true, a pending transaction if committed. For `DbPool`, the default value is false; for `database`, the default value is true. If you specify this parameter, you must also specify the `maxConnections` parameter.

Description The first version of this constructor takes no parameters. It instantiates and allocates memory for a `DbPool` object. This version of the constructor creates and caches one connection. When this connection goes out of scope, pending transactions are rolled back.

The second version of this constructor instantiates a `DbPool` object and then calls the `connect` method to establish a database connection. This version of the constructor also creates and caches one connection. When this connection goes out of scope, pending transactions are rolled back.

The third version of this constructor instantiates a `DbPool` object and then calls the `connect` method to establish a database connection. In addition, it attempts to create as many connections as specified by the `maxConnections` parameter. If successful, it stores those connections for later use. If the runtime engine does not obtain the requested connections, it returns an error. When this connection goes out of scope, pending transactions are rolled back.

The fourth version of this constructor does everything the third version does. In addition, the `commitflag` parameter indicates what to do with pending transactions when the connection goes out of scope. If this parameter is false (the default), a pending transaction is rolled back. If this parameter is true, a pending transaction if committed.

To detect errors, you can use the `majorErrorCode` method.

If possible, your application should call this constructor and make the database connection on its initial page. Doing so prevents conflicts from multiple client requests trying to manipulate the status of the connections at once.

connect

Connects the pool to a particular configuration of database and user.

Method of	`DbPool`
Implemented in	Netscape Server 3.0

Syntax

```
connect (dbtype, serverName, username, password, databaseName)

connect (dbtype, serverName, username, password, databaseName,
maxConnections)

connect (dbtype, serverName, username, password, databaseName,
maxConnections, commitflag)
```

Parameters

`dbtype`	Database type; one of ORACLE, SYBASE, INFORMIX, DB2, or ODBC.

`serverName`	Name of the database server to which to connect. The server name typically is established when the database is installed and is different for different database types: DB2: Local database alias. On both NT and UNIX, this is set up by the client or the DB2 Command Line Processor. Informix: Informix server. On NT, this is specified with the setnet32 utility; on UNIX, in the sqlhosts file. Oracle: Service. On both NT and UNIX, this specified in the `tnsnames.ora` file. On NT, you can use the SQL*Net easy configuration to specify it. When your Oracle database server is local, specify the empty string for this argument. ODBC: Data source name. On NT, this is specified in the ODBC Administrator; on UNIX, in the .odbc.`ini` file. If you are using the Web Server as a user the file `.odbc.ini` must be in your home directory; if as a system, it must be in the root directory. Sybase: Server name (the DSQUERY parameter). On NT, this is specified with the `sqledit` utility; on UNIX, with the `sybinit` utility. If in doubt, see your database or system administrator. For ODBC, this is the name of the ODBC service as specified in Control Panel.
`userName`	Name of the user to connect to the database. Some relational database management systems (RDBMS) require that this be the same as your operating system login name; others maintain their own collections of valid user names. See your system administrator if you are in doubt.
`password`	User's password. If the database does not require a password, use an empty string ("").
`databaseName`	Name of the database to connect to for the given `serverName`. If your database server supports the notion of multiple databases on a single server, supply the name of the database to use. If it does not, use an empty string (""). For Oracle, ODBC, and DB2, you must always use an empty string. For Oracle, specify this information in the `tnsnames.ora` file. For ODBC, if you want to connect to a particular database, specify the database name specified in the datasource definition. For DB2, there is no concept of a database name; the database name is always the server name (as specified with `serverName`).

`maxConnections`	(Optional) Number of connections to be created and cached in the pool. The runtime engine attempts to create as many connections as specified with this parameter. If successful, it stores those connections for later use. If you do not supply this parameter, its value is 1. Remember that your database client license probably specifies a maximum number of connections. Do not set this parameter to a number higher than your license allows. For Sybase, you can have at most 100 connections. If your database client library is not multithreaded, it can only support one connection at a time. In this case, your application performs as though you specified 1 for this parameter. For a current list of which database client libraries are multithreaded, see the *Enterprise Server 3.0 Release Notes*.
`commitFlag`	(Optional) A Boolean value indicating whether to commit a pending transaction when the connection goes out of scope. If this parameter is false, a pending transaction is rolled back. If this parameter is true, a pending transaction if committed. For `DbPool`, the default value is false; for `database`, the default value is true. If you specify this parameter, you must also specify the `maxConnections` parameter.

Returns 0 if the call was successful; otherwise, a nonzero status code based on any error message passed by the database. If the method returns a nonzero status code, use the associated `majorErrorCode` and `majorErrorMessage` methods to interpret the cause of the error.

Description When you call this method, the runtime engine first closes and releases any currently open connections. It then reconnects the pool with the new configuration. You should be sure that all connections have been released before calling this method.

The first version of this method creates and caches one connection. When this connection goes out of scope, pending transactions are rolled back.

The second version of this method attempts to create as many connections as specified by the `maxConnections` parameter. If successful, it stores those connections for later use. If the runtime engine does not obtain the requested connections, it returns an error. When this connection goes out of scope, pending transactions are rolled back.

The third version of this method does everything the second version does. In addition, the `commitflag` parameter indicates what to do with pending transactions when this connection goes out of scope. If this parameter is false (the default), a pending transaction is rolled back. If this parameter is true, a pending transaction if committed.

Example The following statement creates four connections to an Informix database named mydb on a server named myserver, with user name SYSTEM and password MANAGER. Pending transactions are rolled back at the end of a client request:

```
pool.connect("INFORMIX", "myserver", "SYSTEM", "MANAGER", "mydb", 4)
```

connected

Tests whether the database pool and all of its connections are connected to a database.

Method of	`DbPool`
Implemented in	Netscape Server 3.0

Syntax `connected()`

Parameters None.

Returns True if the pool (and hence a particular connection in the pool) is currently connected to a database; otherwise, false.

Description The `connected` method indicates whether this object is currently connected to a database.

If this method returns false for a `Connection` object, you cannot use any other methods of that object. You must reconnect to the database, using the `DbPool` object, and then get a new `Connection` object. Similarly, if this method returns false for the `database` object, you must reconnect before using other methods of that object.

Example **Example 1:** The following code fragment checks to see if the connection is currently open. If it's not, it reconnects the pool and reassigns a new value to the `myconn` variable.

```
if (!myconn.connected()) {
   mypool.connect ("INFORMIX", "myserver", "SYSTEM", "MANAGER", "mydb",
4);
```

```
   myconn = mypool.connection;
}
```

Example 2: The following example uses an `if` condition to determine if an application is connected to a database server. If the application is connected, the `isConnectedRoutine` function runs; if the application is not connected, the `isNotConnected` routine runs.

```
if(database.connected()) {
   isConnectedRoutine() }
else {
   isNotConnectedRoutine() }
```

connection

Retrieves an available connection from the pool.

Method of	`DbPool`
Implemented in	Netscape Server 3.0

Syntax `connection (name, timeout)`

Parameters

`name`	An arbitrary name for the connection. Primarily used for debugging.
`timeout`	The number of seconds to wait for an available connection before returning. The default is to wait indefinitely. If you specify this parameter, you must also specify the `name` parameter.

Returns A new `Connection` object.

disconnect

Disconnects all connections in the pool from the database.

Method of	`DbPool`
Implemented in	Netscape Server 3.0

Syntax `disconnect()`

Parameters None.

Returns 0 if the call was successful; otherwise, a nonzero status code based on any error message passed by the database. If the method returns a nonzero status code, use the associated `majorErrorCode` and `majorErrorMessage` methods to interpret the cause of the error.

Description For the `DbPool` object, before calling the `disconnect` method, you must first call the `release` method for all connections in this database pool. Otherwise, the connection is still considered in use by the system, so the disconnect waits until all connections are released.

After disconnecting from a database, the only methods of this object you can use are `connect` and `connected`.

Examples The following example uses an `if` condition to determine if an application is connected to a database server. If the application is connected, the application calls the `disconnect` method; if the application is not connected, the `isNotConnected` routine runs.

```
if(database.connected()) {
   database.disconnect() }
else {
   isNotConnectedRoutine() }
```

majorErrorCode

Major error code returned by the database server or ODBC.

Method of	`DbPool`
Implemented in	Netscape Server 3.0

Syntax `majorErrorCode()`

Parameters None.

Returns The result returned by this method depends on the database server being used:

- Informix: the Informix error code.
- Oracle: the code as reported by Oracle Call-level Interface (OCI).
- Sybase: the DB-Library error number or the SQL server message number.

Description SQL statements can fail for a variety of reasons, including referential integrity constraints, lack of user privileges, record or table locking in a multiuser database, and so on. When an action fails, the database server returns an error message indicating the reason for failure. The LiveWire™ Database Service provides two ways of getting error information: from the status code returned by various methods or from special properties containing error messages and codes.

Status codes are integers between 0 and 27, with 0 indicating a successful execution of the statement and other numbers indicating an error, as shown in Table 10.3.

Table 10.3 Database status codes.

Status Code	Explanation	Status Code	Explanation
0	No error	14	Null reference parameter
1	Out of memory	15	Connection object not found
2	Object never initialized	16	Required information is missing
3	Type conversion error	17	Object cannot support multiple readers
4	Database not registered	18	Object cannot support deletions
5	Error reported by server	19	Object cannot support insertions
6	Message from server	20	Object cannot support updates
7	Error from vendor's library	21	Object cannot support updates
8	Lost connection	22	Object cannot support indices
9	End of fetch	23	Object cannot be dropped
10	Invalid use of object	24	Incorrect connection supplied
11	Column does not exist	25	Object cannot support privileges
12	Invalid positioning within object (bounds error)	26	Object cannot support cursors
13	Unsupported feature	27	Unable to open

Examples This example updates the `rentals` table within a transaction. The `updateRow` method assigns a database status code to the `statusCode` variable to indicate whether the method is successful.

If `updateRow` succeeds, the value of `statusCode` is 0, and the transaction is committed. If `updateRow` returns a `statusCode` value of either five or seven, the values of `majorErrorCode`, `majorErrorMessage`, `minorErrorCode`, and `minorErrorMessage` are displayed. If `statusCode` is set to any other value, the `errorRoutine` function is called.

```
database.beginTransaction()
statusCode = cursor.updateRow("rentals")

if (statusCode == 0) {
   database.commitTransaction()
   }

if (statusCode == 5 || statusCode == 7) {
   write("The operation failed to complete.<BR>"
   write("Contact your system administrator with the following:<P>"
   write("The value of statusCode is " + statusCode + "<BR>")
   write("The value of majorErrorCode is " +
      database.majorErrorCode() + "<BR>")
   write("The value of majorErrorMessage is " +
      database.majorErrorMessage() + "<BR>")
   write("The value of minorErrorCode is " +
      database.minorErrorCode() + "<BR>")
   write("The value of minorErrorMessage is " +
      database.minorErrorMessage() + "<BR>")
   database.rollbackTransaction()
   }

else {
   errorRoutine()
   }
```

majorErrorMessage

Major error message returned by database server or ODBC. For server errors, this typically corresponds to the server's SQLCODE.

Method of	`DbPool`
Implemented in	Netscape Server 3.0

Syntax `majorErrorMessage()`

Parameters None.

Returns A string describing that depends on the database server:

- Informix: "Vendor Library Error: *string*," where *string* is the error text from Informix.
- Oracle: "Server Error: *string*," where *string* is the translation of the return code supplied by Oracle.
- Sybase: "Vendor Library Error: *string*," where *string* is the error text from DB-Library or "Server Error *string*," where *string* is text from the SQL server, unless the severity and message number are both 0, in which case it returns just the message text.

Description SQL statements can fail for a variety of reasons, including referential integrity constraints, lack of user privileges, record or table locking in a multiuser database, and so on. When an action fails, the database server returns an error message indicating the reason for failure. The LiveWire Database Service provides two ways of getting error information: from the status code returned by `connection` and `DbPool` methods or from special `connection` or `DbPool` properties containing error messages and codes.

Examples See `DbPool.majorErrorCode`.

minorErrorCode

Secondary error code returned by database vendor library.

Method of	`DbPool`
Implemented in	Netscape Server 3.0

Syntax `minorErrorCode()`

Parameters None.

Returns The result returned by this method depends on the database server:

- Informix: the ISAM error code, or 0 if there is no ISAM error.
- Oracle: the operating system error code as reported by OCI.
- Sybase: the severity level, as reported by DB-Library or the severity level, as reported by the SQL server.

minorErrorMessage

Secondary message returned by database vendor library.

Method of	`DbPool`
Implemented in	Netscape Server 3.0

Syntax `minorErrorMessage()`

Parameters None.

Returns The string returned by this method depends on the database server:

- Informix: "ISAM Error: *string*," where *string* is the text of the ISAM error code from Informix, or an empty string if there is no ISAM error.
- Oracle: the Oracle server name.
- Sybase: the operating system error text, as reported by DB-Library or the SQL server name.

storedProcArgs

Creates a prototype for a DB2, ODBC, or Sybase stored procedure.

Method of	`DbPool`
Implemented in	Netscape Server 3.0

Syntax `storedProcArgs (procName, type1, ..., typeN)`

Parameters

`procName`	The name of the procedure.
`type1, ..., typeN`	Each `typeI` is one of: `"IN"`, `"OUT"`, or `"INOUT"` Specifies the type of each parameter: input (`"IN"`), output (`"OUT"`), or both input and output (`"INOUT"`).

Returns Nothing.

Description This method is only for Sybase stored procedures.

This method provides the procedure name and the parameters for that stored procedure. Sybase stored procedures can accept parameters that are only for input ("`IN`"), only for output ("`OUT`"), or for both input and output ("`INOUT`").

You must create one prototype for each Sybase stored procedure you use in your application. Additional prototypes for the same stored procedure are ignored.

You can specify an `INOUT` parameter either as an `INOUT` or as an `OUT` parameter. If you use an `INOUT` parameter of a stored procedure as an `OUT` parameter, the LiveWire Database Service implicitly passes a `NULL` value for that parameter.

Examples Assume the `inoutdemo` stored procedure takes one input parameter and one input/output parameter, as follows:

```
create procedure inoutdemo ( @inparam int, @inoutparam int output)
as
if ( @inoutparam == null)
@inoutparam = @inparam + 1
else
@inoutparam = @inoutparam + 1
```

Assume execute the following code and then call `outParameters(0)`, the result will be 101:

```
database.storedProcArgs("inoutdemo", "IN", "INOUT")
spobj= database.storedProc("inoutdemo", 6, 100);
answer = spobj.outParameters(0);
```

The value of `answer` is 101. On the other hand, assume you execute this code:

```
database.storedProcArgs("inoutdemo", "IN", "OUT")
spobj = database.storedProc("inoutdemo", 6, 100);
answer = spobj.outParameters(0);
```

In this case, the value of `answer` is 7.

toString

Returns a string representing the specified object.

Method of	`DbPool`
Implemented in	Netscape Server 3.0

Syntax `toString()`

Parameters None.

Description Every object has a `toString` method that is automatically called when it is to be represented as a text value or when an object is referred to in a string concatenation.

You can use `toString` within your own code to convert an object into a string, and you can create your own function to be called in place of the default `toString` method.

This method returns a string of the following format:

```
db "name" "userName" "dbtype" "serverName"
```

where

`name`	The name of the database.
`userName`	The name of the user connected to the database.
`dbType`	One of `ORACLE`, `SYBASE`, `INFORMIX, DB2`, or `ODBC`.
`serverName`	The name of the database server.

The method displays an empty string for any of attributes whose value is unknown.

For information on defining your own `toString` method, see the `Object.toString` method.

Connection

Represents a single database connection from a pool of connections.

Server-side object

Implemented in Netscape Server 3.0

Created by The `DbPool.connection` method. You do not call a `connection` constructor directly. Once you have a `Connection` object, you use it for your interactions with the database.

Description You can use the `prototype` property of the `Connection` class to add a property to all `Connection` instances. If you do so, that addition applies to all `Connection` objects running in all applications on your server, not just in the single application that made the change. This allows you to expand the capabilities of this object for your entire server.

Property Summary

Property	Descriptiohn
`prototype`	Allows the addition of properties to a `Connection` object.

Method Summary

Method	Descriptiohn
`beginTransaction`	Begins a new SQL transaction.
`commitTransaction`	Commits the current transaction.
`connected`	Tests whether the database pool (and hence this connection) is connected to a database.
`cursor`	Creates a database cursor for the specified SQL `SELECT` statement.
`execute`	Performs the specified SQL statement. Use for SQL statements other than queries.
`majorErrorCode`	Major error code returned by the database server or ODBC.
`majorErrorMessage`	Major error message returned by database server or ODBC.

Method	Descriptiohn
`minorErrorCode`	Secondary error code returned by database vendor library.
`minorErrorMessage`	Secondary message returned by database vendor library.
`release`	Releases the connection back to the database pool.
`rollbackTransaction`	Rolls back the current transaction.
`SQLTable`	Displays query results. Creates an HTML table for results of an `SQL SELECT` statement.
`storedProc`	Creates a stored-procedure object and runs the specified stored procedure.
`toString`	Returns a string representing the specified object.

Properties

prototype

Represents the prototype for this class. You can use the prototype to add properties or methods to all instances of a class. For information on prototypes, see `Function.prototype`.

Property of	`Connection`
Implemented in	LiveWire 1.0

Methods

beginTransaction

Begins a new SQL transaction.

Method of	`Connection`
Implemented in	Netscape Server 3.0

Syntax `beginTransaction()`

Parameters None.

Returns 0 if the call was successful; otherwise, a nonzero status code based on any error message passed by the database. If the method returns a nonzero status code, use the associated `majorErrorCode` and `majorErrorMessage` methods to interpret the cause of the error.

Description All subsequent actions that modify the database are grouped with this transaction, known as the *current transaction*.

For the `database` object, the scope of a transaction is limited to the current request (HTML page) in the application. If the application exits the page before calling the `commitTransaction` or `rollbackTransaction` method, then the transaction is automatically either committed or rolled back, based on the setting of the `commitflag` parameter when the connection was established. This parameter is provided when you make the connection by calling `database.connect`.

For `Connection` objects, the scope of a transaction is limited to the lifetime of that object. If the connection is released or the pool of connections is closed before calling the `commitTransaction` or `rollbackTransaction` method, then the transaction is automatically either committed or rolled back, based on the setting of the `commitflag` parameter when the connection was established. This parameter is provided when you make the connection by calling the `connect` method or in the `DbPool` constructor.

If there is no current transaction (that is, if the application has not called `beginTransaction`), calls to `commitTransaction` and `rollbackTransaction` are ignored.

The LiveWire Database Service does not support nested transactions. If you call `beginTransaction` when a transaction is already open (that is, you've called `beginTransaction` and have yet to commit or roll back that transaction), you'll get an error message.

Examples This example updates the `rentals` table within a transaction. The values of `customerID` and `videoID` are passed into the `cursor` method as properties of the `request` object. When the `videoReturn` `Cursor` object opens, the `next` method navigates to the only record in the answer set and updates the value in the `returnDate` field.

The variable `x` is assigned a database status code to indicate if the `updateRow` method is successful. If `updateRow` succeeds, the value of `x` is 0, and the transaction is committed; otherwise, the transaction is rolled back.

```
// Begin a transaction
database.beginTransaction();

// Create a Date object with the value of today's date
today = new Date();

// Create a Cursor with the rented video in the answer set
videoReturn = database.Cursor("SELECT * FROM rentals WHERE
   customerId = " + request.customerID + " AND
   videoId = " + request.videoID, true);

// Position the pointer on the first row of the Cursor
// and update the row
videoReturn.next()
videoReturn.returndate = today;
x = videoReturn.updateRow("rentals");

// End the transaction by committing or rolling back
if (x == 0) {
   database.commitTransaction() }
else {
   database.rollbackTransaction() }

// Close the Cursor
videoReturn.close();
```

commitTransaction

Commits the current transaction

Method of	`Connection`
Implemented in	Netscape Server 3.0

Syntax `commitTransaction()`

Parameters None.

Returns 0 if the call was successful; otherwise, a nonzero status code based on any error message passed by the database. If the method returns a nonzero status code, use the associated `majorErrorCode` and `majorErrorMessage` methods to interpret the cause of the error.

Description This method attempts to commit all actions since the last call to `beginTransaction`.

For the `database` object, the scope of a transaction is limited to the current request (HTML page) in the application. If the application exits the page before calling the `commitTransaction` or `rollbackTransaction` method, then the transaction is automatically either committed or rolled back, based on the setting of the `commitflag` parameter when the connection was established. This parameter is provided when you make the connection with the `database` or `DbPool` object.

For `Connection` objects, the scope of a transaction is limited to the lifetime of that object. If the connection is released or the pool of connections is closed before calling the `commitTransaction` or `rollbackTransaction` method, then the transaction is automatically either committed or rolled back, based on the `commitFlag` value.

If there is no current transaction (that is, if the application has not called `beginTransaction`), calls to `commitTransaction` and `rollbackTransaction` are ignored.

The LiveWire Database Service does not support nested transactions. If you call `beginTransaction` when a transaction is already open (that is, you've called `beginTransaction` and have yet to commit or roll back that transaction), you'll get an error message.

connected

Tests whether the database pool and all of its connections are connected to a database.

Method of	`Connection`
Implemented in	Netscape Server 3.0

Syntax `connected()`

Parameters None.

Returns True if the pool (and hence a particular connection in the pool) is currently connected to a database; otherwise, false.

Description The `connected` method indicates whether this object is currently connected to a database.

If this method returns false for a `Connection` object, you cannot use any other methods of that object. You must reconnect to the database, using the `DbPool` object, and then get a new `Connection` object. Similarly, if this method returns false for the `database` object, you must reconnect before using other methods of that object.

Example **Example 1:** The following code fragment checks to see if the connection is currently open. If it's not, it reconnects the pool and reassigns a new value to the `myconn` variable.

```
if (!myconn.connected()) {
   mypool.connect ("INFORMIX", "myserver", "SYSTEM", "MANAGER", "mydb",
4);
   myconn = mypool.connection;
}
```

Example 2: The following example uses an `if` condition to determine if an application is connected to a database server. If the application is connected, the `isConnectedRoutine` function runs; if the application is not connected, the `isNotConnected` routine runs.

```
if(database.connected()) {
   isConnectedRoutine() }
else {
   isNotConnectedRoutine() }
```

cursor

Creates a `Cursor` object.

Method of	`Connection`
Implemented in	Netscape Server 3.0

Syntax `cursor("sqlStatement",updatable)`

Parameters

`sqlStatement`	A JavaScript string representing a SQL SELECT statement supported by the database server.
`updatable`	(Optional) A `Boolean` parameter indicating whether or not the cursor is updatable.

Returns A new `Cursor` object.

Description The `cursor` method creates a `Cursor` object that contains the rows returned by a SQL `SELECT` statement. The `SELECT` statement is passed to the `cursor` method as the `sqlStatement` argument. If the `SELECT` statement does not return any rows, the resulting `Cursor` object has no rows. The first time you use the `next` method on the object, it returns false.

You can perform the following tasks with the `Cursor` object:

- Modify data in a server table.
- Navigate in a server table.
- Customize the display of the virtual table returned by a database query.
- Run stored procedures.

The `cursor` method does not automatically display the returned data. To display this data, you must create custom HTML code. This HTML code may display the rows in an HTML table, as shown in Example 3. The `SQLTable` method is an easier way to display the output of a database query, but you cannot navigate, modify data, or control the format of the output.

The optional parameter `updatable` specifies whether you can modify the `Cursor` object you create with the `cursor` method. To create a `Cursor` object you can modify, specify `updatable` as true. If you do not specify a value for the `updatable` parameter, it is false by default.

If you create an updatable `Cursor` object, the answer set returned by the `sqlStatement` parameter must be updatable. For example, the `SELECT` statement in the `sqlStatement` parameter cannot contain a `GROUP BY` clause; in addition, the query usually must retrieve key values from a table. For more information on constructing updatable queries, consult your database vendor's documentation.

Examples

Example 1. The following example creates the updatable cursor `custs` and returns the columns `ID`, `CUST_NAME`, and `CITY` from the `customer` table:

```
custs = database.Cursor("select id, cust_name, city from customer",
true)
```

Example 2. You can construct the SELECT statement with the string concatenation operator (+) and string variables such as `client` or `request` property values, as shown in the following example:

```
custs = database.Cursor("select * from customer
   where customerID = " + request.customerID);
```

Example 3. The following example demonstrates how to format the answer set returned by the `cursor` method as an HTML table. This example first creates `Cursor` object named `videoSet` and then displays two columns of its data (`videoSet.title` and `videoSet.synopsis`).

```
// Create the videoSet Cursor
<SERVER>
videoSet = database.cursor("select * from videos
   where videos.numonhand > 0 order by title");
</SERVER>

// Begin creating an HTML table to contain the answer set
// Specify titles for the two columns in the answer set
<TABLE BORDER>
<CAPTION> Videos on Hand </CAPTION>
<TR>
   <TH>Title</TH>
   <TH>Synopsis</TH>
</TR>

// Use a while loop to iterate over each row in the cursor
<SERVER>
while(videoSet.next()) {
</SERVER>

// Use write statements to display the data in both columns
<TR>
   <TH><A HREF=`"rent.html?videoID="+videoSet.id`>
       <SERVER>write(videoSet.title)</SERVER></A></TH>
   <TD><SERVER>write(videoSet.synopsis)</SERVER></TD>
</TR>

// End the while loop
<SERVER>
}
</SERVER>

// End the HTML table
</TABLE>
```

The values in the `videoSet.title` column are displayed within the `A` tag so a user can click them as links. When a user clicks a title, the `rent.html` page opens and the column value `videoSet.id` is passed to it as the value of `request.videoID`.

See also `Connection.SQLTable`, `Connection.cursor`

execute

Performs the specified SQL statement. Use for SQL statements other than queries.

Method of	`Connection`
Implemented in	Netscape Server 3.0

Syntax `execute (stmt)`

Parameters

`stmt`	A string representing the SQL statement to execute.

Returns 0 if the call was successful; otherwise, a nonzero status code based on any error message passed by the database. If the method returns a nonzero status code, use the associated `majorErrorCode` and `majorErrorMessage` methods to interpret the cause of the error.

Description This method enables an application to execute any data definition language (DDL) or data manipulation language (DML) SQL statement supported by the database server that does not return a Cursor, such as `CREATE`, `ALTER`, or `DROP`.

Each database supports a standard core of DDL and DML statements. In addition, they may each also support DDL and DML statements specific to that database vendor. You can use `execute` to call any of those statements. However, each database vendor may also provide functions you can use with the database that are not DDL or DML statements. You cannot use `execute` to call those functions. For example, you cannot call the Oracle `describe` function or the Informix `load` function from the `execute` method.

Although technically you can use `execute` to perform data modification (`INSERT`, `UPDATE`, and `DELETE` statements), you should instead use `Cursor` objects. This makes your application more database-independent. Cursors also provide support for binary large object (BLOb) data.

When using the `execute` method, your SQL statement must strictly conform to the syntax requirements of the database server. For example, some servers require each SQL statement to be terminated by a semicolon. See your server documentation for more information.

If you have not explicitly started a transaction, the single statement is automatically committed.

Examples In the following example, the `execute` method is used to delete a customer from the `customer` table. `customer.ID` represents the unique ID of a customer that is in the `ID` column of the `customer` table. The value for `customer.ID` is passed into the `DELETE` statement as the value of the `ID` property of the `request` object.

```
if(request.ID != null) {
   database.execute("delete from customer
      where customer.ID = " + request.ID)
}
```

majorErrorCode

Major error code returned by the database server or ODBC.

Method of	`Connection`
Implemented in	Netscape Server 3.0

Syntax `majorErrorCode()`

Parameters None.

Returns The result returned by this method depends on the database server being used:

- Informix: the Informix error code.
- Oracle: the code as reported by Oracle Call-level Interface (OCI).
- Sybase: the DB-Library error number or the SQL server message number.

Description SQL statements can fail for a variety of reasons, including referential integrity constraints, lack of user privileges, record or table locking in a multiuser database, and so on. When an action fails, the database server returns an error message indicating the reason for failure. The LiveWire™ Database Service provides two ways of getting error information: from the status code returned by various methods or from special properties containing error messages and codes.

Status codes are integers between 0 and 27, with 0 indicating a successful execution of the statement and other numbers indicating an error, as shown in Table 10.4.

Table 10.4 Database status codes.

Status Code	Explanation	Status Code	Explanation
0	No error	14	Null reference parameter
1	Out of memory	15	Connection object not found
2	Object never initialized	16	Required information is missing
3	Type conversion error	17	Object cannot support multiple readers
4	Database not registered	18	Object cannot support deletions

Table 10.4 Database status codes.

Status Code	Explanation	Status Code	Explanation
5	Error reported by server	19	Object cannot support insertions
6	Message from server	20	Object cannot support updates
7	Error from vendor's library	21	Object cannot support updates
8	Lost connection	22	Object cannot support indices
9	End of fetch	23	Object cannot be dropped
10	Invalid use of object	24	Incorrect connection supplied
11	Column does not exist	25	Object cannot support privileges
12	Invalid positioning within object (bounds error)	26	Object cannot support cursors
13	Unsupported feature	27	Unable to open

Examples This example updates the `rentals` table within a transaction. The `updateRow` method assigns a database status code to the `statusCode` variable to indicate whether the method is successful.

If `updateRow` succeeds, the value of `statusCode` is 0, and the transaction is committed. If `updateRow` returns a `statusCode` value of either five or seven, the values of `majorErrorCode`, `majorErrorMessage`, `minorErrorCode`, and `minorErrorMessage` are displayed. If `statusCode` is set to any other value, the `errorRoutine` function is called.

```
database.beginTransaction()
statusCode = cursor.updateRow("rentals")

if (statusCode == 0) {
   database.commitTransaction()
   }

if (statusCode == 5 || statusCode == 7) {
   write("The operation failed to complete.<BR>"
   write("Contact your system administrator with the following:<P>"
   write("The value of statusCode is " + statusCode + "<BR>")
   write("The value of majorErrorCode is " +
      database.majorErrorCode() + "<BR>")
   write("The value of majorErrorMessage is " +
      database.majorErrorMessage() + "<BR>")
   write("The value of minorErrorCode is " +
      database.minorErrorCode() + "<BR>")
   write("The value of minorErrorMessage is " +
      database.minorErrorMessage() + "<BR>")
   database.rollbackTransaction()
   }
else {
   errorRoutine()
   }
```

majorErrorMessage

Major error message returned by database server or ODBC. For server errors, this typically corresponds to the server's SQLCODE.

Method of	`Connection`
Implemented in	Netscape Server 3.0

Syntax `majorErrorMessage()`

Parameters None.

Returns A string describing that depends on the database server:

- Informix: "Vendor Library Error: *string*," where *string* is the error text from Informix.
- Oracle: "Server Error: *string*," where *string* is the translation of the return code supplied by Oracle.
- Sybase: "Vendor Library Error: *string*," where *string* is the error text from DB-Library or "Server Error *string*," where *string* is text from the SQL server, unless the severity and message number are both 0, in which case it returns just the message text.

Description SQL statements can fail for a variety of reasons, including referential integrity constraints, lack of user privileges, record or table locking in a multiuser database, and so on. When an action fails, the database server returns an error message indicating the reason for failure. The LiveWire Database Service provides two ways of getting error information: from the status code returned by `connection` and `DbPool` methods or from special `connection` or `DbPool` properties containing error messages and codes.

Examples See `Connection.majorErrorCode`.

minorErrorCode

Secondary error code returned by database vendor library.

Method of	`Connection`
Implemented in	Netscape Server 3.0

Syntax `minorErrorCode()`

Parameters None.

Returns The result returned by this method depends on the database server:

- Informix: the ISAM error code, or 0 if there is no ISAM error.
- Oracle: the operating system error code as reported by OCI.
- Sybase: the severity level, as reported by DB-Library or the severity level, as reported by the SQL server.

minorErrorMessage

Secondary message returned by database vendor library.

Method of	`Connection`
Implemented in	Netscape Server 3.0

Syntax `minorErrorMessage()`

Parameters None.

Returns The string returned by this method depends on the database server:

- Informix: “ISAM Error: *string*,” where *string* is the text of the ISAM error code from Informix, or an empty string if there is no ISAM error.
- Oracle: the Oracle server name.
- Sybase: the operating system error text, as reported by DB-Library or the SQL server name.

release

Releases the connection back to the database pool.

Method of	`Connection`
Implemented in	Netscape Server 3.0

Syntax `release()`

Parameters None.

Returns 0 if the call was successful; otherwise, a nonzero status code based on any error message passed by the database. If the method returns a nonzero status code, use the associated `majorErrorCode` and `majorErrorMessage` methods to interpret the cause of the error.

Description Before calling the `release` method, you should close all open cursors. When you call the `release` method, the runtime engine waits until all cursors have been closed and then returns the connection to the database pool. The connection is then available to the next user.

If you don't call the `release` method, the connection remains unavailable until the object goes out of scope. Assuming the object has been assigned to a variable, it can go out of scope at different times:

- If the variable is a property of the `project` object (such as `project.engconn`), then it remains in scope until the application terminates.
- If it is a property of the `server` object (such as `server.engconn`), it does not go out of scope until the server goes down. You rarely want to have a connection last the lifetime of the server.
- In all other cases, the variable is a property of the client request. In this situation, the variable goes out of scope when the JavaScript `finalize` method is called; that is, when control leaves the HTML page.

You must call the `release` method for all connections in a database pool before you can call the `DbPool` object's `disconnect` method. Otherwise, the connection is still considered in use by the runtime engine, so the disconnect waits until all connections are released.

rollbackTransaction

Rolls back the current transaction.

Method of	`Connection`
Implemented in	Netscape Server 3.0

Syntax `rollbackTransaction()`

Parameters None.

Returns 0 if the call was successful; otherwise, a nonzero status code based on any error message passed by the database. If the method returns a nonzero status code, use the associated `majorErrorCode` and `majorErrorMessage` methods to interpret the cause of the error.

Description This method will undo all modifications since the last call to `beginTransaction`.

For the `database` object, the scope of a transaction is limited to the current request (HTML page) in the application. If the application exits the page before calling the `commitTransaction` or `rollbackTransaction` method, then the transaction is automatically either committed or rolled back, based on the setting of the `commitflag` parameter when the connection was established. This parameter is provided when you make the connection with the `database` or `DbPool` object.

For `Connection` objects, the scope of a transaction is limited to the lifetime of that object. If the connection is released or the pool of connections is closed before calling the `commitTransaction` or `rollbackTransaction` method, then the transaction is automatically either committed or rolled back, based on the `commitFlag` value.

If there is no current transaction (that is, if the application has not called `beginTransaction`), calls to `commitTransaction` and `rollbackTransaction` are ignored.

The LiveWire Database Service does not support nested transactions. If you call `beginTransaction` when a transaction is already open (that is, you've called `beginTransaction` and have yet to commit or roll back that transaction), you'll get an error message.

SQLTable

Displays query results. Creates an HTML table for results of an SQL SELECT statement.

Method of	`Connection`
Implemented in	Netscape Server 3.0

Syntax `SQLTable (stmt)`

Parameters

`stmt`	A string representing an SQL SELECT statement.

Returns A string representing an HTML table, with each row and column in the query as a row and column of the table.

Description Although `SQLTable` does not give explicit control over how the output is formatted, it is the easiest way to display query results. If you want to customize the appearance of the output, use a `Cursor` object to create your own display function.

Note Every Sybase table you use with a cursor must have a unique index.

Example If `connobj` is a `Connection` object and `request.sql` contains an SQL query, then the following JavaScript statements display the result of the query in a table:

```
write(request.sql)
connobj.SQLTable(request.sql)
```

The first line simply displays the SELECT statement, and the second line displays the results of the query. This is the first part of the HTML generated by these statements:

```
select * from videos
<TABLE BORDER>
<TR>
<TH>title</TH>
<TH>id</TH>
<TH>year</TH>
<TH>category</TH>
<TH>quantity</TH>
<TH>numonhand</TH>
<TH>synopsis</TH>
</TR>
```

```
<TR>
<TD>A Clockwork Orange</TD>
<TD>1</TD>
<TD>1975</TD>
<TD>Science Fiction</TD>
<TD>5</TD>
<TD>3</TD>
<TD> Little Alex, played by Malcolm Macdowell,
and his droogies stop by the Miloko bar for a
refreshing libation before a wild night on the town.
</TD>
</TR>
<TR>
<TD>Sleepless In Seattle</TD>
...
```

As this example illustrates, `SQLTable` generates an HTML table, with column headings for each column in the database table and a row in the table for each row in the database table.

storedProc

Creates a stored-procedure object and runs the specified stored procedure.

Method of	`Connection`
Implemented in	Netscape Server 3.0

Syntax `storedwProc (procName, inarg1, inarg2, ..., inargN)`

Parameters

`procName`	A string specifying the name of the stored procedure to run.
`inarg1, ..., inargN`	The input parameters to be passed to the procedure, separated by commas.

Returns A new `Stproc` object.

Description The scope of the stored-procedure object is a single page of the application. In other words, all methods to be executed for any instance of `storedProc` must be invoked on the same application page as the page on which the object is created.

When you create a stored procedure, you can specify default values for any of the parameters. Then, if a parameter is not included when the stored procedure is executed, the procedure uses the default value. However, when you call a stored procedure from a server-side JavaScript application, you must indicate that you want to use the default value by typing `"/Default/"` in place of the parameter. (Remember that JavaScript is case sensitive.) For example:

```
spObj = connobj.storedProc ("newhire", "/Default/", 3)
```

toString

Returns a string representing the specified object.

Method of	`Connection`
Implemented in	Netscape Server 3.0

Syntax `toString()`

Parameters None.

Description Every object has a `toString` method that is automatically called when it is to be represented as a text value or when an object is referred to in a string concatenation.

You can use `toString` within your own code to convert an object into a string, and you can create your own function to be called in place of the default `toString` method.

This method returns a string of the following format:

```
db "name" "userName" "dbtype" "serverName"
```

where

`name`	The name of the database.
`userName`	The name of the user connected to the database.
`dbType`	One of `ORACLE`, `SYBASE`, `INFORMIX, DB2`, or `ODBC`.
`serverName`	The name of the database server.

The method displays an empty string for any of attributes whose value is unknown.

For information on defining your own `toString` method, see the `Object.toString` method.

Cursor

Server-side object. A `Cursor` object represents a database cursor for a specified SQL `SELECT` statement.

Server-side object	
Implemented in	LiveWire 1.0

Created by The `cursor` method of a `Connection` object or of the `database` object. You do not call a `Cursor` constructor.

Description A database query is said to return a `Cursor`. You can think of a Cursor as a virtual table, with rows and columns specified by the query. A cursor also has a notion of a *current row*, which is essentially a pointer to a row in the virtual table. When you perform operations with a Cursor, they usually affect the current row.

You can perform the following tasks with the `Cursor` object:

- Modify data in a database table.
- Navigate in a database table.
- Customize the display of the virtual table returned by a database query.

You can use a `Cursor` object to customize the display of the virtual table by specifying which columns and rows to display and how to display them. The `Cursor` object does not automatically display the data returned in the virtual table. To display this data, you must create HTML code such as that shown in Example 4 for the `cursor` method.

A pointer indicates the current row in a Cursor. When you create a Cursor, the pointer is initially positioned before the first row of the cursor. The `next` method makes the following row in the cursor the current row. If the `SELECT` statement used in the call to the `cursor` method does not return any rows, the method still creates a `Cursor` object. However, since that object has no rows, the first time you use the `next` method on the object, it returns false. Your application should check for this condition.

Important A database cursor does not guarantee the order or positioning of its rows. For example, if you have an updatable cursor and add a row to the cursor, you have no way of knowing where that row appears in the cursor.

When finished with a `Cursor` object, use the `close` method to close it and release the memory it uses. If you release a connection that has an open cursor, the runtime engine waits until the cursor is closed before actually releasing the connection.

If you do not explicitly close a cursor with the `close` method, the JavaScript runtime engine on the server automatically tries to close all open cursors when the associated `database` or `DbPool` object goes out of scope. This can tie up system resources unnecessarily. It can also lead to unpredictable results.

You can use the `prototype` property of the `Cursor` class to add a property to all `Cursor` instances. If you do so, that addition applies to all `Cursor` instances running in all applications on your server, not just in the single application that made the change. This allows you to expand the capabilities of this object for your entire server.

Note Every Sybase table you use with a cursor must have a unique index.

Property Summary

Property	Descriptiohn
`cursorColumn`	An array of objects corresponding to the columns in a cursor.
`prototype`	Allows the addition of properties to a `Cursor` object.

Method Summary

Method	Descriptiohn
`close`	Closes the cursor and frees the allocated memory.
`columnName`	the name of the column in the cursor corresponding to the specified number.
`columns`	Returns the number of columns in the cursor.
`deleteRow`	Deletes the current row in the specified table.
`insertRow`	Inserts a new row in the specified table.

Method	Descriptiohn
`next`	Moves the current row to the next row in the cursor.
`updateRow`	Updates records in the current row of the specified table in the cursor.

Properties

This section describes the properties of `cursor` objects.

The properties of `cursor` objects vary from instance to instance. Each `Cursor` object has a property for each named column in the cursor. In other words, when you create a cursor, it acquires a property for each column in the virtual table, as determined by the SELECT statement.

Note Unlike other properties in JavaScript, `cursor` properties corresponding to column names are not case sensitive, because SQL is not case sensitive and some databases are not case sensitive.

You can also refer to properties of a `Cursor` object as elements of an array. The 0-index array element corresponds to the first column, the 1-index array element corresponds to the second column, and so on.

SELECT statements can retrieve values that are not columns in the database, such as aggregate values and SQL expressions. You can display these values by using the cursor's property array index for the value.

cursorColumn

An array of objects corresponding to the columns in a cursor.

Property of	`Cursor`
Implemented in	LiveWire 1.0

Examples **Example 1: Using column titles as cursor properties.** The following example creates the `customerSet` `Cursor` object containing the `id`, `name`, and `city` rows from the `customer` table. The `next` method moves the pointer to the first row of the cursor. The `id`, `name`, and `city` columns become the `cursor` properties `customer.id`, `customerSet.name`, and `customerSet.city`. Because the pointer is in the first row of the cursor, the `write` method displays the values of these properties for the first row.

```
// Create a Cursor object
customerSet = database.cursor("SELECT id, name, city FROM customer")

// Navigate to the first row
customerSet.next()

write(customerSet.id + "<BR>")
write(customerSet.name + "<BR>")
write(customerSet.city + "<BR>")

// Close the cursor
customerSet.close()
```

This query might return a virtual table containing the following rows:

```
1 John Smith Anytown
2 Fred Flintstone Bedrock
3 George Jetson Spacely
```

Example 2: Iterating with the cursor properties. In this example, the `cursor` property array is used in a `for` statement to iterate over each column in the `customerSet` cursor.

```
// Create a Cursor object
customerSet = database.cursor("SELECT id, name, city FROM customer")

// Navigate to the first row
customerSet.next()

// Start a for loop
for ( var i = 0; i < customerSet.columns(); i++) {
write(customerSet[i] + "<BR>") }

// Close the cursor
customerSet.close()
```

Because the `next` statement moves the pointer to the first row, the preceding code displays values similar to the following:

```
1
John Smith
Anytown
```

Example 3. Using the cursor properties with an aggregate expression. In this example, the `salarySet` cursor contains a column created by the aggregate function MAX.

```
salarySet = database.cursor("SELECT name, MAX(salary) FROM employee")
```

Because the aggregate column does not have a name, you must use the refer to it by its index number, as follows:

```
write(salarySet[1])
```

prototype

Represents the prototype for this class. You can use the prototype to add properties or methods to all instances of a class. For information on prototypes, see `Function.prototype`.

Property of	`Cursor`
Implemented in	LiveWire 1.0

Methods

close

Closes the cursor and frees the allocated memory.

Method of	`Cursor`
Implemented in	LiveWire 1.0

Syntax `close()`

Parameters None.

Returns 0 if the call was successful; otherwise, a nonzero status code based on any error message passed by the database. If the method returns a nonzero status code, use the associated `majorErrorCode` and `majorErrorMessage` methods to interpret the cause of the error.

Description The `close` method closes a cursor or result set and releases the memory it uses. If you do not explicitly close a cursor or result set with the `close` method, the JavaScript runtime engine on the server automatically closes all open cursors and result sets when the corresponding `client` object goes out of scope.

Examples The following example creates the `rentalSet` cursor, performs certain operations on it, and then closes it with the `close` method.

```
// Create a Cursor object
rentalSet = database.cursor("SELECT * FROM rentals")

// Perform operations on the cursor
cursorOperations()

//Close the cursor
err = rentalSet.close()
```

columnName

Returns the name of the column in the cursor corresponding to the specified number.

Method of	`Cursor`
Implemented in	LiveWire 1.0

Syntax `columnName (n)`

Parameters

`n`	Zero-based integer corresponding to the column in the query. The first column in the result set is 0, the second is 1, and so on.

Returns The name of the column.

The result sets for Informix and DB2 stored procedures do not have named columns. Do not use this method when connecting to those databases. In those cases you should always refer to the result set columns by the index number.

If your `SELECT` statement uses a wildcard (*) to select all the columns in a table, the `columnName` method does not guarantee the order in which it assigns numbers to the columns. That is, suppose you have this statement:

```
custs = connobj.cursor ("select * from customer");
```

If the customer table has 3 columns, ID, NAME, and CITY, you cannot tell ahead of time which of these columns corresponds to `custs.columnName(0)`. (Of course, you are guaranteed that successive calls to `columnName` have the same result.) If the order matters to you, you can instead hard-code the column names in the select statement, as in the following statement:

```
custs = connobj.cursor ("select ID, NAME, CITY from customer");
```

With this statement, `custs.columnName(0)` is ID, `custs.columnName(1)` is NAME, and `custs.columnName(2)` is CITY.

Examples The following example assigns the name of the first column in the `customerSet` cursor to the variable `header`:

```
customerSet=database.cursor(SELECT * FROM customer ORDER BY name)
header = customerSet.columnName(0)
```

columns

Returns the number of columns in the cursor.

Method of	`Cursor`
Implemented in	LiveWire 1.0

Syntax `columns()`

Parameters None.

Returns The number of named and unnamed columns.

Examples See Example 2 of `Cursor` for an example of using the `columns` method with the `cursorColumn` array.

The following example returns the number of columns in the `custs` cursor:

```
custs.columns()
```

deleteRow

Deletes the current row in the specified table.

Method of	`Cursor`
Implemented in	LiveWire 1.0

Syntax `deleteRow (table)`

Parameters

`table`	A string specifying the name of the table from which to delete a row.

Returns 0 if the call was successful; otherwise, a nonzero status code based on any error message passed by the database. If the method returns a nonzero status code, use the associated `majorErrorCode` and `majorErrorMessage` methods to interpret the cause of the error.

Description The `deleteRow` method uses an updatable cursor to delete the current row from the specified table. See `Cursor` for information about creating an updatable cursor.

Examples In the following example, the `deleteRow` method removes a customer from the `customer` database. The `cursor` method creates the `customerSet` cursor containing a single row; the value for `customer.ID` is passed in as a `request` object property. The `next` method moves the pointer to the only row in the cursor, and the `deleteRow` method deletes the row.

```
database.beginTransaction()
   customerSet = database.cursor("select * from customer where
      customer.ID = " + request.ID, true)
   customerSet.next()
   statusCode = customerSet.deleteRow("customer")
   customerSet.close()
   if (statusCode == 0) {
      database.commitTransaction() }
   else {
      database.rollbackTransaction() }
```

In this example, the `deleteRow` method sets the value of `statusCode` to indicate whether `deleteRow` succeeds or fails. If `statusCode` is 0, the method has succeeded and the transaction is committed; otherwise, the transaction is rolled back.

insertRow

Inserts a new row in the specified table.

Method of	`Cursor`
Implemented in	LiveWire 1.0

Syntax `insertRow (table)`

Parameters

`table`	A string specifying the name of the table in which to insert a row.

Returns 0 if the call was successful; otherwise, a nonzero status code based on any error message passed by the database. If the method returns a nonzero status code, use the associated `majorErrorCode` and `majorErrorMessage` methods to interpret the cause of the error.

Description The `insertRow` method uses an updatable cursor to insert a row in the specified table. See the `cursor` method for information about creating an updatable cursor.

The location of the inserted row depends on the database vendor library. If you need to get at the row after calling the `insertRow` method, you must first close the existing cursor and then open a new cursor.

You can specify values for the row you are inserting as follows:

- By explicitly assigning values to each column in the cursor and then calling the `insertRow` method.
- By navigating to a row with the `next` method, explicitly assigning values for some columns, and then calling the `insertRow` method. Columns that are not explicitly assigned values receive values from the row to which you navigated.
- By not navigating to another record and then calling the `insertRow` method. If you do not issue a `next` method, columns that are not explicitly assigned values are null.

The `insertRow` method inserts a null value in any table columns that do not appear in the cursor.

The `insertRow` method returns a status code based on a database server message to indicate whether the method completed successfully. If successful, the method returns a 0; otherwise, it returns a nonzero integer to indicate the reason it failed. See *Writing Server-Side JavaScript Applications* for an explanation of status codes.

Examples In some applications, such as a video-rental application, a husband, wife, and children could all share the same account number but be listed under different names. In this example, a user has just added a name to the `accounts` table and wants to add a spouse's name to the same account.

```
customerSet = database.cursor("select * from customer", true)

x=true
while (x) {
   x = customerSet.next() }

customerSet.name = request.theName
customerSet.insertRow("accounts")
customerSet.close()
```

In this example, the `next` method navigates to the last row in the table, which contains the most recently added account. The value of `theName` is passed in by the `request` object and assigned to the `name` column in the `customerSet` cursor. The `insertRow` method inserts a new row at the end of the table. The value of the `name` column in the new row is the value of `theName`. Because the application used the `next` method to navigate, the value of every other column in the new row is the same as the value in the previous row.

next

Moves the current row to the next row in the cursor.

Method of	`Cursor`
Implemented in	LiveWire 1.0

Syntax `next()`

Parameters None.

Returns False if the current row is the last row; otherwise, true.

Description Initially, the pointer (or current row) for a cursor or result set is positioned before the first row returned. Use the `next` method to move the pointer through the records in the cursor or result set. This method moves the pointer to the next row and returns true as long as there is another row available. When the cursor or result set has reached the last row, the method returns false. Note that if the cursor is empty, this method always returns false.

Examples **Example 1.** This example uses the `next` method to navigate to the last row in a cursor. The variable `x` is initialized to true. When the pointer is in the last row of the cursor, the `next` method returns false and terminates the `while` loop.

```
customerSet = database.cursor("select * from customer", true)

x = true
while (x) {
   x = customerSet.next() }
```

Example 2. In the following example, the `rentalSet` cursor contains columns named `videoId`, `rentalDate`, and `dueDate`. The `next` method is called in a `while` loop that iterates over every row in the cursor. When the pointer is on the last row in the cursor, the `next` method returns false and terminates the `while` loop.

This example displays the three columns of the cursor in an HTML table:

```
<SERVER>
// Create a Cursor object
rentalSet = database.cursor("SELECT videoId, rentalDate, returnDate
   FROM rentals")
</SERVER>

// Create an HTML table
<TABLE BORDER>
<TR>
<TH>Video ID</TH>
<TD>Rental Date</TD>
<TD>Due Date</TD>
</TR>

<SERVER>
// Iterate through each row in the cursor
while (rentalSet.next()) {
</SERVER>

// Display the cursor values in the HTML table
   <TR>
   <TH><SERVER>write(rentalSet.videoId)</SERVER></TH>
   <TD><SERVER>write(rentalSet.rentalDate)</SERVER></TD>
   <TD><SERVER>write(rentalSet.returnDate)</SERVER></TD>
   </TR>

// Terminate the while loop
<SERVER>
}
</SERVER>

// End the table
</TABLE>
```

updateRow

Updates records in the current row of the specified table in the cursor.

Method of	`Cursor`
Implemented in	LiveWire 1.0

Syntax `updateRow (table)`

Parameters

`table`	String specifying the name of the table to update.

Returns 0 if the call was successful; otherwise, a nonzero status code based on any error message passed by the database. If the method returns a nonzero status code, use the associated `majorErrorCode` and `majorErrorMessage` methods to interpret the cause of the error.

Description The `updateRow` method lets you use values in the current row of an updatable cursor to modify a table. See the `cursor` method for information about creating an updatable cursor. Before performing an `updateRow`, you must perform at least one `next` with the cursor so the current row is set to a row.

Assign values to columns in the current row of the cursor, and then use the `updateRow` method to update the current row of the table specified by the `table` parameter. Column values that are not explicitly assigned are not changed by the `updateRow` method.

The `updateRow` method returns a status code based on a database server message to indicate whether the method completed successfully. If successful, the method returns a 0; otherwise, it returns a nonzero integer to indicate the reason it failed. See *Writing Server-Side JavaScript Applications* for an explanation of the individual status codes.

Examples This example uses `updateRow` to update the `returndate` column of the `rentals` table. The values of `customerID` and `videoID` are passed into the `cursor` method as properties of the `request` object. When the `videoReturn Cursor` object opens, the `next` method navigates to the only record returned and updates the value in the `returnDate` field.

```
// Create a cursor containing the rented video
videoReturn = database.cursor("SELECT * FROM rentals WHERE
   customerId = " + request.customerID + " AND
   videoId = " + request.videoID, true)

// Position the pointer on the first row of the cursor
videoReturn.next()

// Assign today’s date to the returndate column
videoReturn.returndate = today

// Update the row
videoReturn.updateRow("rentals")
```

Stproc

Represents a call to a database stored procedure.

Server-side object

Implemented in Netscape Server 3.0

Created by The `storedProc` method of the `database` object or of a `Connection` object. You do not call a `Stproc` constructor.

Description When finished with a `Stproc` object, use the `close` method to close it and release the memory it uses. If you release a connection that has an open stored procedure, the runtime engine waits until the stored procedure is closed before actually releasing the connection.

If you do not explicitly close a stored procedure with the `close` method, the JavaScript runtime engine on the server automatically tries to close all open stored procedures when the associated `database` or `Connection` object goes out of scope. This can tie up system resources unnecessarily. It can also lead to unpredictable results.

You can use the `prototype` property of the `Stproc` class to add a property to all `Stproc` instances. If you do so, that addition applies to all `Stproc` objects running in all applications on your server, not just in the single application that made the change. This allows you to expand the capabilities of this object for your entire server.

Property Summary

Property	Descriptiohn
`prototype`	Allows the addition of properties to a `Stproc` object.

Method Summary

Method	Descriptiohn
`close`	Closes a stored-procedure object.
`outParamCount`	Returns the number of output parameters returned by a stored procedure.
`outParameters`	Returns the value of the specified output parameter.

Method	Descriptiohn
`resultSet`	Returns a new result set object.
`returnValue`	Returns the return value for the stored procedure.

Properties

prototype

Represents the prototype for this class. You can use the prototype to add properties or methods to all instances of a class. For information on prototypes, see `Function.prototype`.

Property of	`Stproc`
Implemented in	LiveWire 1.0

Methods

close

Closes the stored procedure and frees the allocated memory.

Method of	`Stproc`
Implemented in	Netscape Server 3.0

Syntax `close()`

Parameters None.

Returns 0 if the call was successful; otherwise, a nonzero status code based on any error message passed by the database. If the method returns a nonzero status code, use the associated `majorErrorCode` and `majorErrorMessage` methods to interpret the cause of the error.

Description The `close` method closes a stored procedure and releases the memory it uses. If you do not explicitly close a stored procedure with the `close` method, the JavaScript runtime engine on the server automatically closes it when the corresponding `client` object goes out of scope.

outParamCount

Returns the number of output parameters returned by a stored procedure.

Method of	`Stproc`
Implemented in	Netscape Server 3.0

Syntax `outParamCount()`

Parameters None.

Returns The number of output parameters for the stored procedure. Informix stored procedures do not have output parameters. Therefore for Informix, this method always returns 0. You should always call this method before calling `outParameters`, to ensure that the stored procedure has output parameters.

outParameters

Returns the value of the specified output parameter.

Method of	`Stproc`
Implemented in	Netscape Server 3.0

Syntax `outParameters (n)`

Parameters

`n`	Zero-based ordinal for the output parameter to return.

Returns The value of the specified output parameter. This can be a string, number, double, or object.

Description Do not use this method for Informix stored procedures, because they do not have output parameters.

You should always call the `outParamCount` method before you call this method. If `outParamCount` returns 0, the stored procedure has no output parameters. In this situation, do not call this method.

You must retrieve result set objects before you call this method. Once you call this method, you can't get any more data from a result set, and you can't get any additional result sets.

resultSet

Returns a new result set object.

Method of	`Stproc`
Implemented in	Netscape Server 3.0

Syntax `resultSet ()`

Parameters None.

Description Running a stored procedure can create 0 or more result sets. You access the result sets in turn by repeated calls to the `resultSet` method. See the description of the `Resultset` for restrictions on when you can use this method access the result sets for a stored procedure.

```
spobj = connobj.storedProc("getcusts");

// Creates a new result set object
resobj = spobj.resultSet();
```

returnValue

Returns the return value for the stored procedure.

Method of	`Stproc`
Implemented in	Netscape Server 3.0

Syntax `returnValue()`

Parameters None.

Returns For Sybase, this method always returns the return value of the stored procedure.

For Oracle, this method returns null if the stored procedure did not return a value or the return value of the stored procedure.

For Informix, DB2, and ODBC, this method always returns null.

Description You must retrieve result set objects before you call this method. Once you call this method, you can't get any more data from a result set, and you can't get any additional result sets.

Resultset

Represents a virtual table created by executing a stored procedure.

Server-side object

Implemented in Netscape Server 3.0

Created by The `resultSet` method of a `Stproc` object. The `Resultset` object does not have a constructor.

Description For Sybase, Oracle, ODBC, and DB2 stored procedures, the stored-procedure object has one result set object for each `SELECT` statement executed by the stored procedure. For Informix stored procedures, the stored-procedure object always has one result set object.

A result set has a property for each column in the `SELECT` statement used to generate the result set. For Sybase, Oracle, and ODBC stored procedures, you can refer to these properties by the name of the column in the virtual table. For Informix and DB2 stored procedures, the columns are not named. For these databases, you must use a numeric index to refer to the column.

Result set objects are not valid indefinitely. In general, once a stored procedure starts, no interactions are allowed between the database client and the database server until the stored procedure has completed. In particular, there are three circumstances that cause a result set to be invalid:

1. If you create a result set as part of a transaction, you must finish using the result set during that transaction. Once you either commit or rollback the transaction, you can't get any more data from a result set, and you can't get any additional result sets. For example, the following code is illegal:

```
database.beginTransaction();
```

```
spobj = database.storedProc("getcusts");
resobj = spobj.resultSet();
database.commitTransaction();
/* Illegal! Result set no longer valid! */
col1 = resobj[0];
```

2. You must retrieve result set objects before you call a stored-procedure object's `returnValue` or `outParameters` methods. Once you call either of these methods, you can't get any more data from a result set, and you can't get any additional result sets.

```
spobj = database.storedProc("getcusts");
resobj = spobj.resultSet();
retval = spobj.returnValue();
/* Illegal! Result set no longer valid! */
col1 = resobj[0];
```

3. Similarly, you must retrieve result set objects before you call the associated `Connection` object's `cursor` or `SQLTable` method. For example, the following code is illegal:

```
spobj = database.storedProc("getcusts");
cursobj = database.cursor("SELECT * FROM ORDERS;");
/* Illegal! The result set is no longer available! */
resobj = spobj.resultSet();
col1 = resobj[0];
```

When finished with a `Resultset` object, use the `close` method to close it and release the memory it uses. If you release a connection that has an open result set, the runtime engine waits until the result set is closed before actually releasing the connection.

If you do not explicitly close a result set with the `close` method, the JavaScript runtime engine on the server automatically tries to close all open result sets when the associated `database` or `DbPool` object goes out of scope. This can tie up system resources unnecessarily. It can also lead to unpredictable results.

You can use the `prototype` property of the `Resultset` class to add a property to all `Resultset` instances. If you do so, that addition applies to all `Resultset` objects running in all applications on your server, not just in the single application that made the change. This allows you to expand the capabilities of this object for your entire server.

Property Summary

Property	Descriptiohn
`prototype`	Allows the addition of properties to a `Resultset` object.

Method Summary

Method	Descriptiohn
`close`	Closes a result set object.
`columnName`	Returns the name of a column in the result set.
`columns`	Returns the number of columns in the result set.
`next`	Moves the current row to the next row in the result set.

Examples Assume you have the following Oracle stored procedure:

```
create or replace package timpack
as type timcurtype is ref cursor return customer%rowtype;
type timrentype is ref cursor return rentals%rowtype;
end timpack;
```

```
create or replace procedure timset4(timrows1 in out timpack.timcurtype,
timrows in out timpack.timrentype)
as begin
open timrows for select * from rentals;
open timrows1 for select * from customer;
end timset4;
```

Running this stored procedure creates two result sets you can access. In the following code fragment the `resobj1` result set has rows returned by the `timrows` ref cursor and the `resobj2` result set has the rows returned by the `timrows1` ref cursor.

```
spobj = database.storedProc("timset4");
resobj1 = spobj.resultSet();
resobj2 = spobj.resultSet();
```

Properties

prototype

Represents the prototype for this class. You can use the prototype to add properties or methods to all instances of a class. For information on prototypes, see `Function.prototype`.

Property of	`Resultset`
Implemented in	LiveWire 1.0

Methods

close

Closes the result set and frees the allocated memory.

Method of	`Resultset`
Implemented in	Netscape Server 3.0

Syntax `close()`

Parameters None.

Returns 0 if the call was successful; otherwise, a nonzero status code based on any error message passed by the database. If the method returns a nonzero status code, use the associated `majorErrorCode` and `majorErrorMessage` methods to interpret the cause of the error.

Description The `close` method closes a cursor or result set and releases the memory it uses. If you do not explicitly close a cursor or result set with the `close` method, the JavaScript runtime engine on the server automatically closes all open cursors and result sets when the corresponding `client` object goes out of scope.

Examples The following example creates the `rentalSet` cursor, performs certain operations on it, and then closes it with the `close` method.

```
// Create a Cursor object
rentalSet = database.cursor("SELECT * FROM rentals")
```

```
// Perform operations on the cursor
cursorOperations()

//Close the cursor
err = rentalSet.close()
```

See also `Cursor`

columnName

Returns the name of the column in the result set corresponding to the specified number.

Method of	`Resultset`
Implemented in	Netscape Server 3.0

Syntax `columnName (n)`

Parameters

`n`	Zero-based integer corresponding to the column in the query. The first column in the result set is 0, the second is 1, and so on.

Returns The name of the column. For Informix stored procedures, this method for the `Resultset` object always returns the string "Expression".

If your `SELECT` statement uses a wildcard (*) to select all the columns in a table, the `columnName` method does not guarantee the order in which it assigns numbers to the columns. That is, suppose you have this statement:

```
resSet = stObj.resultSet("select * from customer");
```

If the customer table has 3 columns, ID, NAME, and CITY, you cannot tell ahead of time which of these columns corresponds to `resSet.columnName(0)`. (Of course, you are guaranteed that successive calls to `columnName` have the same result.) If the order matters to you, you can instead hard-code the column names in the select statement, as in the following statement:

```
resSet = stObj.resultSet("select ID, NAME, CITY from customer");
```

With this statement, `resSet.columnName(0)` is ID, `resSet.columnName(1)` is NAME, and `resSet.columnName(2)` is CITY.

Examples The following example assigns the name of the first column in the `customerSet` cursor to the variable `header`:

```
customerSet=database.cursor(SELECT * FROM customer ORDER BY name)
header = customerSet.columnName(0)
```

columns

Returns the number of columns in the result set.

Method of	`Resultset`
Implemented in	Netscape Server 3.0

Syntax `columns()`

Parameters None.

Returns The number of named and unnamed columns.

Examples See Example 2 of `Cursor` for an example of using the `columns` method with the `cursorColumn` array.

The following example returns the number of columns in the `custs` cursor:

```
custs.columns()
```

next

Moves the current row to the next row in the result set.

Method of	`Resultset`
Implemented in	Netscape Server 3.0

Syntax `next()`

Parameters None.

Returns False if the current row is the last row; otherwise, true.

Description Initially, the pointer (or current row) for a cursor or result set is positioned before the first row returned. Use the `next` method to move the pointer through the records in the cursor or result set. This method moves the pointer to the next row and returns true as long as there is another row available. When the cursor or result set has reached the last row, the method returns false. Note that if the cursor is empty, this method always returns false.

Examples **Example 1.** This example uses the `next` method to navigate to the last row in a cursor. The variable `x` is initialized to true. When the pointer is in the last row of the cursor, the `next` method returns false and terminates the `while` loop.

```
customerSet = database.cursor("select * from customer", true)

x = true
while (x) {
   x = customerSet.next() }
```

Example 2. In the following example, the `rentalSet` cursor contains columns named `videoId`, `rentalDate`, and `dueDate`. The `next` method is called in a `while` loop that iterates over every row in the cursor. When the pointer is on the last row in the cursor, the `next` method returns false and terminates the `while` loop.

This example displays the three columns of the cursor in an HTML table:

```
<SERVER>
// Create a Cursor object
rentalSet = database.cursor("SELECT videoId, rentalDate, returnDate
   FROM rentals")
</SERVER>

// Create an HTML table
<TABLE BORDER>
<TR>
<TH>Video ID</TH>
<TD>Rental Date</TD>
<TD>Due Date</TD>
</TR>

<SERVER>
// Iterate through each row in the cursor
while (rentalSet.next()) {
</SERVER>

// Display the cursor values in the HTML table
   <TR>
   <TH><SERVER>write(rentalSet.videoId)</SERVER></TH>
   <TD><SERVER>write(rentalSet.rentalDate)</SERVER></TD>
   <TD><SERVER>write(rentalSet.returnDate)</SERVER></TD>
   </TR>

// Terminate the while loop
<SERVER>
}
</SERVER>

// End the table
</TABLE>
```

blob

Server-side object. Provides functionality for displaying and linking to BLOb data.

Server-side object

Implemented in LiveWire 1.0

Created by You do not create a separate `blob` object. Instead, if you know that the value of a `cursor` property contains BLOb data, you use these methods to access that data:

Method	Descriptiohn
`blobImage`	Displays BLOb data stored in a database.
`blobLink`	Displays a link that references BLOb data with a link.

Conversely, to store BLOb data in a database, use the `blob` function.

Methods

blobImage

Displays BLOb data stored in a database.

Method of `blob`

Implemented in LiveWire 1.0

Syntax

```
cursorName.colName.blobImage (format, altText, align,
widthPixels, heightPixels, borderPixels, ismap)
```

Parameters

`format`	The image format. This can be GIF, JPEG, or any other MIME image format. The acceptable formats are specified in the `type=image` section of the file `$nshome\httpd-80\config\mime.types`, where `$nshome` is the directory in which you installed your server. The client browser must also be able to display the image format.
`altText`	(Optional) The value of the ALT attribute of the `image` tag. This indicates text to display if the client browser does not display images.
`align`	(Optional) The value of the `ALIGN` attribute of the `image` tag. This can be `"left"`, `"right"`, or any other value supported by the client browser.
`widthPixels`	(Optional) The width of the image in pixels.
`heightPixels`	(Optional) The height of the image in pixels.
`borderPixels`	(Optional) The size of the outline border in pixels if the image is a link.
`ismap`	(Optional) True if the image is a clickable map. If this parameter is true, the `image` tag has an ISMAP attribute; otherwise it does not.

Returns An HTML IMG tag for the specified image type.

Description Use `blobImage` to create an HTML image tag for a graphic image in a standard format such as GIF or JPEG.

The `blobImage` method fetches a BLOb from the database, creates a temporary file (in memory) of the specified format, and generates an HTML image tag that refers to the temporary file. The JavaScript runtime engine removes the temporary file after the page is generated and sent to the client.

While creating the page, the runtime engine keeps the binary data that `blobImage` fetches from the database in active memory, so requests that fetch a large amount of data can exceed dynamic memory on the server. Generally it is good practice to limit the number of rows retrieved at one time using `blobImage` to stay within the server's dynamic memory limits.

Examples **Example 1.** The following example extracts a row containing a small image and a name. It writes HTML containing the name and a link to the image:

```
cursor = connobj.cursor("SELECT NAME, THUMB FROM FISHTBL WHERE ID=2")
write(cursor.name + " ")
write(cursor.thumb.blobImage("gif"))
write("<BR>")
cursor.close()
```

These statements produce this HTML:

```
Anthia <IMG SRC="LIVEWIRE_TEMP11"><BR>
```

Example 2. The following example creates a cursor from the `rockStarBios` table and uses `blobImage` to display an image retrieved from the `photos` column:

```
cursor = database.cursor("SELECT * FROM rockStarBios
   WHERE starID = 23")
while(cursor.next()) {
   write(cursor.photos.blobImage("gif", "Picture", "left",
      30, 30, 0,false))
}
cursor.close()
```

This example displays an image as if it were created by the following HTML:

```
<IMG SRC="livewire_temp.gif" ALT="Picture" ALIGN=LEFT
   WIDTH=30 HEIGHT=30 BORDER=0>
```

The `livewire_temp.gif` file in this example is the file in which the `rockStarBios` table stores the BLOb data.

blobLink

Returns a link tag that references BLOb data with a link. Creates an HTML link to the BLOb.

Method of	`blob`
Implemented in	LiveWire 1.0

Syntax `cursorName.colName.blobLink (mimeType, linkText)`

Parameters

`mimeType`	The MIME type of the binary data. This can be image/gif or any other acceptable MIME type, as specified in the Netscape server configuration file $nshome\httpd-80\config\mime.types, where `$nshome` is the directory in which you installed your server.
`linkText`	The text to display in the link. This can be any JavaScript string expression.

Returns An HTML link tag.

Description Use `blobLink` if you do not want to display graphics (to reduce bandwidth requirements) or if you want to provide a link to an audio clip or other multimedia content not viewable inline.

The `blobLink` method fetches BLOb data from the database, creates a temporary file in memory, and generates a hypertext link to the temporary file. The JavaScript runtime engine on the server removes the temporary files that `blobLink` creates after the user clicks the link or sixty seconds after the request has been processed.

The runtime engine keeps the binary data that `blobLink` fetches from the database in active memory, so requests that fetch a large amount of data can exceed dynamic memory on the server. Generally it is good practice to limit the number of rows retrieved at one time using `blobLink` to stay within the server's dynamic memory limits.

Example **Example 1.** The following statements extract a row containing a large image and a name. It writes HTML containing the name and a link to the image:

```
cursor = connobj.cursor("SELECT NAME, PICTURE FROM FISHTBL WHERE ID=2")
write(cursor.name + " ")
write(cursor.picture.blobLink("image/gif", "Link" + cursor.id))
write("<BR>")
cursor.close()
```

These statements produce this HTML:

```
Anthia <A HREF="LIVEWIRE_TEMP2">Link2</A><BR>
```

Example 2. The following example creates a cursor from the `rockStarBios` table and uses `blobLink` to create links to images retrieved from the `photos` column:

```
write("Click a link to display an image:<P>")
cursor = database.cursor("select * from rockStarBios")
while(cursor.next()) {
   write(cursor.photos.blobLink("image/gif", "Image " + cursor.id))
   write("<BR>")
}
cursor.close()
```

This example generates the following HTML:

```
Click a link to display an image:<P>
<A HREF="LIVEWIRE_TEMP1">Image 1</A><BR>
<A HREF="LIVEWIRE_TEMP2">Image 2</A><BR>
<A HREF="LIVEWIRE_TEMP3">Image 3</A><BR>
```

Chapter

11

Session Management Service

This chapter contains those server-side objects associated with managing a session, including `request`, `client`, `project`, `server`, and `Lock`.

Table 11.1 summarizes the objects in this chapter.

Table 11.1 Session management objects

Object	Description
`client`	Encapsulates information about a client/application pair, allowing that information to last longer than a single HTTP request.
`Lock`	Provides functionality for safely sharing data among requests, clients, and applications.
`project`	Encapsulates information about an application that lasts until the application is stopped on the server.
`request`	Encapsulates information about a single HTTP request.
`server`	Encapsulates global information about the server that lasts until the server is stopped.

request

Contains data specific to the current client request.

Server-side object

Implemented in LiveWire 1.0

Created by The JavaScript runtime engine on the server automatically creates a `request` object for each client request.

Description The JavaScript runtime engine on the server creates a `request` object each time the client makes a request of the server. The runtime engine destroys the `request` object after the server responds to the request, typically by providing the requested page.

The properties listed below are read-only properties that are initialized automatically when a `request` object is created. In addition to these predefined properties, you can create custom properties to store application-specific data about the current request.

Property Summary

Property	Descriptiohn
`agent`	Provides name and version information about the client software.
`imageX`	The horizontal position of the mouse pointer when the user clicked the mouse over an image map.
`imageY`	The vertical position of the mouse pointer when the user clicked the mouse over an image map.
`inputName`	Represents an input element on an HTML form. (There is not a property whose name is `inputName`. Rather, each instance of `request` has properties named after each input element.)
`ip`	Provides the IP address of the client.
`method`	Provides the HTTP method associated with the request.
`protocol`	Provides the HTTP protocol level supported by the client's software.

Method Summary None.

Examples **Example 1.** This example displays the values of the predefined properties of the `request` object. In this example, an HTML form is defined as follows:

```
<FORM METHOD="post" NAME="idForm" ACTION="hello.html">
<P>Last name:
   <INPUT TYPE="text" NAME="lastName" SIZE="20">
<BR>First name:
   <INPUT TYPE="text" NAME="firstName" SIZE="20">
</FORM>
```

The following code displays the values of the `request` object properties that are created when the form is submitted:

```
agent = <SERVER>write(request.agent)</SERVER><BR>
ip = <SERVER>write(request.ip)</SERVER><BR>
method = <SERVER>write(request.method)</SERVER><BR>
protocol = <SERVER>write(request.protocol)</SERVER><BR>
lastName = <SERVER>write(request.lastName)</SERVER><BR>
firstName = <SERVER>write(request.firstName)</SERVER>
```

When it executes, this code displays information similar to the following:

```
agent = "Mozilla/2.0 (WinNT;I)"
ip = "165.327.114.147"
method = "GET"
protocol = "HTTP/1.0"
lastName = "Schaefer"
firstName = "Jesse"
```

Example 2. The following example creates the `requestDate` property and initializes it with the current date and time:

```
request.requestDate = new Date()
```

Example 3. When a user clicks the following link, the `info.html` page is loaded, `request.accessedFrom` is created and initialized to `"hello.html"`, and `request.formId` is created and initialized to `"047"`.

```
Click here for
<A HREF="info.html?accessedFrom=hello.html&formId=047">
additional information</A>.
```

See also `client`, `project`, `server`

Properties

Custom properties

You can create a property for the `request` object by assigning it a name and a value. For example, you can create a `request` property to store the date and time that a request is received so you can enter the date into the page content.

You can also create `request` object properties by encoding them in a URL. When a user navigates to the URL by clicking its link, the properties are created and instantiated to values that you specify. The properties are valid on the destination page.

Use the following syntax to encode a `request` property in a URL:

```
<A HREF="URL?propertyName=value&propertyName=value...">
```

where:

- `URL` is the URL the page that will get the new `request` properties.
- `propertyName` is the name of the property you are creating.
- `value` is the initial value of the new property.

Use `escape` to encode non-alphanumeric values in the URL string.

You can also create custom properties for the `request` object.

agent

Provides name and version information about the client software.

Property of	`request`
Read-only	
Implemented in	LiveWire 1.0

Description The `agent` property identifies the client software. Use this information to conditionally employ certain features in an application.

The value of the `agent` property is the same as the value of the `userAgent` property of the client-side `navigator` object. The `agent` property specifies client information in the following format:

`codeName`/`releaseNumber` (`platform`; `country`; `platformIdentifier`)

The values contained in this format are the following:

- `codeName` is the code name of the client. For example, `"Mozilla"` specifies Navigator.
- `releaseNumber` is the version number of the client. For example, `"2.0b4"` specifies Navigator 2.0, beta 4.
- `platform` is the platform upon which the client is running. For example, `"Win16"` specifies a 16-bit version of Windows, such as Windows 3.11.
- `country` is either `"I"` for the international release or `"U"` for the domestic U.S. release. The domestic release has a stronger encryption feature than the international release.
- `platformIdentifier` is an optional identifier that further specifies the platform. For example, in Navigator 1.1, `platform` is `"windows"` and `platformIdentifier` is `"32bit"`. In Navigator 2.0, both pieces of information are contained in the `platform` designation. For example, in Navigator 2.0, the previous platform is expressed as `"WinNT"`.

Examples The following example displays client information for Navigator 2.0 on Windows NT:

```
write(request.agent)
\\Displays "Mozilla/2.0 (WinNT;I)"
```

The following example evaluates the `request.agent` property and runs the `oldBrowser` procedure for clients other than Navigator 2.0. If the browser is Navigator 2.0, the `currentBrowser` function executes.

```
<SERVER>
var agentVar=request.agent
if (agentVar.indexOf("2.0")==-1)
   oldBrowser()
else
   currentBrowser()
</SERVER>
```

See also `request.ip`, `request.method`, `request.protocol`

imageX

The horizontal position of the mouse pointer when the user clicked the mouse over an image map.

Property of	`request`
Read-only	
Implemented in	LiveWire 1.0

Description The ISMAP attribute of the `IMG` tag indicates a server-based image map. When the user clicks the mouse with the pointer over an image map, the horizontal and vertical position of the pointer are returned to the server.

The `imageX` property returns the horizontal position of the mouse cursor when the user clicks on an image map.

Examples Suppose you define the following image map:

```
<A HREF="mapchoice.html">
<IMG SRC="images\map.gif" WIDTH=599 WIDTH=424 BORDER=0 ISMAP
ALT="SANTA CRUZ COUNTY">
</A>
```

Note the `ISMAP` attribute that makes the image a clickable map. When the user clicks the mouse on the image, the page `mapchoice.html` will have properties `request.imageX` and `request.imageY` based on the mouse cursor position where the user clicked.

See also `request.imageY`

imageY

The vertical position of the mouse pointer when the user clicked the mouse over an image map.

Property of	`request`
Read-only	
Implemented in	LiveWire 1.0

Description The ISMAP attribute of the `IMG` tag indicates a server-based image map. When the user clicks the mouse with the pointer over an image map, the horizontal and vertical position of the pointer are returned to the server.

The `imageY` property returns the vertical position of the mouse cursor when the user clicks on an image map.

Examples See example for `imageX`.

See also `request.imageX`

inputName

Represents an input element on an HTML form.

Property of	`request`
Read-only	
Implemented in	LiveWire 1.0

Description Each input element in an HTML form corresponds to a property of the `request` object. The name of each of these properties is the name of the field on the associated form. `inputName` is a variable that represents the value of the `name` property of an input field on a submitted form. By default, the value of the JavaScript `name` property is the same as the HTML `NAME` attribute.

Examples The following HTML source creates the `request.lastName` and the `request.firstName` properties when `idForm` is submitted:

```
<FORM METHOD="post" NAME="idForm" ACTION="hello.html">
<P>Last name:
   <INPUT TYPE="text" NAME="lastName" SIZE="20">
<BR>First name:
   <INPUT TYPE="text" NAME="firstName" SIZE="20">
</FORM>
```

ip

Provides the IP address of the client.

Property of	`request`
Read-only	
Implemented in	LiveWire 1.0

Description The IP address is a set of four numbers between 0 and 255, for example, 198.217.226.34. You can use the IP address to authorize or record access in certain situations.

Examples In the following example, the `indexOf` method evaluates `request.ip` to determine if it begins with the string `"198.217.226"`. The `if` statement executes a different function depending on the result of the `indexOf` method.

```
<SERVER>
var ipAddress=request.ip
if (ipAddress.indexOf("198.217.226.")==-1)
   limitedAccess()
else
   fullAccess()
</SERVER>
```

See also `request.agent`, `request.method`, `request.protocol`

method

Provides the HTTP method associated with the request.

Property of	`request`
Read-only	
Implemented in	LiveWire 1.0

Description The value of the `method` property is the same as the value of the `method` property of the client-side `Form` object. That is, `method` reflects the `METHOD` attribute of the `FORM` tag. For HTTP 1.0, the `method` property evaluates to either `"get"` or `"post"`. Use the `method` property to determine the proper response to a request.

Examples The following example executes the `postResponse` function if the `method` property evaluates to `"post"`. If `method` evaluates to anything else, it executes the `getResponse` function.

```
<SERVER>
if (request.method=="post")
   postResponse()
else
   getResponse()
</SERVER>
```

See also `request.agent`, `request.ip`, `request.protocol`

protocol

Provides the HTTP protocol level supported by the client's software.

Property of	`request`
Read-only	
Implemented in	LiveWire 1.0

Description For HTTP 1.0, the protocol value is `"HTTP/1.0"`. Use the `protocol` property to determine the proper response to a request.

Examples In the following example, the `currentProtocol` function executes if `request.protocol` evaluates to `"HTTP/1.0"`.

```
<SERVER>
if (request.protocol=="HTTP/1.0"
   currentProtocol()
else
   unknownProtocol()
</SERVER>
```

See also `request.agent`, `request.ip`, `request.method`

client

Contains data specific to an individual client.

Server-side object	
Implemented in	LiveWire 1.0

Created by The JavaScript runtime engine on the server automatically creates a `client` object for each client/application pair.

Description The JavaScript runtime engine on the server constructs a `client` object for every client/application pair. A browser client connected to one application has a different `client` object than the same browser client connected to a different application. The runtime engine constructs a new `client` object each time a user accesses an application; there can be hundreds or thousands of `client` objects active at the same time.

You cannot use the `client` object on your application's initial page. This page is run when the application is started on the server. At this time, there is not a client request, so there is no available `client` object.

The runtime engine constructs and destroys the `client` object for each client request. However, at the end of a request, the runtime engine saves the names and values of the `client` object's properties so that when the same user returns to the application with a subsequent request, the runtime engine can construct a new `client` object with the saved data. Thus, conceptually you can think of the `client` object as remaining for the duration of a client's session with the application. There are several different ways to maintain `client` property values; for more information, see *Writing Server-Side JavaScript Applications*.

All requests by one client use the same `client` object, as long as those requests occur within the lifetime of that `client` object. By default, a `client` object persists until the associated client has been inactive for 10 minutes. You can use the `expiration` method to change this default lifetime or the `destroy` method to explicitly destroy the `client` object.

Use the `client` object to maintain data that is specific to an individual client. Although many clients can access an application simultaneously, the individual `client` objects keep their data separate. Each `client` object can track the progress of an individual client across multiple requests to the same application.

Method Summary

Method	Descriptiohn
`destroy`	Destroys a client object.
`expiration`	Specifies the duration of a `client` object.

Examples **Example 1.** This example dynamically assigns a customer ID number that is used for the lifetime of an application session. The `assignId` function creates an ID based on the user's IP address, and the `customerId` property saves the ID.

```
<SERVER>client.customerId = assignId(request.ip)</SERVER>
```

See also the examples for the `project` object for a way to sequentially assign a customer ID.

Example 2. This example creates a `customerId` property to store a customer ID that a user enters into a form. The form is defined as follows:

```
<FORM NAME="getCustomerInfo" METHOD="post">
<P>Enter your customer ID:
   <INPUT TYPE="text" NAME="customerNumber">
</FORM>
```

The following code assigns the value entered in the `customerNumber` field from the temporary `request.clientNumber` to the more permanent `client.customerId`:

```
<SERVER>client.customerId=request.customerNumber</SERVER>
```

See also `project`, `request`, `server`

Properties

The `client` object has no predefined properties. You create custom properties to contain any client-specific data that is required by an application. The runtime engine does not save `client` objects that have no property values.

You can create a property for the `client` object by assigning it a name and a value. For example, you can create a `client` property to store a customer ID at the beginning of an application so a user does not have to enter it with each request.

Because of the techniques used to maintain `client` properties across multiple client requests, there is one major restriction on `client` property values. The JavaScript runtime engine on the server converts the values of all of the `client` object's properties to strings.

The runtime engine cannot convert an object to a string. For this reason, you cannot assign an object as the value of a `client` property. If a client property value represents another data type, such as a number, you must convert the value from a string before using it. The core JavaScript `parseInt` and `parseFloat` functions are useful for converting to integer and floating point values.

Methods

destroy

Destroys a `client` object.

Method of	`client`
Implemented in	LiveWire 1.0

Syntax `destroy()`

Description The `destroy` method explicitly destroys the `client` object that issues it and removes all properties from the `client` object. If you do not explicitly issue a `destroy` method, the JavaScript runtime engine on the server automatically destroys the `client` object when its lifetime expires. The `expiration` method sets the lifetime of a `client` object; by default, the lifetime is 10 minutes.

If you are using client-cookies to maintain the `client` object, `destroy` eliminates all `client` property values, but it does not affect what is stored in Navigator cookie file. Use `expiration` with an argument of 0 seconds to remove all client properties stored in the cookie file.

When using client URL encoding to maintain the `client` object, `destroy` removes all `client` properties after the method call. However, any links in a page before the call to `destroy` retain properties in their URLs. Therefore, you should generally call `destroy` either at the top or bottom of the page when using client URL maintenance.

Examples The following method destroys the `client` object that calls it:

```
<server>client.destroy()</server>
```

See also `client.expiration`

expiration

Specifies the duration of a `client` object.

Method of	`client`
Implemented in	LiveWire 1.0

Syntax `expiration(seconds)`

Parameters

`seconds`	An integer representing the number of seconds of client inactivity before the `client` object expires.

Description By default, the JavaScript runtime engine on the server destroys the `client` object after the client has been inactive for 10 minutes. This default lifetime lets the runtime engine clean up `client` objects that are no longer necessary.

Use the `expiration` method to explicitly control the expiration of a `client` object, making it longer or shorter than the default. You must use `expiration` in each page of an application for which you want a `client` expiration other than the default. Any page that does not specify an expiration will use the default of 10 minutes.

Client expiration does not apply if using client URL encoding to maintain the client object. In this case, client properties are stored solely in URLs on HTML pages. The runtime engine cannot remove those properties.

Examples The following example extends the amount of client inactivity before expiration to 1 hour. This code is issued when an application is first launched.

```
<SERVER>client.expiration(3600)</SERVER>
```

See also `client.destroy`

project

Contains data for an entire application.

Server-side object

Implemented in LiveWire 1.0

Created by The JavaScript runtime engine on the server automatically creates a `project` object for each application running on the server.

Description The JavaScript runtime engine on the server creates a `project` object when an application starts and destroys the `project` object when the application or server stops. The typical `project` object lifetime is days or weeks.

Each client accessing the same application shares the same `project` object. Use the `project` object to maintain global data for an entire application. Many clients can access an application simultaneously, and the `project` object lets these clients share information.

The runtime engine creates a set of `project` objects for each distinct Netscape HTTPD process running on the server. Because several server HTTPD processes may be running on different port numbers, the runtime engine creates a set of `project` objects for each process.

You can lock the `project` object to ensure that different clients do not change its properties simultaneously. When one client locks the `project` object, other clients must wait before they can lock it. See `Lock` for more information about locking the `project` object.

Method Summary

Method	Descriptiohn
`lock`	Obtains the lock.
`unlock`	Releases the lock.

Examples **Example 1.** This example creates the `lastID` property and assigns a value to it by incrementing an existing value.

```
project.lastID = 1 + parseInt(project.lastID, 10)
```

Example 2. This example increments the value of the `lastID` property and uses it to assign a value to the `customerID` property.

```
project.lock()
project.lastID = 1 + parseInt(project.lastID, 10);
client.customerID = project.lastID;
project.unlock();
```

In the previous example, notice that the `project` object is locked while the `customerID` property is assigned, so no other client can attempt to change the `lastID` property at the same time.

See also `client`, `request`, `server`

Properties

The `project` object has no predefined properties. You create custom properties to contain project-specific data that is required by an application.

You can create a property for the `project` object by assigning it a name and a value. For example, you can create a `project` object property to keep track of the next available Customer ID. Any client that accesses the application without a Customer ID is sequentially assigned one, and the value of the ID is incremented for each initial access.

Methods

lock

Obtains the lock. If another thread has the lock, this method waits until it can get the lock.

Method of	`project`
Implemented in	LiveWire 1.0

Syntax `lock()`

Parameters None.

Returns Nothing.

Description You can obtain a lock for an object to ensure that different clients do not access a critical section of code simultaneously. When an application locks an object, other client requests must wait before they can lock the object.

Note that this mechanism requires voluntary compliance by asking for the lock in the first place.

See also `Lock`, `project.unlock`

unlock

Releases the lock.

Method of	`project`
Implemented in	LiveWire 1.0

Syntax `unlock()`

Parameters None.

Returns False if it fails; otherwise, true. Failure indicates an internal JavaScript error or that you attempted to unlock a lock that you don't own.

Description If you unlock a lock that is unlocked, the resulting behavior is undefined.

See also `Lock`, `project.lock`

server

Contains global data for the entire server.

Server-side object	
Implemented in	LiveWire 1.0

Created by The JavaScript runtime engine on the server automatically creates a single `server` object to store information common to all JavaScript applications running on the web server.

Description The JavaScript runtime engine on the server creates a `server` object when the server starts and destroys it when the server stops. Every application on a server shares the same `server` object. Use the `server` object to maintain global data for the entire server. Many applications can run on a server simultaneously, and the `server` object lets them share information.

The runtime engine creates a `server` object for each distinct Netscape HTTPD process running on the server.

The properties listed below are read-only properties that are initialized automatically when a `server` object is created. These properties provide information about the server process. In addition to these predefined properties, you can create custom properties.

You can lock the `server` object to ensure that different applications do not change its properties simultaneously. When one application locks the `server` object, other applications must wait before they can lock it.

Property Summary

Property	Descriptiohn
`host`	String specifying the server name, subdomain, and domain name.
`hostname`	String containing the full hostname of the server, including the server name, subdomain, domain, and port number.
`port`	String indicating the port number used for the server.
`protocol`	String indicating the communication protocol used by the server.

Method Summary

Method	Descriptiohn
`lock`	Obtains the lock.
`unlock`	Releases the lock.

Examples The following example displays the values of the predefined `server` object properties:

```
<P>server.host = <SERVER>write(server.host);</SERVER>
<BR>server.hostname = <SERVER>write(server.hostname);</SERVER>
<BR>server.protocol = <SERVER>write(server.protocol);</SERVER>
<BR>server.port = <SERVER>write(server.port);</SERVER>
```

The preceding code displays information such as the following:

```
server.host = www.myWorld.com
server.hostname = www.myWorld.com:85
server.protocol = http:
server.port = 85
```

See also `client`, `project`, `request`

Properties

host

A string specifying the server name, subdomain, and domain name.

Property of	`server`
Read-only	
Implemented in	LiveWire 1.0

Description The `host` property specifies a portion of a URL. The `host` property is a substring of the `hostname` property. The `hostname` property is the concatenation of the `host` and `port` properties, separated by a colon. When the `port` property is 80 (the default), the `host` property is the same as the `hostname` property.

See Section 3.1 of RFC 1738 (`http://www.cis.ohio-state.edu/htbin/rfc/rfc1738.html`) for complete information about the hostname and port.

See also `server.hostname`, `server.port`, `server.protocol`

hostname

A string containing the full hostname of the server, including the server name, subdomain, domain, and port number.

Property of	`server`
Read-only	
Implemented in	LiveWire 1.0

Description The `hostname` property specifies a portion of a URL. The `hostname` property is the concatenation of the `host` and `port` properties, separated by a colon. When the `port` property is 80 (the default), the `host` property is the same as the `hostname` property.

See Section 3.1 of RFC 1738 (`http://www.cis.ohio-state.edu/htbin/rfc/rfc1738.html`) for complete information about the hostname and port.

See also `server.host`, `server.port`, `server.protocol`

port

A string indicating the port number used for the server.

Property of	`server`
Read-only	
Implemented in	LiveWire 1.0

Description The `port` property specifies a portion of the URL. The `port` property is a substring of the `hostname` property. The `hostname` property is the concatenation of the `host` and `port` properties, separated by a colon.

The default value of the `port` property is 80. When the `port` property is set to the default, the values of the `host` and `hostname` properties are the same.

See Section 3.1 of RFC 1738 (`http://www.cis.ohio-state.edu/htbin/rfc/rfc1738.html`) for complete information about the port.

See also `server.host`, `server.hostname`, `server.protocol`

protocol

A string indicating the communication protocol used by the server.

Property of	`server`
Read-only	
Implemented in	LiveWire 1.0

Description The `protocol` property specifies the beginning of the URL, up to and including the first colon. The protocol indicates the access method of the URL. For example, a protocol of `"http:"` specifies HyperText Transfer Protocol.

The `protocol` property represents the scheme name of the URL. See Section 2.1 of RFC 1738 (`http://www.cis.ohio-state.edu/htbin/rfc/rfc1738.html`) for complete information about the protocol.

See also `server.host`, `server.hostname`, `server.port`

Methods

lock

Obtains the lock. If another thread has the lock, this method waits until it can get the lock.

Method of	`server`
Implemented in	LiveWire 1.0

Syntax `lock()`

Parameters None

Returns Nothing.

Description You can obtain a lock for an object to ensure that different clients do not access a critical section of code simultaneously. When an application locks an object, other client requests must wait before they can lock the object.

Note that this mechanism requires voluntary compliance by asking for the lock in the first place.

See also `Lock`, `server.lock`

unlock

Releases the lock.

Method of	`server`
Implemented in	LiveWire 1.0

Syntax `unlock()`

Parameters None.

Returns False if it fails; otherwise, true. Failure indicates an internal JavaScript error or that you attempted to unlock a lock that you don't own.

Description If you unlock a lock that is unlocked, the resulting behavior is undefined.

See also `Lock`, `server.unlock`

Lock

Provides a way to lock a critical section of code.

Server-side object

Implemented in Netscape Server 3.0

Created by The `Lock` constructor:

```
Lock();
```

Parameters None.

Failure to construct a new `Lock` object indicates an internal JavaScript error, such as out of memory.

Method Summary

Method	Descriptiohn
`lock`	Obtains the lock.
`isValid`	Verifies that this `Lock` object was properly constructed.
`unlock`	Releases the lock.

See also `project.lock`, `project.unlock`, `server.lock`, `server.unlock`

Methods

Syntax

lock

Obtains the lock. If someone else has the lock, this method blocks until it can get the lock, the specified timeout period has elapsed, or an error occurs.

Method of `Lock`

Implemented in Netscape Server 3.0

Syntax `lock(timeout)`

Parameters

`timeout`	An integer indicating the number of seconds to wait for the lock. If 0, there is no timeout; that is, the method waits indefinitely to obtain the lock. The default value is 0, so if you do not specify a value, the method waits indefinitely.

Returns True if it succeeds in obtaining the lock within the specified timeout. False if it did not obtain the lock.

Description You can obtain a lock for an object to ensure that different clients do not access a critical section of code simultaneously. When an application locks an object, other client requests must wait before they can lock the object.

Note that this mechanism requires voluntary compliance by asking for the lock in the first place.

See also `Lock.unlock`, `Lock.isValid`, `project.lock`, `server.lock`

isValid

Verifies that this `Lock` object was properly constructed.

Method of	`Lock`
Implemented in	Netscape Server 3.0

Syntax `isValid()`

Parameters None.

Returns True, if this object was properly constructed; otherwise, false.

Description It is very rare that your `Lock` object would not be properly constructed. This happens only if the runtime engine runs out of system resources while creating the object.

Examples This code creates a `Lock` object and verifies that nothing went wrong creating it:

```
// construct a new Lock and save in project
project.ordersLock = new Lock();
if (! project.ordersLock.isValid()) {
   // Unable to create a Lock. Redirect to error page
   ...
}
```

See also `Lock.lock`, `Lock.unlock`

unlock

Releases the lock.

Method of	`Lock`
Implemented in	Netscape Server 3.0

Syntax `unlock()`

Parameters None.

Returns False if it fails; otherwise, true. Failure indicates an internal JavaScript error or that you attempted to unlock a lock that you don't own.

Description If you unlock a lock that is unlocked, the resulting behavior is undefined.

See also `Lock.lock`, `Lock.isValid`, `project.unlock`, `server.unlock`

Chapter

12

Utilities

This chapter contains the server-side objects `File` and `SendMail`.

Table 12.1 summarizes the objects in this chapter.

Table 12.1 Miscellaneous objects

Object	Description
`File`	Provides access to the server's file system.
`SendMail`	Provides functionality for sending electronic mail from your JavaScript application.

File

Lets an application interact with a physical file on the server.

Server-side object

Implemented in LiveWire 1.0

Created by The `File` constructor:

```
new File("path")
```

Parameters

`path`	The path and filename in the format of the server's file system (not a URL path).

Description You can use the `File` object to write to or read from a file on the server. For security reasons, you cannot programmatically access the file system of client machines.

You can use the `File` object to generate persistent HTML or data files without using a database server. Information stored in a file is preserved when the server goes down.

Exercise caution when using the `File` object. An application can read and write files anywhere the operating system allows. If you create an application that writes to or reads from your file system, you should ensure that users cannot misuse this capability.

Specify the full path, including the filename, for the `path` parameter of the `File` object you want to create. The path must be an absolute path; do not use a relative path.

If the physical file specified in the path already exists, the JavaScript runtime engine references it when you call methods for the object. If the physical file does not exist, you can create it by calling the `open` method.

You can display the name and path of a physical file by calling the `write` function and passing it the name of the related `File` object.

A pointer indicates the current position in a file. If you open a file in the `a` or `a+` mode, the pointer is initially positioned at the end of the file; otherwise, it is initially positioned at the beginning of the file. In an empty file, the beginning and end of the file are the same. Use the `eof`, `getPosition`, and `setPosition` methods to specify and evaluate the position of the pointer. See the `open` method for a description of the modes in which you can open a file.

You can use the `prototype` property of the `File` object to add a property to all `File` instances. If you do so, that addition applies to all `File` objects running in all applications on your server, not just in the single application that made the change. This allows you to expand the capabilities of this object for your entire server.

Property Summary

Property	Descriptiohn
`prototype`	Allows the addition of properties to a `File` object.

Method Summary

Method	Descriptiohn
`byteToString`	Converts a number that represents a byte into a string.
`clearError`	Clears the current file error status.
`close`	Closes an open file on the server.
`eof`	Determines whether the pointer is beyond the end of an open file.
`error`	Returns the current error status.
`exists`	Tests whether a file exists.
`flush`	Writes the content of the internal buffer to a file.
`getLength`	Returns the length of a file.
`getPosition`	Returns the current position of the pointer in an open file.
`open`	Opens a file on the server.
`read`	Reads data from a file into a string.
`readByte`	Reads the next byte from an open file and returns its numeric value.
`readln`	Reads the current line from an open file and returns it as a string.
`setPosition`	Positions a pointer in an open file.
`stringToByte`	Converts the first character of a string into a number that represents a byte.
`write`	Writes data from a string to a file on the server.
`writeByte`	Writes a byte of data to a binary file on the server.
`writeln`	Writes a string and a carriage return to a file on the server.

Examples **Example 1.** The following example creates the `File` object `userInfo` that refers to a physical file called `info.txt`. The `info.txt` file resides in the same directory as the application's `.web` file:

```
userInfo = new File("info.txt")
```

Example 2. In the following example, the `File` object refers to a physical file with an absolute path:

```
userInfo = new File("c:\\data\\info.txt")
```

Example 3. The following example displays the name of a `File` object onscreen.

```
userInfo = new File("c:\\data\\info.txt")
write(userInfo)
```

Properties

prototype

Represents the prototype for this class. You can use the prototype to add properties or methods to all instances of a class. For information on prototypes, see `Function.prototype`.

Property of	`File`
Implemented in	LiveWire 1.0

Methods

byteToString

Converts a number that represents a byte into a string.

Method of	`File`
Static	
Implemented in	LiveWire 1.0

Syntax `byteToString(number)`

Parameters

`number`	A number that represents a byte.

Description Use the `stringToByte` and `byteToString` methods to convert data between binary and ASCII formats. The `byteToString` method converts the `number` argument into a string.

Because `byteToString` is a static method of `File`, you always use it as `File.byteToString()`, rather than as a method of a `File` object you created.

If the argument you pass into the `byteToString` method is not a number, the method returns an empty string.

Examples The following example creates a copy of a text file, one character at a time. In this example, a `while` loop executes until the pointer is positioned past the end of the file. Inside the loop, the `readByte` method reads the current character from the source file, and the `byteToString` method converts it into a string; the `write` method writes it to the target file. The last `readByte` method positions the pointer past the end of the file, ending the `while` loop. See the `File` object for a description of the pointer.

```
// Create the source File object
source = new File("c:\data\source.txt")

// If the source file opens successfully, create a target file
if (source.open("r")) {
   target = new File("c:\data\target.txt")
   target.open("w")

// Copy the source file to the target
   while (!source.eof()) {
      data = File.byteToString(source.readByte())
      target.write(data);
   }
   source.close()
}
   target.close()
```

This example is similar to the example used for the `write` method of `File`. However, this example reads bytes from the source file and converts them to strings, instead of reading strings from the source file.

See also `File.stringToByte`

clearError

Clears the current file error status.

Method of	`File`
Implemented in	LiveWire 1.0

Syntax `clearError()`

Parameters None.

Description The `clearError` method clears both the file error status (the value returned by the `error` method) and the value returned by the `eof` method.

Examples See the example for the `error` method.

See also `File.error`, `File.eof`

close

Closes an open file on the server.

Method of	`File`
Implemented in	LiveWire 1.0

Syntax `close()`

Parameters None.

Description When your application is finished with a file, you should close the file by calling the `close` method. If the file is not open, the `close` method fails. This method returns true if it is successful; otherwise, it returns false.

Examples See the examples for the `open` method.

See also `File.open`, `blob`

eof

Determines whether the pointer is beyond the end of an open file.

Method of	`File`

Implemented in LiveWire 1.0

Syntax `eof()`

Parameters None.

Description Use the `eof` method to determine whether the position of the pointer is beyond the end of a file. See `File` for a description of the pointer.

A call to `setPosition` resulting in a location greater than `fileObjectName.getLength` places the pointer beyond the end of the file. Because all read operations also move the pointer, a read operation that reads the last byte of data (or character) in a file positions the pointer beyond the end of the file.

The `eof` method returns true if the pointer is beyond the end of the file; otherwise, it returns false.

Examples In this example, a `while` loop executes until the pointer is positioned past the end of the file. While the pointer is not positioned past the end of the file, the `readln` method reads the current line, and the `write` method displays it. The last `readln` method positions the pointer past the end of the file, ending the `while` loop.

```
x = new File("c:\data\userInfo.txt")
if (x.open("r")) {
   while (!x.eof()) {
      line = x.readln()
      write(line+"<br>");
   }
   x.close();
}
```

See also `File.getPosition`, `File.setPosition`

error

Returns the current error status.

Method of `File`

Implemented in LiveWire 1.0

Syntax `error()`

Parameters None

Returns 0 if there is no error.

-1 if the file specified in `fileObjectName` is not open

Otherwise, the method returns a nonzero integer indicating the error status. Specific error status codes are platform-dependent. Refer to your operating system documentation for more information.

Examples The following example uses the `error` method in an `if` statement to take different actions depending on whether a call to the `open` method succeeded. After the `if` statement completes, the error status is reset with the `clearError` method.

```
userInput = new File("c:\data\input.txt")
userInput.open("w")
if (userInput.error() == 0) {
   fileIsOpen() }
else {
   fileIsNotOpen() }
userInput.clearError()
```

See also `File.clearError`

exists

Tests whether a file exists.

Method of	`File`
Implemented in	LiveWire 1.0

Syntax `exists()`

Parameters None.

Returns True if the file exists; otherwise, false.

Examples The following example uses an `if` statement to take different actions depending on whether a physical file exists. If the file exists, the JavaScript runtime engine opens it and calls the `writeData` function. If the file does not exist, the runtime engine calls the `noFile` function.

```
dataFile = new File("c:\data\mytest.txt")
```

```
if (dataFile.exists() ==true) {
   dataFile.open("w")
   writeData()
   dataFile.close()
}
else {
   noFile()
}
```

flush

Writes the content of the internal buffer to a file.

Method of	`File`
Implemented in	LiveWire 1.0

Syntax `flush()`

Parameters None.

Description When you write to a file with any of the `File` object methods (`write`, `writeByte`, or `writeln`), the data is buffered internally. The `flush` method writes the buffer to the physical file. The `flush` method returns true if it is successful; otherwise, it returns false.

Do not confuse the `flush` method of the `File` object with the top-level `flush` function. The `flush` function flushes a buffer of data and causes it to display in the client browser; the `flush` method flushes a buffer of data to a physical file.

Examples See the `write` method for an example of the `flush` method.

See also `File.write`, `File.writeByte`, `File.writeln`

getLength

Returns the length of a file.

Method of	`File`
Implemented in	LiveWire 1.0

Syntax `getLength()`

Parameters None.

Description If this method is successful, it returns the number of bytes in a binary file or characters in a text file; otherwise, it returns -1.

Examples The following example copies a file one character at a time. This example uses `getLength` as a counter in a `for` loop to iterate over every character in the file.

```
// Create the source File object
source = new File("c:\data\source.txt")

// If the source file opens successfully, create a target file
if (source.open("r")) {
   target = new File("c:\data\target.txt")
   target.open("a")

   // Copy the source file to the target
   for (var x = 0; x < source.getLength(); x++) {
      source.setPosition(x)
      data = source.read(1)
      target.write(data)
   }
   source.close()
}
   target.close()
```

getPosition

Returns the current position of the pointer in an open file.

Method of	`File`
Implemented in	LiveWire 1.0

Syntax `getPosition()`

Parameters None

Returns -1 if there is an error.

Description Use the `getPosition` method to determine the position of the pointer in a file. See the `File` object for a description of the pointer. The `getPosition` method returns the current pointer position; the first byte in a file is byte 0.

Examples The following examples refer to the file `info.txt`, which contains the string "Hello World." The length of `info.txt` is 11 bytes.

Example 1. In the following example, the first call to `getPosition` shows that the default pointer position is 0 in a file that is opened for reading. This example also shows that a call to the `read` method repositions the pointer.

```
dataFile = new File("c:\data\info.txt")
dataFile.open("r")

write("The position is " + dataFile.getPosition() + "<BR>")
write("The next character is " + dataFile.read(1) + "<BR>")
write("The new position is " + dataFile.getPosition() + "<BR>")

dataFile.close()
```

This example displays the following information:

```
The position is 0
The next character is H
The new position is 1
```

Example 2. This example uses `setPosition` to position the pointer one byte from the end of the eleven-byte file, resulting in a pointer position of offset 10.

```
dataFile = new File("c:\data\info.txt")
dataFile.open("r")

dataFile.setPosition(-1,2)
write("The position is " + dataFile.getPosition() + "<BR>")
write("The next character is " + dataFile.read(1) + "<BR>")

dataFile.close()
```

This example displays the following information:

```
The position is 10
The next character is d
```

Example 3. You can position the pointer beyond the end of the file and still evaluate `getPosition` successfully. However, a call to `eof` indicates that the pointer is beyond the end of the file.

```
dataFile.setPosition(1,2)
write("The position is " + dataFile.getPosition() + "<BR>")
write("The value of eof is " + dataFile.eof() + "<P>")
```

This example displays the following information:

```
The position is 12
The value of eof is true
```

See also `File.eof`, `File.open`, `File.setPosition`

open

Opens a file on the server.

Method of	`File`
Implemented in	LiveWire 1.0

Syntax `open("mode")`

Parameters

`mode`	A string specifying whether to open the file to read, write, or append, according to the list below.

Description Use the `open` method to open a file on the server before you read from it or write to it. If the file is already open, the method fails and has no effect. The `open` method returns true if it is successful; otherwise, it returns false.

The `mode` parameter is a string that specifies whether to open the file to read, write, or append data. You can optionally use the `b` parameter anytime you specify the mode. If you do so, the JavaScript runtime engine on the server opens the file as a binary file. If you do not use the `b` parameter, the runtime engine opens the file as a text file. The `b` parameter is available only on Windows platforms.

The possible values for `mode` are as follows:

- `r[b]` opens a file for reading. If the file exists, the method succeeds and returns true; otherwise, the method fails and returns false.
- `w[b]` opens a file for writing. If the file does not already exist, it is created; otherwise, it is overwritten. This method always succeeds and returns true.
- `a[b]` opens a file for appending (writing at the end of the file). If the file does not already exist, it is created. This method always succeeds and returns true.
- `r+[b]` opens a file for reading and writing. If the file exists, the method succeeds and returns true; otherwise, the method fails and returns false. Reading and writing commence at the beginning of the file. When writing, characters at the beginning of the file are overwritten.

- `w+[b]` opens a file for reading and writing. If the file does not already exist, it is created; otherwise, it is overwritten. This method always succeeds and returns true.
- `a+[b]` opens a file for reading and appending. If the file does not already exist, it is created. This method always succeeds and returns true. Reading and appending commence at the end of the file.

When your application is finished with a file, you should close the file by calling the `close` method.

Examples **Example 1.** The following example opens the file `info.txt` so an application can write information to it. If `info.txt` does not already exist, the `open` method creates it; otherwise, the `open` method overwrites it. The `close` method closes the file after the `writeData` function is completed.

```
userInfo = new File("c:\data\info.txt")
userInfo.open("w")
writeData()
userInfo.close()
```

Example 2. The following example opens a binary file so an application can read data from it. The application uses an `if` statement to take different actions depending on whether the `open` statement finds the specified file.

```
entryGraphic = new File("c:\data\splash.gif")
if (entryGraphic.open("rb") == true) {
   displayProcedure()
   }
else {
   errorProcedure()
   }
entryGraphic.close()
```

See also `File.close`

read

Reads data from a file into a string.

Method of	`File`
Implemented in	LiveWire 1.0

Syntax `read(count)`

Parameters

`count`	An integer specifying the number of characters to read.

Description The `read` method reads the specified number of characters from a file, starting from the current position of the pointer. If you attempt to read more characters than the file contains, the method reads as many characters as possible. This method moves the pointer the number of characters specified by the `count` parameter. See the `File` object for a description of the pointer.

The `read` method returns the characters it reads as a string.

Use the `read` method to read information from a text file; use the `readByte` method to read data from a binary file.

Examples The following example references the file `info.txt`, which contains the string "Hello World." The first `read` method starts from the beginning of the file and reads the character "H." The second `read` method starts from offset six and reads the characters "World."

```
dataFile = new File("c:\data\info.txt")
dataFile.open("r")

write("The next character is " + dataFile.read(1) + "<BR>")
dataFile.setPosition(6)
write("The next five characters are " + dataFile.read(5) + "<BR>")

dataFile.close()
```

This example displays the following information:

```
The next character is H
The next five characters are World
```

See also `File.readByte`, `File.readln`, `File.write`

readByte

Reads the next byte from an open file and returns its numeric value.

Method of	`File`
Implemented in	LiveWire 1.0

Syntax `readByte()`

Parameters None.

Description The `readByte` method reads the next byte from a file, starting from the current position of the pointer. This method moves the pointer one byte. See the `File` object for a description of the pointer.

The `readByte` method returns the byte it reads as a number. If the pointer is at the end of the file when you issue `readByte`, the method returns -1.

Use the `readByte` method to read information from a binary file. You can use the `readByte` method to read from a text file, but you must use the `byteToString` method to convert the value to a string. Generally it is better to use the `read` method to read information from a text file.

You can use the `writeByte` method to write data read by the `readByte` method to a file.

Examples This example creates a copy of a binary file. In this example, a `while` loop executes until the pointer is positioned past the end of the file. While the pointer is not positioned past the end of the file, the `readByte` method reads the current byte from the source file, and the `writeByte` method writes it to the target file. The last `readByte` method positions the pointer past the end of the file, ending the `while` loop.

```
// Create the source File object
source = new File("c:\data\source.gif")

// If the source file opens successfully, create a target file
if (source.open("rb")) {
   target = new File("c:\data\target.gif")
   target.open("wb")

// Copy the source file to the target
   while (!source.eof()) {
      data = source.readByte()
      target.writeByte(data);
   }
   source.close();
}
target.close()
```

See also `File.read`, `File.readln`, `File.writeByte`

readln

Reads the current line from an open file and returns it as a string.

Method of `File`

Implemented in	LiveWire 1.0

Syntax `readln()`

Parameters None

Description The `readln` method reads the current line of characters from a file, starting from the current position of the pointer. If you attempt to read more characters than the file contains, the method reads as many characters as possible. This method moves the pointer to the beginning of the next line. See the `File` object for a description of the pointer.

The `readln` method returns the characters it reads as a string.

The line separator characters ("`\r`" and "`\n`" on Windows platforms and "`\n`" on UNIX platforms) are not included in the string that the `readln` method returns. The `\r` character is skipped; `\n` determines the actual end of the line.

Use the `readln` method to read information from a text file; use the `readByte` method to read data from a binary file. You can use the `writeln` method to write data read by the `readln` method to a file.

Examples See `File.eof`

See also `File.read`, `File.readByte`, `File.writeln`

setPosition

Positions a pointer in an open file.

Method of	`File`
Implemented in	LiveWire 1.0

Syntax `setPosition(position, reference)`

Parameters

`position`	An integer indicating where to position the pointer.
`reference`	(Optional) An integer that indicates a reference point, according to the list below.

Description Use the `setPosition` method to reposition the pointer in a file. See the `File` object for a description of the pointer.

The `position` argument is a positive or negative integer that moves the pointer the specified number of bytes relative to the `reference` argument. Position 0 represents the beginning of a file. The end of a file is indicated by `fileObjectName.getLength()`.

The optional `reference` argument is one of the following values, indicating the reference point for `position`:

- 0: relative to beginning of file.
- 1: relative to current position.
- 2: relative to end of file.
- Other (or unspecified): relative to beginning of file.

The `setPosition` method returns true if it is successful; otherwise, it returns false.

Examples The following examples refer to the file `info.txt`, which contains the string "Hello World." The length of `info.txt` is 11 bytes. The first example moves the pointer from the beginning of the file, and the second example moves the pointer to the same location by navigating relative to the end of the file. Both examples display the following information:

```
The position is 10
The next character is d
```

Example 1. This example moves the pointer from the beginning of the file to offset 10. Because no value for `reference` is supplied, the JavaScript runtime engine assumes it is 0.

```
dataFile = new File("c:\data\info.txt")
dataFile.open("r")

dataFile.setPosition(10)
write("The position is " + dataFile.getPosition() + "<BR>")
write("The next character is " + dataFile.read(1) + "<P>")

dataFile.close()
```

Example 2. This example moves the pointer from the end of the file to offset 10.

```
dataFile = new File("c:\data\info.txt")
dataFile.open("r")

dataFile.setPosition(-1,2)
write("The position is " + dataFile.getPosition() + "<BR>")
write("The next character is " + dataFile.read(1) + "<P>")

dataFile.close()
```

See also `File.eof`, `File.getPosition`, `File.open`

stringToByte

Converts the first character of a string into a number that represents a byte.

Method of	`File`
Static	
Implemented in	LiveWire 1.0

Syntax `stringToByte(string)`

Parameters

`string`	A JavaScript string.

Description Use the `stringToByte` and `byteToString` methods to convert data between binary and ASCII formats. The `stringToByte` method converts the first character of its `string` argument into a number that represents a byte.

Because `stringToByte` is a static method of `File`, you always use it as `File.stringToByte()`, rather than as a method of a `File` object you created.

If this method succeeds, it returns the numeric value of the first character of the input string; if it fails, it returns 0.

Examples In the following example, the `stringToByte` method is passed "Hello" as an input argument. The method converts the first character, "H," into a numeric value representing a byte.

```
write("The stringToByte value of Hello = " +
   File.stringToByte("Hello") + "<BR>")
write("Returning that value to byteToString = " +
   File.byteToString(File.stringToByte("Hello")) + "<P>")
```

The previous example displays the following information:

```
The stringToByte value of Hello = 72
Returning that value to byteToString = H
```

See also `File.byteToString`

write

Writes data from a string to a file on the server.

Method of	`File`
Implemented in	LiveWire 1.0

Syntax `write(string)`

Parameters

`string`	A JavaScript string.

Description The `write` method writes the string specified as `string` to the file specified as `fileObjectName`. This method returns true if it is successful; otherwise, it returns false.

Use the `write` method to write data to a text file; use the `writeByte` method to write data to a binary file. You can use the `read` method to read data from a file to a string for use with the `write` method.

Do not confuse the `write` method of the `File` object with the `write` function. The `write` function outputs data to the client browser; the `write` method outputs data to a physical file on the server.

Examples This example creates a copy of a text file, one character at a time. In this example, a `while` loop executes until the pointer is positioned past the end of the file. While the pointer is not positioned past the end of the file, the `read` method reads the current character from the source file, and the `write` method writes it to the target file. The last `read` method positions the pointer past the end of the file, ending the `while` loop. See the `File` object for a description of the pointer.

```
// Create the source File object
source = new File("c:\data\source.txt")

// If the source file opens successfully, create a target file
if (source.open("r")) {
   target = new File("c:\data\target.txt")
   target.open("w")

// Copy the source file to the target
   while (!source.eof()) {
      data = source.read(1)
      target.write(data);
   }
   source.close();
}
   target.flush()
   target.close()
```

See also `File.flush`, `File.read`, `File.writeByte`, `File.writeln`

writeByte

Writes a byte of data to a binary file on the server.

Method of	`File`
Implemented in	LiveWire 1.0

Syntax `writeByte(number)`

Parameters

`number`	A number that specifies a byte of data.

Description The `writeByte` method writes a byte that is specified as `number` to a file that is specified as `fileObjectName`. This method returns true if it is successful; otherwise, it returns false.

Use the `writeByte` method to write data to a binary file; use the `write` method to write data to a text file. You can use the `readByte` method to read bytes of data from a file to numeric values for use with the `writeByte` method.

Examples See the example for the `readByte` method.

See also `File.flush`, `File.readByte`, `File.write`, `File.writeln`

writeln

Writes a string and a carriage return to a file on the server.

Method of	`File`
Implemented in	LiveWire 1.0

Syntax `writeln(string)`

Parameters

`string`	A JavaScript string.

Description The `writeln` method writes the string specified as `string` to the file specified as `fileObjectName`. Each string is followed by the carriage return/line feed character "`\n`" ("`\r\n`" on Windows platforms). This method returns true if the write is successful; otherwise, it returns false.

Use the `writeln` method to write data to a text file; use the `writeByte` method to write data to a binary file. You can use the `readln` method to read data from a file to a string for use with the `writeln` method.

Examples This example creates a copy of a text file, one line at a time. In this example, a `while` loop executes until the pointer is positioned past the end of the file. While the pointer is not positioned past the end of the file, the `readln` method reads the current line from the source file, and the `writeln` method writes it to the target file. The last `readln` method positions the pointer past the end of the file, ending the `while` loop. See the `File` object for a description of the pointer.

```
// Create the source File object
source = new File("c:\data\source.txt")

// If the source file opens successfully, create a target file
if (source.open("r")) {
   target = new File("c:\data\target.txt")
   target.open("w")

// Copy the source file to the target
   while (!source.eof()) {
      data = source.readln()
      target.writeln(data);
   }
   source.close();
}
   target.close()
```

Note that the `readln` method ignores the carriage return/line feed characters when it reads a line from a file. The `writeln` method appends these characters to the string that it writes.

See also `File.flush`, `File.readln`, `File.write`, `File.writeByte`

SendMail

Sends an email message.

Server-side object

Implemented in Netscape Server 3.0

The `To` and `From` attributes are required. All other properties are optional.

Created by The `SendMail` constructor:

```
new SendMail();
```

Parameters None.

Description Whatever properties you specify for the `SendMail` object are sent in the header of the mail message.

The `SendMail` object allows you to send either simple text-only mail messages or complex MIME-compliant mail or add attachments to your message. To send a MIME message, set the `Content-Type` property to the MIME type of the message.

You can use the `prototype` property of the `SendMail` object to add a property to all `SendMail` instances. If you do so, that addition applies to all `SendMail` objects running in all applications on your server, not just in the single application that made the change. This allows you to expand the capabilities of this object for your entire server.

Property Summary

Property	Descriptiohn
`Bcc`	Comma-delimited list of recipients of the message whose names should not be visible in the message.
`Body`	Text of the message.
`Cc`	Comma-delimited list of additional recipients of the message.
`Errorsto`	Address to which to send errors concerning the message. Defaults to the sender's address.
`From`	User name of the person sending the message.
`Organizatio n`	Organization information.

Property	Descriptiohn
`prototype`	Allows the addition of properties to a `SendMail` object.
`Replyto`	User name to which replies to the message should be sent. Defaults to the sender's address.
`Smtpserver`	Mail (SMTP) server name. Defaults to the value specified through the setting in the Administration server.
`Subject`	Subject of the message.
`To`	Comma-delimited list of primary recipients of the message.

Method Summary

Method	Descriptiohn
`errorCode`	Returns an integer error code associated with sending this message.
`errorMessage`	Returns a string associated with sending this message.
`send`	Sends the mail message represented by this object.

Examples

Example 1: The following script sends mail to vpg and gwp, copying jaym, with the specified subject and body for the message:

```
<server>
SMName = new SendMail();
SMName.To = "vpg@co1.com, gwp@co2.com"
SMName.From = "me@myco.com"
SMName.Cc = "jaym@hisco.com"
SMName.Subject = "The State of the Universe"
SMName.Body = "The universe, contrary to what you may have heard, is in
none too shabby shape. Not to worry! --me"
SMName.send()
</server>
```

Example 2: The following example sends an image in a GIF file:

```
sm = new SendMail();
sm.To = "satish";
sm.From = "satish@netscape.com";
sm.Smtpserver = "fen.mcom.com";
sm["Errors-to"] = "satish";
sm["Content-type"] = "image/gif";
sm["Content-Transfer-Encoding"] = "base64";
file = new File("/u/satish/LiveWire/mail/banner.gif");
```

```
openFlag = file.open("r");
if ( openFlag ) {
   len = file.getLength();
   str = file.read(len);
   sm.Body = str;
}
sm.send();
```

Example 3: The following example sends a multipart message:

```
sm = new SendMail();
sm.To = "chandra@cs.uiowa.edu, satish@netscape.com";
sm.From = "satish@netscape.com";
sm.Smtpserver = "fen.mcom.com";
sm.Organization = "Netscape Comm Corp";
sm["Content-type"] = "multipart/mixed; boundary=\"------------
8B3F7BA67B67C1DDE6C25D04\"";
file = new File("/u/satish/LiveWire/mail/mime");
openFlag = file.open("r");
if ( openFlag ) {
   len = file.getLength();
   str = file.read(len);
   sm.Body = str;
}
sm.send();
```

The file `mime` has HTML text and an Microsoft Word document separated by the specified boundary. The resulting message appears as HTML text followed by the Microsoft Word attachment.

Properties

Bcc

Comma-delimited list of recipients of the message whose names should not be visible in the message.

Property of	`SendMail`
Implemented in	Netscape Server 3.0

Body

Text of the message.

Property of	`SendMail`
Implemented in	Netscape Server 3.0

Cc

Comma-delimited list of additional recipients of the message.

Property of	`SendMail`
Implemented in	Netscape Server 3.0

Errorsto

Address to which to send errors concerning the message. Defaults to the sender's address.

Property of	`SendMail`
Implemented in	Netscape Server 3.0

From

User name of the person sending the message.

Property of	`SendMail`
Implemented in	Netscape Server 3.0

Organization

Organization information.

Property of	`SendMail`
Implemented in	Netscape Server 3.0

prototype

Represents the prototype for this class. You can use the prototype to add properties or methods to all instances of a class. For information on prototypes, see `Function.prototype`.

Property of	`SendMail`
Implemented in	LiveWire 1.0

Replyto

User name to which replies to the message should be sent. Defaults to the sender's address.

Property of	`SendMail`
Implemented in	Netscape Server 3.0

Smtpserver

Mail (SMTP) server name. Defaults to the value specified through the setting in the Administration server.

Property of	`SendMail`
Implemented in	Netscape Server 3.0

Subject

Subject of the message.

Property of	`SendMail`
Implemented in	Netscape Server 3.0

To

Comma-delimited list of primary recipients of the message.

Property of	`SendMail`
Implemented in	Netscape Server 3.0

Methods

errorCode

Returns an integer error code associated with sending this message.

Method of	`SendMail`
Implemented in	Netscape Server 3.0

Syntax `public errorCode();`

Returns The possible return values and their meanings are as follows:

`0`	Successful send.
`1`	SMTP server not specified.
`2`	Specified mail server is down or doesn't exist.
`3`	At least one receiver's address must be specified to send the message.
`4`	Sender's address must be specified to send the message.
`5`	Mail connection problem; data not sent.

errorMessage

Returns a string associated with sending this message.

Method of	`SendMail`
Implemented in	Netscape Server 3.0

Syntax `public errorMessage();`

Returns An error string.

send

Sends the mail message represented by this object.

Method of	`SendMail`
Implemented in	Netscape Server 3.0

Syntax `public send ();`

Returns This method returns a Boolean value to indicate whether or not the mail was successfully sent. If the mail was not successfully sent, you can use the `errorMessage` and `errorCode` methods to determine the nature of the error.

This method returns a string indicating the nature of the error that occurred sending the message.

Chapter

13

Global Functions

This chapter contains all JavaScript functions not associated with any object.

Table 13.1 summarizes these functions.

Table 13.1 Global functions

Function	Description
`addClient`	Appends client information to URLs.
`addResponseHeader`	Adds new information to the response header sent to the client.
`blob`	Assigns BLOb data to a column in a cursor.
`callC`	Calls a native function.
`debug`	Displays values of expressions in the trace window or frame.
`deleteResponseHeader`	Removes information from the header of the response sent to the client.
`escape`	Returns the hexadecimal encoding of an argument in the ISO Latin-1 character set; used to create strings to add to a URL.
`eval`	Evaluates a string of JavaScript code without reference to a particular object.
`flush`	Flushes the output buffer.

Table 13.1 Global functions

Function	Description
`getOptionValue`	Gets values of individual options in an HTML `SELECT` form element.
`getOptionValueCount`	Gets the number of options in an HTML `SELECT` form element.
`isNaN`	Evaluates an argument to determine if it is not a number.
`Number`	Converts an object to a number.
`parseFloat`	Parses a string argument and returns a floating-point number.
`parseInt`	Parses a string argument and returns an integer.
`redirect`	Redirects the client to the specified URL.
`registerCFunction`	Registers a native function for use in server-side JavaScript.
`ssjs_generateClientID`	Returns an identifier you can use to uniquely specify the `client` object.
`ssjs_getCGIVariable`	Returns the value of the specified environment variable set in the server process, including some CGI variables.
`ssjs_getClientID`	Returns the identifier for the `client` object used by some of JavaScript's client-maintenance techniques.
`String`	Converts an object to a string.
`taint`	Adds tainting to a data element or script.
`unescape`	Returns the ASCII string for the specified value; used in parsing a string added to a URL.
`untaint`	Removes tainting from a data element or script.
`write`	Adds statements to the client-side HTML page being generated.

addClient

Adds `client` object property values to a dynamically generated URL or the URL used with the `redirect` function.

Server-side function

Implemented in LiveWire 1.0

Syntax `addClient(URL)`

Parameters

`URL`	A string representing a URL

Description The `addClient` function is a top-level server-side JavaScript function not associated with any object.

Use `addClient` to preserve `client` object property values when you use `redirect` or generate dynamic links. This is necessary if an application uses client or server URL encoding to maintain the `client` object; it does no harm in other cases. Since the client maintenance technique can be changed after the application has been compiled, it is always safer to use `addClient`, even if you do not anticipate using a URL encoding scheme.

See *Writing Server-Side JavaScript Applications* for information about using URL encoding to maintain client properties.

Examples In the following example, `addClient` is used with the `redirect` function to redirect a browser:

```
redirect(addClient("mypage.html"))
```

In the following example, `addClient` preserves `client` object property values when a link is dynamically generated:

```
<A HREF='addClient("page" + project.pageno + ".html")'>
   Jump to new page</A>
```

See also `redirect`

addResponseHeader

Adds new information to the response header sent to the client.

Server-side function

Implemented in	Netscape Server 3.0

Syntax `addResponseHeader(field, value)`

Parameters

`field`	A field to add to the response header.
`value`	The information to specify for that field.

Description You can use the `addResponseHeader` function to add information to the header of the response you send to the client.

For example, if the response you send to the client uses a custom content type, you should encode this content type in the response header. The JavaScript runtime engine automatically adds the default content type (`text/html`) to the response header. If you want a custom header, you must first remove the old default content type from the header and then add the new one. If your response uses `royalairways-format` as a custom content type, you would specify it this way:

```
deleteResponseHeader("content-type");
addResponseHeader("content-type","royalairways-format");
```

You can use the `addResponseHeader` function to add any other information you want to the response header.

Remember that the header is sent with the first part of the response. Therefore, you should call these functions early in the script on each page. In particular, you should ensure that the response header is set *before* any of these happen:

- The runtime engine generates 64KB of content for the HTML page (it automatically flushes the output buffer at this point).
- You call the `flush` function to clear the output buffer.
- You call the `redirect` function to change client requests.

See also `deleteResponseHeader`

blob

Assigns BLOb data to a column in a cursor.

Server-side function

Implemented in LiveWire 1.0

Syntax `blob (path)`

Parameters

`path`	A string representing the name of a file containing `BLOb` data. This string must be an absolute pathname.

Returns A `blob` object.

Description Use this function with an updatable cursor to insert or update a row containing BLOb data. To insert or update a row using SQL and the `execute` method, use the syntax supported by your database vendor.

On DB2, blobs are limited to 32 KBytes.

Remember that back slash ("\") is the escape character in JavaScript. For this reason, in NT filenames you must either use 2 backslashes or a forward slash.

Example The following statements update BLOb data from the specified GIF files in columns `PHOTO` and `OFFICE` of the current row of the `EMPLOYEE` table.

```
// Create a cursor
cursor = database.cursor("SELECT * FROM customer WHERE
   customer.ID = " + request.customerID

// Position the pointer on the row
cursor.next()

// Assign the blob data
cursor.photo = blob("c:/customer/photos/myphoto.gif")
cursor.office = blob("c:/customer/photos/myoffice.gif")

// And update the row
cursor.updateRow("employee")
```

callC

Calls an external function and returns the value that the external function returns.

Server-side function

Implemented in LiveWire 1.0

Syntax `callC(JSFunctionName, arg1,..., argN)`

Parameters

`JSFunctionName`	The name of the function as it is identified with `RegisterCFunction`.
`arg1...argN`	A comma-separated list of arguments to the external function. The arguments can be any JavaScript values: strings, numbers, or Boolean values. The number of arguments must match the number of arguments required by the external function.

Description The `callC` function is a top-level server-side JavaScript function that is not associated with any object.

The `callC` function returns the string value that the external function returns; `callC` can only return string values.

Examples The following example assigns a value to the variable `isRegistered` according to whether the attempt to register the external function `echoCCallArguments` succeeds or fails. If `isRegistered` is true, the `callC` function executes.

```
var isRegistered =
   registerCFunction("echoCCallArguments",
      "c:/mypath/mystuff.dll",
      "mystuff_EchoCCallArguments")
if (isRegistered == true) {
   var returnValue =
   callC("echoCCallArguments", "first arg", 42, true, "last arg")
   write(returnValue)
}
```

See also `registerCFunction`

debug

Displays a JavaScript expression in the trace facility.

Server-side function

Implemented in LiveWire 1.0

Syntax `debug(expression)`

Parameters

`expression`	Any valid JavaScript expression.

Description The `debug` function is a top-level server-side JavaScript function that is not associated with any object.

Use this function to display the value of an expression for debugging purposes. The value is displayed in the trace facility of the Application Manager following the brief description "Debug message:".

Examples The following example displays the value of the variable `data`:

```
debug("The final value of data is " + data)
```

deleteResponseHeader

Removes information from the header of the response sent to the client.

Server-side function

Implemented in Netscape Server 3.0

Syntax `deleteResponseHeader(field)`

Parameters

`field`	A field to remove from the response header.

Description You can use the `deleteResponseHeader` function to remove information from the header of the response you send to the client. The most frequent use of this function is to remove the default content-type information before adding your own content-type information with `addResponseHeader`.

For more information, see `addResponseHeader`.

escape

Returns the hexadecimal encoding of an argument in the ISO-Latin-1 character set.

Core function

Implemented in Navigator 2.0, LiveWire 1.0

Syntax

```
escape("string")
```

Parameters

`string`	A string in the ISO-Latin-1 character set.

Description The `escape` function is a top-level JavaScript function that is not associated with any object. Use the `escape` and `unescape` functions to add property values manually to a URL.

The `escape` function encodes special characters in the specified string and returns the new string. It encodes spaces, punctuation, and any other character that is not an ASCII alphanumeric character, with the exception of these characters:

```
* @ - _ + . /
```

Examples **Example 1.** The following example returns `"%26"`:

```
escape("&")
```

Example 2. This statement

```
escape("The_rain. In Spain, Ma’am")
```

returns

```
"The_rain.%20In%20Spain%2C%20Ma%92am":
```

Example 3. In the following example, the value of the variable `theValue` is encoded as a hexadecimal string and passed on to the `request` object when a user clicks the link:

```
<A HREF=`"mypage.html?val1="+escape(theValue)`)>Click Here</A>
```

See also `unescape`

eval

Evaluates a string of JavaScript code without reference to a particular object.

Core function

Implemented in Navigator 2.0

Syntax `eval(string)`

Parameters

`string`	A string representing a JavaScript expression, statement, or sequence of statements. The expression can include variables and properties of existing objects.

Description The argument of the `eval` function is a string. If the string represents an expression, `eval` evaluates the expression. If the argument represents one or more JavaScript statements, `eval` performs the statements. Do not call `eval` to evaluate an arithmetic expression; JavaScript evaluates arithmetic expressions automatically.

If you construct an arithmetic expression as a string, you can use `eval` to evaluate it at a later time. For example, suppose you have a variable `x`. You can postpone evaluation of an expression involving `x` by assigning the string value of the expression, say `"3 * x + 2"`, to a variable, and then calling `eval` at a later point in your script.

`eval` is also a method of all objects. This method is described for the `Object` class.

Examples The following examples display output using `document.write`. In server-side JavaScript, you can display the same output by calling the `write` function instead of using `document.write`.

Example 1. Both of the write statements below display 42. The first evaluates the string `"x + y + 1"`; the second evaluates the string `"42"`.

```
var x = 2
var y = 39
var z = "42"
```

```
document.write(eval("x + y + 1"), "<BR>")
document.write(eval(z), "<BR>")
```

Example 2. In the following example, the `getFieldName(n)` function returns the name of the specified form element as a string. The first statement assigns the string value of the third form element to the variable `field`. The second statement uses `eval` to display the value of the form element.

```
var field = getFieldName(3)
document.write("The field named ", field, " has value of ",
   eval(field + ".value"))
```

Example 3. The following example uses `eval` to evaluate the string `str`. This string consists of JavaScript statements that open an Alert dialog box and assign `z` a value of 42 if `x` is five, and assigns 0 to `z` otherwise. When the second statement is executed, `eval` will cause these statements to be performed, and it will also evaluate the set of statements and return the value that is assigned to `z`.

```
var str = "if (x == 5) {alert('z is 42'); z = 42;} else z = 0; "
document.write("<P>z is ", eval(str))
```

Example 4. In the following example, the `setValue` function uses `eval` to assign the value of the variable `newValue` to the text field `textObject`:

```
function setValue (textObject, newValue) {
   eval ("document.forms[0]." + textObject + ".value") = newValue
}
```

Example 5. The following example creates `breed` as a property of the object `myDog`, and also as a variable. The first write statement uses `eval('breed')` without specifying an object; the string `"breed"` is evaluated without regard to any object, and the `write` method displays `"Shepherd"`, which is the value of the `breed` variable. The second write statement uses `myDog.eval('breed')` which specifies the object `myDog`; the string `"breed"` is evaluated with regard to the `myDog` object, and the `write` method displays `"Lab"`, which is the value of the `breed` property of the `myDog` object.

```
function Dog(name,breed,color) {
   this.name=name
   this.breed=breed
   this.color=color
}
myDog = new Dog("Gabby")
myDog.breed="Lab"
var breed='Shepherd'
document.write("<P>" + eval('breed'))
document.write("<BR>" + myDog.eval('breed'))
```

See also `Object.eval` method

flush

Sends data from the internal buffer to the client.

Server-side function

Implemented in LiveWire 1.0

Syntax `flush()`

Parameters None.

Description To improve performance, JavaScript buffers the HTML page it constructs. The `flush` function immediately sends data from the internal buffer to the client. If you do not explicitly call the `flush` function, JavaScript sends data to the client after each 64KB of content in the constructed HTML page.

Use the `flush` function to control when data is sent to the client. For example, call the `flush` function before an operation that creates a delay, such as a database query. If a database query retrieves a large number of rows, you can flush the buffer after retrieving a small number of rows to prevent long delays in displaying data.

Because the `flush` function updates the client's cookie file as part of the HTTP header, you should perform any changes to the `client` object before flushing the buffer, if you are using client cookie to maintain the `client` object. For more information, see *Writing Server-Side JavaScript Applications*.

Do not confuse the `flush` method of the `File` object with the top-level `flush` function. The `flush` function is a top-level server-side JavaScript function that is not associated with any object.

Examples The following example iterates through a text file and outputs each line in the file, preceded by a line number and five spaces. The `flush` function then causes the client to display the output.

```
while (!In.eof()) {
   AscLine = In.readln();
   if (!In.eof())
      write(LPad(LineCount + ": ", 5), AscLine, "\n");
   LineCount++;
   flush();
}
```

See also `write`

getOptionValue

Returns the text of a selected `OPTION` in a `SELECT` form element.

Server-side function

Implemented in LiveWire 1.0

Syntax `getOptionValue(name, index)`

Parameters

`name`	A name specified by the `NAME` attribute of the `SELECT` tag
`index`	Zero-based ordinal index of the selected option.

Returns A string containing the text for the selected option, as specified by the associated `OPTION` tag.

Description The `getOptionValue` function is a top-level server-side JavaScript function not associated with any object. It corresponds to the `Option.text` property available to client-side JavaScript.

The `SELECT` tag allows multiple values to be associated with a single form element, with the `MULTIPLE` attribute. If your application requires select lists that allow multiple selected options, you use the `getOptionValue` function to get the values of selected options in server-side JavaScript.

Examples Suppose you have the following form element:

```
<SELECT NAME="what-to-wear" MULTIPLE SIZE=8>
   <OPTION SELECTED>Jeans
   <OPTION>Wool Sweater
   <OPTION SELECTED>Sweatshirt
   <OPTION SELECTED>Socks
   <OPTION>Leather Jacket
   <OPTION>Boots
   <OPTION>Running Shoes
   <OPTION>Cape
</SELECT>
```

You could process the input from this select list in server-side JavaScript as follows:

```
<SERVER>
var loopIndex = 0
var loopCount = getOptionValueCount("what-to-wear") // 3 by default
while ( loopIndex < loopCount ) {
   var optionValue = getOptionValue("what-to-wear",loopIndex)
   write("<br>Item #" + loopIndex + ": " + optionValue + "\n")
   loopIndex++
}
</SERVER>
```

If the user kept the default selections, this script would return

Item #1: Jeans
Item #3: Sweatshirt
Item #4: Socks

See also `getOptionValueCount`

getOptionValueCount

Returns the number of options selected by the user in a `SELECT` form element.

Server-side function

Implemented in LiveWire 1.0

Syntax `getOptionValueCount(name)`

Parameters

`name`	Specified by the `NAME` attribute of the `SELECT` tag.

Description The `getOptionValueCount` function is a top-level server-side JavaScript function not associated with any object.

Use this function with `getOptionValue` to process user input from `SELECT` form elements that allow multiple selections.

Examples See the example for `getOptionValue`.

See also `getOptionValue`

isNaN

Evaluates an argument to determine if it is not a number.

Core function

Implemented in	Navigator 2.0: Unix only Navigator 3.0, LiveWire 1.0: all platforms

Syntax `isNaN(testValue)`

Parameters

`testValue`	The value you want to evaluate.

Description `isNaN` is a built-in JavaScript function. It is not a method associated with any object, but is part of the language itself.

On platforms that support NaN, the `parseFloat` and `parseInt` functions return `"NaN"` when they evaluate a value that is not a number. `isNaN` returns true if passed `"NaN"`, and false otherwise.

Examples The following example evaluates `floatValue` to determine if it is a number and then calls a procedure accordingly:

```
floatValue=parseFloat(toFloat)

if (isNaN(floatValue)) {
   notFloat()
} else {
   isFloat()
}
```

See also `Number.NaN`, `parseFloat`, `parseInt`

Number

Converts the specified object to a number.

Core function

Implemented in	Navigator 4.0, Netscape Server 3.0

Syntax `Number(obj)`

Parameter

`obj`	An object

Description When the object is a `Date` object, `Number` returns a value in milliseconds measured from 01 January, 1970 UTC (GMT), positive after this date, negative before.

If `obj` is a string that does not contain a well-formed numeric literal, `Number` returns NaN.

Example The following example converts the `Date` object to a numerical value:

```
<SCRIPT>
d = new Date ("December 17, 1995 03:24:00");
document.write (Number(d) + "<BR>");
</SCRIPT>
```

This prints "819199440000."

See also `Number`

parseFloat

Parses a string argument and returns a floating point number.

Core function

Implemented in	Navigator 2.0: If the first character of the string specified in parseFloat(`string`) cannot be converted to a number, returns `"NaN"` on Solaris and Irix and 0 on all other platforms. Navigator 3.0, LiveWire 1.0: Returns `"NaN"` on all platforms if the first character of the string specified in parseFloat(`string`) cannot be converted to a number.

Syntax `parseFloat(string)`

Parameters

`string`	A string that represents the value you want to parse.

Description The `parseFloat` function is a built-in JavaScript function.

`parseFloat` parses its argument, a string, and returns a floating point number. If it encounters a character other than a sign (+ or -), numeral (0-9), a decimal point, or an exponent, then it returns the value up to that point and ignores that character and all succeeding characters.

If the first character cannot be converted to a number, `parseFloat` returns `"NaN"`.

For arithmetic purposes, the `"NaN"` value is not a number in any radix. You can call the `isNaN` function to determine if the result of `parseFloat` is `"NaN"`. If `"NaN"` is passed on to arithmetic operations, the operation results will also be `"NaN"`.

Examples The following examples all return 3.14:

```
parseFloat("3.14")
parseFloat("314e-2")
parseFloat("0.0314E+2")
var x = "3.14"
parseFloat(x)
```

The following example returns `"NaN"`:

```
parseFloat("FF2")
```

See also `isNaN`, `parseInt`

parseInt

Parses a string argument and returns an integer of the specified radix or base.

Core function

Implemented in	Navigator 2.0: If the first character of the string specified in parseInt(`string`) cannot be converted to a number, returns `"NaN"` on Solaris and Irix and 0 on all other platforms. Navigator 3.0, LiveWire 2.0: Returns `"NaN"` on all platforms if the first character of the string specified in parseInt(`string`) cannot be converted to a number.

Syntax `parseInt(string,radix)`

Parameters

`string`	A string that represents the value you want to parse.

`radix`	(Optional) An integer that represents the radix of the return value.

Description The `parseInt` function is a built-in JavaScript function.

The `parseInt` function parses its first argument, a string, and attempts to return an integer of the specified radix (base). For example, a radix of 10 indicates to convert to a decimal number, 8 octal, 16 hexadecimal, and so on. For radixes above 10, the letters of the alphabet indicate numerals greater than 9. For example, for hexadecimal numbers (base 16), A through F are used.

If `parseInt` encounters a character that is not a numeral in the specified radix, it ignores it and all succeeding characters and returns the integer value parsed up to that point. `parseInt` truncates numbers to integer values.

If the radix is not specified or is specified as 0, JavaScript assumes the following:

- If the input `string` begins with `"0x"`, the radix is 16 (hexadecimal).
- If the input `string` begins with `"0"`, the radix is eight (octal).
- If the input `string` begins with any other value, the radix is 10 (decimal).

If the first character cannot be converted to a number, `parseInt` returns `"NaN"`.

For arithmetic purposes, the `"NaN"` value is not a number in any radix. You can call the `isNaN` function to determine if the result of `parseInt` is `"NaN"`. If `"NaN"` is passed on to arithmetic operations, the operation results will also be `"NaN"`.

Examples The following examples all return 15:

```
parseInt("F", 16)
parseInt("17", 8)
parseInt("15", 10)
parseInt(15.99, 10)
parseInt("FXX123", 16)
parseInt("1111", 2)
parseInt("15*3", 10)
```

The following examples all return `"NaN"`:

```
parseInt("Hello", 8)
parseInt("0x7", 10)
parseInt("FFF", 10)
```

Even though the radix is specified differently, the following examples all return 17 because the input `string` begins with `"0x"`.

```
parseInt("0x11", 16)
parseInt("0x11", 0)
parseInt("0x11")
```

See also `isNaN`, `parseFloat`, `Object.valueOf`

redirect

Redirects the client to the specified URL.

Server-side function

Implemented in LiveWire 1.0

Syntax `redirect(location)`

Parameters

`location`	The URL to which you want to redirect the client.

Description The `redirect` function is a top-level server-side JavaScript function that is not associated with any object.

The `redirect` function redirects the client browser to the URL specified by the `location` parameter. The value of `location` can be relative or absolute.

When the client encounters a `redirect` function, it loads the specified page immediately and discards the current page. The client does not execute or load any HTML or script statements in the page following the `redirect` function.

You can use the `addClient` function to preserve `client` object property values. See `addClient` for more information.

Examples The following example uses the `redirect` function to redirect a client browser:

```
redirect("http://www.royalairways.com/lw/apps/newhome.html")
```

The page displayed by the `newhome.html` link could contain content such as the following:

```
<H1>New location</H1>
The URL you tried to access has been moved to:<BR>
```

```
<LI><A HREF=http://www.royalairways.com/lw/apps/index.html>
   http://www.royalairways.com/lw/apps/index.html</A>
<P>This notice will remain until 12/31/97.
```

See also `addClient`

registerCFunction

Registers an external function for use with a server-side JavaScript application.

Server-side function

Implemented in LiveWire 1.0

Syntax

```
registerCFunction(JSFunctionName, libraryPath,
   externalFunctionName)
```

Parameters

`JSFunctionName`	The name of the function as it is called in JavaScript.
`libraryPath`	The full filename and path of the library, using the conventions of your operating system.
`externalFunctionName`	The name of the function as it is defined in the library.

Description `registerCFunction` is a top-level server-side JavaScript function that is not associated with any object.

Use `registerCFunction` to make an external function available to a server-side JavaScript application. The function can be written in any language, but you must use C calling conventions.

To use an external function in a server-side JavaScript application, register the function with `registerCFunction`, and then call it with the `callC` function. Once an application registers a function, you can call the function any number of times.

The `registerCFunction` function returns true if the external function is registered successfully; otherwise, it returns false. For example, `registerCFunction` can return false if the JavaScript runtime engine cannot find either the library or the specified function inside the library.

To use a backslash (\) character as a directory separator in the `libraryPath` parameter, you must enter a double backslash (\\). The single backslash is a reserved character.

Examples See the example for the `callC` function.

See also `callC`

ssjs_generateClientID

Returns a unique string you can use to uniquely specify the `client` object.

Server-side function

Implemented in Netscape Server 3.0

Syntax `ssjs_generateClientID()`

Parameters None.

Description This function is closely related to `ssjs_getClientID`. See the description of that function for information on these functions and the differences between them.

ssjs_getCGIVariable

Returns the value of the specified environment variable set in the server process, including some CGI variables.

Server-side function

Implemented in Netscape Server 3.0

Syntax `ssjs_getCGIVariable(varName)`

Parameters

`varName`	A string containing the name of the environment variable to retrieve.

Description `ssjs_getCGIVariable` lets you access the environment variables set in the server process, including the CGI variables listed in Table 13.2.

Table 13.2 CGI variables accessible through `ssjs_getCGIVariable`

Variable	Description
`AUTH_TYPE`	The authorization type, if the request is protected by any type of authorization. Netscape web servers support HTTP basic access authorization. Example value: `basic`
`HTTPS`	If security is active on the server, the value of this variable is `ON`; otherwise, it is `OFF`. Example value: `ON`
`HTTPS_KEYSIZE`	The number of bits in the session key used to encrypt the session, if security is on. Example value: `128`
`HTTPS_SECRETKEYSIZE`	The number of bits used to generate the server's private key. Example value: `128`
`PATH_INFO`	Path information, as sent by the browser. Example value: `/cgivars/cgivars.html`
`PATH_TRANSLATED`	The actual system-specific pathname of the path contained in `PATH_INFO`. Example value: `/usr/ns-home/myhttpd/js/samples/cgivars/cgivars.html`
`QUERY_STRING`	Information from the requesting HTML page; if "?" is present, the information in the URL that comes after the "?". Example value: `x=42`
`REMOTE_ADDR`	The IP address of the host that submitted the request. Example value: `198.93.95.47`
`REMOTE_HOST`	If DNS is turned on for the server, the name of the host that submitted the request; otherwise, its IP address. Example value: `www.netscape.com`
`REMOTE_USER`	The name of the local HTTP user of the web browser, if HTTP access authorization has been activated for this URL. Note that this is not a way to determine the user name of any person accessing your program. Example value: `ksmith`
`REQUEST_METHOD`	The HTTP method associated with the request. An application can use this to determine the proper response to a request. Example value: `GET`
`SCRIPT_NAME`	The pathname to this page, as it appears in the URL. Example value: `cgivars.html`

Table 13.2 CGI variables accessible through `ssjs_getCGIVariable` (Continued)

Variable	Description
`SERVER_NAME`	The hostname or IP address on which the JavaScript application is running, as it appears in the URL. Example value: `piccolo.mcom.com`
`SERVER_PORT`	The TCP port on which the server is running. Example value: `2020`
`SERVER_PROTOCOL`	The HTTP protocol level supported by the client's software. Example value: `HTTP/1.0`
`SERVER_URL`	The URL that the user typed to access this server. Example value: `https://piccolo:2020`

If you supply an argument that isn't one of the CGI variables listed in n, the runtime engine looks for an environment variable by that name in the server environment. If found, the runtime engine returns the value; otherwise, it returns null. For example, the following code assigns the value of the standard `CLASSPATH` environment variable to the JavaScript variable `classpath`:

```
classpath = ssjs_getCGIVariable("CLASSPATH");
```

ssjs_getClientID

Returns the identifier for the `client` object used by some of JavaScript's client-maintenance techniques.

Server-side function

Implemented in Netscape Server 3.0

Syntax `ssjs_getClientID()`

Parameters None.

Description For some applications, you may want to store information specific to a client/application pair in the `project` or `server` objects. In these situations, you need a way to refer uniquely to the client/application pair. JavaScript provides two functions for this purpose, `ssjs_generateClientID` and `ssjs_getClientID`.

Each time you call `ssjs_generateClientID`, the runtime engine returns a new identifier. For this reason, if you use this function and want the identifier to last longer than a single client request, you need to store the identifier, possibly as a property of the `client` object.

If you use this function and store the ID in the `client` object, you may need to be careful that an intruder cannot get access to that ID and hence to sensitive information.

An alternative approach is to use the `ssjs_getClientID` function. If you use one of the server-side maintenance techniques for the `client` object, the JavaScript runtime engine generates and uses a identifier to access the information for a particular client/application pair.

When you use these maintenance techniques, `ssjs_getClientID` returns the identifier used by the runtime engine. Every time you call this function from a particular client/application pair, you get the same identifier. Therefore, you do not need to store the identifier returned by `ssjs_getClientID`. However, if you use any of the other maintenance techniques, this function returns "undefined"; if you use those techniques you must instead use the `ssjs_generateClientID` function.

If you need an identifier and you're using a server-side maintenance technique, you probably should use the `ssjs_getClientID` function. If you use this function, you do not need to store and track the identifier yourself; the runtime engine does it for you. However, if you use a client-side maintenance technique, you cannot use the `ssjs_getClientID` function; you must use the `ssjs_generateClientID` function.

String

Converts the specified object to a string.

Core function

Implemented in	Navigator 4.0, Netscape Server 3.0

Syntax `String(obj)`

Parameter

`obj`	An object.

Description When the object is a `Date` object, `String` returns a string representation of the date. Its format is: Thu Aug 18 04:37:43 Pacific Daylight Time 1983.

Example The following example converts the `Date` object to a readable string.

```
<SCRIPT>
D = new Date (430054663215);
document.write (String(D) +" <BR>");
</SCRIPT>
```

This prints "Thu Aug 18 04:37:43 Pacific Daylight Time 1983."

See also `String`

taint

Adds tainting to a data element or script.

Core function	
Implemented in	Navigator 3.0; removed in Navigator 4.0

Syntax `taint(dataElementName)`

Parameters

`dataElementName`	(Optional) The property, variable, function, or object to taint. If omitted, taint is added to the script itself.

Description Tainting prevents other scripts from passing information that should be secure and private, such as directory structures or user session history. JavaScript cannot pass tainted values on to any server without the end user's permission.

Use `taint` to mark data that otherwise is not tainted.

In some cases, control flow rather than data flow carries tainted information. In these cases, taint is added to the script's window. You can add taint to the script's window by calling `taint` with no arguments.

`taint` does not modify its argument; instead, it returns a marked copy of the value, or, for objects, an unmarked reference to the value.

Examples The following statement adds taint to a property so that a script cannot send it to another server without the end user's permission:

```
taintedStatus=taint(window.defaultStatus)
// taintedStatus now cannot be sent in a URL or form post without
// the end user's permission
```

See also `navigator.taintEnabled`, `untaint`

unescape

Returns the ASCII string for the specified value.

Core function

Implemented in Navigator 2.0

Syntax `unescape(string)`

Parameters

`string`	A string containing characters in the form `"%xx"`, where `xx` is a 2-digit hexadecimal number.

Description The string returned by the `unescape` function is a series of characters in the ISO-Latin-1 character set. The `unescape` function is a top-level JavaScript function not associated with any object. In server-side JavaScript, use this function to decode name/value pairs in URLs.

Examples The following client-side example returns `"&"`:

```
unescape("%26")
```

The following client-side example returns `"!#"`:

```
unescape("%21%23")
```

In the following server-side example, `val1` has been passed to the `request` object as a hexadecimal value. The statement assigns the decoded value of `val1` to `myValue`.

```
myValue = unescape(request.val1)
```

See also `escape`

untaint

Removes tainting from a data element or script.

Core function	
Implemented in	Navigator 3.0; removed in Navigator 4.0

Syntax `untaint(dataElementName)`

Parameters

`dataElementName`	(Optional) The property, variable, function, or object to remove tainting from. If omitted, taint is removed from the script itself.

Description Tainting prevents other scripts from passing information that should be secure and private, such as directory structures or user session history. JavaScript cannot pass tainted values on to any server without the end user's permission.

Use `untaint` to clear tainting that marks data that should not to be sent by other scripts to different servers.

A script can untaint only data that originated in that script (that is, only data that has the script's taint code or has the identity (null) taint code). If you use `untaint` with a data element from another server's script (or any data that you cannot untaint), `untaint` returns the data without change or error.

In some cases, control flow rather than data flow carries tainted information. In these cases, taint is added to the script's window. You can remove taint from the script's window by calling `untaint` with no arguments, if the window contains taint only from the current window.

`untaint` does not modify its argument; instead, it returns an unmarked copy of the value, or, for objects, an unmarked reference to the value.

Examples The following statement removes taint from a property so that a script can send it to another server:

```
untaintedStatus=untaint(window.defaultStatus)
// untaintedStatus can now be sent in a URL or form post by other
// scripts
```

See also `navigator.taintEnabled`, `taint`

write

Generates HTML based on an expression and sends it to the client.

Server-side function	
Implemented in	LiveWire 1.0

Syntax `write(expression)`

Parameters

`expression`	A valid JavaScript expression.

Description The `write` function causes server-side JavaScript to generate HTML that is sent to the client. The client interprets this generated HTML as it would static HTML. The server-side `write` function is similar to the client-side `document.write` method.

To improve performance, the JavaScript engine on the server buffers the output to be sent to the client and sends it in large blocks of at most 64 KBytes in size. You can control when data are sent to the client by using the `flush` function.

The `write` function is a top-level server-side JavaScript function that is not associated with any object. Do not confuse the `write` method of the `File` object with the `write` function. The `write` function outputs data to the client; the `write` method outputs data to a physical file on the server.

Examples In the following example, the `write` function is passed a string, concatenated with a variable, concatenated with a `BR` tag:

```
write("The operation returned " + returnValue + "<BR>")
```

If `returnValue` is 57, this example displays the following:

```
The operation returned 57
```

See also `flush`

Chapter 14

Java packages for LiveConnect

The LiveConnect facility allows your JavaScript application to work with Java objects and for those Java objects to work with JavaScript objects.

LiveConnect provides two Java applet API packages for communicating with JavaScript. These packages are `netscape.javascript` and `netscape.plugin.`

The `netscape.javascript` applet package is available both on the client and on the server and has the following classes:

- `netscape.javascript.JSObject`
- `netscape.javascript.JSException`

The `netscape.plugin` applet API package can be used only on the client. It has the following class:

- `netscape.plugin.Plugin`

The following sections describe these classes and list their constructors and methods.

netscape.javascript.JSObject

The public final class `JSObject` extends `Object`.

```
java.lang.Object
   |
   +----netscape.javascript.JSObject
```

`JSObject` allows Java to manipulate objects that are defined in JavaScript. Values passed from Java to JavaScript are converted as follows:

- `JSObject` is converted to the original JavaScript object.
- Any other Java object is converted to a JavaScript wrapper, which can be used to access methods and fields of the Java object. Converting this wrapper to a string will call the `toString` method on the original object, converting to a number will call the `floatValue` method if possible and fail otherwise. Converting to a boolean will try to call the `booleanValue` method in the same way.
- Java arrays are wrapped with a JavaScript object that understands `array.length` and `array[index]`.
- A Java boolean is converted to a JavaScript boolean.
- Java byte, char, short, int, long, float, and double are converted to JavaScript numbers.

Values passed from JavaScript to Java are converted as follows:

- Objects that are wrappers around Java objects are unwrapped.
- Other objects are wrapped with a `JSObject`.
- Strings, numbers, and booleans are converted to String, Float, and Boolean objects respectively.

This means that all JavaScript values show up as some kind of `java.lang.Object` in Java. In order to make much use of them, you will have to cast them to the appropriate subclass of `Object`, as shown in the following examples:

```
(String) window.getMember("name")
(JSObject) window.getMember("document")
```

Note If you call a Java method from JavaScript, this conversion happens automatically—you can pass in "int" argument and it works.

Methods and static methods

The `netscape.javascript.JSObject` class has the following methods:

Method	Description
`call`	Calls a JavaScript method
`eval`	Evaluates a JavaScript expression
`getMember`	Retrieves a named member of a JavaScript object
`getSlot`	Retrieves an indexed member of a JavaScript object
`removeMember`	Removes a named member of a JavaScript object
`setMember`	Sets a named member of a JavaScript object
`setSlot`	Sets an indexed member of a JavaScript object
`toString`	Converts a `JSObject` to a string

The `netscape.javascript.JSObject` class has the following static methods:

Method	Description
`getWindow`	Gets a `JSObject` for the window containing the given applet

The following sections show the declaration and usage of these methods.

call

Method. Calls a JavaScript method. Equivalent to "`this.methodName(args[0], args[1], ...)`" in JavaScript.

Declaration

```
public Object call(String methodName,
   Object args[])
```

eval

Method. Evaluates a JavaScript expression. The expression is a string of JavaScript source code which will be evaluated in the context given by "this".

Declaration

```
public Object eval(String s)
```

getMember

Method. Retrieves a named member of a JavaScript object. Equivalent to "`this.name`" in JavaScript.

Declaration `public Object getMember(String name)`

getSlot

Method. Retrieves an indexed member of a JavaScript object. Equivalent to "`this[index]`" in JavaScript.

Declaration `public Object getSlot(int index)`

getWindow

Static method. Returns a `JSObject` for the window containing the given applet. This method is available only on the client.

Declaration `public static JSObject getWindow(Applet applet)`

removeMember

Method. Removes a named member of a JavaScript object.

Declaration `public void removeMember(String name)`

setMember

Method. Sets a named member of a JavaScript object. Equivalent to "`this.name = value`" in JavaScript.

Declaration

```
public void setMember(String name,
   Object value)
```

setSlot

Method. Sets an indexed member of a JavaScript object. Equivalent to "`this[index] = value`" in JavaScript.

Declaration

```
public void setSlot(int index,
   Object value)
```

toString

Method. Converts a `JSObject` to a `String`.

Overrides: `toString` in class `Object`

Declaration `public String toString()`

netscape.javascript.JSException

The public class `JSException` extends `Exception`.

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----netscape.javascript.JSException
```

`JSException` is an exception that is thrown when JavaScript code returns an error.

Constructors

The `netscape.javascript.JSException` class has the following constructors:

Constructor	Description
`JSException`	Constructs a `JSException.` You specify whether the `JSException` has a detail message and other information.

The following sections show the declaration and usage of these constructors.

JSException

Constructor. Constructs a `JSException`. You specify whether the `JSException` has a detail message and other information.

Declaration

```
1. public JSException()

2. public JSException(String s)

3. public JSException(String s,
   String filename,
   int lineno,
   String source,
   int tokenIndex)
```

Arguments

`s`	The detail message.
`filename`	The URL of the file where the error occurred, if possible.
`lineno`	The line number if the file, if possible.
`source`	The string containing the JavaScript code being evaluated.
`tokenIndex`	The index into the source string where the error occurred.

Description A detail message is a string that describes this particular exception.

Each form constructs a `JSException` with different information:

- Form 1 of the declaration constructs a `JSException` without a detail message.
- Form 2 of the declaration constructs a `JSException` with a detail message.
- Form 3 of the declaration constructs a `JSException` with a detail message and all the other information that usually comes with a JavaScript error.

netscape.plugin.Plugin

The public class `Plugin` extends `Object`.

```
java.lang.Object
   |
   +----netscape.plugin.Plugin
```

This class represents the Java reflection of a plug-in. Plug-ins that need to have Java methods associated with them should subclass this class and add new (possibly native) methods to it. This allows other Java entities (such as applets and JavaScript code) to manipulate the plug-in.

Constructors and methods

The `netscape.plugin.Plugin` class has the following constructors:

Constructor	Description
`Plugin`	Constructs a Plugin.

The `netscape.plugin.Plugin` class has the following methods:

Method	Description
`destroy`	Called when the plug-in is destroyed
`getPeer`	Returns the native NPP object—the plug-in instance that is the native part of a Java `Plugin` object
`getWindow`	Returns the JavaScript window on which the plug-in is embedded
`init`	Called when the plug-in is initialized
`isActive`	Determines whether the Java reflection of a plug-in still refers to an active plug-in

The following sections show the declaration and usage of these constructors and methods.

destroy

Method. Called when the plug-in is destroyed. You never need to call this method directly, it is called when the plug-in is destroyed. At the point this method is called, the plug-in will still be active.

Declaration `public void destroy()`

See also `init`

getPeer

Method. Returns the native NPP object—the plug-in instance that is the native part of a Java `Plugin` object. This field is set by the system, but can be read from plug-in native methods by calling:

```
NPP npp = (NPP)netscape_plugin_Plugin_getPeer(env, thisPlugin);
```

Declaration `public int getPeer()`

getWindow

Method. Returns the JavaScript window on which the plug-in is embedded.

Declaration `public JSObject getWindow()`

init

Method. Called when the plug-in is initialized. You never need to call this method directly, it is called when the plug-in is created.

Declaration `public void init()`

See also `destroy`

isActive

Method. Determines whether the Java reflection of a plug-in still refers to an active plug-in. Plug-in instances are destroyed whenever the page containing the plug-in is left, thereby causing the plug-in to no longer be active.

Declaration `public boolean isActive()`

Plugin

Constructor. Constructs a `Plugin`.

Declaration `public Plugin()`

Index

Note: This index has not yet been updated.