Version 2.8, March 12, 1996

(Next scheduled update - March 17, 1996)

As taken from Netscape Corporation's World Wide Web Site

NOTE : Every effort was made to release Version 3 on time, but due to a tempermental BIOS flash chip, it was delayed. We put this version 2.8 out to satisfy your hunger for the docs. Version 3.0 will be out shortly. The only sections that have NOT been updated are the **Objects** and **Properties** sections.

As a new feature, all changes (significant changes, that is) will be highlighted in BLUE for your convenience.

Also, if you have some code snippets to donate, PLEASE email them to me at ajbrown@ajbrown.com so we may include them in a common place inside this document for all to enjoy.

Table of Contents

JavaScript Working Document

Current Version : Version 3, dated March 3, 1996
Next Update : April 7, 1996

Ongoing JavaScript Development

Development of the JavaScript language and its documentation continues. Additional features are planned; some current features could be modified if necessary.

This Acrobat document was converted from the original HTML docuemtnation found on Netscape Corporation's World Wide Web site : http://home.netscape.com by Aj Brown of IPST - Internet Professional Services and Training.

This document is meant soley as an easier way to view and scan the JavaScript documentation.

You may reach me at any of the following email addresses :

ajbrown@ipst.com
webmaster@ipst.com
http://www.ipst.com

We encourage any and all comments, suggestions, or code snippets that we could include in this working document to help fellow developers explore and take advantage of JavaScript.

If you have a code snippet you feel would be of value, please email it to me, and we will include it in the next version.

This document will be updated on a monthly basis until Netscape institutes a charge for it.

Happy Scripting !

The Mother of all Disclaimers

JavaScript and its documentation are currently under development. Some of the language is not yet implemented. That which is implemented is subject to change. Information provided at this time is incomplete and should not be considered a language specification. JavaScript is a work in progress whose potential we'd like to share with you, the beta users, in this developmental form.

Learning JavaScript

JavaScript is a compact, object-based scripting language for developing client and server Internet applications. Netscape Navigator 2.0 interprets JavaScript statements embedded directly in an HTML page, and LiveWire enables you to create server-based applications similar to common gateway interface (CGI) programs.

In a client application for Navigator, JavaScript statements embedded in an HTML page can recognize and 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, an HTML page with embedded JavaScript can interpret the entered text and alert the user with a message dialog if the input is invalid. Or you can use JavaScript to perform an action (such as play an audio file, execute an applet, or communicate with a plug-in) in response to the user opening or exiting a page.

JavaScript and Java

The JavaScript language resembles Java, but without Java's static typing and strong type checking. JavaScript supports most of Java's expression syntax and basic control flow constructs. In contrast to Java's compile-time system of classes built by declarations, JavaScript supports a run-time system based on a small number of data types representing numeric, Boolean, and string values. JavaScript has a simple instance-based object model that still provides significant capabilities.

JavaScript also supports functions, again without any special declarative requirements. Functions can be properties of objects, executing as loosely typed methods.

JavaScript complements Java by exposing useful properties of Java applets to script authors. JavaScript statements can get and set exposed properties to query the state or alter the performance of an applet or plug-in.

Java is an extension language designed, in particular, for fast execution and type safety. Type safety is reflected by being unable to cast a Java int into an object reference or to get at private memory by corrupting Java bytecodes.

Java programs consist exclusively of classes and their methods. Java's requirements for declaring classes, writing methods, and ensuring type safety make programming more complex than JavaScript authoring. Java's inheritance and strong typing also tend to require tightly coupled object hierarchies.

In contrast, JavaScript descends in spirit from a line of smaller, dynamically typed languages like HyperTalk and dBASE. These scripting languages offer programming tools to a much wider audience because of their easier syntax, specialized built-in functionality, and minimal requirements for object creation.

The following table compares and contrasts JavaScript and Java.

JavaScript	Java
Interpreted (not compiled) by client	Compiled on server before execution on client.
Object-based. Code uses built-in, extensible objects, but no classes or inhereitance	Object-orientated. Applets consist of object classes with inheritance.
Code integrated with, and embedded in, HTML.	Applets distinct from HTML (accessed from HTML pages)
Variable data types not declared (loose typing).	Variable data types must be declared (strong typing).
Dynamic binding; object references checked at run-time	Static binding; object references must exist at compile-time
Secure. cannot write to hard disk	Secure. cannot write to hard disk

JavaScript Development

A script author is not required to extend, instantiate, or know about classes. Instead, the author acquires finished components exposing high-level properties such as "visible" and "color", then gets and sets the properties to cause desired effects.

As an example, suppose you want to design an HTML page that contains some catalog text, a picture of a shirt available in several colors, a form for ordering the shirt, and a color selector tool that's visually integrated with the form. You could write a Java applet that draws the whole page, but you'd face complicated source encoding and forgo the simplicity of HTML page authoring.

A better route would use Java's strengths by implementing only the shirt viewer and color picker as applets, and using HTML for the framework and order form. A script that runs when a color is picked could set the shirt applet's color property to the picked color. With the availability of general-purpose components like a color picker or image viewer, a page author would not be required to learn or write Java. Components used by the script would be reusable by other scripts on pages throughout the catalog.

Navigator Scripting

- Using JavaScript in HTML
- Scripting Event Handlers
- Tips and Techniques

Using JavaScript in HTML

JavaScript can be embedded in an HTML document in two ways:

- As statements and functions using the SCRIPT tag.
- As event handlers using HTML tags.

The SCRIPT tag

A script embedded in HTML with the SCRIPT tag uses the format:

```
<SCRIPT>
   JavaScript statements...
</SCRIPT>
```

The optional LANGUAGE attribute specifies the scripting language as follows:

```
<SCRIPT LANGUAGE="JavaScript">
   JavaScript statements...
</SCRIPT>
```

The HMTL tag, <SCRIPT>, and its closing counterpart, </SCRIPT> can enclose any number of JavaScript statements in a document.

JavaScript is case sensitive.

Example 1: a simple script.

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
document.write("Hello net.")
</SCRIPT>
</HEAD>
<BODY>
That's all, folks.
</BODY>
</HTML>
```

Example 1 page display.

Hello net. That's all folks.

Code Hiding

Scripts can be placed inside comment fields to ensure that your JavaScript code is not displayed by old browsers that do not recognize JavaScript. The entire script is encased by HTML comment tags:

```
<!-- Begin to hide script contents from old browsers.
// End the hiding here. -->
```

Defining and Calling Functions

Scripts placed within SCRIPT tags are evaluated after the page loads. Functions are stored, but not executed. Functions are executed by events in the page.

It's important to understand the difference between defining a function and calling the function. Defining the function simply names the function and specifies what to do when the function is called. Calling the function actually performs the specified actions with the indicated parameters.

Example 2: a script with a function and comments.

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
<!--  to hide script contents from old browsers
  function square(i) {
    document.write("The call passed ", i ," to the function.","<BR>")
    return i * i
  }
  document.write("The function returned ",square(5),".")
// end hiding contents from old browsers  -->
</SCRIPT>
</HEAD>
<BODY>
<BR>
All done.
</BODY>
</HTML>
```

Example 2 page display.

We passed 5 to the function.
The function returned 25.
All done.

The HEAD tag

Generally, you should define the functions for a page in the HEAD portion of a document. Since the HEAD is loaded first, this practice guarantees that functions are loaded before the user has a chance to do anything that might call a function.

Example 3: a script with two functions.

```
<HTML>
<HEAD>
<SCRIPT>
<!--- hide script from old browsers
function bar() {
   document.write("<HR ALIGN='left' WIDTH=25%>")
}
function output(head, level, string) {
   document.write("<H" + level + ">" + head + "</H" + level + "><P>" + string)
}
// end hiding from old browsers -->
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT>
<!--- hide script from old browsers
document.write(bar(),output("Make Me Big",3,"Make me ordinary."))
// end hiding from old browsers -->
</SCRIPT>
<P>
Thanks.
</BODY>
</HTML>
```

Example 3 results.

Make Me Big

Make me ordinary.

Thanks.

Quotes

Use single quotes (') to delimit string literals so that scripts can distinguish the literal from attribute values enclosed in double quotes. In the previous example, function bar contains the literal 'left' within a double-quoted attribute value. Here's another example:

```
<INPUT TYPE="button" VALUE="Press Me" onClick="myfunc('astring')">
```

Scripting Event Handlers

JavaScript applications in the Navigator are largely event-driven. Events are actions that occur, usually as a result of something the user does. For example, a button click is an event, as is giving focus to a form element. There is a specific set of events that Navigator recognizes. You can define Event handlers, scripts that are automatically executed when an event occurs.

Event handlers are embedded in documents as attributes of HTML tags to which you assign JavaScript code to execute. The general syntax is

```
<TAG eventHandler="JavaScript Code">
```

where TAG is some 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 Navigator to perform this function when the user clicks on 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 quotes following *onClick*. These statements get executed when the user clicks on the button. If you want to include more than one statement, separate statements with a semicolon (;).

In general, it is a good idea to define functions for your event handlers because:

- it makes your code modular-you can use the same function as an event handler for many different items.
- it makes your code easier to read.

Notice in this example the use of **this.form** to refer to the current form. The keyword **this** refers to the current object-in the above example, the button object. The construct **this.form** then refers to the form containing the button. In the above example, the onClick event handler is a call to the compute() function, with **this.form**, the current form, as the parameter to the function.

Events apply to HTML tags as follows:

- Focus, Blur, Change events: text fields, textareas, and selections
- Click events: buttons, radio buttons, checkboxes, submit buttons, reset buttons, links
- Select events: text fields, textareas
- MouseOver event: links

If an event applies to an HTML tag, then you can define an event handler for it. In general, an event handler has the name of the event, preceded by "on." For example, the event handler for the Focus event is onFocus.

Many objects also have methods that emulate events. For example, button has a click method that emulates the button being clicked. **Note:** The event-emulation methods do not trigger event-handlers. So, for example, the click method does not trigger an onClick event-handler. However, you can always call an event-handler directly (for example, you can call onClick explicitly in a script).

Event	Occurs when ...	Event Handler
blur	User removes input focus from form element	onBlur
click	User clicks on form element or link	onClick
change	User changes value of text, textarea, or select element	onChange
focus	User gives form element input focus	onFocus
load	User loads the page in the Navigator	onLoad
mouseover	User moves mouse pointer over a link or anchor	onMouseOver
select	User selects form element's input field	onSelect
submit	User submits a form	onSubmit
unload	User exits the page	onUnload

Example 4: a script with a form and an event handler attribute.

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
function compute(form) {
   if (confirm("Are you sure?"))
      form.result.value = eval(form.expr.value)
   else
      alert("Please come back again.")
}
</SCRIPT>
</HEAD>

<BODY>
<FORM>
Enter an expression:
<INPUT TYPE="text" NAME="expr" SIZE=15 >
<INPUT TYPE="button" VALUE="Calculate" ONCLICK="compute(this.form)">
<BR>
Result:
<INPUT TYPE="text" NAME="result" SIZE=15 >
<BR>
</FORM>
</BODY>
</HTML>
```

Example 4 page display.

Enter an expression: 9 + 5

Result: 14

Example 5: a script with a form and event handler attribute within a BODY tag.

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
function checkNum(str, min, max) {
```

```
    if (str == "") {
        alert("Enter a number in the field, please.")
        return false
    }
    for (var i = 0; i < str.length; i++) {
        var ch = str.substring(i, i + 1)
        if (ch < "0" || ch >; "9") {
            alert("Try a number, please.")
            return false
        }
    }
    var val = parseInt(str, 10)
    if ((val < min) || (val > max)) {
        alert("Try a number from 1 to 10.")
        return false
    }
    return true
}
function thanks() {
    alert("Thanks for your input.")
}
</SCRIPT>
</HEAD>

<BODY>
<FORM>
Please enter a small number:
<INPUT NAME="num"
    ONCHANGE="if (!checkNum(this.value, 1, 10))
        {this.focus();this.select();} else {thanks()}"
    VALUE="0">
</FORM>
</BODY>
</HTML>
```

Example 5 page display.

Please enter a small number: 7
Field name: num
Field value: 7

Tips and Techniques

This section describes various useful scripting techniques.

Updating Pages

JavaScript in Navigator generates its results from the top of the page down. Once something has been formatted, you can't change it without reloading the page. Currently, you cannot update a particular part of a page without updating the entire page. However, you can update a "sub-window" in a frame separately.

Printing

You cannot currently print output created with JavaScript. For example, if you had the following in a page:

```
<P>This is some text.
<SCRIPT>document.write("<P>And some generated text")</SCRIPT>
```

And you printed it, you would get only "This is some text", even though you would see both lines on-screen.

Using Quotes

Be sure to alternate double quotes with single quotes. Since event handlers in HTML must be enclosed in quotes, you must use single quotes to delimit arguments.
For example

```
<FORM NAME="myform">
<INPUT TYPE="button" NAME="Button1" VALUE="Open Sesame!"
onClick="window.open('stmtsov.html', 'newWin', 'toolbar=no,directories=no')">
</FORM>
```

Alternatively, you can escape quotes by preceding them by a backslash (\).

Defining Functions

It is always a good idea to define all of your functions in the HEAD of your HTML page. This way, all functions will be defined before any content is displayed. Otherwise, the user might perform some action while the page is still loading that triggers an event handler and calls an undefined function, leading to an error.

Creating Arrays

An array is an ordered set of values that you reference through an array name and an index. For example, you could have an array called emp, that contains employees' names indexed by their employee number. So emp[1] would be employee number one, emp[2] employee number two, and so on.

JavaScript does not have an explicit array data type, but because of the intimate relationship between arrays and object properties (see JavaScript Object Model), it is easy to create arrays in JavaScript. You can define an array object type, as follows:

```
function MakeArray(n) {
   this.length = n;
   for (var i = 1; i <= n; i++) {
     this[i] = 0 }
     return this
     }
}
```

This defines an array such that the first property, length, (with index of zero), represents the number of elements in the array. The remaining properties have an integer index of one or greater, and are initialized to zero.

You can then create an array by a call to new with the array name, specifying the number of elements it has. For example:

```
emp = new MakeArray(20);
```

This creates an array called emp with 20 elements, and initializes the elements to zero.

Populating an Array

You can populate an array by simply assigning values to its elements. For example:

```
emp[1] = "Casey Jones"
emp[2] = "Phil Lesh"
emp[3] = "August West"
```

and so on.

You can also create arrays of objects. For example, suppose you define an object type named Employees, as follows:

```
function Employee(empno, name, dept) {
   this.empno = empno;
   this.name = name;
   this.dept = dept;
}
```

Then the following statements define an array of these objects:

```
emp = new MakeArray(3)
emp[1] = new Employee(1, "Casey Jones", "Engineering")
emp[2] = new Employee(2, "Phil Lesh", "Music")
emp[3] = new Employee(3, "August West", "Admin")
```

Then you can easily display the objects in this array using the show_props function (defined in the section on the JavaScript Object Model) as follows:

```
for (var n =1; n <= 3; n++) {
   document.write(show_props(emp[n], "emp") + "
");
}
```

JavaScript Values, Names, and Literals

- Values
- Variable Names
- Literals

Values

JavaScript recognizes the following types of values:

- numbers, such as 42 or 3.14159
- logical (Boolean) values, either true or false
- strings, such as "Howdy!"
- null, a special keyword denoting a null value

This relatively small set of types of values, or data types, enables you to perform useful functions with your applications.Notice that there is no explicit distinction between integer and real-valued numbers. Nor is there an explicit date data type in Navigator. However, the date object and related built-in functions enable you to handle dates.

Objects and functions are the other fundamental elements in the language. You can think of objects as named containers for values, and functions as procedures that your application can perform.

Datatype Conversion

JavaScript is a loosely typed language. That means that you do not have to specify the datatype of a variable when you declare it, and datatypes are converted automatically as needed during the course of script execution. So, for example, you could define a variable as follows:

```
var answer = 42
```

And later, you could assign the same variable a string value, for example:

```
answer = "Thanks for all the fish..."
```

Because JavaScript is loosely typed, this will not cause an error message.

In general, in expressions involving numeric and string values, JavaScript converts the numeric values to strings. For example, consider the following statements:

```
x = "The answer is " + 42
y = 42 + " is the answer."
```

The first statement will the string "The answer is 42". The second statement will return the string "42 is the answer".

JavaScript provides several special functions for manipulating string and numeric values:

- eval attempts to evaluate a string representing any JavaScript literals or variables, converting it to a number.
- parseInt converts a string to an integer of the specified radix (base), if possible.
- parseFloat converts a string to a floating-point number, if possible.

NOTE: Much of the functionality specified in this table is not implemented as of Navigator beta4.

	Converted to data type :				
Data type	**Function**	**Object**	**Number**	**Boolean**	**String**
Function	-	function	error	error	decompile
Object **Null Object**	error funobj OK	-	error 0	true false	toString "null"
Number (non-zero) **0** **Error (NaN)** **+infinity** **-infinity**	error	Number null Number Number Number	-	true false false true true	toString "0" "NaN" "+Infinity" "-Infinity"
Boolean: false **true**	error	Boolean	0 1	-	"false" "true"
String (non-null) **Null String**	funstr OK error	String	numstr OK error	true false	-

Variable Names

You use variables to hold values in your application. You give these variables names by which you reference them, and there are certain rules to which the names must conform.

A JavaScript identifier or name must start with a letter or underscore ("_"); subsequent characters can also be digits (0-9). Letters include the characters "A" through "Z" (uppercase) and the characters "a" through "z" (lowercase). JavaScript is case-sensitive.
Some examples of legal names are:

- Number_hits
- temp99
- _name

Literals

Literals are the way you represent values in JavaScript. These are fixed values that you literally provide in your application source, and are not variables. Examples of literals include:

- 42
- 3.14159
- "To be or not to be"

Integers

Integers can be expressed in decimal (base 10), hexadecimal (base 16), or octal (base 8) format. A decimal integer literal consists of a sequence of digits (optionally suffixed as described below) without a leading 0 (zero).

An integer can be expressed in octal or hexadecimal rather than decimal. A leading 0 (zero) on an integer literal means it is in octal; a leading 0x (or 0X) means hexadecimal. Hexadecimal integers can include digits (0-9) and the letters a-f and A-F. Octal integers can include only the digits 0-7.

Floating Point Literals

A floating point literal can have the following parts: a decimal integer, a decimal point ("."), a fraction (another decimal number), an exponent, and a type suffix. The exponent part is an "e" or "E" followed by an integer, which can be signed (preceded by a "+" or "-"). A floating point literal must have at least one digit, plus either a decimal point or "e" (or "E"). Some examples of floating point literals are:

- 3.1415
- -3.1E12
- .1e12
- 2E-12

Boolean Literals

The boolean type has two literal values: **true** and **false**.

String Literals

A string literal is zero or more characters enclosed in double (") or single (') quotes. A string must be delimited by quotes of the same type; that is, either both single quotes or double quotes. The following are examples of string literals:

- "blah"
- 'blah'
- "1234"
- "one line \n another line"

Special Characters

You can use the following special characters in JavaScript string literals:

- \b indicates a backspace.
- \f indicates a a form feed.
- \n indicates a new line character.
- \r indicates a carriage return.
- \t indicates a tab character.

Escaping Characters

You can insert quotes inside of strings by preceding them by a backslash.
This is known as *escaping* the quotes.
For example,

```
var quote = "<P>He read \"The Cremation of Sam McGee\" by R.W. Service"
document.write(quote)
```

The result of this would be

He read "The Cremation of Sam McGee" by R.W. Service

JavaScript Expressions and Operators

- Expressions
- Operators
    - Arithmetic Operators
    - Bitwise Operators
    - Logical Operators
    - Comparison Operators
    - String Operators
    - Operator Precedence

Expressions

An expression is any valid set of literals, variables, operators, and expressions that evaluates to a single value. The value may be a number, a string, or a logical value. Conceptually, there are two types of expressions: those that assign a value to a variable, and those that simply have a value. For example, the expression

```
x = 7
```

is an expression that assigns x the value 7. This expression itself evaluates to 7. Such expressions use assignment operators. On the other hand, the expression

```
3 + 4
```

simply evaluates to 7; it does not perform an assignment. The operators used in such expressions are referred to simply as *operators*.

JavaScript has the following kinds of expressions:

- Arithmetic: evaluates to a number, for example
- String: evaluates to a character string, for example "Fred" or "234"
- Logical: evaluates to true or false

The special keyword **null** denotes a null value. In contrast, variables that have not been assigned a value are *undefined*, and cannot be used without a run-time error.

Conditional Expressions

A conditional expression can have one of two values based on a condition. The syntax is

```
(condition) ? val1 : val2
```

If *condition* is true, the expression has the value of *val1*, Otherwise it has the value of *val2*. You can use a conditional expression anywhere you would use a standard expression.

For example,

```
status = (age >= 18) ? "adult" : "minor"
```

This statement assigns the value "adult" to the variable status if age is eighteen or greater. Otherwise, it assigns the value "minor" to status.

Assignment Operators (=, +=, -=, *=, /=)

An assignment operator assigns a value to its left operand based on the value of its right operand. 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 operators are shorthand for standard arithmetic operations as follows:

- x += y means x = x + y
- x -= y means x = x - y
- x *= y means x = x * y
- x /= y means x = x / y
- x %= y means x = x % y

There are additional assignment operators for bitwise operations:

- x <<= y means x = x << y
- x >>= y means x = x >> y
- x >>>= means x = x >>> y
- x &= means x = x & y
- x ^= means x = x ^ y
- x |= means x = x | y

Operators

LiveScript has arithmetic, string, and logical operators. There are both binary and unary operators. A binary operator requires two operands, one before the operator and one after the operator:

```
operand1 operator operand2
```

For example, `3 + 4` or `x * y`

A unary operator requires a single operand, either before or after the operator:

```
operator operand
```

or

```
operand operator
```

For example `x++` or `++x`.

Arithmetic Operators

Arithmetic operators take numerical values (either literals or variables) as their operands and return a

single numerical value.

Standard Arithmetic Operators

The standard arthmetic operators are addition (+), subtraction (-), multiplication (*), and division (/). These operators work in the standard way.

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 statement above, where *var1* and *var2* are variables. The modulo function is the remainder of integrally 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 3, then the statement

```
y = x++
```

increments x to 4 and sets y to 3.

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 3, then the statement

```
y = x--
```

decrements x to 2 and sets y to 3.
If x is 3, then the statement

```
y = --x
```

decrements x to 2 and sets y to 2.

Unary negation (-)

The unary negation operator must precede its operand. It negates its operand. For example,

```
x = -x
```

negates the value of x; that is if x were 3, it would become -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 9 has a binary representation of 101. Bitwise operators perform their operations on such binary representations, but they return standard JavaScript numerical values.

Bitwise Logical Operators

The bitwise operators are:

- Bitwise AND &. Returns a one if both operands are ones.
- Bitwise OR |. Returns a one if either operand is one.
- Bitwise XOR ^. Returns a one if one but not both operands are one.

The bitwise logical operators work conceptually as follows:

- The operands are converted to 32-bit integers, and expressed 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.

Bitwise Shift Operators

The bitwise shift operators are:

- Left Shift (<<)
- Sign-propagating Right Shift (>>)
- Zero-fill Right shift (>>>)

The 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 32-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.

Example TBD.

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.

Example TBD.

Zero-fill right shift (>>>)

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

Example TBD.

Logical Operators

Logical operators take logical (Boolean) values as operands. They return a logical value. Logical values are **true** and **false**.

And (&&)

> **Usage:** `expr1 && expr2`

The logical "and" operator returns true if both logical expressions *expr1* and *expr2* are true. Otherwise, it returns false.

Or (||)

> **Usage:** `expr1 || expr2`

The logical "or" operator returns true if either logical expression expr1 or expr2 is true. If both expr1 and expr2 are false, then it returns false.

Not (!)

Usage: `!expr`

The logical "not" operator is a unary operator that negates its operand expression expr. That is, if expr is true, it returns false, and if expr is false, then it returns true.

Short-Circuit Evaluation

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

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

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

Comparison Operators (= =, >, >=, <, <=, !=)

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

The operators are:

- Equal (= =): returns true if the operands are equal.
- Not equal (!=): returns true if the operands are not equal.
- Greater than (>): returns true if left operand is greater than right operand. Example: x > y returns true if x is greater than y.
- Greater than or equal to (>=): returns true if left operand is greater than or equal to right operand. Example: x >= y returns true if x is greater than or equal to y.
- Less than (<): returns true if left operand is less than right operand. Example: x < y returns true if x is less than y.
- Less than or equal to (<=): returns true if left operand is less than or equal to right operand. Example: x <= y returns true if x is less than or equal to y.

String Operators

In addition to the comparison operators, which may 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"
```

The shorthand assignment operator += can also be used to concatenate strings. For example, if the variable mystring is a string that has the value "alpha", then the expression

```
mystring += "bet"
```

evaluates to "alphabet" and assigns this value to `mystring`.

Operator Precedence

The precedence of operators determines the order they are applied when evaluating an expression. You can override operator precedence by using parentheses.

The precedence of operators, from lowest to highest is as follows:

comma ,
assignment = += -= *= /= %= <<= >>= >>>= &= ^= |=
conditional ?:
logical-or ||
logical-and &&
bitwise-or |
bitwise-xor ^
bitwise-and &
equality == !=
relational < <= > >=
shift << >> >>>
addition/subtraction + -
multiply/divide * / %
negation/increment ! ~ - ++ --
call, member () [] .

The JavaScript Object Model

JavaScript is based on a simple object-oriented paradigm. An object is a construct with properties that are JavaScript variables. Properties can be other objects. Functions associated with an object are known as the object's methods.

In addition to objects that are built into the Navigator client and the LiveWire server, you can define your own objects.

- Objects and Properties
- Functions and Methods
- Creating New Objects

Objects and Properties

A JavaScript object has properties associated with it. You access the properties of an object with a simple notation:

```
objectName.propertyName
```

Both the object name and property name are case sensitive.
You define a property by assigning it a value. For example, suppose there is an object named *myCar* (we'll discuss how to create objects later-for now, just assume the object already exists). You can give it properties named **make**, **model**, and **year** as follows:

```
myCar.make = "Ford"

myCar.model = "Mustang"

myCar.year = 69;
```

An array is an ordered set of values associated with a single variable name. Properties and arrays in JavaScript are intimately related; in fact, they are different interfaces to the same data structure. So, for example, you could access the properties of the *myCar* object described above as follows:

```
myCar["make"] = "Ford"

myCar["model"] = "Mustang"

myCar["year"] = 67;
```

Equivalently, each of these elements can be accessed by its index, as follows:

```
myCar[0] = "Ford"

myCar[1] = "Mustang"

myCar[2] = 67;
```

This type of an array is known as an associative array, because each index element is also associated with a string value. To illustrate how this works, the following function displays the properties of the object, when you pass the

object and the object's name as arguments to the function:

```
function show_props(obj, obj_name) {

    var result = ""
    for (var i in obj)
       result += obj_name + "." + i + " = " + obj[i] + "\n"
    return result;
}
```

So, the function call `show_props(myCar, "car")` would return the following:

```
myCar.make = Ford

myCar.model = Mustang

myCar.year = 67
```

Functions and Methods

Functions are one of the fundamental building blocks in JavaScript. A function is a JavaScript procedure--a set of statments that performs a specific task. In a Navigator application, you can use any functions defined in the current page. It is generally a good idea to define all your functions in the HEAD of a page. When a user loads the page, the functions will then be loaded first.

The statements in a function can include other function calls defined for the current application.

A function definition consists of the **function** keyword, followed by

- the name of the function
- a list of parameters to the function, enclosed in parentheses, and separated by commas
- the JavaScript statements that define the function, enclosed in curly braces, {...}

The statements in a function can include other function calls defined for the current application. A function can be recursive, that is, it can call itself.

For example, here is the definition of a simple function named pretty_print:

```
function pretty_print(string) {

   document.write("<HR><P>" + string)

}
```

This function takes a string as its argument, adds some HTML tags to it using the concatenation operator (+), then displays the result to the current document.

Defining a function does not execute it. You have to *call* the function for it to do its work. For example, you could call the pretty_print function as follows:

```
<SCRIPT>
pretty_print("This is some text to display")
</SCRIPT>
```

The parameters of a function are not limited to just strings and numbers. You can pass whole objects to a function, too.

The show_props function from the previous section is an example of a function that takes an object as an argument.

A function can even be recursive, that is, it can call itself. For example, here is a function that computes factorials:

```
function factorial(n) {
  if ((n == 0) || (n == 1))
    return 1
  else {
    result = (n * factorial(n-1) )
    return result
  }
}
```

You could call this function with an argument of one through five inside a loop as follows:

```
for (x = 0; x < 5; x++) {
   document.write(x, " factorial is ", factorial(x))
   document.write("<BR>")
}
```

The results would be:

0 factorial is 1
1 factorial is 1
2 factorial is 2
3 factorial is 6
4 factorial is 24
5 factorial is 120

Methods

A *method* is a function associated with an object. You define a method in the same way as you define a standard function. Then, use the following syntax to associate the function with an existing object:

```
object.methodname = function_name
```

where object is an existing object, *methodname* is the name you are assigning to the method, and function_name is the name of the function.

You can then call the method in the context of the object as follows:

```
object.methodname(params);
```

Using this for Object References

JavaScript has a special keyword, **this**, that you can use to refer to the current object. For example, suppose you

have a function called *validate* that 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!")

}
```

Then, 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:

```
<INPUT TYPE = "text" NAME = "age" SIZE = 3 onChange="validate(this, 18, 99)">
```

In general, in a method **this** refers to the calling object.

Creating New Objects

Both client and server JavaScript have a number of predefined objects. In addition, you can create your own objects. Creating your own object requires two steps:

- Define the object type by writing a function.
- Create an instance of the object with **new**.

To define an object type, create a function for the object type that specifies its name, and its properties and methods. For example, 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, year, and color. To do this, you would write the following function:

```
function car(make, model, year) {

   this.make = make;

   this.model = model;

   this.year = year;
}
```

Notice the use of **this** to assign values to the object's properties based on the values passed to the function.

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", `car1.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)
```

An object can have a property that is itself another object. For example, suppose I 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 we 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)
```

Notice that instead of passing a literal string or integer valuewhen creating the new objects, the above statements pass the objects rand and ken as the parameters of the owners. Then if you want to find out the name of the owner of car2, you can access the following property:

```
car2.owner.name
```

Note that 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 have to add the property to the

definition of the *car* object type.

Defining Methods

You can define methods for an object type by including a method defnition in the object type definition. For example, suppose you have a set of image GIF files, and you want to define a method that displays the information for the cars, along with the corresponding image. You could define a function such as:

```
function displayCar() {
   var result = "A Beautiful " + this.year
                + " " + this.make + " " + this.model;
   pretty_print(result)
}
```

where *pretty_print* is the previously defined function to display a string. Notice the use of **this** to refer to the object to which the method belongs.
You can make this function a method of *car* by adding the statement

```
this.displayCar = displayCar;
```

to the object definition. So, the full definition of *car* would now look like:

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

Then you can call this new method as follows:

```
car1.displayCar()

car2.displayCar()
```

This will produce output like this:

A Beautiful 1993 Eagle Talon TSi

A Beautiful 1992 Nissan 300ZX

Using Built-in Objects and Functions

The JavaScript Language contains the following built-in objects and functions:

- String object
- Math object
- Date object
- Built-in functions

These objects and their properties and methods are built into the language. You can use these objects in both client applications with Netscape Navigator and server applications with LiveWire.

Using the String Object

Whenever you assign a string value to a variable or property, you create a string object. String literals are also string objects. For example, the statement

```
mystring = "Hello, World!"
```

creates a string object called mystring. The literal "blah" is also a string object.

The string object has methods that return:

- a variation on the string itself, such as *substring* and *toUpperCase*.
- an HTML formatted version of the string, such as *bold* and *link*.

For example, given the above object, `mystring.toUpperCase()` returns "HELLO, WORLD!", and so does `"hello, world!".toUpperCase()`.

Using the Math Object

The built-in Math object has properties and methods for mathematical constants and functions. For example, the Math object's PI property has the value of pi, which you would use in an application as

```
Math.PI
```

Similarly, standard mathematical functions are methods of Math. These include trigonometric, logarithmic, exponential, and
other functions. For example, if you want to use the trigonometric function sine, you would write

```
Math.sin(1.56)
```

Note that all trigonometric methods of math take arguments in radians.

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)

}
```

Using the Date Object

JavaScript does not have a date data type. However, the date object and its methods enable you to work with dates and times in your applications. The date object has a large number of methods for setting, getting, and manipulating dates. It does not have any properties.

JavaScript handles dates very similarly to Java. The two languages have many of the same date methods, and both languages store dates as the number of milliseconds since January 1, 1970 00:00:00.

NOTE: You cannot currently work with dates prior to 1/1/70.

To create a date object:

```
varName = new Date(parameters)
```

where *varName* is a JavaScript variable name for the date object being created; it can be a new object or a property of an existing object.

The *parameters* for the Date constructor can be any of the following:

- Nothing: creates today's date and time. For example, `today = new Date()`
- A string representing a date in the following form: "Month day, year hours:minutes:seconds". For example, `Xmas95= new Date("December 25, 1995 13:30:00")` If you omit hours, minutes, or seconds, the value will be set to zero.
- A set of integer values for year, month, and day. For example, `Xmas95 = new Date(95,11,25)`
- A set of values for year, month, day, hour, minute, and seconds For example, `Xmas95 = new Date(95,11,25,9,30,0)`

The Date object has a large number of methods for handling dates and times. The methods fall into these broad categories:

- "set" methods, for setting date and time values in date objects
- "get" methods, for getting date and time values from date objects
- "to" methods, for returning string values from date objects.
- parse and UTC methods, for parsing date strings.

The "get" and "set" methods enable you to get and set seconds, minutes, hours, day of the month, day of the week, months, and years separately. There is a getDay method that returns the day of the week, but no corresponding setDay method, because the day of the week is set automatically. These methods use integers to represent these values as follows:

- seconds and minutes: 0 to 59

- hours: 0 to 23
- day: 0 to 6 (day of the week)
- date: 1 to 31 (day of the month)
- months: 0 (January) to 11 (December)
- year: years since 1900

For example, suppose you define the following date:

```
Xmas95 = new Date("December 25, 1995")
```

Then `Xmas95.getMonth()` returns 11, and `Xmas95.getYear()` returns 95.

The getTime and setTime methods are useful for comparing dates. The getTime method returns the number of milliseconds since the epoch for a date object.

For example, the following code displays the number of shopping days left until Christmas:

```
today = new Date()

nextXmas = new Date("December 25, 1990")

nextXmas.setYear(today.getYear())

msPerDay = 24 * 60 * 60 * 1000 ; // Number of milliseconds per day

daysLeft = (nextXmas.getTime() - today.getTime()) / msPerDay;

daysLeft = Math.round(daysLeft);

document.write("Number of Shopping Days until Christmas: " + daysLeft);
```

This example creates a date object named today that contains today's date. It then creates a date object named nextXmas, and sets the year to the current year. Then, using the number of milliseconds per day, it computes the number of days between today and nextXmas, using getTime, and rounding to a whole number of days.

The parse method is useful for assigning values from date strings to existing date objects. For example, the following code uses parse and setTime to assign a date to the IPOdate object.

```
IPOdate = new Date()

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

Using Built-in Functions

JavaScript has several “top-level” functions built-in to the language.
They are :

- eval
- parseInt
- parseFloat

The eval Function

The built-in function *eval* takes a string as its argument.The string can be is any string representing a JavaScript expression, statement, or sequence of statements. The expression can include variables and properties of existing objects.

If the argument represents an expression, eval evaluates the expression.
If the argument represents one or more JavaScript statements, eval performs the statements.

This function is useful for evaluating a string representing a numerical expression to a number. For example, input from a form element is always in a string, but you often want to convert it to a numerical value.

The following example takes input in a text field, applies the eval function and displays the result in another text field. If you type a numerical expression in the first field, and click on the button, the expression will be evaluted. For example, enter "(666 * 777) / 3", and click on the button to see the result.

```
<SCRIPT>

function compute(obj) {

   obj.result.value = eval(obj.expr.value)

}

</SCRIPT>

<FORM NAME="evalform">

Enter an expression: <INPUT TYPE=text NAME="expr" SIZE=20 >

<BR>

Result: <INPUT TYPE=text NAME="result" SIZE=20 >

<BR>

<INPUT TYPE="button" VALUE="Click Me" onClick="compute(this.form)">

</FORM>
```

The eval function is not limited to evaluating numerical expressions, however. Its argument can include object references or even JavaScript statements. For example, you could define a function called setValue that would take two arguments: and object and a value, as follows:

```
function setValue (myobj, myvalue) {
  eval ("document.forms[0]." + myobj + ".value") = myvalue;
}
```

Then, for example, you could call this function to set the value of a form element "text1" as follows:

```
setValue(text1, 42)
```

The parseInt and parseFloat Functions

These two built-in functions return a numeric value when given a string as an argument.

ParseFloat parses its argument, a string, and attempts to return 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, it returns *NaN*.

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. If the first character cannot be converted to a number in the specified radix, it returns *NaN*. ParseInt truncates numbers to integer values.

Overview of JavaScript Statements

JavaScript supports a compact set of statements that nevertheless enables you to incorporate a great deal of interactivity in web pages.

- Variable Declaration / Assignment
- Function Definition
- Conditionals
- Loops
    - for loop
    - while loop
    - for...in loop
    - break and continue statements
- with statement
- Comments

Further overview information TBD. Refer to statements reference for specific information.

Navigator Objects

- Using Navigator Objects
- Navigator Object Hierarchy
- JavaScript and HTML Layout
- Key Navigator Objects

Using Navigator Objects

When you load a page in Navigator, it creates a number of objects corresponding to the page, its contents, and other pertinent information.

Every page always has the following objects:

- window: the top-level object; contains properties that apply to the entire window. There is also a window object for each "child window" in a frames document.
- location: contains properties on the current URL
- history: contains properties representing URLs the user has previously visited
- document: contains properties for content in the current document, such as title, background color, and forms

The properties of the document object are largely content-dependent. That is, they are created based on the content that you put in the document. For example, the document object has a property for each form and each anchor in the document.

For example, suppose you create a page named simple.html that contains the following HTML:

```
<TITLE>A Simple Document</TITLE>
<BODY><FORM NAME="myform" ACTION="FormProc()" METHOD="get" >Enter a value: <INPUT
TYPE=text NAME="text1" VALUE="blahblah" SIZE=20 >
Check if you want:
<INPUT TYPE="checkbox" NAME="Check1" CHECKED onClick="update(this.form)"> Option
#1
<P>
<INPUT TYPE="button" NAME="Button1" VALUE="Press Me" onClick="update(this.form)">
</FORM></BODY>
```

As always, there would be window, location, history, and document objects. These would have properties such as:

- location.href = "http://www.terrapin.com/samples/vsimple.html"
- document.title = "A Simple Document"
- document.fgColor = #000000
- document.bgColor = #ffffff
- history.length = 7

These are just some example values. In practice, these values would be based on the document's actual location, its title, foreground and background colors, and so on.
Navigator would also create the following objects based on the contents of the page:

- document.myform
- document.myform.Check1
- document.myform.Button1

These would have properties such as:

- document.myform.action = http://terrapin/mocha/formproc()
- document.myform.method = get
- document.myform.length = 5
- document.myform.Button1.value = Press Me
- document.myform.Button1.name = Button1
- document.myform.text1.value = blahblah
- document.myform.text1.name = text1
- document.myform.Check1.defaultChecked = true
- document.myform.Check1.value = on
- document.myform.Check1.name = Check1

Notice that each of the property references above starts with "document," followed by the name of the form, "myform," and then the property name (for form properties) or the name of the form element. This sequence follows the Navigator's object hierarchy, discussed in the next section.

Navigator Object Hierarchy

The objects in Navigator exist in a hierarchy that reflects the hierarchical structure of the HTML page itself. Although you cannot derive object classes from these objects, as you can in languages such as Java, it is useful to understand the Navigator's JavaScript object hierarchy. In the strict object-oriented sense, this type of hierarchy is known as an *instance hierarchy*, since it concerns specific instances of objects rather than object classes.

In this hierarchy, an object's "descendants" are properties of the object. For example, a form named "form1" is an object, but is also a property of document, and is referred to as "document.form1". The Navigator object hierarchy is illustrated below:

```
navigator
window
  |
  +--parent, frames, self, top
  |
  +--location
  |
  +--history
  |
  +--document
       |
       +--forms
       |    |
       |  elements (text fields, textarea, checkbox, password
       |            radio, select, button, submit, reset)
```

```
        +--links
        |
        +--anchors
```

To refer to specific properties of these objects, you must specify the object name and all its ancestors. **Exception:** You are not required to include the window object.

JavaScript and HTML Layout

To use JavaScript properly in the Navigator, it is important to have a basic understanding of how the Navigator performs *layout*. Layout refers to transforming the plain text directives of HTML into graphical display on your computer. Generally speaking, layout happens sequentially in the Navigator. That is, the Navigator starts from the top of the HTML file and works its way down, figuring out how to display output to the screen as it goes. So, it starts with the HEAD of an HTML document, then starts at the top of the BODY and works its way down.

Because of this "top-down" behavior, JavaScript only reflects HTML that it has encountered. For example, suppose you define a form with a couple of text input elments:

```
<FORM NAME="statform">
<input type = "text" name = "username" size = 20>
<input type = "text" name = "userage" size = 3>
```

Then these form elements are reflected as JavaScript objects *document.statform.username* and *document.statform.userage*, that you can use anywhere **after** the form is defined. However, you could not use these objects **before** the form is defined. So, for example, you could display the value of these objects in a script after the form definition:

```
<SCRIPT>
document.write(document.statform.username.value)
document.write(document.statform.userage.value)
</SCRIPT>
```

However, if you tried to do this before the form definition (i.e. above it in the HTML page), you would get an error, since the objects don't exist yet in the Navigator.

Likewise, once layout has occurred, setting a property value does not affect its value or its appearance. For example, suppose you have a document title defined as follows:

```
<TITLE>My JavaScript Page</TITLE>
```

This is reflected in JavaScript as the value of *document.title*. Once the Navigator has displayed this in layout (in this case, in the title bar of the Navigator window), you cannot change the value in JavaScript. So, if later in the page, you have the following script:

```
document.title = "The New Improved JavaScript Page"
```

it will not change the value of *document.title* nor affect the appearance of the page, nor will it generate an error.

Key Navigator Objects

Some of the most useful Navigator objects include document, form, and window.

Using the document Object

One of the most useful Navigator objects is the document object, because its write and writeln methods can generate HTML. These methods are the way that you display JavaScript expressions to the user. The only difference between write and writeln is that writeln adds a carriage return at the end of the line. However, since HTML ignores carriage returns, this will only affect preformatted text, such as that inside a PRE tag.

The document object also has onLoad and onUnload event-handlers to perform functions when a user first loads a page and when a user exits a page.

There is only one document object for a page, and it is the ancestor for all the form, link, and anchor objects in the page.

Using the form Object

Navigator creates a form object for each form in a document. You can name a form with the NAME attribute, as in this example:

```
<FORM NAME="myform">
<INPUT TYPE="text" NAME="quantity" onChange="...">
...
</FORM>
```

There would be a JavaScript object named *myform* based on this form. The form would have a property corresponding to the text object, that you would refer to as

```
document.myform.quantity
```

You would refer to the value property of this object as

```
document.myform.quantity.value
```

The forms in a document are stored in an array called *forms*. The first (topmost in the page) form is *forms[0]*, the second *forms[1]*, and so on. So the above references could also be:

```
document.forms[0].quantity
document.forms[0].quantity.value
```

Likewise, the elements in a form, such as text fields, radio buttons, and so on, are stored in an *elements* array.

Using the window Object

The window object is the "parent" object for all other objects in Navigator. You can always omit the object name in references to window properties and methods.

Window has several very useful methods that create new windows and pop-up dialog boxes:

- open and close: Opens and closes a browser window
- alert: Pops up an alert dialog box
- confirm: Pops up a confirmation dialog box

The window object has properties for all the frames in a frameset. The frames are stored in the frames array. The frames array contains an entry for each child frame in a window. For example, if a window contains three child frames, these frames are reflected as window.frames[0], window.frames[1], and window.frames[2].

The status property enables you to set the message in the status bar at the bottom of the client window.

Objects

The following objects are available in JavaScript:

- anchor(*anchors* array)
- button
- checkbox
- Date
- document
- *elements* array
- form(*forms* array)
- frame(*frames* array)
- hidden
- history
- link(*links* array)
- location
- Math
- navigator
- password
- radio
- reset
- select(*options* array)
- string
- submit
- text
- textarea
- window

---

anchor object (*anchors* array)

A piece of text that can be the target of a hypertext link.

Syntax

To define an anchor, use standard HTML syntax:

```
<A [HREF=locationOrURL]
   NAME="anchorName"
   [TARGET="windowName"]>
   anchorText
</A>
```

HREF=locationOrURL identifies a destination anchor or URL. If this attribute is present, the anchor object is also a link object. See link for details.
NAME="anchorName" specifies a tag that becomes an available hypertext target within the current document.
TARGET="windowName" specifies the window that the link is loaded into. This attribute is meaningful only if *HREF=locationOrURL* is present. See link for details.
anchorText specifies the text to display at the anchor.

You can also define an anchor using the anchor method.

Description

If an anchor object is also a link object, the object has entries in both the anchors and links arrays.

The *anchors* array

You can reference 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 in source order. For example, if a document contains three named anchors, these anchors are reflected as `document.anchors[0], document.anchors[1], and document.anchors[2].`

To use the *anchors* array:

```
1. document.anchors[index]
2. document.anchors.length
```

index is an integer representing an anchor in a document.

To obtain the number of anchors in a document, use the length property: `document.anchors.length`.

Even though the *anchors* array represents named anchors, the value of anchors[*index*] is always null. But 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`. See the example below.

Elements in the *anchors* array are read-only. For example, the statement `document.anchors[0]="anchor1"` has no effect.

Properties

The anchors object has no properties. The *anchors* array has the following properties:

- length reflects the number of named anchors in a document

Methods

None.

Event handlers

None.

Property of

document

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>
<LI>one
<LI>two
<LI>three
<LI>four
<LI>five
<LI>six
```

```
<LI>seven
<LI>eight
<LI>nine
<P><A NAME="1"><B>Some colors</B> (Anchor 1)</A>
<LI>red
<LI>orange
<LI>yellow
<LI>green
<LI>blue
<LI>purple
<LI>brown
<LI>black
<P><A NAME="2"><B>Some music types</B> (Anchor 2)</A>
<LI>R&B
<LI>Jazz
<LI>Soul
<LI>Reggae
<LI>Rock
<LI>Country
<LI>Classical
<LI>Opera
<P><A NAME="3"><B>Some countries</B> (Anchor 3)</A>
<LI>Afghanistan
<LI>Brazil
<LI>Canada
<LI>Finland
<LI>India
<LI>Italy
<LI>Japan
<LI>Kenya
<LI>Mexico
<LI>Nigeria
</BODY>
</HTML>
```

See also

- link object
- anchors method

---

button object (client)

A pushbutton on an HTML form.

Syntax

To define a button:

```
<INPUT
   TYPE="button"
   NAME="buttonName"
   VALUE="buttonText"
   [onClick="handlerText"]>
```

NAME="buttonName" specifies the name of the button object. You can access this value using the name

property.
VALUE="buttonText" specifies the label to display on the button face. You can access this value using the value property.

To use a button object's properties and methods:

```
1. buttonName.propertyName
2. buttonName.methodName(parameters)
3. formname.elements.[index].propertyName
4. formname.elements.[index].methodname(parameters)
```

buttonName is the value of the NAME attribute of a button object.
formName is either the value of the NAME attribute of a form object or an element in the *forms* array.
index is an integer representing a button object on a form.
propertyName is one of the properties listed below. *methodName* is one of the methods listed below.
methodName is one of the methods listed below.

Description

A button object is a form element and must be defined with a <FORM>...</FORM> tag.
The button object is a custom button that you can use to perform an action you define.

Properties

- name reflects the NAME attibute
- value reflects the VALUE attribute

Methods

- click

Event handlers

- onClick

Property of

- form

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, and submit objects

checkbox object (client)

A checkbox on an HTML form. A checkbox is a toggle switch that lets the user set a value on or off.

Syntax

To define a checkbox, use standard HTML syntax with the addition of the onClick event handler:

```
<INPUT
   TYPE="checkbox"
   NAME="checkboxName"
   [CHECKED]
   [onClick="handlerText"]>
   textToDisplay
```

NAME="checkboxName" specifies the name of the checkbox object. You can access this value using the name property.
VALUE="checkboxValue" specifies a value that is returned to the server when the checkbox is selected and the form is submitted. This defaults to "on". You can access this value using the value property.
CHECKED specifies that the checkbox is displayed as checked. You can access this value using the defaultChecked property.
textToDisplay specifies the label to display beside the checkbox.

To use a checkbox object's properties and methods:

```
1. checkboxName.propertyName
2. checkboxName.methodName(parameters)
3. formName.elements[index].propertyName
4. formName.elements[index].methodName(parameters)
```

checkboxName is the value of the NAME attribute of a checkbox object.
formName is either the value of the NAME attribute of a form object or an element in the forms array.
index is an integer representing a checkbox object on a form.
propertyName is one of the properties listed below.
methodName is one of the methods listed below.

Description

A checkbox object is a form element and must be defined within a <FORM>...</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.

Properties

- checked lets you programatically check a checkbox
- defaultChecked reflects the CHECKED attribute

- name reflect the NAME attribute
- value reflects the VALUE attribute

Methods

- click

Event handlers

- onClick

Property of

form

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 checkbox lets the user choose whether the text fields are converted to upper case. Each text field has an onChange event handler that converts the field value to upper case if the checkbox is checked. The checkbox has an onClick event handler that converts all fields to upper case 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 and radio object s

Date object (common)

Lets you work with dates and times.

Syntax

To create a Date object:

```
1. dateObjectName = new Date()
2. dateObjectName = new Date("month day, year hours:minutes:seconds")
3. dateObjectName = new Date(year, month, day)
4. dateObjectName = new Date(year, month, day, hours, minutes, seconds)
```

dateObjectName is either the name of a new object or a property of an existing object.
month, day, year, hours, minutes, and seconds are string values for form 2 of the syntax. For forms 3 and 4, they are integer values.

To use Date methods:

```
dateObjectName.methodName(parameters)
```

dateObjectName is either the name of an existing Date object or a property of an existing object..
methodName is one of the methods listed below.

Exceptions: The Date object's parse and UTC methods are static methods that you use as follows:

```
Date.UTC(parameters)
Date.parse(parameters)
```

Description

The Date object is a built-in JavaScript object.

Form 1 of the syntax creates today's date and time. If you omit hours, minutes, or seconds from form 2 or 4 of the syntax, the value will be set to zero.

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.

Properties

None.

Methods

- getDate
- getDay
- getHours
- getMinutes
- getMonth
- getSeconds
- getTime
- getTimeZoneoffset
- getYear
- parse
- setDate
- setHours
- setMinutes
- setMonth
- setSeconds
- setTime
- setYear
- toGMTString
- toLocaleString
- UTC

Event handlers

None. Built-in objects do not have event handlers.

Property of

None

Examples

```
today = new Date()
birthday = new Date("December 17, 1995 03:24:00")
birthday = new Date(95,12,17)
birthday = new Date(95,12,17,3,24,0)
```

---

document object

Contains information on the current document, and provides methods for displaying HTML output to the user.

Syntax

To define a document object, use standard HTML syntax :

```
<BODY
   BACKGROUND="backgroundImage"
   BGCOLOR="backgroundColor"
   FGCOLOR="foregroundColor"
```

```
    LINK="unfollowedLinkColor"
    ALINK="activatedLinkColor"
    VLINK="followedLinkColor"
    [onLoad="handlerText"]
    [onUnLoad="handlerText"]>
</BODY>
```

BACKGROUND specifies an image that fills the background of the document.
BGCOLOR, TEXT, LINK, ALINK, and *VLINK* are color specifications expressed as a hexadecimal RGB triplet (in the format "rrggbb" or "#rrggbb") or as one of the string literals listed in Color Values.

To use a document object's properties and methods:

```
1. document.propertyName
2. document.methodName(parameters)
```

propertyName is one of the properties listed below.
methodName is one of the methods listed below.

Description

An HTML document consists of a <HEAD> and <BODY> tag. The <HEAD> 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 reference 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.

Properties

- alinkColor reflects the ALINK attribute
- anchors is an array reflecting all the anchors in a document
- bgColor reflects the BGCOLOR attribute
- cookie specifies a cookie
- fgColor reflects the TEXT attribute
- forms is an array reflecting all the forms in a document
- lastModified reflects the date a document was last modified
- linkColor reflects the LINK attribute
- links is an array reflecting all the links in a document
- location reflects the complete URL of a document
- referrer reflects the URL of the calling document
- title reflects the contents of the &LTTITLE&GT tag
- vlinkColor reflects the VLINK attribute

Methods

- clear
- close
- open
- write

- writeln

Event handlers

- None. The onLoad and onUnload event handlers are specified in the <BODY> tag but are actually event handlers for the window object.

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>
<LI>one
<LI>two
<LI>three
```

```
<LI>four
<LI>five
<LI>six
<LI>seven
<LI>eight
<LI>nine
<P><A NAME="colors"><B>Some colors</B></A>
<LI>red
<LI>orange
<LI>yellow
<LI>green
<LI>blue
<LI>purple
<LI>brown
<LI>black
<P><A NAME="musicTypes"><B>Some music types</B></A>
<LI>R&B
<LI>Jazz
<LI>Soul
<LI>Reggae
<LI>Rock
<LI>Country
<LI>Classical
<LI>Opera
<P><A NAME="countries"><B>Some countries</B></A>
<LI>Afghanistan
<LI>Brazil
<LI>Canada
<LI>Finland
<LI>India
<LI>Italy
<LI>Japan
<LI>Kenya
<LI>Mexico
<LI>Nigeria
</BODY>
</HTML>
```

See also

- frame and window objects

---

elements array

An array of objects corresponding to form elements (such as checkbox, radio, and text objects) in source order.

Syntax

```
1. formName.elements[index]
2. formName.elements.length
```

formName is either the name of a form or an element in the *forms* array.
index is an integer representing an object on a form.

Description

You can reference a form's elements in your code by using the *elements* array. This array contains an entry for each object (button, checkbox, hidden, password, radio, reset, select, submit, text, or textarea object) in a form in source order. For example, if a form has a text field and two checkboxes, these elements are reflected as *formName*.elements[0], *formName*.elements[1], and *formName*.elements[2].

Although you can also reference a form's elements by using the element's name (from the NAME attribute), the elements array provides a way to reference form objects programatically 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
```

To obtain the number of elements on a form, use the length property: *formName*.elements.length. Each radio button in a radio object appears as a separate element in the *elements* array.

Elements in the *elements* array are read-only. For example, the statement *formName*.elements[0]="music" has no effect.

The value of each element in the <I>elements</I> array is the full HTML statement for the object.

Properties

- length reflects the number of elements on a form

Property of

- form

Examples

See the examples for the name property.

See also

- form object

---

form object (*forms* array)

Lets users input text and make choices from form objects such as checkboxes, radio buttons, and selection lists. You can also use a form to post data to a server.

Syntax

To define a form, use standard HTML syntax with the addition of the onSubmit event handler:

```
<FORM
   NAME="formName"
   TARGET="windowName"
   ACTION="serverURL"
```

```
    METHOD=GET | POST
    ENCTYPE="encodingType"
    [onSubmit="handlerText"]>
</FORM>
```

TARGET="windowName" specifies the window that form responses go to. When you submit a form with a TARGET attribute, server responses are displayed in the specified window instead of the window that contains the form. *windowName* can be an existing window; it can be a frame name specified in a <FRAMESET> tag; or it can be one of the literal frame names _top, _parent, _self, or _blank; it cannot be a JavaScript expression (for example, it cannot be parent.*frameName* or *windowName.frameName*). Some values for this attribute may require specific values for other attributes. See RFC 1867 for details. You can access this value using the target property.

ACTION="serverURL" specifies the URL of the server to which form field input information is sent. This attribute can specify a CGI or LiveWire application on the server; it can also be a mailto: URL if the form is to be mailed. See the location object for a description of the URL components. Some values for this attribute may require specific values for other attributes. See RFC 1867 for details. You can access this value using the action property.

METHOD=GET | POST specifies how information is sent to the server specified by *ACTION*. GET (the default) appends the input information to the URL which on most receiving systems becomes the value of the environment variable *QUERY_STRING*. POST sends the input information in a data body which is available on *stdin* with the data length set in the environment variable *CONTENT_LENGTH*. Some values for this attribute may require specific values for other attributes. See RFC 1867 for details. You can access this value using the method property.

ENCTYPE="encodingType" specifies the MIME encoding of the data sent: "application/x-www-form-urlencoded" (the default) or "multipart/form-data". Some values for this attribute may require specific values for other attributes. See RFC 1867 for details. You can access this value using the encoding property.

To use a form object's properties and methods:

1. *formName.propertyName*
2. *formName.methodName*(*parameters*)
3. forms[*index*].*propertyName*
4. forms[*index*].*methodName*(*parameters*)

formName is the value of the NAME attribute of a form object.
index is an integer representing a form object.
propertyName is one of the properties listed below.
methodName is one of the methods listed below.

Description

Each form in a document is a distinct object.

You can reference the form objects in your code by using the forms property of the document object. The forms property is an array that contains an entry for each form in a document.

You can reference a form's elements in your code by using the form name or the forms property of the document object. The forms property is an array that contains an entry for each form in a document.

Properties

- action reflects the ACTION argument.
- elements
- encoding reflects the ENCTYPE argument.
- method reflects the METHOD argument.
- target reflects the TARGET argument.

Methods

- submit

Event handlers

- onSubmit

Examples

xxx to be supplied

See also

- elements and forms properties

---

frame object (client)

A frame is a sub-HTML document; a series of frames makes up the page.

Syntax

To define a frame object, use standard HTML syntax. The onLoad and onUnload event handlers are specified in the <FRAMESET> tag but are actually event handlers for the window object :

```
<FRAMESET
   ROWS="rowHeightValueList"
   COLS="columnWidthList">
   textToDisplay
   [onLoad="handlerText"]
   [onUnload="handlerText"]
   [<FRAME SRC="locationOrURL" NAME="frameName">]
</FRAMESET>
```

ROWS = "rowHeightValueList" is a comma-separated list of values specifying the row-height of the frame. An optional suffix defines the units. Default units are pixels.
COLS= "columnWidthList" is a comma-separated list of values specifying the column-width of the frame. An optional suffix defines the units. Default units are pixels.
textToDisplay specifies the text to display in the frame.
FRAME defines a frame.
SRC= "locationOrURL" specifies the URL of the document to be displayed in the frame.

NAME=”frameName” specifies a name to be used as a target of hypertext links.

To use a frame's properties and methods:

1. `frameName.propertyName`
2. `frameName.methodName.(properties)`
3. `frames[index].propertyName`
4. `frames[index].methodName(properties)`

frameName is the value of the NAME attribute in the <FRAME> tag of a frame object.
propertyName is one of the properties listed below.
methodName is one of the methods listed below.
index is an integer representing a frame object.

Description

The <FRAMESET> tag is used in an HTML document whose sole purpose is to define the layout of the sub-HTML documents, or frames, that make up the page. Each frame is a window object.

You can reference the frame objects in your code by using the frames property of the window object. The frames property is an array that contains an entry for each frame in a window containing a <FRAMESET> tag.

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]`.

Properties

A frame object is a type of window and has the same properties as a window object. Some properties, however, have no effect on a frame because frames do not have all the features of windows; for example, setting status and defaultStatus has no effect because a frame has no status bar.

- defaultStatus
- frames
- parent
- self
- status
- top
- window

Methods

A frame object is a type of window and has the same methods as a window object.

- alert
- close
- confirm
- open
- prompt
- setTimeout

- clearTimeout

Event handlers

None. The onLoad and onUnload event handlers are specified in the <FRAMESET> tag but are actually event handlers for the window object.

Examples

xxx to be supplied

See also

- document and window objects
- frames property

---

hidden object (client)

A text object that is suppressed from form display on an HTML form.

Syntax

To define a hidden object:

```
<INPUT
   TYPE="hidden"
   NAME="hiddenName"
   [VALUE="textValue"]
```

NAME="hiddenName" specifies the name of the hidden object as a property of the enclosing form object and can be accessed using the name property.
VALUE="textValue" specifies the value of the hidden object and can be accessed using the value property.

To use a hidden object's properties:

1. *hiddenName.propertyName*
2. *formName.elements*[*index*].*propertyName*

hiddenName is the value of the NAME attribute of a password object.
formName is either the value of the NAME attribute of a form object or an element in the <I>forms</I> array.
index is an integer representing a hidden object on a form.
propertyName is one of the properties listed below.

Description

A hidden object is a form element and must be defined within a `<FORM>...</FORM>` tag.
You can use hidden objects instead of cookies for client/server communication.

Properties

- defaultValue
- name reflects the NAME argument
- value reflects the VALUE argument

Methods

None.

Event handlers

None.

Examples

xxx to be supplied

See also

- cookie property

---

history object (client)

The history object contains information on the URLs that the client has visited. This information is stored in a history list, and is accessible through the Navigator's Go menu.

Syntax

To use a history object:

```
1. history.propertyName
2. history.methodName(parameters)
```

propertyName is one of the properties listed below.
methodName is one of the methods listed below.

Description

The history object is a linked list of URLs the user has visited, as shown in the Navigator's Go menu.

Properties

- length

Methods

- back
- forward

- go

Event handlers

None.

Examples

The following example goes to the URL the user visited three clicks ago.

```
history.go(-3)
```

See also

- location object

---

link object (client)

A link is a piece of text identified as a hypertext link. When the user clicks the link text, the link hypertext reference is loaded into its target window.

Syntax

To define a link, use standard HTML syntax with the addition of the onClick and onMouseOver event handlers:

```
<A [NAME="anchorName"]
   HREF=locationOrURL
   TARGET="windowName"
   [onClick="handlerText"]
   [onMouseOver="handlerText"]>
   linkText
</A>
```

NAME="anchorName" specifies a tag that becomes an available hypertext target within the current document.
HREF=locationOrURL identifies a destination anchor or URL.
TARGET="windowName" specifies the window that the link is loaded into. *WindowName* can be an existing window created by previous targeted form submits or link clicks; it can be a frame name specified in a <FRAMESET> tag; or it can be one of the magic frame names _top, _parent, _self, or _blank.
linkText is rendered as a hypertext link to the URL.

To use a link's properties:

```
document.links[index].propertyName
```

index is an integer representing a link object.
propertyName is one of the properties listed below.

Description

Each link object is a location object.

You can reference the link objects in your code by using the links property of the document object. The links property is an array that contains an entry for each link in a document.

When you specify a URL, you can use JavaScript statements in addition to using standard URL formats. The following list shows the syntax for specifying some of the most common types of URLs.

URL Type	Protocol	Example
JavaScript code	javascript	javascript:history.go(-1)
World Wide Web	http://	http://www.netscape.com/
File	file://	file:///javascript/methods.html
FTP	ftp://	ftp://ftp.mine.com/home/mine
MailTo	mailto://	mailto:info@netscape.com
Gopher	gopher://	gopher.myhost.com

Properties

- target reflects the TARGET argument.

Methods

None.

Event handlers

- onClick
- onMouseOver

Examples

The following example creates a hypertext link to an anchor named *javascript_intro*.

```
<A HREF="#javascript_intro">Introduction to JavaScript</A>
```

The following example creates a hypertext link to a URL.

```
<A HREF="http://www.netscape.com">Netscape Home Page</A>
```

The following example takes the user back x entries in the history list:

```
<A HREF="javascript:history.go(-1 * x)">Click here</A>
```

See also

- anchor object
- links property

---

location object (client)

The location object contains information on the current URL.

Syntax

To use a location object:

```
1. location.propertyName
2. location.methodName(parameters)
```

propertyName is one of the properties listed below.
methodName is one of the methods listed below.

Description

The location object represents a complete URL. Each property of the location object represents a different portion of the URL.

The following diagram of a URL shows the relationships between the location properties:

```
protocol//hostname:port pathname search hash
```

protocol represents the beginning of the URL, up to and including the first colon.
hostname represents the domain name or IP address of a network host.
port represents the port number to connect to.
pathname represents the url-path portion of the URL.
search represents any query information in the URL, beginning with a question mark.
hash represents an anchor name fragment in the URL, beginning with a hash mark (#).

The location object has two other properties not shown in the diagram above:

href represents a complete URL.
host represents the concatenation *hostname*:*port*.

The location object is contained by the window object and is within its scope. If you reference a location object without specifying a window, the location object represents the current location. If you reference a location object and specify a window name, for example, `windowName.location.propertyName`, the location object represents the location of the specified window.

Do not confuse the location object with the location property of the document object. You cannot change the

value of the location object, but you can change the value of the location property. Also, the location object has properties, and the location property does not. `document.location` is a string-valued property that usually matches what `window.location` is set to when you load the document, but redirection may change it.

Properties

- hash
- host
- hostname
- href
- pathname
- port
- protocol
- search

Methods

- assign
- toString

Event handlers

None.

Examples

xxx to be supplied

See also

- history object
- location property

Math object (common)

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

Syntax

To use a Math object:

```
1. Math.propertyName
2. Math.methodName(parameters)
```

propertyName is one of the properties listed below.
methodName is one of the methods listed below.

Description

You reference the constant PI as `Math.PI`. Constants are defined with the full precision of real numbers in JavaScript.

Similarly, you reference Math functions as methods. For example, the sine function is `Math.sin(argument)`, where *argument* is the argument.

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)
}
```

Properties

- E
- LN10
- LN2
- PI
- SQRT1_2
- SQRT2

Methods

- abs
- acos
- asin
- atan
- ceil
- cos
- exp
- floor
- max
- min
- pow
- random
- round
- sin
- sqrt
- tan

Event handlers

None. Built-in objects do not have event handlers.

Examples

xxx to be supplied

navigator object (client)

The navigator object contains information about the version of Navigator in use.

Syntax

To use a navigator object:

```
navigator.<I>propertyName</I>
```

propertyName is one of the properties listed below.

Description

Use the navigator object to determine which version of the Navigator your users have.

Properties

- appName
- appVersion
- appCodeName
- userAgent

Methods

None.

Event handlers

None.

Examples

```
<A NAME="javascript_intro"><H2>Welcome to JavaScript</H2></A>
```

See also

- link object
- anchors property

---

password object (client)

A password object is a text field on an HTML form. When the user enters text into the field, asterisks (*) hide anything entered from view.

Syntax

To define a password object, use standard HTML syntax:

```
<INPUT
  TYPE="password"
  NAME="passwordName"
  [VALUE="textValue"]
  SIZE=integer>
```

NAME="passwordName" specifies the name of the password object as a property of the enclosing form object and can be accessed using the name property.
VALUE="textValue" specifies the value of the password object and can be accessed using the value property.
SIZE=integer specifies the number of characters in the password object.

To use a password object's properties and methods:

1. *passwordName.propertyName*
2. *passwordName.methodName*(*parameters*)
3. *formName*.elements[*index*].*propertyName*
4. *formName*.elements[*index*].*methodName*(*parameters*)

passwordName is the value of the NAME attribute of a password object.
formName is either the value of the NAME attribute of a form object or an element in the *forms* array.
index is an integer representing a password object on a form.
propertyName is one of the properties listed below.
methodName is one of the methods listed below.

Description

A password object is a form element and must be defined within a <FORM>...</FORM> tag.

Properties

- defaultValue
- name reflects the NAME argument
- value reflects the VALUE argument

Methods

- focus
- blur
- select

Event handlers

None.

Examples

```
<B>Password:</B> <INPUT TYPE="password" NAME="password" VALUE="" SIZE=25>
```

See also

- form and text objects

---

radio object (client)

A radio object is a set of radio buttons on an HTML form. A set of radio buttons lets the user choose one item from a list.

Syntax

To define a set of radio buttons, use standard HTML syntax with the addition of the onClick event handler:

```
<INPUT
   TYPE="radio"
   NAME="radioName"
   VALUE="buttonValue"
   [CHECKED]
   [onClick="handlerText"]>
   textToDisplay
```

NAME="radioName" specifies the name of the radio object as a property of the enclosing form object and can be accessed using the name property. All radio buttons in a group should have the same NAME attribute.
VALUE="buttonValue" specifies the value returned when the radio button is selected and can be accessed using the value property. This defaults to "on".
CHECKED specifies that the radio button is selected and can be accessed using the checked property.
textToDisplay specifies the label to display beside the radio button and can be accessed using the value property

To use a radio button's properties and methods:

```
1. radioName[index1].propertyName
2. radioName[index1].methodName(parameters)
3. formName.elements[index2].propertyName
4. formName.elements[index2].methodName(parameters)
```

radioName is the value of the NAME attribute of a radio object.
index1 is an integer representing a radio button in a radio object.
formName is either the value of the NAME attribute of a form object or an element in the forms array.
index2 is an integer representing a radio button on a form. The *elements* array contains an entry for each radio button in a radio object.
propertyName is one of the properties listed below.
methodName is one of the methods listed below.

Description

A radio object is a form element and must be defined within a <FORM>...</FORM> tag.

All radio buttons in a radio button group use the same name property. To access the individual radio buttons in your code, follow the object name with an index starting from zero, one for each button the same way you would for an array such as forms: `document.forms[0].radioName[0]` is the first, `document.forms[0].radioName[1]` is the second, etc.

Properties

- checked reflects the CHECKED argument
- defaultChecked
- index
- length
- name reflects the NAME argument
- value reflects the VALUE argument

Methods

- click

Event handlers

- onClick

Examples

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. JavaScript automatically sets the catalog name input field based on the radio buttons.

```
<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
```

See also

- checkbox, form, and select objects

---

reset object (client)

A reset object is a reset button on an HTML form. A reset button resets all elements in a form to their defaults.

Syntax

To define a reset button, use standard HTML syntax with the addition of the onClick event handler:

```
<INPUT
   TYPE="reset"
   NAME="resetName"
   VALUE="buttonText"
   [onClick="handlerText"]>
```

NAME="resetName" specifies the name of the reset object as a property of the enclosing form object and can be accessed using the name property.
VALUE="buttonText" specifies the text to display on the button face and can be accessed using the value property.

To use a reset button's properties and methods:

```
1. resetName.propertyName
2. resetName.methodName(parameters)
3. formName.elements[index].propertyName
4. formName.elements[index].methodName(parameters)
```

resetName is the value of the NAME attribute of a reset object.
formName is either the value of the NAME attribute of a form object or an element in the *forms* array.
index is an integer representing a reset object on a form.
propertyName is one of the properties listed below.
methodName is one of the methods listed below.

Description

A reset object is a form element and must be defined within a <FORM>...</FORM> tag.

Properties

- name reflects the NAME argument
- value reflects the VALUE argument

Methods

- click

Event handlers

- onClick

Examples

The following example displays a text object containing "CA". If the user types a different 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">
```

See also

- button, form, and submit objects

---

select object (client)

A select object is a selection list or scrolling list on an HTML form. A selection list lets the user choose one item from a list. A scrolling list lets the user choose one or more items from a list.

Syntax

To define a select object, use standard HTML syntax with the addition of the onBlur, onChange, and onFocus event handlers:

```
<SELECT
   NAME="selectName"
   [SIZE="integer"]
   [MULTIPLE]
   [onBlur="handlerText"]
   [onChange="handlerText"]
   [onFocus="handlerText"]>
   <OPTION [SELECTED]> textToDisplay [ ... <OPTION> textToDisplay]
</SELECT>
```

NAME="selectName" specifies the name of the select object as a property of the enclosing form object and can be accessed using the name property.
SIZE="integer" specifies the number of options visible when the form is displayed and can be accessed using the length property.
MULTIPLE specifies that the select object is a scrolling list (not a selection list).
OPTION specifies a selection element in the list.
SELECTED specifies that the option is selected by default and can be accessed using the selected property.
textToDisplay specifies the text to display in the list and can be accessed using the value property.

To use a select object's properties and methods:

```
1. selectName.propertyName
2. selectName.methodName(parameters)
3. formName.elements[index].propertyName
4. formName.elements[index].methodName(parameters)
```

selectName is the value of the NAME attribute of a select object.
formName is either the value of the NAME attribute of a form object or an element in the <I>forms</I> array.
index is an integer representing a select object on a form.
propertyName is one of the properties listed below.
methodName is one of the methods listed below.

To use a select object's option's properties:

```
1. selectName.options[index1].propertyName
2. formName.elements[index2].options[index1].propertyName
```

selectName is the value of the NAME attribute of a select object.
index1 is an integer representing an option in a select object.
formName is either the value of the NAME attribute of a form object or an element in the *forms* array.
index2 is an integer representing a select object on a form.
propertyName is one of the properties listed below.

Description

A select object is a form element and must be defined within a <FORM>...</FORM> tag.

You can reference the options of a select object in your code by using the options property. The options property is an array that contains an entry for each option in a select object: selectName.options[0] is the first, selectName.options[1] is the second, etc. Each option has the properties listed below.

Properties

The select object has the following properties:

- length reflects the SIZE argument
- name reflects the NAME argument
- options
- selectedIndex

The options property has the following properties:

- defaultSelected
- index
- selected reflects the SELECTED argument
- text
- value

Methods

None.

Event handlers

- onBlur
- onChange
- onFocus

Examples

The following example displays a selection list.

```
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>
```

See also

- form and radio objects
- options property

---

string object (common)

A string object consists of a series of characters.

Syntax

To use a string object:

```
1. stringName.propertyName
2. stringName.methodName(parameters)
```

stringName is the name of a string variable.
propertyName is one of the properties listed below.
methodName is one of the methods listed below.

Description

A string can be represented as a literal enclosed by single or double quotes; for example, "Netscape" or 'Netscape'.

Properties

- length

Methods

- anchor
- big
- blink
- bold
- charAt
- fixed
- fontcolor
- fontsize
- indexOf
- italics
- lastIndexOf
- link
- small
- strike
- sub
- substring
- sup
- toLowerCase
- toUpperCase

Event handlers

None. Built-in objects do not have event handlers.

Examples

The following statement creates a string variable.

```
var last_name = "Schaefer"

last_name.length is 8.
last_name.toUpperCase() is "SCHAEFER".
last_name.toLowerCase() is "schaefer".
```

See also

- text and textarea objects

---

submit object (client)

A submit object is a submit button on an HTML form. A submit button causes a form to be submitted.

Syntax

To define a submit button, use standard HTML syntax with the addition of the onClick event handler:

```
<INPUT
   TYPE="submit"
   NAME="submitName"
   VALUE="buttonText"
   [onClick="handlerText"]>
```

NAME="submitName" specifies the name of the submit object as a property of the enclosing form object and can be accessed using the name property.
VALUE="buttonText" specifies the label to display on the button face and can be accessed using the value property.

To use a submit button's properties and methods:

```
1. submitName.propertyName
2. submitName.methodName(parameters)
3. formName.elements[index].propertyName
4. formName.elements[index].methodName(parameters)
```

submitName is the value of the NAME attribute of a submit object.
formName is either the value of the NAME attribute of a form object or an element in the *forms* array.
index is an integer representing a submit object on a form.
propertyName is one of the properties listed below.
methodName is one of the methods listed below.

Description

A submit object is a form element and must be defined within a <FORM>...</FORM> tag.
Clicking a submit button submits a form to the program 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.

Properties

- name reflects the NAME argument
- value reflects the VALUE argument

Methods

- click

Event handlers

- onClick

Examples

```
<INPUT TYPE="submit" NAME="submit_button" VALUE="Done">
```

See also

- button, form, and reset objects
- submit method

---

text object (client)

A text object is a text input field on an HTML form. A text field lets the user enter a word, phrase, or series of numbers.

Syntax

To define a text object, use standard HTML syntax with the addition of the onBlur, on Change, onFocus, and onSelect event handlers:

```
<INPUT
   TYPE="text"
   NAME="textName"
   VALUE="textValue"
   SIZE=integer
   [onBlur="handlerText"]
   [onChange="handlerText"]
   [onFocus="handlerText"]
   [onSelect="handlerText"]>
```

NAME="textName" specifies the name of the text object as a property of the enclosing form object and can be accessed using the name property.
VALUE="textValue" specifies the value of the text object and can be accessed using the value property.
SIZE=integer specifies the number of characters in the text object.

To use a text object's properties and methods:

1. *textName.propertyName*
2. *textName.methodName*(*parameters*)
3. *formName*.elements[*index*].*propertyName*
4. *formName*.elements[*index*].*methodName*(*parameters*)

textName is the value of the NAME attribute of a text object.
formName is either the value of the NAME attribute of a form object or an element in the *forms* array.
index is an integer representing a text object on a form.
propertyName is one of the properties listed below.
methodName is one of the methods listed below.

Description

A text object is a form element and must be defined within a <FORM>...</FORM> tag.

text objects can be updated (redrawn) dynamically by setting the value property (this.value).

Properties

- defaultValue
- name reflects the NAME argument
- value reflects the VALUE argument

Methods

- focus
- blur
- select

Event handlers

- onBlur
- onChange
- onFocus
- onSelect

Examples

```
<B>Last name:</B> <INPUT TYPE="text" NAME="last_name" VALUE="" SIZE=25>
```

See also

- form, password, string, and textarea objects

---

textarea object (client)

A textarea object is a multiline input field on an HTML form. A textarea field lets the user enter words, phrases, or numbers.

Syntax

To define a text area, use standard HTML syntax with the addition of the onBlur, onChange, onFocus, and onSelect event handlers:

```
<TEXTAREA
   NAME="textareaName"
   ROWS="integer"
   COLS="integer"
   [onBlur="handlerText"]
   [onChange="handlerText"]
   [onFocus="handlerText"]
   [onSelect="handlerText"]>
   textToDisplay
</TEXTAREA>
```

NAME="textareaName" specifies the name of the textarea object as a property of the enclosing form object and can be accessed using the name property.
ROWS="integer" and *COLS="integer"* define the physical size of the displayed input field in numbers of characters.
textToDisplay specifies the value of the textarea object and can be accessed using the value property. A textarea allows only ASCII text, and new lines are respected.

To use a textarea object's properties and methods:

1. *textareaName.propertyName*
2. *textareaName.methodName*(*parameters*)
3. *formName*.elements[*index*].*propertyName*
4. *formName*.elements[*index*].*methodName*(*parameters*)

textareaName is the value of the NAME attribute of a textarea object.
formName is either the value of the NAME attribute of a form object or an element in the *forms* array.
index is an integer representing a textarea object on a form.
propertyName is one of the properties listed below.
methodName is one of the methods listed below.

Description

A textarea object is a form element and must be defined within a <FORM>...</FORM> tag.

textarea objects can be updated (redrawn) dynamically by setting the value property (this.value).

Properties

- defaultValue
- name reflects the NAME argument
- value reflects the VALUE argument

Methods

- focus
- blur
- select

Event handlers

- onBlur
- onChange
- onFocus
- onSelect

Examples

```
<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>
```

See also

- form, password, string, and text objects

---

window object (client)

A window object is the top-level object for each document, location, and history object group.

Syntax

To define a window, use the open method :

```
windowVar = window.open("URL", "windowName", ["windowFeatures"])
```

windowVar is the name of a new window. Use this variable when referring to a window's properties, methods, and containership.
windowName is the window name to use in the TARGET argument of a <FORM> or <A> tag.
xxx and location object? For details on defining a window, see the open method.

To use a window's properties and methods:

```
1. window.propertyName
2. window.methodName
3. self.propertyName
4. self.methodName
5. windowVar.propertyName
6. windowVar.methodName
```

windowVar is a variable referring to a window object. See the preceding syntax for defining a window.
propertyName is one of the properties listed below.
methodName is one of the methods listed below.

To define an onLoad or onUnload event handler for a window object, use the <BODY> or <FRAMESET> tags:

```
<BODY
   ...
   [onLoad="<I>handlerText</I>"]
   [onUnload="<I>handlerText</I>"]>
</BODY>

<FRAMESET
   ROWS="<I>rowHeightValueList</I>"
   COLS="<I>columnWidthList</I>"
   [onLoad="<I>handlerText</I>"]
   [onUnload="<I>handlerText</I>"]>
   [<FRAME SRC="locationOrURL" NAME="frameName">]
</FRAMESET>
</PRE>
```

For information on the <BODY> and <FRAMESET> tags, see the document and frame objects.

Description

The window object is the top-level object in the JavaScript client hierarchy. The top window is a "document window" or "Web Browser window". Frame objects are also windows.

Because the existence of the current window is assumed, you do not have to reference 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.

The self and window properties 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 either `window.close()` or `self.close()`. 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.

You can reference a window's frame objects in your code by using the frames property. The frames property is an array that contains an entry for each frame in a window containing a <FRAMESET> tag.

Windows lack event handlers until some HTML is loaded into them containing a <BODY> or <FRAMESET> tag.

Properties

- defaultStatus
- frames
- parent
- self
- status
- top
- window

Methods

- alert

- close
- confirm
- open
- prompt
- setTimeout
- clearTimeout

Event handlers

- onLoad
- onUnload

Examples

xxx to be supplied

See also

- document and frame objects
- frames property

Methods and Functions

The following methods and functions are available in JavaScript:

- abs
- acos
- alert
- anchor
- asin
- atan
- back
- big
- blink
- blur
- bold
- ceil
- charAt
- clear
- clearTimeout
- click
- close (document)
- close
- confirm
- cos
- escape
- eval
- exp
- fixed
- floor
- focus
- fontcolor
- fontsize
- forward
- getDate
- getDay
- getHours
- getMinutes
- getMonth
- getSeconds
- getTime
- getTimeZoneoffset
- getYear
- go
- indexOf
- italics
- lastIndexOf
- link
- log
- max
- min
- open (document)
- open (window)
- parse
- parseFloat
- isNaN
- parseInt
- pow
- prompt
- random
- round
- select
- setDate
- setHours
- setMinutes
- setMonth
- setSeconds
- setTimeout
- setTime
- setYear
- sin
- small
- sqrt
- strike
- sub
- submit
- substring
- sup
- tan
- toGMTString
- toLocaleString
- toLowerCase
- toString
- toUpperCase
- unEscape
- UTC
- write
- writeln

---

abs method

Returns the absolute value of a number.

Syntax

```
Math.abs(number)
```

number is any numeric expression or a property of an existing object.

Method of

Math

Examples

In the following example, the user enters a number in the first text box and presses the Calculate button to display the absolute value of the number.

```
<FORM>
<P>Enter a number:
<INPUT TYPE="text" NAME="absEntry">
<P>The absolute value is:
<INPUT TYPE="text" NAME="result">
<P>
<INPUT TYPE="button" VALUE="Calculate"
   onClick="form.result.value = Math.abs(form.absEntry.value)">
</FORM>
```

acos method

Returns the arc cosine (in radians) of a number.

Syntax

```
Math.acos(number)
```

number is a numeric expression between -1 and 1, or a property of an existing object.

Description

The acos method returns a numeric value between 0 and pi radians. If the value of *number* is outside the suggested range, the return value is always 0.

Method of

Math

Examples

```
// Displays the value 0
document.write("The arc cosine of 1 is " + Math.acos(1))

// Displays the value 3.141592653589793
document.write("<P>The arc cosine of -1 is " + Math.acos(-1))

// Displays the value 0
document.write("<P>The arc cosine of 2 is " + Math.acos(2))
```

See also

- asin, atan, cos, sin, tan methods

---

alert method

Displays an Alert dialog box with a message and an OK button.

Syntax

```
alert("message")
```

message is any string or a property of an existing object.

Description

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.

Although alert is a method of the window object, you do not need to specify a *windowReference* when you call it. For example, `windowReference.alert()` is unnecessary.

Method of

- window

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

- confirm, prompt methods

---

anchor method

Creates an HTML anchor that is used as a hypertext target.

Syntax

```
text.anchor(nameAttribute)
```

text is any string or a property of an existing object.
nameAttribute is any string or a property of an existing object.

Description

Use the anchor method with the write or writeln methods to programatically create and display an anchor in a document. Create the anchor with the anchor method, then call write or writeln to display the anchor in a document.

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 anchors array. See the anchor object for information about the anchors array.

Method of

string

Examples

The following example opens the msgWindow window and creates an anchor for the Table of Contents:

```
var myString="Table of Contents"

msgWindow=window.open("","displayWindow")
msgWindow.document.writeln(myString.anchor("contents_anchor"))
msgWindow.document.close()
```

The previous example produces the same output as the following HTML:

```
<A NAME="contents_anchor">Table of Contents</A>
```

See also

- link method

---

asin method

Returns the arc sine (in radians) of a number.

Syntax

```
Math.asin(number)
```

number is a numeric expression between -1 and 1, or a property of an existing object.

Description

The asin method returns a numeric value between -pi/2 and pi/2 radians. If the value of *number* is outside the suggested range, the return value is always 0.

Method of

Math

Examples

```
// Displays the value 1.570796326794897 (pi/2)
document.write("The arc sine of 1 is " + Math.asin(1))

// Displays the value -1.570796326794897 (-pi/2)
```

```
document.write("<P>The arc sine of -1 is " + Math.asin(-1))

// Displays the value 0 because the argument is out of range
document.write("<P>The arc sine of 2 is " + Math.asin(2))
```

See also

- acos, atan, cos, sin, tan methods

---

atan method

Returns the arc tangent (in radians) of its argument.

Syntax

```
Math.atan(number)
```

number is a numeric expression representing the tangent of an angle. The atan method returns a numeric value between -pi/2 and pi/2 radians.

Method of

Math

Examples

```
// Displays the value 0.7853981633974483
document.write("The arc tangent of 1 is " + Math.atan(1))

// Displays the value -0.7853981633974483
document.write("<P>The arc tangent of -1 is " + Math.atan(-1))

// Displays the value 0.4636476090008061
document.write("<P>The arc tangent of .5 is " + Math.atan(.5))
```

See also

- acos, asin, cos, sin, tan methods

---

back method

Loads the previous URL in the history list.

Syntax

```
history.back()
```

Description

This method performs the same action as a user choosing the Back button in the Navigator. The back method is

the same as history.go(-1).

Applies to

history

Examples

The following custom buttons perform the same operations as the Navigator Back and Forward buttons:

```
<P><INPUT TYPE="button" VALUE="< Back" onClick="history.back()">
<P><INPUT TYPE="button" VALUE="> Forward" onClick="history.forward()">
```

See also

- forward, go methods

---

big method

Causes a string to be displayed in a big font as if it were in a <BIG> tag.

Syntax

```
stringName.big()
```

stringName is any string or a property of an existing object.

Description

Use the big method with the write or writeln methods to format and display a string in a document.

Method of

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

- fontsize, small methods

---

blink method

Causes a string to blink as if it were in a <BLINK> tag.

Syntax

```
stringName.blink()
```

stringName is any string or a property of an existing object.

Description

Use the blink method with the write or writeln methods to format and display a string in a document.

Method of

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

- bold, italics, strike methods

---

blur method

Removes focus from the specified object.

Syntax

```
1. passwordName.blur()
2. selectName.blur()
3. textName.blur()
4. textareaName.blur()
```

passwordName is either the value of the NAME attribute of a password object or an element in the *elements* array.
selectName is either the value of the NAME attribute of a select object or an element in the *elements* array.
textName is either the value of the NAME attribute of a text object or an element in the *elements* array.
textareaName is either the value of the NAME attribute of a textarea object or an element in the *elements* array.

Description

Use the blur method to remove focus from a specific form element.

Method of

password, text, textarea

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

- focus, select methods

---

bold method

Causes a string to be displayed as bold as if it were in a <B> tag.

Syntax

```
stringName.bold()
```

stringName is any string or a property of an existing object.

Description

Use the bold method with the write or writeln methods to format and display a string in a document.

Method of

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

- blink, italics, strike methods

---

ceil method

Returns the least integer greater than or equal to a number.

Syntax

```
Math.ceil(number)
```

number is any numeric expression or a property of an existing object.

Method of

Math

Examples

```
//Displays the value 46
document.write("The ceil of 45.95 is " + Math.ceil(45.95))

//Displays the value -45
document.write("<P>The ceil of -45.95 is " + Math.ceil(-45.95))
```

See also

- floor method

---

charAt method

Returns the character at the specified index.

Syntax

```
stringName.charAt(index)
```

stringName is any string or a property of an existing object.
index is any integer from 0 to *stringName*.length - 1, or a property of an existing object.

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. If the *index* you supply is out of range, JavaScript returns an empty string.

Method of

string

Examples

The following example displays characters at different locations in the string "Brave new world".

```
var anyString="Brave new world"

document.write("The character at index 0 is " + anyString.charAt(0))
document.write("The character at index 1 is " + anyString.charAt(1))
document.write("The character at index 2 is " + anyString.charAt(2))
document.write("The character at index 3 is " + anyString.charAt(3))
document.write("The character at index 4 is " + anyString.charAt(4))
```

See also

- indexOf, lastIndexOf methods

---

clear method

Clears the document in a window.

Syntax

```
document.clear()
```

Description

The clear method empties the content of a window, regardless of how the content of the window has been painted.

Method of

document

Examples

When the following function is called, the clear method empties the contents of the *msgWindow* window:

```
function windowCleaner() {
   msgWindow.document.clear()
   msgWindow.document.close()
}
```

See also

- close, open, write, writeln methods

---

clearTimeout method

Cancels a timeout that was set with the setTimeout method.

Syntax

```
clearTimeout(timeoutID)
```

timeoutID is a timeout setting that was returned by a previous call to the setTimeout method.

Description

See the description for the setTimeout method.

Method of

frame, window

Examples

See the examples for the setTimeout method.

See also

- setTimeout method

---

click method

Simulates a mouse click on the calling form element.

Syntax

```
1. buttonName.click()
2. radioName[index].click()
3. checkboxName.click()
```

buttonName is either the value of the NAME attribute of a button, reset, or submit object or an element in the *elements* array.
radioName is the value of the NAME attribute of a radio object or an element in the *elements* array.
index is an integer representing a radio button in a radio object.
checkboxName is either the value of the NAME attribute of a checkbox object or an element in the *elements* array.

Description

The effect of the click method varies according to the calling element:

- For button, reset, and submit, performs the same action as clicking the button.
- For a radio, selects a radio button.
- For a checkbox, checks the check box and sets its value to on.

Method of

button, checkbox, radio, reset, submit

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()
```

---

close method (document object)

Closes an output stream and forces data sent to layout to display.

Syntax

```
document.close()
```

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.

Applies to

document

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("<P>" + myString)
   msgWindow.document.close()
}
```

See also

- clear, open, write, writeln methods

---

close method (window object)

Closes the specified window.

Syntax

```
windowReference.close()
```

windowReference is a valid way of referring to a window, as described in the window object.

Description

The close method closes the specified window. If you call close without specifying a *windowReference*, JavaScript closes the current window.

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().

Method of

window

Examples

Any of the following examples close the current window:

```
window.close()
self.close()
close()
```

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

- open method

---

confirm method

Displays a Confirm dialog box with the specified message and OK and Cancel buttons.

Syntax

```
confirm("message")
```

message is any string or a property of an existing object.

Description

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.

Although confirm is a method of the window object, you do not need to specify a *windowReference* when you call it. For example, *windowReference*`.confirm()` is unnecessary.

Method of

window

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 pushbutton, as shown in the following example:

```
<INPUT TYPE="button" VALUE="Quit" onClick="confirmCleanUp()">
```

See also

- alert, prompt methods

---

cos method

Returns the cosine of a number.

Syntax

```
Math.cos(number)
```

number is a numeric expression representing the size of an angle in radians, or a property of an existing object.

Description

The cos method returns a numeric value between -1 and 1, which represents the cosine of the angle.

Method of

Math

Examples

```
 //Displays the value 6.123031769111886e-017
document.write("The cosine of PI/2 radians is " + Math.cos(Math.PI/2))

//Displays the value -1
document.write("<P>The cosine of PI radians is " + Math.cos(Math.PI))

//Displays the value 1
document.write("<P>The cosine of 0 radians is " + Math.cos(0))
```

See also

- acos, asin, atan, sin, tan methods

---

escape function

Returns the ASCII encoding of an argument in the ISO Latin-1 character set.

Syntax

```
escape(string)
```

string is a non-alphanumeric string in the ISO Latin-1 character set, or a property of an existing object.

Description

The escape function is not a method associated with any object, but is part of the language itself.

The value returned by the escape function is a string of the form "%xx", where *xx* is the ASCII encoding of a character in the argument. If you pass the escape function an alphanumeric character, the escape function returns the same character.

Examples

The following example returns "!#"

```
escape("!#")
```

See also

- unEscape function

---

eval function

The eval function evaluates a string and returns a value.

Syntax

```
eval(string)
```

string is any string representing a JavaScript expression, statement, or sequence of statements. The expression can include variables and properties of existing objects.

Description

The eval function is a built-in JavaScript function. It is not a method associated with any object, but is part of the language itself.

The argument of the eval function is a string. Do not call eval to evaluate an arithmetic expression. JavaScript evaluates arithmetic expressions automatically. If the argument represents an expression, eval evaluates the expression.

If the argument represents one or more JavaScript statements, eval performs the statements.

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.

Example

Both of the write statements below display 42. The first evaluates the string "x + y + 1", and the second evaluates the string "42".

```
var x = 2
var y = 39
var z = "42"
document.write(eval("x + y + 1"), "
" )
document.write(eval(z), "
" )
```

In the following example, the getFieldName(n) function returns the name of the nth 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"))
```

The following example uses eval to evaluate the string str. This string consists of JavaScript statements that opens an alert dialog box and assigns z a value of 42 if x is five, and assigns zero 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("
z is ", eval(str))
```

---

exp method

Returns e^{number}, where *number* is the argument, and *e* is Euler's constant, the base of the natural logarithms.

Syntax

```
Math.exp(number)
```

number is any numeric expression or a property of an existing object.

Method of

Math

Examples

```
//Displays the value 2.718281828459045
document.write("The value of e<SUP>1</SUP> is " + Math.exp(1))
```

See also

- log, pow methods

---

fixed method

Causes a string to be displayed in fixed-pitch font as if it were in a <TT> tag.

Syntax

```
stringName.fixed()
```

stringName is any string or a property of an existing object.

Description

Use the fixed method with the write or writeln methods to format and display a string in a document.

Method of

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>
```

---

floor method

Returns the greatest integer less than or equal to a number.

Syntax

```
Math.floor(number)
```

number is any numeric expression or a property of an existing object.

Method of

Math

Examples

```
//Displays the value 45
document.write("<P>The floor of 45.95 is " + Math.floor(45.95))

//Displays the value -46
document.write("<P>The floor of -45.95 is " + Math.floor(-45.95))
```

See also

- ceil method

---

focus method

Gives focus to the specified object.

Syntax

```
1. passwordName.focus()
2. selectName.focus()
3. textName.focus()
4. textareaName.focus()
```

passwordName is either the value of the NAME attribute of a password object or an element in the *elements* array.
selectName is either the value of the NAME attribute of a select object or an element in the *elements* array.
textName is either the value of the NAME attribute of a text object or an element in the elements array.
textareaName is either the value of the NAME attribute of a textarea object or an element in the *elements* array.

Description

Use the focus method to navigate to a specific form element and give it focus. You can then either programatically enter a value in the element or let the user enter a value.

Method of

password, select, text, textarea

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 re-enter 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

- blur, select methods

---

fontcolor method

Causes a string to be displayed in the specified color as if it were in a <FONT COLOR=*color*> tag.

Syntax

```
stringName.fontcolor(color)
```

stringName is any string or a property of an existing object.
color is a string or a property of an existing object, expressing the color as a hexadecimal RGB triplet or as one of the string literals listed in Color Values.

Description

Use the fontcolor method with the write or writeln methods to format and display a string in a document.

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.

Method of

string

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 method

Causes a string to be displayed in the specified font size as if it were in a <FONTSIZE=*size*> tag.

Syntax

```
stringName.fontsize(size)
```

stringName is any string or a property of an existing object.
size is an integer between one and seven, or a string representing a signed integer between 1 and 7, or a property of an existing object.

Description

Use the fontsize method with the write or writeln methods to format and display a string in a document. When you specify *size* as an integer, you set the size of *stringName* to one of the seven 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.

Method of

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

- big, small methods

---

forward method

Loads the next URL in the history list.

Syntax

```
history.forward()
```

Description

This method performs the same action as a user choosing the Forward button in the Navigator. The forward method is the same as `history.go(1)`.

Method of

history

Examples

The following custom buttons perform the same operations as the Navigator Back and Forward buttons:

```
<P><INPUT TYPE="button" VALUE="< Back" onClick="history.back()">
<P><INPUT TYPE="button" VALUE="> Forward" onClick="history.forward()">
```

See also

- back, go methods

---

getDate method

Returns the day of the month for the specified date.

Syntax

```
dateObjectName.getDate()
```

dateObjectName is either the name of a date object or a property of an existing object.

Description

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

Method of

Date

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

- setDate method

---

getDay method

Returns the day of the week for the specified date.

Syntax

```
dateObjectName.getDay()
```

dateObjectName is either the name of a date object or a property of an existing object.

Description

The value returned by getDay is an integer corresponding to the day of the week: zero for Sunday, one for Monday, two for Tuesday, and so on.

Method of

Date

Examples

The second statement below assigns the value 1 to *weekday*, based on the value of the date object *Xmas95*. This

is because December 25, 1995 is a Monday.

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

---

getHours method

Returns the hour for the specified date.

Syntax

```
dateObjectName.getHours()
```

dateObjectName is either the name of a date object or a property of an existing object.

Description

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

Method of

Date

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

- setHours method

---

getMinutes method

Returns the minutes in the specified date.

Syntax

```
dateObjectName.getMinutes()
```

dateObjectName is either the name of a date object or a property of an existing object.

Description

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

Method of

Date

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

- setMinutes method

---

getMonth method

Returns the month in the specified date.

Syntax

```
dateObjectName.getMonth()
```

dateObjectName is either the name of a date object or a property of an existing object.

Description

The value returned by getMonth is an integer between zero and eleven. Zero corresponds to January, one to February, and so on.

Applies to

Date

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.getDate()
```

See also

- setMonth method

getSeconds method

Returns the seconds in the current time.

Syntax

```
dateObjectName.getSeconds()
```

dateObjectName is either the name of a date object or a property of an existing object.

Description

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

Method of

Date

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

- setSeconds method

getTime method

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

Syntax

```
dateObjectName.getTime()
```

dateObjectName is either the name of a date object or a property of an existing object.

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.

Method of

Date

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

- setTime method

---

getTimezoneOffset method

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

Syntax

```
dateObjectName.getTimezoneOffset()
```

dateObjectName is either the name of a date object or a property of an existing object.

Description

The time zone offset is the difference between local time and GMT. Daylight savings time prevents this value from being a constant.

Method of

Date

Examples

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

---

getYear method

Returns the year in the specified date.

Syntax

```
dateObjectName.getYear()
```

dateObjectName is either the name of a date object or a property of an existing object.

Description

The value returned by getYear is the year less 1900. For example, if the year is 1976, the value returned is 76.

Method of

Date

Examples

The second statement below assigns the value 95 to the variable *year*, based on the value of the date object *Xmas95*.

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

See also

- setYear method

---

go method

Loads a URL from the history list.

Syntax

```
history.go(delta | "location")
```

delta is an integer or a property of an existing object, representing a relative position in the history list.
location is a string or a property of an existing object, 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 argument that you specify. You can interactively display the history list by choosing History from the Window menu. Up to 10 items in the history list are also displayed on the Go menu.

The *delta* argument is a positive or negative integer. If *delta* is greater than zero, the go method loads the URL that is that number of entries forward in the history list; otherwise, it loads the URL that is that number of entries backward in the history list. If *delta* is zero, Navigator reloads the current page.

The *location* argument is a string. Use location to load the nearest history entry whose URL contains *location* as a substring. The location to URL matching is case-insensitive. Each section of a URL contains different information. See the location object for a description of the URL components.

Method of

history

Examples

The following button navigates to the nearest history entry that contains the string "home.netscape.com":

```
<P><INPUT TYPE="button" VALUE="Go" onClick="history.go('home.netscape.com')">
```

The following button navigates to the URL that is three entries backward in the history list:

```
<P><INPUT TYPE="button" VALUE="Go" onClick="history.go(-3)">
```

See also

- back, forward methods

---

indexOf method

Returns the index within the calling string object of the first occurrence of the specified value, starting the search at *fromIndex*.

Syntax

```
stringName.indexOf(searchValue, [fromIndex])
```

stringName is any string or a property of an existing object.
searchValue is a string or a property of an existing object, representing the value to search for.
fromIndex is the location within the calling string to start the search from. It can be any integer from 0 to *stringName*.length - 1 or a property of an existing object.

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.

If you do not specify a value for *fromIndex*, JavaScript assumes 0 by default.

Method of

string

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"))
```

See also

- charAt, lastIndexOf methods

---

isNaN function

On Unix platforms, evaluates an argument to determine if it is "NaN" (not a number).

Syntax

```
isNaN(testValue)
```

testValue is the value you want to evaluate.

Description

The isNaN function is a built-in JavaScript function. It is not a method associated with any object, but is part of the language itself. isNaN is available on Unix platforms only.

On all platforms except Windows, the parseFloat and parseInt functions return "NaN" when they evaluate a value that is not a number. The "NaN" value is not a number in any radix. You can call the isNaN function to determine if the result of parseFloat or parseInt is "NaN". If "NaN" is passed on to arithmetic operations, the operation results will also be "NaN".

The isNaN function returns true or false.

Examples

The following example evaluates *floatValue* to determine if it is a number, then calls a procedure accordingly.

```
floatValue=parseFloat(toFloat)
```

```
if isNaN(floatValue) {
   notFloat()
} else {
   isFloat()
}
```

See also

- parseFloat and parseInt functions

---

italics method

Causes a string to be italicized as if it were in an <I> tag.

Syntax

```
stringName.italics()
```

stringName is any string or a property of an existing object.

Description

Use the italics method with the write or writeln methods to format and display a string in a document.

Method of

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

- blink, bold, strike methods

---

lastIndexOf method

Returns the index within the calling string object of the last occurrence of the specified value. The calling string is searched backwards, starting at *fromIndex*.

Syntax

```
stringName.lastIndexOf(searchValue, [fromIndex])
```

stringName is any string or a property of an existing object.
searchValue is a string or a property of an existing object, representing the value to search for.
fromIndex is the location within the calling string to start the search from. It can be any integer from 0 to *stringName*.length - 1 or a property of an existing object.

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.

If you do not specify a value for *fromIndex*, JavaScript assumes *stringName*.length() - 1 (the end of the string) by default. If a *searchValue* is not found, JavaScript returns 0.

Method of

string

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"))
```

See also

- charAt, indexOf methods

link method

Creates an HTML hypertext link that jumps to another URL.

Syntax

```
linkText.link(hrefAttribute)
```

linkText is any string or a property of an existing object.
hrefAttribute is any string or a property of an existing object.

Description

Use the link method with the write or writeln methods to programatically create and display a hypertext link in a document. Create the link with the link method, then call write or writeln to display the link in a document.

In the syntax, the *linkName* string represents the literal text that you want the user to see. The *hrefAttribute* string represents the HREF attribute of the HTML A tag, and it should be a valid URL. Each section of a URL contains different information. See the location object for a description of the URL components.

Links created with the link method become elements in the links array. See the link object for information about the links array.

Method of

string

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://www.netscape.com"

document.open()
document.write("Click to return to " + hotText.link(URL))
document.close()
```

The previous example produces the same output as the following HTML:

```
Click to return to <A HREF="http://www.netscape.com">Netscape</A>
```

See also

- anchor method

log method

Returns the natural logarithm (base *e*) of a number.

Syntax

```
Math.log(number)
```

number is any positive numeric expression or a property of an existing object.

Description

If the value of *number* is outside the suggested range, the return value is always -1.797693134862316e+308.

Method of

Math

Examples

```
//Displays the value 2.302585092994046
document.write("The natural log of 10 is " + Math.log(10))

//Displays the value 0
document.write("<P>The natural log of 1 is " + Math.log(1))

//Displays the value -1.797693134862316e+308 because the argument is out of range
document.write("<P>The natural log of 0 is " + Math.log(0))
```

See also

- exp, pow methods

---

max method

Returns the greater of two numbers.

Syntax

```
max(number1, number2)
```

number1 and *number2* are any numeric arguments or the properties of existing objects.

Method of

Math

Examples

```
//Displays the value 20
document.write("The maximum value is " + Math.max(10,20))

//Displays the value -10
document.write("<P>The maximum value is " + Math.max(-10,-20))
```

See also

- min method

---

min method

Returns the lesser of two numbers.

Syntax

```
min(number1, number2)
```

number1 and *number2* are any numeric arguments or the properties of existing objects.

Method of

Math

Examples

```
//Displays the value 10
document.write("
The minimum value is " + Math.min(10,20))

//Displays the value -20
document.write("<P>The minimum value is " + Math.min(-10,-20))
```

See also

- max method

---

open method (document object)

Opens a stream to collect the output of write or writeln methods.

Syntax

```
document.open(["mimeType"])
```

mimeType specifies any of the following document types:

```
text/html
```

```
text/plain
image/gif
image/jpeg
image/x-bitmap
plugIn
```

plugIn is any two-part plug-in MIME type that Netscape supports.

Description

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(), issue document.open() again when you want to begin another output stream.

mimeType is an optional argument that specifies the type of document to which you are writing. If you do not specify a *mimeType*, the open method assumes `text/html` by default.

Following is a description of *mimeType*:

- 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 XBM 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.

Method of

document

Examples

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()
}
```

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

- clear, close, write, writeln methods

---

open method (window object)

Opens a new web browser window.

Syntax

```
[windowVar = ][window].open("URL", "windowName", ["windowFeatures"])
```

windowVar is the name of a new window. Use this variable when referring to a window's properties, methods, and containership.
URL specifies the URL to open in the new window. See the location object for a description of the URL components.
windowName is the window name to use in the TARGET attribute of a <FORM>or <A> tag. *windowName* can contain only alphanumeric or underscore (_) characters.
windowFeatures is a comma-separated list of any of the following options and values:

```
toolbar[=yes|no]|[=1|0]
location[=yes|no]|[=1|0]
directories[=yes|no]|[=1|0]
status[=yes|no]|[=1|0]
menubar[=yes|no]|[=1|0]
scrollbars[=yes|no]|[=1|0]
resizable[=yes|no]|[=1|0]
width=pixels
height=pixels
```

You may use any subset of these options. Separate options with a comma. Do not put spaces between the options.

pixels is a positive integer specifying the dimension in pixels.

Description

The open method opens a new web browser window on the client, similar to choosing File|New Web Browser

from the menu of the Navigator. The *URL* argument specifies the *URL* contained by the new window. If *URL* is an empty string, a new, empty window is created.

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().

windowFeatures is an optional, comma-separated list of options for the new window. The boolean *windowFeatures* options are set to true if they are specified without values, or as `yes` or `1`. For example, `open("", "messageWindow", "toolbar")` and `open("", "messageWindow", "toolbar=1")` both set the toolbar option to true. If *windowName* does not specify an existing window and you do not specify *windowFeatures*, all boolean *windowFeatures* are true by default. If you specify any item in *windowFeatures*, all other Boolean *windowFeatures* are false unless you explicitly specify them.

Following is a description of the *windowFeatures*:

- *toolbar* creates the standard Navigator toolbar, with buttons such as "Back" and "Forward", if true
- *location* creates a Location entry field, if true
- *directories* creates the standard Navigator directory buttons, such as "What's New" and "What's Cool", if true
- *status* creates the status bar at the bottom of the window, if true
- *menubar* creates the menu at the top of the window, if true
- *scrollbars* creates horizontal and vertical scrollbars when the document grows larger than the window dimensions, if true
- *resizable* allows a user to resize the window, if true
- *copyhistory* gives the new window the same session history as the current window, if true
- *width* specifies the width of the window in pixels
- *height* specifies the height of the window in pixels

Method of

window

Examples

In the following example, the windowOpener function opens a window and uses write methods to display a message:

```
function windowOpener() {
   msgWindow=window.open("","Display
   window","toolbar=no,directories=no,menubar=no")
   msgWindow.document.write("<HEAD><TITLE>Message window</TITLE></HEAD>")
   msgWindow.document.write("<CENTER><BIG><B>Hello, world!</B></BIG></CENTER>")
}
```

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>
```

Notice the use of single quotes (') inside the onClick event handler.

See also

- close method

---

parse method

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

Syntax

```
Date.parse(dateString)
```

dateString is a string representing a date or the property of an existing object.

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

Method of

Date

Examples

If *IPOdate* is an existing date object, then

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

See also

- UTC method

parseFloat function

Parses a string argument and returns a floating point number.

Syntax

```
parseFloat(string)
```

string is a string that represents the value you want to parse.

Description

The parseFloat function is a built-in JavaScript function. It is not a method associated with any object, but is part of the language itself.

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 one of the following values:

- 0 on Windows platforms.
- "NaN" on any other platform, indicating that the value is not a number.

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” or 0:

```
parseFloat("FF2")
```

See also

- isNaN mthod and parseInt function

parseInt function

Parses a string argument and returns an integer of the specified radix or base.

Syntax

```
parseInt(string, [radix])
```

string is a string that represents the value you want to parse.
radix is an integer that represents the radix of the return value.

Description

The parseInt function is a built-in JavaScript function. It is not a method associated with any object, but is part of the language itself.

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 8 (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, parseFloat returns one of the following values:

- 0 on Windows platforms.
- "NaN" on any other platform, indicating that the value is not a number.

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” or zero:

```
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 method or parseFloat function

---

pow method

Returns *base* to the *exponent* power, that is, $base^{exponent}$.

Syntax

```
pow(base, exponent)
```

base is any numeric expression or a property of an existing object.
exponent is any numeric expression or a property of an existing object.

Method of

Math

Examples

```
//Displays the value 49
document.write("7 to the power of 2 is " + Math.pow(7,2))

//Displays the value 1024
document.write("<P>2 to the power of 10 is " + Math.pow(2,10))
```

See also

- exp, log methods

---

prompt method

Displays a Prompt dialog box with a message and an input field.

Syntax

```
prompt(message, [inputDefault])
```
message is any string or a property of an existing object; the string is displayed as the message.
inputDefault is a string, integer, or property of an existing object that represents the default value of the input field.

Description

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 the value <undefined>.

Although prompt is a method of the window object, you do not need to specify a *windowReference* when you call it. For example, *`windowReference`*`.prompt()` is unnecessary.

Method of

window

Examples

```
prompt("Enter the number of cookies you want to order:", 12)
```

See also

- alert, confirm methods

---

random method

Returns a pseudo-random number between zero and one. This method is available on UNIX platforms only.

Syntax

```
Math.random()
```

Method of

Math

Examples

```
//Displays a random number between 0 and 1
document.write("The random number is " + Math.random())
```

---

round method

Returns the value of a number rounded to the nearest integer.

Syntax

```
round(number)
```

number is any numeric expression or a property of an existing object.

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.

Method of

Math

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))
```

---

select method

Selects the input area of the specified password, text, or textarea object.

Syntax

```
1. passwordName.select()
2. textName.select()
3. textareaName.select()
```

passwordName is either the value of the NAME attribute of a password object or an element in the *elements* array.
textName is either the value of the NAME attribute of a text object or an element in the *elements* array.
textareaName is either the value of the NAME attribute of a textarea object or an element in the *elements* array.

Description

Use the select method to highlight the input area of a form element. You can use the select method with the focus method to highlight a field and position the cursor for a user response.

Method of

password, text, textarea

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 re-enter 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

- blur, focus methods

---

setDate method

Sets the day of the month for a specified date.

Syntax

```
dateObjectName.setDate(dayValue)
```

dateObjectName is either the name of a date object or a property of an existing object.
dayValue is an integer from 1 to 31 or a property of an existing object, representing the day of the month.

Method of

Date

Examples

The second statement below changes the day for *theBigDay* to the 24th of July from its original value.

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

See also

- getDate method

---

setHours method

Sets the hours for a specified date.

Syntax

```
dateObjectName.setHours(hoursValue)
```

dateObjectName is either the name of a date object or a property of an existing object.
hoursValue is an integer between 0 and 23, or a property of an existing object representing the hour.

Method of

Date

Examples

```
theBigDay.setHours(7)
```

See also

- getHours method

---

setMinutes method

Sets the minutes for a specified date.

Syntax

```
dateObjectName.setMinutes(minutesValue)
```

dateObjectName is either the name of a date object or a property of an existing object.
minutesValue is an integer between 0 and 59, or a property of an existing object representing the minutes.

Method of

Date

Examples

```
theBigDay.setMinutes(45)
```

See also

- getMinutes method

setMonth method

Sets the month in the current date.

Syntax

```
dateObjectName.setMonth(monthValue)
```

dateObjectName is either the name of a date object or a property of an existing object.
monthValue is an integer between 0 and 11 representing the month.

Method of

Date

Examples

```
theBigDay.setMonth(6)
```

See also

- getMonth method

setSeconds method

Sets the seconds for a specified date.

Syntax

```
dateObjectName.setSeconds(secondsValue)
```

dateObjectName is either the name of a date object or a property of an existing object.
secondsValue is an integer between 0 and 59 or a property of an existing object.

Method of

Date

Examples

```
theBigDay.setSeconds(30)
```

See also

- getSeconds method

setTime method

Sets the value of a specified date.

Syntax

```
dateObjectName.setTime(timevalue)
```

dateObjectName is either the name of a date object or a property of an existing object.
timevalue is an integer or a property of an existing object representing the number of milliseconds since January 1, 1970 00:00:00.

Description

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

Method of

Date

Examples

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

See also

- getTime method

setTimeout method

Evaluates an expression after a specified number of milliseconds have elapsed.

Syntax

```
timeoutID=setTimeout(expression, msec)
```

timeoutID is an identifier that is used only to cancel the evaluation with the clearTimeout method.
expression is a string expression or a property of an existing object.
msec is a numeric value, numeric string, or a property of an existing object in millisecond units.

Description

The setTimeout method evaluates an expression after a specified amount of time. It does not evaluate the expres-

sion repeatedly. For example, if a setTimeout method specifies 5 seconds, the expression is evaluated after 5 seconds, not every 5 seconds.

Method of

frame, window

Examples

Example 1. The following example displays an alert message 5 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 cancelled 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 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

- clearTimeout method

---

setYear method

Sets the year for a specified date.

Syntax

```
dateObjectName.setYear(yearValue)
```

dateObjectName is either the name of a date object or a property of an existing object.
yearValue is an integer greater than 1900 or a property of an existing object.

Method of

Date

Examples

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

See also

- getYear method

---

sin method

Returns the sine of a number.

Syntax

```
Math.sin(number)
```

number is a numeric expression, or a property of an existing object representing the size of an angle in radians. The sin method returns a numeric value between -1 and 1, which represents the sine of the angle.

Description

The sin method returns a numeric value between -1 and 1, which represents the sine of the angle.

Method of

Math

Examples

```
//Displays the value 1
document.write("The sine of pi/2 radians is " + Math.sin(Math.PI/2))

//Displays the value 1.224606353822377e-016
document.write("<P>The sine of pi radians is " + Math.sin(Math.PI))

//Displays the value 0
document.write("<P>The sine of 0 radians is " + Math.sin(0))
```

See also

- acos, asin, atan, cos, tan methods

---

small method

Causes a string to be displayed in a small font as if it were in a <SMALL> tag.

Syntax

```
stringName.small()
```

stringName is any string or a property of an existing object.

Description

Use the small method with the write or writeln methods to format and display a string in a document.

Method of

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

- big, fontsize methods

---

sqrt method

Returns the square root of a number.

Syntax

```
Math.sqrt(number)
```

number is any non-negative numeric expression or a property of an existing expression.

Description

If the value of *number* is outside the suggested range, the return value is always 0.

Method of

Math

Examples

```
//Displays the value 3
document.write("The square root of 9 is " + Math.sqrt(9))

//Displays the value 1.414213562373095
document.write("<P>The square root of 2 is " + Math.sqrt(2))

//Displays the value 0 because the argument is out of range
document.write("<P>The square root of -1 is " + Math.sqrt(-1))
```

---

strike method

Causes a string to be displayed as struck out text as if it were in a <STRIKE> tag.

Syntax

```
stringName.strike()
```

stringName is any string or a property of an existing object.

Description

Use the strike method with the write or writeln methods to format and display a string in a document.

Method of

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

- blink, bold, italics methods

---

sub method

Causes a string to be displayed as a subscript as if it were in a <SUB> tag.

Syntax

```
stringName.sub()
```

stringName is any string or a property of an existing object.

Description

Use the sub method with the write or writeln methods to format and display a string in a document.

Method of

string

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
```

See also

- sup method

---

submit method

Submits a form.

Syntax

```
formName.submit()
```

formName is the name of any form or an element in the forms array.

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 the method property.

Method of

form

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 the form object.

See also

- submit object, onSubmit event handler

---

substring method

Returns a subset of a string object.

Syntax

```
stringName.substring(indexA, indexB)
```

stringName is any string or a property of an existing object.
indexA is any integer from 0 to stringName.length() - 1 or a property of an existing object.
indexB is any integer from 0 to stringName.length() - 1 or a property of an existing object.

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.

If *indexA* is less than *indexB*, the substring method returns the subset starting with the character at *indexA* and ending with the character before *indexB*. If *indexA* is greater than *indexB*, the substring method returns the subset starting with the character at indexB and ending with the character before *indexA*. If *indexA* is equal to *indexB*, the substring method returns the empty string.

Method of

string

Examples

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))
```

---

sup method

Causes a string to be displayed as a superscript as if it were in a <SUP> tag.

Syntax

```
stringName.sup()
```

stringName is the name of any string variable.

Description

Use the sup method with the write or writeln methods to format and display a string in a document.

Method of

string

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

- sub method

---

tan method

Returns the tangent of a number.

Syntax

```
Math.tan(number)
```

number is a numeric expression representing the size of an angle in radians or a property of an existing object.

Description

The tan method returns a numeric value which represents the tangent of the angle.

Method of

Math

Examples

```
//Displays the value 0.9999999999999999
document.write("The tangent of pi/4 radians is " + Math.tan(Math.PI/4))

//Displays the value 0
document.write("<P>The tangent of 0 radians is " + Math.tan(0))
```

See also

- acos, asin, atan, cos, sin methods

---

toGMTString method

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

Syntax

```
dateObjectName.toGMTString()
```

dateObjectName is either the name of a date object or a property of an existing object.

Description

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

Method of

Date

Examples

In the following example, today is a date object:

```
today.toGMTString()
```

In this example, toGMTString 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

- toLocaleString method

---

toLocaleString method

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

Syntax

```
dateObjectName.toLocaleString()
```

dateObjectName is either the name of a date object or a property of an existing object.

Description

If you are trying to pass a date using toLocaleString, be aware that different locales assemble the string in different ways. Using methods such as getHours, getMinutes, and getSeconds will give more portable results.

Method of

Date

Examples

In the following example, today is a date object:

```
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

- toGMTString method

---

toLowerCase method

Returns the calling string value converted to lower case.

Syntax

```
stringName.toLowerCase()
```

stringName is any string or a property of an existing object.

Description

The toLowerCase method returns the value of *stringName* converted to lower case. toLowerCase does not affect the value of *stringName* itself.

Method of

string

Examples

The following examples both yield "alphabet".

```
var upperText="ALPHABET"
document.write(upperText.toLowerCase())

"ALPHABET".toLowerCase()
```

See also

- toUpperCase method

---

toString method

Returns the calling string value converted to upper case.

Syntax

```
1. dateObjectName.getDate()
```

dateObjectName is the name of a date object.

```
2. location.toString()
```

Description

The value returned by the method location.toString() is the same as the value of the property location.href.

Method of

Date, location objects

Examples

The following example converts the Date object *theBigDay* to a string:

```
theBigDay.toString()
```

The following example displays the value of the current location:

```
document.write("The value of location.toString() is "+ location.toString())
```

---

toUpperCase method

Returns the calling string value converted to upper case.

Syntax

```
stringName.toUpperCase()
```

stringName is any string or a property of an existing object.

Description

The toUpperCase method returns the value of *stringName* converted to upper case. toUpperCase does not affect the value of *stringName* itself.

Method of

string

Examples

The following examples both yield "ALPHABET".

```
var lowerText="alphabet"
document.write(lowerText.toUpperCase())

"alphabet".toUpperCase()
```

See also

- toLowerCase method

---

unEscape function

Returns the ASCII character for the specified value.

Syntax

```
unescape("string")
```

string is a string or a property of an existing object, containing characters in either of the following forms:

- "%*integer*", where *integer* is a number between 0 and 255 (decimal)
- "*hex*", where hex is a number between 0x0 and 0xFF (hexadecimal)

Description

The unescape function is not a method associated with any object, but is part of the language itself.

The string returned by the unescape function is a series of characters in the ISO Latin-1 character set.

Examples

The following example returns "&"

```
unescape("%26")
```

The following example returns "!#"

```
unescape("%21%23")
```

See also

- escape function

UTC method

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

Syntax

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

year is a year after 1900.
month is a month between 0-11.
date is a day of the month between 1-31.
hrs is hours between 0-23.
min is minutes between 0-59.
sec is seconds between 0-59.

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.

Method of

Date

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

- parse method

---

write method

Writes one or more HTML expressions to a document in the specified window.

Syntax

```
write(expression1 [,expression2], ...[,expressionN])
```

expression1 through *expressionN* are any JavaScript expressions or the properties of existing objects.

Description

The write method displays any number of expressions in a document window. You can specify any JavaScript expression with the write method, including numerics, strings, or logicals.

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 will implicitly open a new document of *mimeType* `text/html` if you do not explicitly issue a document.open() method in the event handler.

Method of

document

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

- close, clear, open, writeln methods

---

writeln method

Writes one or more HTML expressions to a document in the specified window and follows them with a newline character.

Syntax

```
writeln(expression1 [,expression2], ...[,expressionN])
```

expression1 through *expressionN* are any JavaScript expressions or the properties of existing objects.

Description

The writeln method displays any number of expressions in a document window. You can specify any JavaScript expression, including numerics, strings, or logicals.

The writeln method is the same as the write method, except the writeln method appends a newline character to the end of the output.

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.

Methods of

document

Examples

All the examples used for the write method are also valid with the writeln method.

See also

- close, clear, open, write methods

Properties

The following properties are available in JavaScript:

- action
- alinkColor
- anchors
- appCodeName
- appName
- appVersion
- bgColor
- checked
- cookie
- defaultChecked
- defaultSelected
- defaultStatus
- defaultValue
- E
- encoding
- elements
- fgColor
- forms
- frames
- hash
- host
- hostname
- href
- index
- lastModified
- length
- linkColor
- links
- LN2
- LN10
- location
- method
- name
- options
- parent
- pathname
- PI
- port
- protocol
- referrer
- search
- selected
- selectedIndex
- self
- SQRT1_2
- SQRT2
- status
- target
- text
- title
- top
- userAgent
- value
- vlinkColor
- window

---

action property

A string specifying the URL of the server to which form field input information is sent.

Syntax

```
formName.action
```

formName is the name of any form or an element in the forms array.

Description

The action property is a reflection of the ACTION attribute of the HTML FORM tag. You can set this property before or after the HTML source has been through layout.

Applies to

form

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

- method, target properties

alinkColor property

The color of an active link (after mouse-button down, but before mouse-button up).

Syntax

```
document.alinkColor
```

Description

The alinkColor property is expressed as a hexadecimal RGB triplet or as one of the string literals listed in the Color Appendix. This property is the JavaScript reflection of the ALINK attribute of the HTML 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".

Applies to

document

Examples

The following example sets the color of active links to aqua 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

- bgColor, fgColor, linkColor, and vlinkColor properties

---

anchors property

xxx

Syntax

xxx

Description

Array of objects corresponding to named anchors (`<A NAME="">` tags) in source order.
The *anchors* array contains an entry for each anchor in a document. For example, if a document contains three

anchors, these anchors are reflected as `document.anchors[0], document.anchors[1],` and `document.anchors[2].`

To obtain the number of anchors in a document, use the length property: `document.anchors.length`.

Applies to

document

Examples

xxx Examples to be supplied.

See also

- links, length properties

---

appCodeName property

A string specifying the code name of the browser.

Syntax

```
navigator.appCodeName
```

Description

appCodeName is a read-only property.

Applies to

navigator

Examples

The following example displays the value of the appCodeName property:

```
document.write("The value of navigator.appCodeName is " + navigator.appCodeName)
```

This example displays information such as the following:

```
The value of navigator.appCodeName is Mozilla
```

See also

- appName, appVersion, userAgent properties

---

appName property

A string specifying the name of the browser.

Syntax

```
navigator.appName
```

Description

appName is a read-only property

Applies to

navigator

Examples

The following example displays the value of the appName property:

```
document.write("The value of navigator.appName is " + navigator.appName)
```

This example displays information such as the following:

```
The value of navigator.appName is Netscape
```

See also

- appVersion, appCodeName, userAgent properties

---

appVersion property

A string specifying version information for the Navigator.

Syntax

```
navigator.appVersion
```

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 3.11.
- *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.

appVersion is a read-only property.

Applies to

navigator

Examples

The following example displays version information for the Navigator:

```
document.write("The value of navigator.appVersion is " + navigator.appVersion)
```

This example displays information such as the following:

```
The value of navigator.appVersion is 2.0b5 (Win95, I)
```

See also

- appName, appCodeName, userAgent properties

---

bgColor property

The color of the document background.

Syntax

```
document.bgColor
```

Description

The bgColor property is expressed as a hexadecimal RGB triplet or as one of the string literals listed in the Color Appendix. This property is the JavaScript reflection of the BGCOLOR attribute of the HTML BODY tag. The default value of this property is set by the user on the colors tab of the Preferences dialog box, which is displayed by choosing General Preferences from the Options menu. You can set this property before or 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".

Applies to

document

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

- alinkColor, fgColor, linkColor, and vlinkColor properties

---

checked property

A Boolean value specifying the selection state of a checkbox or radio element.

Syntax

```
1. checkboxName.checked
2. radioName[index].checked
```

checkboxName is the value of the NAME attribute of a checkbox object.
radioName is the value of the NAME attribute of a radio object.
index is an integer representing a radio button in a radio object.

Description

If a checkbox or radio element is selected, the value of the checked property is true; otherwise, it is false.

You can set the checked property at any time. The display of the checkbox element updates immediately when you set the checked property.

Applies to

checkbox, radio

Examples

The following example examines an array of radio elements 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 i = ""
   var checkedButton = ""
   for (i in document.musicForm.musicType) {
      if (document.musicForm.musicType[i].checked=="1") {
```

```
            checkedButton=document.musicForm.musicType[i].value
         }
      }
   }
```

See also

- defaultChecked property

---

cookie property

String value of a cookie, which is a small piece of information stored by the Navigator in the cookies.txt file.

Syntax

```
document.cookie
```

Description

Use string methods such as substring, charAt, indexOf, and lastIndexOf to determine the value stored in the cookie. See the Netscape cookie specification for a complete specification of the cookie syntax.

You can set the cookie property at any time.

Applies to

document

Examples

The following function uses the cookie property to record a reminder for users of an application. The "expires=" component sets an expiration date for the cookie, so it persists beyond the current browser session.

```
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=" + Date(time + 24*60*60*1000)
}
```

When the user loads the page that contains this function, another function uses indexOf("@") and indexOf("=") to determine the date and time stored in the cookie.

See Also

- hidden object

---

defaultChecked property

A Boolean value indicating the default selection state of a checkbox or radio element.

Syntax

```
1. checkboxName.defaultChecked
2. radioName[index].defaultChecked
```

checkboxName is the value of the NAME attribute of a checkbox object.
radioName is the value of the NAME attribute of a radio object.
index is an integer representing a radio button in a radio object.

Description

If an checkbox or radio element is selected by default, the value of the defaultChecked property is true; otherwise, it is false. defaultChecked initially reflects whether the HTML CHECKED attribute is used within an INPUT tag; however, setting defaultChecked may override the CHECKED attribute.

You can set the defaultChecked property at any time. The display of the checkbox element does not update when you set the defaultChecked property, only when you set the checked property.

Applies to

checkbox, radio

Examples

The following example resets an array of radio elements 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

checked property

---

defaultSelected property

A Boolean value indicating the default selection state of an option in a select element.

Syntax

```
selectName.options[index].defaultSelected
```

selectName is the value of the NAME attribute of a select object.
index is an integer representing an option in a select object.

Description

If an option in a select element is selected by default, the value of the defaultSelected property is true; otherwise, it is false. defaultSelected initially reflects whether the HTML SELECTED attribute is used within an OPTION tag; however, setting defaultSelected may override the SELECTED attribute.

Applies to

options property

Examples

In the following example, the restoreDefault() function returns the *musicType* element to its default state. The for loop evaluates every option in the select element. 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 element is similar to the following:

```
<SELECT NAME="musicType">
   <OPTION SELECTED> R&B
   <OPTION> Jazz
   <OPTION> Blues
   <OPTION> New Age
</SELECT>
```

See also

- index, selected and selectedIndex properties

---

defaultStatus property

The default message displayed in the status bar at the bottom of the window.

Syntax

```
windowReference.defaultStatus
```

windowReference is a valid way of referring to a window, as described in the window object.

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 onMouseOver event handler.

Applies to

window

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://www.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

- status property

---

defaultValue property

xxx

Syntax

xxx

Description

For hidden, password, text, and textarea, string, the initial contents of the field.

Applies to

hidden, password, text, textarea

Examples

xxx Examples to be supplied.

See also

- value property

E property

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

Syntax

```
Math.E
```

Description

Because E is a constant, it is a read-only property of Math.

Applies to

Math

Examples

The following example displays Euler's constant:

```
document.write("Euler's constant is " + Math.E)
```

See also

- LN2, LN10, PI, SQRT1_2, SQRT2 properties

---

elements property

xxx

Syntax

Description

Array of objects corresponding to form elements (such as checkbox, radio, and text objects) in source order.

The *elements* array contains an entry for each object in a form. For example, if a form has a text field, a radio button group, and a checkbox, these elements are reflected as `formName.elements[0]`, `formName.elements[1]`, and `formName.elements[2]`.

Applies to

form

Examples

xxx Examples to be supplied.

---

encoding property

A string specifying the MIME encoding of the form.

Syntax

```
formName.encoding
```

formName is the name of any form or an element in the forms array.

Description

The encoding property is a reflection of the ENCTYPE attribute of the HTML FORM tag.

Applies to

form

Examples

xxx To be supplied

See also

xxx To be supplied

---

fgColor property

The color of the document foreground text.

Syntax

```
document.fgColor
```

Description

The fgColor property is expressed as a hexadecimal RGB triplet or as one of the string literals listed in the Color Appendix. This property is the JavaScript reflection of the FGCOLOR attribute of the HTML BODY tag. The default value of this property is set by the user on the colors tab of the Preferences dialog box, which is displayed by choosing General Preferences from the Options menu. 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".

Applies to

document

Examples

The following example sets the color of the foreground to aqua using a string literal:

```
document.fgColor="aqua"
```

The following example sets the color of the foreground to aqua using a hexadecimal triplet:

```
document.fgColor="00FFFF"
```

See also

- alinkColor, bgColor, linkColor, and vlinkColor properties

---

forms property

xxx

Syntax

xxx

Description

Array of objects corresponding to named forms (`<FORM NAME="">` tags) in source order.

The *forms* array contains an entry for each form object in a document. For example, if a document contains three forms, these forms are reflected as `document.forms[0], document.forms[1]`, and `document.forms[2]`.

You can refer to a form's elements by using the *forms* array. 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
```

To obtain the number of forms in a document, use the length property: `document.forms.length`.

Applies to

document

Examples

xxx Examples to be supplied.

See also

- length property

---

frames property

xxx

Syntax

xxx

Description

Array of objects corresponding to child frame windows (<FRAMESET> tag) in source order.

The *frames* array contains an entry for each child frame in a window. For example, if a window contains three child frames, these frames are reflected as `window.frames[0], window.frames[1]`, and `window.frames[2]`.

To obtain the number of number of child frames in a window, use the length property: `window.frames.length`.

Applies to

window

Examples

xxx Examples to be supplied.

See also

- length property

---

hash property

A string beginning with a hash mark (#) that specifies an anchor name fragment in the URL.

Syntax

```
location.hash
```

Description

The hash property specifies a portion of the URL.

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 will get an error.

See RFC 1738 (http://www.cis.ohio-state.edu/htbin/rfc/rfc1738.html) for complete information about the hash.

Applies to

location

Examples

See the examples for the href property.

See also

- host, hostname, href, pathname, port, protocol, search properties

---

host property

A string specifying the hostname:port portion of the URL.

Syntax

```
location.host
```

Description

The host property specifies a portion of the URL. The host property is the concatenation of the hostname 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 will get an error.

See Section 3.1 of RFC 1738 for complete information about the hostname and port.

Applies to

location

Examples

See the examples for the href property.

See also

- hash, hostname, href, pathname, port, protocol, search properties

---

hostname property

A string specifying the domain name or IP address of a network host.

Syntax

```
location.hostname
```

Description

The hostname property specifies a portion of the URL. The hostname property is a substring of the host property. The host property is the concatenation of the hostname 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 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 will get an error.

See Section 3.1 of RFC 1738 for complete information about the hostname.

Applies to

location

Examples

See the examples for the href property.

See also

- hash, host, href, pathname, port, protocol, search properties

---

href property

A string specifying the entire URL.

Syntax

```
location.href
```

Description

The href property specifies the entire URL. Other location object properties are substrings of the href property. You can set the href property at any time.

Applies to

location

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.search = " +
newWindow.location.search + "<P>")
```

```
msgWindow.document.write("newWindow.location.hash = " + newWindow.location.hash +
"<P>")
msgWindow.document.close()
```

The previous example displays the following output:

```
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.search =
newWindow.location.hash = #checkbox_object
```

See also

- hash, host, hostname, pathname, port, protocol, search properties

---

index property

An integer representing the index of a radio element or an option in a select element.

Syntax

1. *radioName*.index
2. *selectName*.options[*indexValue*].index

radioName is the value of the NAME attribute of a radio object.
selectName is the value of the NAME attribute of a select object.
indexValue is an integer representing an option in a select object.

Description

For radio, number, the ordinal number of the radio field, 0-based. For a select object option, the number identifying the position of the option in the selection, starting from zero.

Applies to

radio object, options property

Examples

xxx Examples to be supplied.

See also

For options:

- defaultSelected, selected, selectedIndex properties

---

lastModified property

A string representing the date that a document was last modified.

Syntax

```
document.lastModified
```

Description

lastModified is a read-only property.

Applies to

document

Examples

In the following example, the value of the lastModified property is assigned to a variable called *currentDate*:

```
var currentDate = ""
newWindow = window.open("http://www.netscape.com")
currentDate = newWindow.document.lastModified
```

---

length property

xxx

Syntax

xxx

Description

For a history object, the length of the history list. For a string object, the integer length of the string. For a radio object, the number of radio buttons in the object. For an *anchors*, *forms*, *frames*, *links*, or *options* array, the number of elements in the array. For a select element, the number of options in the element.

For a null string, length is zero.

Applies to

- history, radio, select, string objects
- anchors, forms, frames, links, options properties

Examples

xxx Example with history to be supplied.

If the string object `mystring` is "netscape", then `mystring.length` returns the integer 8.

If the current document contains five forms, then `document.forms.length` returns the integer 5.

---

linkColor property

The color of the document hyperlinks.

Syntax

```
document.linkColor
```

Description

The linkColor property is expressed as a hexadecimal RGB triplet or as one of the string literals listed in the Color Appendix. This property is the JavaScript reflection of the LINK attribute of the HTML BODY tag. The default value of this property is set by the user on the colors tab of the Preferences dialog box, which is displayed by choosing General Preferences from the Options menu. 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".

Applies to

document

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

- alinkColor, bgColor, fgColor, and vlinkColor properties

---

links property

xxx

Syntax

xxx

Description

Array of objects corresponding to link objects (`<A HREF=URL>` tags) in source order.

The links array contains an entry for each link object in a document. For example, if a document contains three link objects, these links are reflected as `document.links[0], document.links[1]`, and `document.links[2].`

To obtain the number of links in a document, use the length property: `document.links.length`.

Applies to

document

Examples

xxx Examples to be supplied.

See also

- anchors, length properties

LN2 property

The natural logarithm of two, approximately 0.693.

Syntax

```
Math.LN2
```

Description

Because LN2 is a constant, it is a read-only property of Math.

Applies to

Math

Examples

The following example displays the natural log of 2:

```
document.write("The natural log of 2 is " + Math.LN2)
```

See also

- E, LN10, PI, SQRT1_2, SQRT2 properties

LN10 property

The natural logarithm of ten, approximately 2.302.

Syntax

```
Math.LN10
```

Description

Because LN10 is a constant, it is a read-only property of Math.

Applies to

Math

Examples

The following example displays the natural log of 10:

```
document.write("The natural log of 10 is " + Math.LN10)
```

See also

- E, LN2, PI, SQRT1_2, SQRT2 properties

---

location property

A string specifying the complete URL of the document.

Syntax

```
document.location
```

Description

Do not confuse the location object with the location property of the document object. The location object has properties with values you can change, and the location property does not. `document.location` is a string-

valued property that usually matches what `window.location` is set to when you load the document, but redirection may change it.

location is a read-only property of document.

Applies to

document

Examples

The following example displays the URL of the current document:

```
document.write("The current URL is " + document.location)
```

See also

- location object

---

method property

A string specifying how form field input information is sent to the server.

Syntax

```
formName.method
```

formName is the name of any form or an element in the forms array.

Description

The method property is a reflection of the METHOD attribute of the HTML FORM tag. You cannot set this property after the HTML source has been through layout. The method property can evaluate to either "get" or "post". See the form object for more information.

Applies to

form

Examples

The following example sets the method property of the *musicInfo* form to "post":

```
musicInfo.method="post"
```

See also

- action, target properties

---

name property

xxx

Syntax

xxx

Description

A string whose value is the same as the NAME attribute of the object. Note that for button, reset, and submit objects, this is the internal name for the button, not the label that appears onscreen.

Applies to

- button, checkbox, hidden, password, radio, reset, select, submit, text, textarea

Examples

xxx Examples to be supplied.

See also

- value property

---

options property

xxx

Syntax

xxx

Description

Array of objects corresponding to options in a select object (`<OPTION>` tags) in source order.

The options array contains an entry for each option in a select object. For example, if a select object named musicStyle contains three options, these options are reflected as `musicStyle.options[0]`, `musicStyle.options[1]`, and `musicStyle.options[2]`.

To obtain the number of options in a select object, use the length property of the select object: *`selectName`*`.length`.

Applies to

select

Properties

- defaultSelected
- index

- selected reflects the SELECTED argument
- text
- value

Examples

xxx Examples to be supplied.

See also

- length property

---

parent property

xxx

Syntax

xxx

Description

In a <FRAMESET> and <FRAME> relationship, the <FRAMESET> window.

Applies to

window

Examples

xxx Examples to be supplied.

---

pathname property

A string specifying the url-path portion of the URL.

Syntax

```
location.pathname
```

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 will get an error.

See Section 3.1 of RFC 1738 for complete information about the pathname.

Applies to

location

Examples

See the examples for the href propeerty.

See also

- hash, host, hostname, href, port, protocol, search properties

---

PI property

The ratio of the circumference of a circle to its diameter, approximately 3.1415.

Syntax

```
Math.PI
```

Description

Because PI is a constant, it is a read-only property of Math.

Applies to

Math

Examples

The following example displays the value of pi:

```
document.write("The value of pi is " + Math.PI)
```

See also

- E, LN2, LN10, SQRT1_2, SQRT2 properties

---

port property

A string specifying the port number to connect to, if any.

Syntax

```
location.port
```

Description

The port property specifies a portion of the URL. The port property is a substring of the host property. The host property is the concatenation of the hostname 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 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.

See Section 3.1 of RFC 1738 for complete information about the port.

Applies to

location

Examples

See the examples for the href property.

See also

- hash, host, hostname, href, pathname, protocol, search properties

---

protocol property

A string specifying the beginning of the URL, up to and including the first colon.

Syntax

```
location.protocol
```

Description

The protocol property specifies a portion of the URL. The protocol indicates the access method of the URL. For example, a protocol of "http:" specifies Hypertext Transfer Protocol, and a protocol of "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 will get an error.
The protocol property represents the scheme name of the URL. See Section 2.1 of RFC 1738 for complete information about the protocol.

Applies to

location

Examples

See the examples for the href property.

See also

- hash, host, hostname, href, pathname, port, search properties

---

referrer property

xxx

Syntax

xxx

Description

xxx Description to be supplied.

Applies to

document

Examples

xxx Examples to be supplied.

---

search property

A string beginning with a question mark that specifies any query information in the URL.

Syntax

```
location.search
```

Description

The search property specifies a portion of the URL.
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 will get an error.

See Section 3.3 of RFC 1738 for complete information about the search.

Applies to

location

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://guide-p.infoseek.com/WW/NS/
Titles?qt=RFC+1738+&col=WW")
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.search = " +
newWindow.location.search + "<P>")
msgWindow.document.write("newWindow.location.hash = " + newWindow.location.hash +
"<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.protocol = http:
newWindow.location.host = guide-p.infoseek.com
newWindow.location.hostName = guide-p.infoseek.com
newWindow.location.port =
newWindow.location.pathname = /WW/NS/Titles
newWindow.location.search = ?qt=RFC+1738+&col=WW
newWindow.location.hash =
```

See also

- hash, host, hostname, href, pathname, port, protocol properties

---

selected property

A Boolean value specifying the current selection state of an option in a select object.

Syntax

```
selectName.options[index].selected
```

selectName is the value of the NAME attribute of a select object.
index is an integer representing an option in a select object.

Description

If an option in a select element 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 select element updates immediately when you set the selected property.

In general, the selected property is more useful than the selectedIndex property for select elements that are created with the MULTIPLE attribute. With the 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.

Applies to

options property

Examples

See the examples for the defaultSelected property.

See also

- defaultSelected property

---

selectedIndex property

An integer specifying the index of the selected item.

Syntax

```
selectName.selectedIndex
```

selectName is the value of the NAME attribute of a select object.

Description

Options in a select element 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 form element updates immediately when you set the selectedIndex property.

In general, the selectedIndex property is more useful for select elements 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 element.

The selected property of the select element's options array is more useful for select elements that are created with the MULTIPLE attribute. With the 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.

Applies to

select

Examples

In the following example, the getSelectedIndex() function assigns the selected index in the *musicType* element to the variable *chosenIndex*:

```
function getSelectedIndex() {
   var chosenIndex=""
   chosenIndex=document.musicForm.musicType.selectedIndex
}
```

The previous example assumes that the select element is similar to the following:

```
<SELECT NAME="musicType">
   <OPTION SELECTED> R&B
   <OPTION> Jazz
   <OPTION> Blues
   <OPTION> New Age
</SELECT>
```

See also

- defaultSelected, index, selected properties

---

self property

The self property refers to the current window.

Syntax

xxx

Description

The *self* property refers to the current window. Use the *self* property to disambiguate a window property from a form of the same name. You can also use the *self* property to make your code more readable.

Applies to

window

Examples

In the following example, `self.status` is used to set the status property. This usage disambiguates the status property of a window from a form called "status".

```
<A HREF=""
   onClick="this.href=pickRandomURL()"
   onMouseOver="self.status='Pick a random URL' ; return true">
Go!</A>
```

See also

- window property

SQRT1_2 property

The square root of one-half; equivalently, one over the square root of two, approximately 0.707.

Syntax

```
Math.SQRT1_2
```

Description

Because SQRT1_2 is a constant, it is a read-only property of Math.

Applies to

Math

Examples

The following example displays 1 over the square root of 2:

```
document.write("1 over the square root of 2 is " + Math.SQRT1_2)
```

See also

- E, LN2, LN10, PI, SQRT2 properties

SQRT2 property

The square root of two, approximately 1.414.

Syntax

```
Math.SQRT2
```

Description

Because SQRT2 is a constant, it is a read-only property of Math.

Applies to

Math

Examples

The following example displays the square root of 2:

```
document.write("The square root of 2 is " + Math.SQRT2)
```

See also

- E, LN2, LN10, PI, SQRT1_2 properties

---

status property

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.

Syntax

```
windowReference.status
```

windowReference is a valid way of referring to a window, as described in the window object.

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.

Applies to

window

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 above example, the status property of the window is assigned to the window's self property, as self.status. As this example shows, you must return true to set the status property in the onMouseOver event handler.

See also

- defaultStatus property

---

target property

For form, a string specifying the name of the window that responses go to after a form has been submitted. For link, a string specifying the name of the window that displays the content of a clicked hypertext link.

Syntax

```
1. formName.target
2. linkName.target
```

formName is the name of any form or an element in the forms array.
linkName is an element in the links array.

Description

The target property is a reflection of the TARGET attribute of the HTML FORM and A tags. You can set this property before or after the HTML source has been through layout.

Applies to

form, link

Examples

The following example specifies that responses to the musicInfo form are displayed in the "msgWindow" window:

```
document.musicInfo.target="msgWindow"
```

See also

For form:
- action, method properties

---

text property

xxx

Syntax

xxx

Description

String, reflection of the text after the <OPTION> tag.

Applies to

select

Examples

xxx Examples to be supplied.

---

title property

A string representing the title of a document.

Syntax

```
document.title
```

Description

The title property is a reflection of the value specified within the <TITLE> and </TITLE> tags. If a document does not have a title, the title property is null.

title is a read-only property.

Applies to

document

Examples

In the following example, the value of the title property is assigned to a variable called <I>docTitle</I>:

```
var docTitle = ""
newWindow = window.open("http://www.netscape.com")
docTitle = newWindow.document.title
```

top property

xxx

Syntax

xxx

Description

The top-most ancestor window, which is its own parent.

Applies to

window

Examples

xxx Examples to be supplied.

userAgent property

A string representing the value of the user-agent header sent in the HTTP protocol from client to server.

Syntax

```
navigator.userAgent
```

Description

Servers use the value sent in the user-agent header to identify the client.

userAgent is a read-only property.

Applies to

navigator

Examples

The following example displays userAgent information for the Navigator:

```
document.write("The value of navigator.userAgent is " + navigator.userAgent)
```

This example displays information such as the following:

```
The value of navigator.userAgent is Mozilla/2.0b5 (Win16; I)
```

See also

- appName, appVersion, appCodeName properties

---

value property

For button, reset, and submit objects, a string that is the same as the VALUE attribute (this is the label that appears onscreen, not the internal name for the button). For checkbox, a string, "on" if item is checked; "off" otherwise. For radio, a string, reflection of the VALUE attribute. For select objects, reflection of VALUE attribute, sent to server on submit. For hidden, text, textarea, and string, the contents of the field.

Syntax

xxx

Description

For button, reset, and submit objects, a string that is the same as the VALUE attribute (this is the label that appears onscreen, not the internal name for the button). For checkbox, a string, "on" if item is checked; "off" otherwise. For radio, a string, reflection of the VALUE attribute. For select objects, reflection of VALUE attribute, sent to server on submit. For text and textarea, string, the contents of the field.

If you change the value property of a text or textArea object, the object on the form is updated dynamically. If you change the value property of any other type of object, the object on the form is not updated.

Applies to

button, checkbox, hidden, password, radio, reset, select, submit, text, textarea

Examples

xxx Examples to be supplied.

See also

For password, text, and textarea:
- defaultValue property

---

vlinkColor property

The color of visited links.

Syntax

```
document.vlinkColor
```

Description

The vlinkColor property is expressed as a hexadecimal RGB triplet or as one of the string literals listed in the Color Appendix. This property is the JavaScript reflection of the VLINK attribute of the HTML BODY tag. The default value of this property is set by the user on the colors tab of the Preferences dialog box, which is displayed by choosing General Preferences from the Options menu. 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".

Applies to

document

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

- alinkColor, bgColor, fgColor, and linkColor properties

---

window property

xxx

Syntax

xxx

Description

The window property refers to the current window. Use the window property to disambiguate a property of the window object from a form of the same name. You can also use the window property to make your code more readable.

Applies to

window

Examples

In the following example, `window.status` is used to set the status property. This usage disambiguates the status property of a window from a form called "status".

```
<A HREF=""
   onClick="this.href=pickRandomURL()"
   onMouseOver="window.status='Pick a random URL' ; return true">
Go!</A>
```

See also

self property

Event handlers

The following event handlers are available in JavaScript:

- onBlur
- onChange
- onClick
- onFocus
- onLoad
- onMouseOver
- onSelect
- onSubmit
- onUnload

---

onBlur event handler

A blur event occurs when a select, text, or textarea field on a form loses focus. The onBlur event handler executes JavaScript code when a blur event occurs.

See the relevant objects for the onBlur syntax.

Event Handler of

select, text, textarea

Examples

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)">
```

See also

- onChange , onFocus event handlers

---

onChange event handler

A change event occurs when a select, text, or textarea field loses focus and its value has been modified. The onChange event handler executes JavaScript code when a change event occurs.

Use the onChange event handler to validate data after it is modified by a user.

See the relevant objects for the onChange syntax.

Event Handler of

select, text, textarea

Examples

In the following example, *userName* is a text field. When a user attempts to leave the field, the onBlur event handler calls the checkValue() function to confirm that *userName* has a legal value.

```
<INPUT TYPE="text" VALUE="" NAME="userName" onBlur="checkValue(this.value)">
```

See also

- onBlur , onFocus event handlers

---

onClick event handler

A click event occurs when an object on a form is clicked. The onClick event handler executes JavaScript code when a click event occurs.

See the relevant objects for the onClick syntax.

Event Handler of

button, checkbox, radio, link, reset, submit

Examples

For example, 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 above 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 the onClick event handler of a link 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, the onMouseOver event handler specifies a custom message for the Navigator 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.

---

onFocus event handler

A focus event occurs when a field receives input focus by tabbing with the keyboard or clicking with the mouse. Selecting within a field results in a select event, not a focus event. The onFocus event handler executes JavaScript code when a focus event occurs.

See the relevant objects for the onFocus syntax.

Event Handler of

select, text, textarea

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

- onBlur , onChange event handlers

---

onLoad event handler

A load event occurs when Navigator finishes loading a window or all frames within a <FRAMESET>. The onLoad event handler executes JavaScript code when a load event occurs.

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).

Event Handler of

window

Examples

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!")>
```

See also

- onUnload event handler

---

onMouseOver event handler

A mouseOver event occurs once each time the mouse pointer moves over an object from outside that object. The onMouseOver event handler executes JavaScript code when a mouseOver event occurs.

You must return true within the event handler if you want to set the status or defaultStatus properties with the onMouseOver event handler.

See the relevant objects for the onMouseOver syntax.

Event Handler of

link

Examples

By default, the HREF value of an anchor displays in the status bar at the bottom of the Navigator when a user places the mouse pointer over the anchor. In the following example, the onMouseOver event handler 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.

---

onSelect event handler

A select event occurs when a user selects some of the text within a text or textarea field. The onSelect event handler executes JavaScript code when a select event occurs.

See the relevant objects for the onSelect syntax.

Applies to

text, textarea

Examples

The following example uses an onSelect handler in the valueField text object to call the selectState() function.

```
<INPUT TYPE="text" VALUE="" NAME="valueField" onFocus="selectState()">
```

onSubmit event handler

A submit event occurs when a user submits a form. The onSubmit event handler executes JavaScript code when a submit event occurs.

You can use the onSubmit event handler 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.

See the relevant objects for the onSubmit syntax.

Event Handler of

form

Examples

In the following example, the onSubmit event handler calls the formData() 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 formData(this)"
```

See also the examples for the form object.

See Also

- submit object, submit method

onUnload event handler

An unload event occurs when you exit a document. The onUnload event handler executes JavaScript code when an unload event occurs.

Use the onUnload event handler 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 Handler of

window

Examples

In the following example, the onUnload event handler calls the cleanUp() function to perform some shut down processing when the user exits a web page:

```
<BODY onUnload="cleanUp()">
```

See also

- onLoad event handler

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 semi-colon.

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 {}.

The following statements are available in JavaScript:

- break
- comment
- continue
- for
- for...in
- function
- if...else
- new
- return
- this
- var
- while
- with

NOTE: **new** and **this** are not really statements, but are included in this section for convenience.

---

break

A statement that terminates the current **while** or **for** loop and transfers program control to the statement following the terminated loop.

Syntax

```
break
```

Examples

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

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

---

comment

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

Syntax

```
1. // comment text
2. /* multiple line comment text */
```

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

A 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. 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.

Syntax

```
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
}
```

---

for

A 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. The parts of the **for** statement are:

- The *initial expression*, generally used to initialize a counter variable. This statement may optionally declare new variables with the **var** keyword. This expression is optional.
- The *condition* that is evaluated on each pass through the loop. If this condition is true, the statements in the succeeding block are performed. This conditional test is optional. If omitted, then the condition always evaluates to true.
- An *update expression* generally used to update or increment the counter variable. This expres sion is optional.
- A block of statements that are executed as long as the *condition* is 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.

Syntax

```
for ([initial expression]; [condition]; [update expression]) {
    statements
}
initial expression = statement | variable declaration
```

Examples

This **for** statement starts by declaring the variable *i* and initializing it to zero. It checks that *i* is less than nine, and performs the two succeeding statements, and increments *i* by one after each pass through the loop.

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

---

for...in

A statement that iterates a variable *var* over all the properties of object *obj*. For each distinct property, it executes the statements in *statements*.

Syntax

```
for (var in obj) {
    statements }
```

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, obj_name) {
   var result = ""
   for (var i in obj) {
      result += obj_name + "." + i + " = " + obj[i] + "<BR>"
   }
   result += "<HR>"

    return result
}
```

---

function

A statement that declares a JavaScript function *name* with the specified parameters *param*. Acceptable parameters include strings, numbers, and objects.

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.

Syntax

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

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

A conditional statement that executes the statements in *statements* if *condition* is true. In the optional **else** clause, it executes the statements in *else statements* if *condition* is false. These may be any JavaScript statements, including further nested **if** statements.

Syntax

```
if (condition) {
    statements
} [else {
    else statements
}]
```

Examples

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

---

new

An operator that lets you create an instance of a user-defined object type.

Creating an object type requires two steps:

- Define the object type by writing a function.
- Create an instance of the object with <B>new</B>.

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.

Syntax

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

objectName is the name of the new object instance.
objectType is the object type. It must be a function that defines an object type.
param1...*paramN* are the property values for the object. These properties are parameters defined for the *objectType* function.

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, year, and color. 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
```

---

return

A statement specifies the value to be returned by a function.

Syntax

```
return expression;
```

Examples

The following simple function returns the square of its argument, x, where x is an number.

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

---

this

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

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:

```
<INPUT TYPE = "text" NAME = "age" SIZE = 3
   onChange="validate(this, 18, 99)">
```

---

var

A statement that declares a variable *varname*, optionally initializing it to *value*. The variable name *varname* can be any legal identifier, and *value* can be any legal expression. 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 there is a global variable of the same name.

Syntax

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

Examples

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

---

while

A statement that creates a loop that evaluates the expression *condition*, and if it is true, executes *statements*. It then repeats this process, as long as *condition* is true. When *condition* evaluates to false, execution continues with the statement following *statements*.

Although not required, it is good practice to indent the statements a **while** loop four spaces from the beginning of a **for** statement.

Syntax

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

Examples

The following **while** loop iterates as long as *n* is less than three. Each iteration, it increments *n* and adds it to *x*. Therefore, *x* and *n* take on the following values:

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

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

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

---

with

A statement that establishes *object* as the default object for the *statements*. Any property references without an object are then assumed to be for *object*. Note that the parentheses are required around *object*.

Syntax

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

Examples

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

Reserved words

The reserved words in this list cannot be used as JavaScript variables, functions, methods, or object names. Some of these words are keywords used in JavaScript; others are reserved for future use.

- abstract
- boolean
- break
- byte
- case
- catch
- char
- class
- const
- continue
- default
- do
- double
- else
- extends
- false
- final
- finally
- float
- for
- function
- goto
- if
- implements
- import
- in
- instanceof
- int
- interface
- long
- native
- new
- null
- package
- private
- protected
- public
- return
- short
- static
- super
- switch
- synchronized
- this
- throw
- throws
- transient
- true
- try
- var
- void
- while
- with

Color values

The string literals in this table can be used to specify colors in the JavaScript alinkColor, bgColor, fgColor, linkColor, and vlinkColor properties and the fontcolor method.

You can also use these string literals to set the color in the HTML reflections of these properties, for example `<BODY BGCOLOR="bisque">`, and to set the COLOR attribute of the FONT tag, for example, `<FONT COLOR="blue">color</font>`.

The following red, green, and blue values are in Decimal and Hexidecimal.

	Decimal			Hexidecimal		
Color	**Red**	**Green**	**Blue**	**Red**	**Green**	**Blue**
aliceblue	240	248	255	F0	F8	FF
antiquewhite	250	235	215	FA	EB	D7
aqua	0	255	255	00	FF	FF
aquamarine	127	255	212	7F	FF	D4
azure	240	255	255	F0	FF	FF
beige	245	245	220	F5	F5	DC
bisque	255	228	196	FF	E4	C4
black	0	0	0	00	00	00
blanchedalmond	255	235	205	FF	EB	CD
blue	0	0	255	00	00	FF
blueviolet	138	43	226	8A	2B	E2
brown	165	42	42	A5	2A	2A
burlywood	222	184	135	DE	B8	87
cadetblue	95	158	160	5F	9E	A0
chartreuse	127	255	0	7F	FF	00
chocolate	210	105	30	D2	69	1E
coral	255	127	80	FF	7F	50
cornflowerblue	100	149	237	64	95	ED
cornsilk	255	248	220	FF	F8	DC
crimson	220	20	60	DC	14	3C
cyan	0	255	255	00	FF	FF
darkblue	0	0	139	00	00	8B
darkcyan	0	139	139	00	8B	8B
darkgoldenrod	184	134	11	B8	86	0B
darkgray	169	169	169	A9	A9	A9
darkgreen	0	100	0	00	64	00
darkkhaki	189	183	107	BD	B7	6B
darkmagenta	139	0	139	8B	00	8B
darkolivegreen	85	107	47	55	6B	2F
darkorange	255	140	0	FF	8C	00
darkorchid	153	50	204	99	32	CC
darkred	139	0	0	8B	00	00
darksalmon	233	150	122	E9	96	7A
darkseagreen	143	188	143	8F	BC	8F
darkslateblue	72	61	139	48	3D	8B
darkslategray	47	79	79	2F	4F	4F
darkturquoise	0	206	209	00	CE	D1

	Decimal			Hexidecimal		
Color	**Red**	**Green**	**Blue**	**Red**	**Green**	**Blue**
darkviolet	148	0	211	94	00	D3
deeppink	255	20	147	FF	14	93
deepskyblue	0	191	255	00	BF	FF
dimgray	105	105	105	69	69	69
dodgerblue	30	144	255	1E	90	FF
firebrick	178	34	34	B2	22	22
floralwhite	255	250	240	FF	FA	F0
forestgreen	34	139	34	22	8B	22
fuchsia	255	0	255	FF	00	FF
gainsboro	220	220	220	DC	DC	DC
ghostwhite	248	248	255	F8	F8	FF
gold	255	215	0	FF	D7	00
goldenrod	218	165	32	DA	A5	20
gray	128	128	128	80	80	80
green	0	128	0	00	80	00
greenyellow	173	255	47	AD	FF	2F
honeydew	240	255	240	F0	FF	F0
hotpink	255	105	180	FF	69	B4
indianred	205	92	92	CD	5C	5C
indigo	75	0	130	4B	00	82
ivory	255	255	240	FF	FF	F0
khaki	240	230	140	F0	E6	8C
lavender	230	230	250	E6	E6	FA
lavenderblush	255	240	245	FF	F0	F5
lawngreen	124	252	0	7C	FC	00
lemonchiffon	255	250	205	FF	FA	CD
lightblue	173	216	230	AD	D8	E6
lightcoral	240	128	128	F0	80	80
lightcyan	224	255	255	E0	FF	FF
lightgoldenrodyellow	250	250	210	FA	FA	D2
lightgreen	144	238	144	90	EE	90
lightgrey	211	211	211	D3	D3	D3
lightpink	255	182	193	FF	B6	C1
lightsalmon	255	160	122	FF	A0	7A
lightseagreen	32	178	170	20	B2	AA
lightskyblue	135	206	250	87	CE	FA
lightslategray	119	136	153	77	88	99
lightsteelblue	176	196	222	B0	C4	DE
lightyellow	255	255	224	FF	FF	E0
lime	0	255	0	00	FF	00
limegreen	50	205	50	32	CD	32
linen	250	240	230	FA	F0	E6
magenta	255	0	255	FF	00	FF
maroon	128	0	0	80	00	00
mediumaquamarine	102	205	170	66	CD	AA
mediumblue	0	0	205	00	00	CD
mediumorchid	186	85	211	BA	55	D3
mediumpurple	147	112	219	93	70	DB
mediumseagreen	60	179	113	3C	B3	71
mediumslateblue	123	104	238	7B	68	EE
mediumspringgreen	0	250	154	00	FA	9A
mediumturquoise	72	209	204	48	D1	CC

	Decimal			Hexidecimal		
Color	**Red**	**Green**	**Blue**	**Red**	**Green**	**Blue**
mediumvioletred	199	21	133	C7	15	85
midnightblue	25	25	112	19	19	70
mintcream	245	255	250	F5	FF	FA
mistyrose	255	228	225	FF	E4	E1
moccasin	255	228	181	FF	E4	B5
navajowhite	255	222	173	FF	DE	AD
navy	0	0	128	00	00	80
oldlace	253	245	230	FD	F5	E6
olive	128	128	0	80	80	00
olivedrab	107	142	35	6B	8E	23
orange	255	165	0	FF	A5	00
orangered	255	69	0	FF	45	00
orchid	218	112	214	DA	70	D6
palegoldenrod	238	232	170	EE	E8	AA
palegreen	152	251	152	98	FB	98
paleturquoise	175	238	238	AF	EE	EE
palevioletred	219	112	147	DB	70	93
papayawhip	255	239	213	FF	EF	D5
peachpuff	255	218	185	FF	DA	B9
peru	205	133	63	CD	85	3F
pink	255	192	203	FF	C0	CB
plum	221	160	221	DD	A0	DD
powderblue	176	224	230	B0	E0	E6
purple	128	0	128	80	00	80
red	255	0	0	FF	00	00
rosybrown	188	143	143	BC	8F	8F
royalblue	65	105	225	41	69	E1
saddlebrown	139	69	19	8B	45	13
salmon	250	128	114	FA	80	72
sandybrown	244	164	96	F4	A4	60
seagreen	46	139	87	2E	8B	57
seashell	255	245	238	FF	F5	EE
sienna	160	82	45	A0	52	2D
silver	192	192	192	C0	C0	C0
skyblue	135	206	235	87	CE	EB
slateblue	106	90	205	6A	5A	CD
slategray	112	128	144	70	80	90
snow	255	250	250	FF	FA	FA
springgreen	0	255	127	00	FF	7F
steelblue	70	130	180	46	82	B4
tan	210	180	140	D2	B4	8C
teal	0	128	128	00	80	80
thistle	216	191	216	D8	BF	D8
tomato	255	99	71	FF	63	47
turquoise	64	224	208	40	E0	D0
violet	238	130	238	EE	82	EE
wheat	245	222	179	F5	DE	B3
white	255	255	255	FF	FF	FF
whitesmoke	245	245	245	F5	F5	F5
yellow	255	255	0	FF	FF	00
yellowgreen	154	205	50	9A	CD	32

Persistant Client State
HTTP Cookies

Preliminary Specification - Use with caution

Introduction

Cookies are a general mechanism which server side connections (such as CGI scripts) can use to both store and retrieve information on the client side of the connection. The addition of a simple, persistent, client-side state significantly extends the capabilities of Web-based client/server applications.

Overview

A server, when returning an HTTP object to a client, may also send a piece of state information which the client will store. Included in that state object is a description of the range of URLs for which that state is valid. Any future HTTP requests made by the client which fall in that range will include a transmittal of the current value of the state object from the client back to the server. The state object is called a **cookie**, for no compelling reason.

This simple mechanism provides a powerful new tool which enables a host of new types of applications to be written for web-based environments. Shopping applications can now store information about the currently selected items, for fee services can send back registration information and free the client from retyping a user-id on next connection, sites can store per-user preferences on the client, and have the client supply those preferences every time that site is connected to.

Specification

A cookie is introduced to the client by including a **Set-Cookie** header as part of an HTTP response, typically this will be generated by a CGI script.

Syntax of the Set-Cookie HTTP Response Header

This is the format a CGI script would use to add to the HTTP headers a new piece of data which is to be stored by the client for later retrieval.

```
Set-Cookie: NAME=VALUE; expires=DATE;
path=PATH; domain=DOMAIN_NAME; secure
```

NAME=VALUE

This string is a sequence of characters excluding semi-colon, comma and white space. If there is a need to place such data in the name or value, some encoding method such as URL style %XX encoding is recommended, though no encoding is defined or required.

This is the only required attribute on the **Set-Cookie** header.

expires=DATE

The **expires** attribute specifies a date string that defines the valid life time of that cookie. Once the expiration date has been reached, the cookie will no longer be stored or given out.

The date string is formatted as:

```
Wdy, DD-Mon-YY HH:MM:SS GMT
```

This is based on RFC 850, RFC 1036, and RFC 822, with the variations that the only legal time zone is **GMT** and the separators between the elements of the date must be dashes.

expires is an optional attribute. If not specified, the cookie will expire when the user's session ends.

Note: There is a bug in Netscape Navigator version 1.1 and earlier. Only cookies whose **path** attribute is set explicitly to "/" will be properly saved between sessions if they have an **expires** attribute.

domain=DOMAIN_NAME

When searching the cookie list for valid cookies, a comparison of the **domain** attributes of the cookie is made with the Internet domain name of the host from which the URL will be fetched. If there is a tail match, then the cookie will go through **path** matching to see if it should be sent. "Tail matching" means that **domain** attribute is matched against the tail of the fully qualified **domain** name of the host. A **domain** attribute of "acme.com" would match host names "anvil.acme.com" as well as "shipping.crate.acme.com".

Only hosts within the specified domain can set a cookie for a domain and domains must have at least two (2) or three (3) periods in them to prevent domains of the form: ".com", ".edu", and "va.us". Any domain that fails within one of the seven special top level domains listed below only require two periods. Any other domain requires at least three. The seven special top level domains are: "COM", "EDU", "NET", "ORG", "GOV", "MIL", and "INT".

The default value of **domain** is the host name of the server which generated the cookie response.

path=PATH

The **path** attribute is used to specify the subset of URLs in a domain for which the cookie is valid. If a cookie has already passed **domain** matching, then the pathname component of the URL is compared with the path attribute, and if there is a match, the cookie is considered valid and is sent along with the URL request. The path "/foo" would match "/foobar" and "/foo/bar.html". The path "/" is the most general path.

If the **path** is not specified, it as assumed to be the same path as the document being described by the header which contains the cookie.

secure

If a cookie is marked **secure**, it will only be transmitted if the communications channel with the host is a secure one. Currently this means that secure cookies will only be sent to HTTPS (HTTP over SSL) servers.

If **secure** is not specified, a cookie is considered safe to be sent in the clear over unsecured channels.

Syntax of the Cookie HTTP Request Header

When requesting a URL from an HTTP server, the browser will match the URL against all cookies and if any of them match, a line containing the name/value pairs of all matching cookies will be included in the HTTP request. Here is the format of that line:

```
Cookie: NAME1=OPAQUE_STRING1; NAME2=OPAQUE_STRING2 ...
```

Additional Notes

- Multiple Set-Cookie headers can be issued in a single server response.
- Instances of the same path and name will overwrite each other, with the latest instance taking prece dence. Instances of the same path but different names will add additional mappings.
- Setting the path to a higher-level value does not override other more specific path mappings. If there are multiple matches for a given cookie name, but with separate paths, all the matching cookies will be sent. (See examples below.)
- The expires header lets the client know when it is safe to purge the mapping but the client is not re quired to do so. A client may also delete a cookie before it's expiration date arrives if the number of cookies exceeds its internal limits.
- When sending cookies to a server, all cookies with a more specific path mapping should be sent before cookies with less specific path mappings. For example, a cookie "name1=foo" with a path mapping of "/" should be sent after a cookie "name1=foo2" with a path mapping of "/bar" if they are both to be sent.
- There are limitations on the number of cookies that a client can store at any one time. This is a specifi cation of the minimum number of cookies that a client should be prepared to receive and store.
    - 300 total cookies
    - 4 kilobytes per cookie, where the name and the OPAQUE_STRING combine to form the 4 kilobyte limit.
    - 20 cookies per server or domain. (note that completely specified hosts and domains are treated as separate entities and have a 20 cookie limitation for each, not combined)

  Servers should not expect clients to be able to exceed these limits. When the 300 cookie limit or the 20 cookie per server limit is exceeded, clients should delete the least recently used cookie. When a cookie larger than 4 kilobytes is encountered the cookie should be trimmed to fit, but the name should remain intact as long as it is less than 4 kilobytes.
- If a CGI script wishes to delete a cookie, it can do so by returning a cookie with the same name, and an **expires** time which is in the past. The path and name must match exactly in order for the expiring cookie to replace the valid cookie. This requirement makes it difficult for anyone but the originator of a cookie to delete a cookie.
- When caching HTTP, as a proxy server might do, the **Set-cookie** response header should never be cached.
- If a proxy server receives a response which contains a **Set-cookie** header, it should propagate the **Set-**

cookie header to the client, regardless of whether the response was 304 (Not Modified) or 200 (OK).

Similarly, if a client request contains a Cookie: header, it should be forwarded through a proxy, even if the conditional If-modified-since request is being made.

Examples

Here are some sample exchanges which are designed to illustrate the use of cookies.

First Example transaction sequence:

Client requests a document, and receives in the response:

```
Set-Cookie: CUSTOMER=WILE_E_COYOTE; path=/; expires=Wednesday, 09-Nov-99 23:12:40
GMT
```

When client requests a URL in path "/" on this server, it sends:

```
Cookie: CUSTOMER=WILE_E_COYOTE
```

Client requests a document, and receives in the response:

```
Set-Cookie: PART_NUMBER=ROCKET_LAUNCHER_0001; path=/
```

When client requests a URL in path "/" on this server, it sends:

```
Cookie: CUSTOMER=WILE_E_COYOTE; PART_NUMBER=ROCKET_LAUNCHER_0001
```

Client receives:

```
Set-Cookie: SHIPPING=FEDEX; path=/foo
```

When client requests a URL in path "/" on this server, it sends:

```
Cookie: CUSTOMER=WILE_E_COYOTE; PART_NUMBER=ROCKET_LAUNCHER_0001
```

When client requests a URL in path "/foo" on this server, it sends:

```
Cookie: CUSTOMER=WILE_E_COYOTE; PART_NUMBER=ROCKET_LAUNCHER_0001; SHIPPING=FEDEX
```

Second Example transaction sequence:

Assume all mappings from above have been cleared.

Client receives:

```
Set-Cookie: PART_NUMBER=ROCKET_LAUNCHER_0001; path=/
```

When client requests a URL in path "/" on this server, it sends:

```
Cookie: PART_NUMBER=ROCKET_LAUNCHER_0001
```

Client receives:

```
Set-Cookie: PART_NUMBER=RIDING_ROCKET_0023; path=/ammo
```

When client requests a URL in path "/ammo" on this server, it sends:

```
Cookie: PART_NUMBER=RIDING_ROCKET_0023; PART_NUMBER=ROCKET_LAUNCHER_0001
```

NOTE: There are two name/value pairs named "PART_NUMBER" due to the inheritance of the "/" mapping in addition to the "/ammo" mapping.

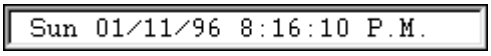
JavaScript Snippets

The Digital Clock

This is a submission that's extremely popular. It's mainly a ticking digital clock originally taken from Netscape's sight, and altered to taste by various sources. This is how it ended up as used on my company's page (http://www.ipst.com)

```
<HTML>
<HEAD>
<TITLE>IPST • Internet Professional Services & Training</TITLE>
<!--Updated 2/6/96-->
<SCRIPT language="JavaScript">
<!--
var timerID = null;
var timerRunning = false;
function stopclock (){
        if(timerRunning)
                cleartimeout(timerID);
                timerRunning=false;
}
function startclock (){
        stopclock();
        showtime();
}
function showtime (){
        var now = new Date();
        var dow = now.getDay();
                if(dow == 0){
                        dow = "Sun";
                } else {
                        if(dow == 1){
                                dow = "Mon";
                        } else {
                                if(dow == 2){
                                        dow = "Tues";
                                } else {
                                        if(dow == 3){
                                                dow = "Wed";
                                        } else {
                                                if(dow == 4){
                                                        dow = "Thur";
                                                } else {
                                                        if(dow == 5){
                                                                dow = "Fri";
                                                        } else {
                                                                dow = "Sat";
                                                        }}}}}}
        var month = now.getMonth() + 1;
                if(month < 10){
                        month = "0" + month;
                }
        var date = now.getDate();
```

```
                if(date < 10){
                        date = "0" + date;
                }
        var year = now.getYear();
        var hours = now.getHours();
        var minutes = now.getMinutes();
        var seconds = now.getSeconds();
        var timeValue = " " + dow;
        timeValue += " " + month + "/" + date + "/" + year;
        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 BGCOLOR=#000000 TEXT=#ffffff LINK=#a68c24 VLINK=#404040
onLoad="startclock()">
<FORM name="clock" onSubmit="0">
<INPUT type="text" name="face" size=29 value="      WELCOME TO IPST !      "
onClick="this.Blur()"><FORM name="clock" onSubmit="0">
</FORM>
```

This will output a text field looking like this : 

Setting the *value* of the INPUT field will allow whatever you want to show while the page is loading. When the page fully loads, the clock will load.

Thanks to **Netscape** and the **JavaScript Index** (http://www.c2.org/~andreww/javascript) for this example.

Remember, if anyone has any examples of value, please email us so we may include it in the next version.