Version 1, January 22, 1996

As taken from Netscape Corporation's World Wide Web Site

Table of Contents

JavaScript Working Document

Current Version : Version 1, dated January 22, 1996

This document was prepared by Aj Brown of IPST from the original JavaScript Authoring Guide from Netscape's World Wide Web site : http://home.netscape.com

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

ajbrown@ajbrown.com
ajbrown@ipst.com
ajbrown@shore.net
webmaster@ipst.com
Compuserve : 102636,362
http://www.ipst.com

I 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 I will include it in the next version.

I'll try to keep it updated every two weeks or so.

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 by client - not compiled	Compiled on server before execution on client.
Object-based. No classes or inheritance; built-in, extensible objects	Object-orientated. Programs consist of object classes, with inheritance, etc.
Integrated with / embedded in HTML	Applets distinct from HTML (accessed from HTML pages)
Do not declare variables' datatypes (loose typing)	Must delcare variables' datatypes (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 Authoring

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.

Using JavaScript in HTML

Embedding JavaScript in documents

A script is embedded in HTML within a SCRIPT tag.

```
<SCRIPT>...</SCRIPT>
```

The text of a script is inserted between SCRIPT and its end tag.

Attributes within the SCRIPT tag can be specified as follows:

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

The LANGUAGE attribute is mandatory unless the SRC attribute is present and specifies the scripting language.

```
<SCRIPT SRC="http://myscript.js">...</SCRIPT>
```

The SRC attribute is optional and, if given, specifies a URL that loads the text of a script.
NOTE: Not yet implemented for Beta4 release.

```
<SCRIPT LANGUAGE="language" SRC=url>...</SCRIPT>
```

Both attributes may be present. NOTE: Not yet implemented for Beta4 release.

Usage Notes

- 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.
- SRC URL information is read in and evaluated as script container content. SRC script is evaluated before in-page script.
- The SRC URL should use the .js suffix.
- A named SCRIPT tag may contain a function body that can be called in an onChange or other event-handler attribute.
- Scripts may be placed inside comment fields to ensure that the script is not displayed when the page's HTML is viewed with a browser unaware of the SCRIPT tag. The entire script is encased by HTML comment tags:

```
<!-- Begin to hide script contents from old browsers.

// End the hiding here. -->
```

- Like Java, JavaScript is case-sensitive.
- Use single quotes (') to delimit string literals so that scripts can be distinguished from attribute values enclosed in double quotes. Example:

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

Some Introductory Examples

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.

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.

Example 3, 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 3 page display.

Enter an expression: 9 + 5

Result: 14

Example 4, 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 num = 0 + str

    if (num < min || num > 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>

<SCRIPT LANGUAGE="JavaScript">

document.write("<PRE>")

document.writeln("Field name: " + document.forms[0].num.name)

document.writeln("Field value: " + document.forms[0].num.value)

document.write("</PRE>")

</SCRIPT>

</BODY>
</HTML>
```

Example 4 page display.

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

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. 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.

JavaScript will attempt to convert an expression to the datatype of the left-hand operand. Expressions are always evaluated from left to right, so JavaScript applies this rule at each step in the evaluation of a complex expression.

For example, suppose you define the following variables

```
var astring = "7"

var anumber = 42
```

Then consider the following statements:

```
x = astring + anumber

y = anumber + astring
```

The first statement will convert *anumber* to a string value, because the left-hand operand, *astring*, is a string. The statement will then concatenate the two strings, so x will have a value of "742".

Conversely, the second statement will convert *astring* to a numeric value, because the left-hand operand, *anumber*, is a number. The statement then adds the two numbers, so y will have a value of 49.

JavaScript cannot convert some strings to numbers. For example, the statements

```
var anumber = 42

var astring = "Phil"

y = anumber + astring
```

will generate an error, becuase "Phil" cannot be converted to a number.

The following table summarizes conversion between data types.

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.

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
```

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--that you can then call anywhere in the current application. In a Navigator application, you can use any functions defined in the current page. You use the **function** statement to define a function. 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.

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, {...}

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.

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:

```
car1 = new car("Eagle", "Talon TSi", 1993);
```

This statement creates *mycar* and assigns it the specified values for its properties. Then the value of `car1.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,

```
car2 = 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)
```

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
- eval function

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*.
- methods that return the string, with HTML formatting, such as *bold* and *link*.

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

More introductory and overview information TBD.

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 the eval function

The special built-in function *eval* takes an expression as its argument, evaluates the expression, and returns the value.

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 might 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>
```

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.

Authoring with JavaScript

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

Using JavaScript in HTML

There are two ways to embed JavaScript in an HTML document:

- with the SCRIPT tag. This is the way you define functions and perform statements within a page.
- as event handlers in HTML tags. Scripting event handlers is discussed in the next section.

SCRIPT is an HTML tag that enables you to include any JavaScript statements in a document. The tag, <SCRIPT>, and its closing counterpart, </SCRIPT> may enclose any number of JavaScript statements. It is a good idea to enclose the SCRIPT tag in HTML comments to hide the script from old browsers that do not recognize JavaScript. That way, your JavaScript code won't appear as a bunch of unformatted gobble-dygook to users with old browsers.

It is 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.

Generally, it is good practice to define the functions for a page in the HEAD portion of a document. Since the HEAD is loaded first, this practice guarantees that the functions will be loaded before the user has a chance to do anything that might call a function. For example:

```
<HEAD>
<SCRIPT>
<!--- hide script from old browsers
function bar() {
   document.write("<HR>")
}
function output(head, level, string) {
   document.write("<H" + level + ">" + head + "</H" + level + "><P>" + string)
}
// end hiding from old browsers -->
</SCRIPT>
</HEAD>
```

This part of a document HEAD defines two simple functions: *bar*, that displays an HTML horizontal rule, and *output*, that displays a string as an HTML paragraph with a heading of the specified level.

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

Tips and Techniques

This section describes various useful scripting techniques.

Caveats:

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.

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>
```

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 Functions with Optional Arguments

To process extra arguments beyond the ones declared by the formal parameter names in the function definition, you refer to a property of the function named "arguments", as if the function were an object:

```
function foo(x) {
    var argv = foo.arguments;

    print("foo.caller is " + foo.caller);
    print("foo.arguments.length is " + foo.arguments.length);
    print("formal x is " + foo.x);
    for (var i = 0; i < argv.length; i++)
        print("argument " + i + " is " + argv[i]);
}

function bar(x,y,z) {
    foo(x,y,z)
}
foo(1, "two", 3);
bar(1, "two", 3);
```

As this example shows, there is also a "caller" property. Both of these properties are non-null only within the function; if evaluated from some other context (top-level or another function), they're null.

The arguments property is like a real JavaScript array: it has a length property that tells how many actual arguments were passed. Actuals are indexed starting from 0 up to and including (arguments.length-1).

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") + "");
}
```

More information in this section TBD.

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
- button
- checkbox
- Date
- document
- form
- frame
- history
- link
- location
- Math
- navigator
- password
- radio
- reset
- select
- string
- submit
- text
- textarea
- window

NOTE: Each object topic indicates whether the object is part of the client (in Navigator), server (in LiveWire), or is common (built-in to JavaScript). Server objects are not included in this version of the documentation.

---

anchor object (client)

An anchor is a piece of text identified as the target of a hypertext link.

Syntax

To define an anchor, use standard HTML syntax:

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

NAME specifies a tag that becomes an available hypertext target within the current document.
HREF identifies a destination anchor or URL.
anchorText specifies the text to display at the anchor.

To use an anchor's properties and methods:

```
xxx to be supplied
```

Description

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

Properties

```
xxx to be supplied
```

Methods

```
xxx to be supplied
```

Event handlers

None.

Examples

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

See also

- link object
- anchors property

---

button object (client)

A button object is a pushbutton on an HTML form.

Syntax

To define a button:

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

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

To use a button's properties and methods:

1. *buttonName.propertyName*
2. *buttonName.methodName(parameters)*

buttonName is the value of the NAME attribute of a button object.
propertyName is one of the properties listed below.
methodName is one of the methods listed below.

Description

The button object is a custom button that you can use to perform an action you define.

Properties

- name
- value

Methods

- click

Event handlers

- onClick

Examples

A custom button does not necessarily load a new page into the client; it merely executes the script specified by the onClick event handler. In the following example, *myfunction()* is a JavaScript function.

```
<INPUT TYPE="button" VALUE="Calculate" NAME="calc_button"
  onClick="myfunction(this.form)">
```

See also

- form, reset, and submit objects

---

checkbox object (client)

A checkbox object is 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
```

To use a checkbox object's properties and methods:

1. *checkboxName.propertyName*
2. *checkboxName.methodName(parameters)*

checkboxName is the value of the NAME attribute of a checkbox object.
propertyName is one of the properties listed below.
methodName is one of the methods listed below.

Description

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
- defaultChecked
- name
- value

Methods

- click

Event handlers

- onClick

Examples

```
<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
```

See also

- form object

Date object (common)

The Date object 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 the value of the NAME attribute of a Date object.

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

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

Description

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.

JavaScript handles dates 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.

JavaScript does not have a date data type, but you can use the Date object and its methods to work with dates and times in your applications. The Date object has many methods for setting, getting, and manipulating dates.

Properties

None.

Methods

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

Event handlers

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

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)
```

See also

xxx to be supplied

---

document object (client)

The document object contains information on the current document.

Syntax

To define a document object, use standard HTML syntax with the addition of the onLoad and onUnLoad event handlers:

```
<BODY
   BACKGROUND="backgroundImage"
   BGCOLOR="#backgroundColor"
   FGCOLOR="#foregroundColor"
   LINK="#unfollowedLinkColor"
   ALINK="#activatedLinkColor"
   VLINK="#followedLinkColor"
   [onLoad="handlerText"]
   [onUnLoad="handlerText"]>
</BODY>
```

BGCOLOR, *FGCOLOR*, *LINK*, *ALINK*, and *VLINK* are color specifications expressed as a hexadecimal RGB triplet (in
the format "#rrggbb") or as one of the string literals listed in the Color Appendix.

To use the current document'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

The <BODY>...</BODY> tag encloses an entire 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>...</BODY> tag.

You can reference the anchors, forms, and links of a document by using the anchors, forms, and links properties. These properties are arrays that contain an entry for each anchor, form, or link in a document.

The document object's title property reflects the contents of <TITLE>...</TITLE>. Other properties reflect the contents of the document; for example, bgColor reflects the background color, and lastModified reflects the time last modified. Some of the properties are reflections from HTML attributes; for example, the links property is a reflection of all the links in the document, and the forms property is a reflection of all the forms in the document.

Properties

- alinkColor
- anchors
- bgColor
- cookie
- fgColor
- forms
- lastModified
- linkColor
- links
- location
- referrer
- title
- vlinkColor

Methods

- clear
- close
- open
- write
- writeln

Event handlers

None.

Examples

xxx to be supplied

See also

- frame and window objects

form object (client)

A form 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 or retrieve data from 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
   [onSubmit="handlerText"]>
</FORM>
```

TARGET specifies the window that form responses go to. When you submit a form with a TARGET attribute, instead of seeing the server's responses in the same window that contains the form, you see them in a (possibly) new window. The *windowName* may be an existing window created by previous targetted form submits or link clicks; it may also refer to named frames in a <FRAMESET> tag; it may also be _top, _parent, _self, or _blank.

ACTION secifies the URL of the server to which form field input information is sent.

METHOD 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*.

To use a form'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 corresponds to 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 elements property. The elements property is an array that contains an entry for each element (such as a checkbox, radio, or text object) in a form.

Properties

- action
- elements
- method
- name
- target

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

```
<FRAMESET
   ROWS="number"
   COLS="number">
   textToDisplay
   [<FRAME SRC="locationOrURL" NAME="frameName">]
</FRAMESET>
```

ROWS is an integer specifying the row-height of the frame. An optional suffix defines the units. Default units are pixels.
COLS is an integer 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 specifies the URL of the document to be displayed in the frame.
NAME specifies a name to be used as a target of hyperlinks.

To use a frame's properties and methods:

```
xxx to be supplied
```

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.

xxx to be supplied

Properties

xxx to be supplied

Methods

xxx to be supplied

Event handlers

None.

Examples

xxx to be supplied

See also

- document and window objects
- frames 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 specifies a tag that becomes an available hypertext target within the current document.
HREF identifies a destination anchor or URL.
TARGET specifies the window that the link is loaded into.
linkText is rendered as a hypertext link to the URL.

To use a link's properties and methods:

```
xxx to be supplied
```

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.

Properties

- target

Methods

xxx to be supplied

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>
```

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

xxx to be supplied

Properties

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

Methods

- assign
- toString

Event handlers

None.

Examples

xxx to be supplied

See also

- history object

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

See also

```
xxx to be supplied
```

---

navigator object (client)

xxx description to be supplied

Syntax

xxx to be supplied

Description

xxx to be supplied

Properties

- appName
- appVersion
- appCodeName
- userAgent

Methods

xxx to be supplied

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>
```

To use a password object's properties and methods:

1. *passwordName.propertyName*
2. *passwordName.methodName*(*parameters*)

passwordName is the value of the NAME attribute of a password object.
propertyName is one of the properties listed below.
methodName is one of the methods listed below.

Description

xxx to be supplied

Properties

- defaultValue
- name
- value

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 should contain the same value for all radio buttons in a group.

To use a radio button's properties and methods:

1. *radioName*[*index*].*propertyName*
2. *radioName*[*index*].*methodName*(*parameters*)

radioName is the value of the NAME attribute of a radio object.
index is an integer representing a radio button in a radio object.
propertyName is one of the properties listed below.
methodName is one of the methods listed below.

Description

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
- defaultChecked
- index
- length
- name
- value

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

- form and select objects

---

reset object (client)

A reset object is a reset button on an HTML form.

Syntax

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

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

VALUE 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)
```

resetName is the value of the NAME attribute of a reset object.
propertyName is one of the properties listed below.
methodName is one of the methods listed below.

Description

A reset button resets all elements in a form to their defaults.

Properties

- name
- value

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="value"]
   [MULTIPLE]
   [onBlur="handlerText"]
   [onChange="handlerText"]
   [onFocus="handlerText"]>
   <OPTION [SELECTED]> textToDisplay [ ... <OPTION> textToDisplay]
</SELECT>
```

SIZE specifies the number of options visible when the form is displayed.

To use a select object's properties and methods:

1. *selectName.propertyName*
2. *selectName.methodName*(*parameters*)

selectName is the value of the NAME attribute of a select object.
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:

```
selectName.options[index].propertyName
```

selectName is the value of the NAME attribute of a select object.
index is an integer representing an option in a select object.
propertyName is one of the properties listed below.

Description

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.

The options on select objects can be updated dynamically. xxx NYI.

Properties

The select object has the following properties:

- options
- selectedIndex

The options property has the following properties:

- defaultSelected
- index
- selected
- 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)
```

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
- fontsize
- indexOf
- italics
- lastIndexOf
- link
- sub
- substring
- sup
- toLowerCase
- toUpperCase

- fixed
- fontcolor
- small
- strike

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.

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"]>
```

VALUE specifies the text 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*)

submitName is the value of the NAME attribute of a submit object.
propertyName is one of the properties listed below.
methodName is one of the methods listed below.

Description

A submit button causes a form to be submitted.

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
- value

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"]>
```

To use a text object's properties and methods:

1. *textName.propertyName*
2. *textName.methodName*(*parameters*)

textName is the value of the NAME attribute of a text object.

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

Description

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

xxx to be supplied

Properties

- defaultValue
- name
- value

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>
```

textToDisplay allows only ASCII text, and new lines are respected.
ROWS and *COLS* define the physical size of the displayed input field in numbers of characters.

To use a textarea object's properties and methods:

1. *textareaName.propertyName*
2. *textareaName.methodName*(*parameters*)

textareaName is the value of the NAME attribute of a textarea object.
propertyName is one of the properties listed below.
methodName is one of the methods listed below.

Description

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

Properties

- defaultValue
- name
- value

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:

```
windowName = window.open()
```

windowName is the name of a new window.

To use a window's properties and methods:

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

windowName is the value of the NAME attribute of a window object.
propertyName is one of the properties listed below.
methodName is one of the methods listed below.

Description

The window object is the top-level object in the JavaScript client hierarchy. Because the existence of the current window is assumed, you don't 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.

Properties

- frames
- parent
- self
- top
- status
- defaultStatus
- 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
- 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 its argument.

Syntax

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

number is any numeric expression.

Applies to

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 its argument.

Syntax

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

number should be a numeric expression between -1 and 1. 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.

Applies to

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.

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.

Applies to

- window

Examples

In the following example, the testValue function checks the name entered by a user in the text element 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 of an application 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 element, 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.
nameAttribute is any string.

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 HTML A tag.

Applies to

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 its argument.

Syntax

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

number should be a numeric expression between -1 and 1. 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.

Applies to

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.

Applies to

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 the calling string object to be displayed in a big font by surrounding it with the HTML font tags `<BIG>` and `</BIG>`.

Syntax

```
stringName.big()
```

stringName is the name of any string variable.

Description

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

Applies to

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 the calling string object to blink by surrounding it with the HTML tags `<BLINK>` and `</BLINK>`.

Syntax

```
stringName.blink()
```

stringName is the name of any string variable.

Description

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

Applies to

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. textName.blur()
3. textareaName.blur()
```

passwordName is the value of the NAME attribute of a password object.
textName is the value of the NAME attribute of a text object.
textareaName is the value of the NAME attribute of a textarea object.

Description

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

Applies to

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 the calling string object to be displayed as bold by surrounding it with the HTML tags `<B>` and `</B>`.

Syntax

```
stringName.bold()
```

stringName is the name of any string variable.

Description

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

Applies to

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 its argument.

Syntax

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

number is any numeric expression.

Applies to

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 the name of any string variable or string object.
index is any integer from 0 to stringName.length() - 1.

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.

Applies to

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 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.

Applies to

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.

Applies to

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.click()
3. checkboxName.click()
```

buttonName is the value of the NAME attribute of a button, reset, or submit object.
radioName is the name of an element in a radio array.
checkboxName is the value of the NAME attribute of a checkbox object.

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.

Applies to

button, checkbox, radio, reset, submit

Examples

The following example toggles the selection status of the first element in the *musicType* radio group 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 close a layout stream without calling the close method.

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 window.

Syntax

```
window.close()
```

Description

The close method closes the current window.

Applies to

window

Examples

Any of the following examples close the current window:

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

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.

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 decison. The confirm method returns true if the user chooses OK and false if the user chooses Cancel.

Applies to

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 its argument.

Syntax

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

number is a numeric expression representing the size of an angle in radians. The cos method returns a numeric value between -1 and 1, which represents the cosine of the angle.

Applies to

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 its argument in the ISO Latin-1 character set.

Syntax

```
escape(char)
```

char is a non-alphanumeric character in the ISO Latin-1 character set.

Description

The escape function is a built-in JavaScript function. It is not a method associated with any object, but is part of the language itself. The value it returns is a string of the form "%xx", where *xx* is the ASCII encoding of the argument.

Examples

The following returns "%26"

```
escape("&")
```

See also

- unEscape function

---

eval function

The eval function takes a JavaScript arithmetic expression as its argument and returns the value of the argument as a number.

Syntax

```
eval(expression)
```

expression is any expression or sequence of statements.

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.

Example

In the following example, both uses of eval assign the value 42 to the variable result.

```
x = 6
result = eval((3+3)*7)
result = eval(x*7)
```

---

exp method

Returns *e* to the power of its argument, i.e. e^x, where *x* is the argument, and *e* is Euler's constant, the base of the natural logarithms.

Syntax

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

number is any numeric expression.

Applies to

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 the calling string object to be displayed in fixed-pitch font by surrounding it with the HTML tags `<TT>` and `</TT>`.

Syntax

```
stringName.fixed()
```

stringName is the name of any string variable.

Description

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

Applies to

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 its argument.

Syntax

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

number is any numeric expression.

Applies to

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. textName.focus()
3. textareaName.focus()
```

passwordName is the value of the NAME attribute of a password object.
textName is the value of the NAME attribute of a text object.
textareaName is the value of the NAME attribute of a textarea object.

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.

Applies to

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 focus method returns focus to the password field 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 the calling string object to be displayed in the specified color by surrounding it with the HTML tags `<FONT COLOR=color>` and `</FONT>`.

Syntax

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

stringName is the name of any string variable.
color is a string expressing the color as a hexadecimal RGB triplet or as one of the string literals listed in the Color Appendix.

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".

Applies to

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 the calling string object to be displayed in the specified font size by surrounding it with the HTML font size tags `<FONTSIZE=size> ... </FONTSIZE>`.

Syntax

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

stringName is the name of any string variable.
size is an integer between one and seven, or a string representing a signed integer between 1 and 7.

Description

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

Applies to

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)`.

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

- back, go methods

---

getDate method

Returns the day of the month for a date object.

Syntax

```
dateObjectName.getDate()
```

dateObjectName is the name of a date object.

Description

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

Applies to

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 a date object.

Syntax

```
dateObjectName.getDay()
```

dateObjectName is the name of a date 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.

Applies to

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 a date object.

Syntax

```
dateObjectName.getHours()
```

dateObjectName is the name of a date object.

Description

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

Applies to

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 a date object.

Syntax

```
dateObjectName.getMinutes()
```

dateObjectName is the name of a date object.

Description

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

Applies to

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 a date object.

Syntax

```
dateObjectName.getMonth()
```

dateObjectName is the name of a date object.

Description

The value returned by getMonth is an integer between zero and eleven. Zero corresponds to January, one to Februrary, 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 the name of a date object.

Description

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

Applies to

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 a date object.

Syntax

```
dateObjectName.getTime()
```

dateObjectName is the name of a date object.

Description

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

Applies to

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 the name of a date object.

Description

The time zone offset is the difference between local time and GMT. This value would be a constant except for daylight savings time.

Applies to

Date

Examples

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

---

getYear method

Returns the year in the date object.

Syntax

```
dateObjectName.getYear()
```

dateObjectName is the name of a date object.

Description

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

Applies to

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 in the history list.

Syntax

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

delta is an integer representing a relative position in the history list.
location is a string representing all or part of a URL in the history list.

Description

The go method navigates to the location in the history list determined by the argument that you specify.

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.

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

Applies to

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 character, starting the search at fromIndex.

Syntax

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

stringName is the name of any string variable or string object.
character is a string representing the character to search for.
fromIndex is the location within the calling string to start the search from, any integer from 0 to *stringName*.length() - 1.

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.

Applies to

string

Examples

The following example uses indexOf and lastIndexOf to locate a character 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.indexOf("w"))
```

See also

- charAt, lastIndexOf methods

---

italics method

Causes the calling string object to be italicized by surrounding it with the HTML tags <I> and </I>.

Syntax

```
stringName.italics()
```

stringName is the name of any string variable.

Description

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

Applies to

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 character. The calling string is searched backwards, starting at *fromIndex*.

Syntax

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

stringName is the name of any string variable or string object.
character is a string representing the character to search for.
fromIndex is the location within the calling string to start the search from, any integer from 0 to *stringName*.length() - 1.

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.

Applies to

string

Examples

The following example uses indexOf and lastIndexOf to locate a character 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.indexOf("w"))
```

See also

- charAt, indexOf methods

---

link method

Creates an HTML hypertext link that jumps to another URL.

Syntax

```
linkName.link(hrefAttribute)
```

linkName is any string.
hrefAttribute is any string.

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.

Applies to

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 its argument.

Syntax

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

number is any positive numeric expression. If the value of number is outside the suggested range, the return value is always -1.797693134862316e+308.

Applies to

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 its two arguments.

Syntax

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

number1 and *number2* are any numeric arguments.

Applies to

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 its two arguments.

Syntax

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

number1 and *number2* are any numeric arguments.

Applies to

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/xbm
x-world/plugIn
```

plugIn is any 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/xbm specifies a document with encoded bytes constituting a XBM header and pixel data.

- x-world/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 Communi cations.

Applies to

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();
}
```

See also

- clear, close, write, writeln methods

---

open method (window object)

Opens a new web browser window.

Syntax

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

URL specifies the URL to open in the new window.
windowName specifies a name for the window object being opened.
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.

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

Applies to

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*. It opens it with the specified option settings and names the corresponding window object newWin.

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

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.

Applies to

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 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 zero.

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 0:

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

See also

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

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 zero:

```
parseInt("Hello", 8)
parseInt("0x7", 10)
parseInt("FFF", 10)
```

See also

- parseFloat function

---

pow method

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

Syntax

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

base is any numeric expression.
exponent is any numeric expression.

Applies to

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 a string that is displayed as the message.
inputDefault is a string or integer 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>.

Applies to

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 X-platforms only.

Syntax

```
Math.random()
```

Applies to

Math

Examples

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

---

round method

Returns the value of the argument rounded to the nearest integer. If the decimal portion of the argument is .5 or greater, the argument is rounded to the next highest integer. If the decimal portion of the argument is less than .5, the argument is rounded to the next lowest integer.

Syntax

```
round(number)
```

number is any numeric expression.

Applies to

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 object.

Syntax

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

passwordName is the value of the NAME attribute of a password object.
textName is the value of the NAME attribute of a text object.
textareaName is the value of the NAME attribute of a textarea object.

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.

Applies to

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 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 date object.

Syntax

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

dateObjectName is the name of a date object.
dayValue is an integer from 1 to 31 representing the day of the month.

Applies to

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 in the current time.

Syntax

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

dateObjectName is the name of a date object.
hoursValue is an integer between 0 and 23 representing the hour.

Applies to

Date

Examples

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

See also

- getHours method

---

setMinutes method

Sets the minutes in the current time.

Syntax

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

dateObjectName is the name of a date object.
minutesValue is an integer between 0 and 59 representing the minutes.

Applies to

Date

Examples

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

See also

- getMinutes method

---

setMonth method

Sets the month in the current date.

Syntax

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

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

Applies to

Date

Examples

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

See also

- getMonth method

---

setSeconds method

Sets the seconds in the current time.

Syntax

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

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

Applies to

Date

Examples

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

See also

- getSeconds method

---

setTime method

Sets the value of a date object.

Syntax

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

dateObjectName is the name of a date object.
timevalue is an integer representing the number of milliseconds since the epoch (1 January 1970 00:00:00).

Description

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

Applies to

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.
msec is a numeric value or numeric string in millisecond units.

Description

The setTimeout method evaluates an expression after a specified amount of time. It does not evaluate the expression repeatedly. For example, if a setTimeout method specifies 5 seconds, the expression is evaluated after 5 seconds, not every 5 seconds.

Applies to

window

Examples

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>
```

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(){
    // cannot directly test timerID on DEC OSF/1 in beta 4.
    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 in the current date.

Syntax

```
dateObjectName.setYear(yearValue)
```

dateObjectName is the name of a date object.
yearValue is an integer greater than 1900.

Applies to

Date

Examples

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

See also

- getYear method

---

sin method

Returns the sine of its argument.

Syntax

```
Math.sin(number)
```

number is a numeric expression 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.

Applies to

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 the calling string object to be displayed in a small font by surrounding it with the HTML font tags `<SMALL>...</SMALL>`.

Syntax

```
stringName.small()
```

stringName is the name of any string variable.

Description

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

Applies to

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 its argument.

Syntax

```
Math.sqrt(number)
```

number is any non-negative numeric expression. If the value of number is outside the suggested range, the

return value is always 0.

Applies to

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 the calling string object to be displayed as struck out text by surrounding it with the HTML tags `<STRIKE>` and `</STRIKE>`.

Syntax

```
stringName.strike()
```

stringName is the name of any string variable.

Description

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

Applies to

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 the calling string object to be displayed as a subscript by surrounding it with the HTML tags _{and}.

Syntax

```
stringName.sub()
```

stringName is the name of any string variable.

Description

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

Applies to

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.

Applies to

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

- submit object

---

substring method

The substring method returns a subset of a string object.

Syntax

```
stringName.substring(indexA, indexB)
```

stringName is the name of any string variable or string object.
indexA is any integer from 0 to stringName.length() - 1.
indexB is any integer from 0 to stringName.length() - 1.

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.

Applies to

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 the calling string object to be displayed as a superscript by surrounding it with the HTML tags `<SUP>` and `</SUP>`.

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.

Applies to

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 its argument.

Syntax

```
Math.tan(number)
```

number is a numeric expression representing the size of an angle in radians. The tan method returns a numeric which represents the tangent of the angle.

Applies to

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 the name of a date object.

Applies to

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 in the following form:

Mon, 18 Dec 1995 17:28:35 GMT

See also

- toLocaleString method

---

toLocaleString method

Converts a date to a string, using the locale conventions.

Syntax

```
dateObjectName.toLocaleString()
```

dateObjectName is the name of a date object.

Applies to

Date

Examples

In the following example, today is a date object:

```
today.toLocaleString()
```

In this example, toLocaleString returns a string value in the following form:

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

See also

- toGMTString method

toLowerCase method

Converts the calling string to lower case.

Syntax

```
stringName.toLowerCase()
```

stringName is the name of any string variable or string object.

Applies to

string

Examples

The following examples both yield "alphabet".

```
var upperText="ALPHABET"
document.write(upperText.toLowerCase())

"ALPHABET".toLowerCase
```

See also

- toUpperCase method

---

toString method

Converts the value of a Date object or the current location object to a string.

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.

Applies to

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

Converts the calling string to upper case.

Syntax

```
stringName.toUpperCase()
```

stringName is the name of any string variable or string object.

Applies to

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 of the form "%xx", where xx is a number between 0 and 255 (decimal) or 0x0 and 0xFF (hexadecimal).

Description

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

The string it returns is a non-alphanumeric character in the ISO Latin-1 character set.

Examples

The following returns "&"

```
unescape("%26")
```

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.

Applies to

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.

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.

Applies to

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.

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.

Applies to

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
- appName
- appVersion
- appCodeName
- bgColor
- checked
- cookie
- defaultChecked
- defaultSelected
- defaultStatus
- defaultValue
- E
- 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 cannot set this property after the Navigator has laid out the HTML source.

Applies to

form

Examples

The following example sets the action property to the value of the variable *urlName*:

```
forms[0].action=urlName
```

See also

- method, target properties

alinkColor property

xxx

Syntax

xxx

Description

The color of an active link (after mouse-button down, but before mouse-button up), 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.

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

---

appName property

xxx

Syntax

xxx

Description

xxx Description to be supplied

Applies to

navigator

Examples

xxx Examples to be supplied.

See also

- appVersion, appCodeName, userAgent properties

---

appVersion property

xxx

Syntax

xxx

Description

xxx Description to be supplied

Applies to

navigator

Examples

xxx Examples to be supplied.

See also

- appName, appCodeName, userAgent properties

---

appCodeName property

xxx

Syntax

xxx

Description

xxx Description to be supplied

Applies to

navigator

Examples

xxx Examples to be supplied.

See also

- appName, appVersion, userAgent properties

---

bgColor property

xxx

Syntax

xxx

Description

The color of the document background, 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.

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

xxx

Syntax

xxx

Description

For checkbox, Boolean, false if not checked, true if checked. For radio, Boolean, false if not pressed, true if pressed.

Applies to

checkbox, radio

Examples

xxx To be supplied.

See also

- defaultChecked property

---

cookie property

xxx

Syntax

xxx

Description

String value of a cookie, which is a small piece of information stored by the Navigator in the cookies.txt file. Use string methods such as substring, charAt, indexOf, and lastIndexOf to determine the value stored in the cookie. See the Netscape cookie specification for a complete specification of the cookie syntax.

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.

---

defaultChecked property

xxx

Syntax

xxx

Description

For checkbox, Boolean property that indicates if the element is selected by default, by the CHECKED attribute. For radio, Boolean property that indicates if the element is selected by default, by the CHECKED attribute.

Applies to

checkbox, radio

Examples

xxx Examples to be supplied.

See also

checked property

---

defaultSelected property

xxx

Syntax

xxx

Description

Boolean property that indicates if the option is selected by default, by the presence of the SELECTED attribute in the HTML OPTION tag.

Applies to

select

Examples

xxx Examples to be supplied.

See also

- selected property

---

defaultStatus property

xxx

Syntax

xxx

Description

For a window, the defaultStatus property reflects the default message displayed in the status bar at the bottom of the window. Do not confuse defaultStatus with status. 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.

Applies to

window

Examples

xxx Examples to be supplied.

See also

- status property

---

defaultValue property

xxx

Syntax

xxx

Description

For password, text, and textarea, string, the initial contents of the field.

Applies to

password, text, textarea

Examples

xxx Examples to be supplied.

See also

- value property

E property

xxx

Syntax

xxx

Description

E is Euler's constant, the base of natural logarithms, roughly 2.718.

Applies to

Math

Examples

xxx Examples to be supplied.

See also

- LN2, LN10, PI, SQRT1_2, SQRT2 properties

---

elements property

xxx

Syntax

xxx

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.

---

fgColor property

xxx

Syntax

xxx

Description

The color of the document foreground text, 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.

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

xxx

Syntax

xxx

Description

The anchor name following the # symbol.

Applies to

location

Examples

xxx Examples to be supplied.

See also

- host, hostname, href, pathname, port, protocol, search properties

---

host property

xxx

Syntax

xxx

Description

The hostname:port part of the location or URL.

Applies to

location

Examples

xxx Examples to be supplied.

See also

- hash, hostname, href, pathname, port, protocol, search properties

---

hostname property

xxx

Syntax

xxx

Description

The hostname part of the location or URL.

Applies to

location

Examples

xxx Examples to be supplied.

See also

- hash, host, href, pathname, port, protocol, search properties

---

href property

xxx

Syntax

xxx

Description

The entire URL as a JavaScript string.

Applies to

location

Examples

xxx Examples to be supplied.

See also

- hash, host, hostname, pathname, port, protocol, search properties

---

index property

xxx

Syntax

xxx

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, select

Examples

xxx Examples to be supplied.

See also

For select:

- selectedIndex property

---

lastModified property

xxx

Syntax

xxx

Description

A string containing the last-modified date.

Applies to

document

Examples

xxx Examples to be supplied.

---

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 null string, length is zero.

Applies to
history, radio, 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

xxx

Syntax

xxx

Description

The color of the document hyperlinks, expressed as a hexadecimal RGB triplet or as one of the string literals listed in the Color Appendix.

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

xxx

Syntax

xxx

Description

LN2 is the natural logarithm of two, roughly 0.693.

Applies to

Math

Examples

xxx Examples to be supplied.

See also

- E, LN10, PI, SQRT1_2, SQRT2 properties

LN10 property

xxx

Syntax

xxx

Description

LN10 is the natural logarithm of ten, roughly 2.302.

Applies to

Math

Examples

xxx Examples to be supplied.

See also

- E, LN2, PI, SQRT1_2, SQRT2 properties

---

location property

xxx

Syntax

xxx

Description

The full URL of the document.

Applies to

document

Examples

xxx Examples to be supplied.

---

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 Navigator has laid out the HTML source. 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, form, password, radio, reset, submit, text, textarea

Examples

xxx Examples to be supplied.

See also

For form:
- action, elements, method, target properties

For button:
- 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: `objectName.options.length`.

Applies to

select

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

xxx

Syntax

xxx

Description

The file or object path name following the third slash.

Applies to

location

Examples

xxx Examples to be supplied.

See also

- hash, host, hostname, href, port, protocol, search properties

---

PI property

xxx

Syntax

xxx

Description

Pi is the ratio of the circumference of a circle to its diameter, roughly 3.1415.

Applies to

Math

Examples

xxx Examples to be supplied.

See also

- E, LN2, LN10, SQRT1_2, SQRT2 properties

---

port property

xxx

Syntax

xxx

Description

The port number in a URL, if any; otherwise "".

Applies to

location

Examples

xxx Examples to be supplied.

See also

- hash, host, hostname, href, pathname, protocol, search properties

---

protocol property

xxx

Syntax

xxx

Description

The initial substring up to and including the first colon, which indicates the URL's access method.

Applies to

location

Examples

xxx Examples to be supplied.

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

xxx

Syntax

xxx

Description

Any query string or form data after ?.

Applies to

location

Examples

xxx Examples to be supplied.

See also

- hash, host, hostname, href, pathname, port, protocol properties

---

selected property

xxx

Syntax

xxx

Description

Boolean property that indicates the current selected state of an option in a select object.

Applies to

select

Examples

xxx Examples to be supplied.

See also

- defaultSelected property

---

selectedIndex property

xxx

Syntax

xxx

Description

xxx to be described

Applies to

select

Examples

xxx Examples to be supplied.

See also

- index property

---

self property

xxx

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 disambiguate 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

xxx

Syntax

xxx

Description

SQRT1_2 is the square root of one-half; equivalently, one over the square root of two, roughly 0.707.

Applies to

Math

Examples

xxx Examples to be supplied.

See also

- E, LN2, LN10, PI, SQRT2 properties

SQRT2 property

xxx

Syntax

xxx

Description

SQRT2 is the square root of two, roughly 1.414.

Applies to

Math

Examples

xxx Examples to be supplied.

See also

- E, LN2, LN10, PI, SQRT1_2 properties

---

status property

xxx

Syntax

xxx

Description

For a window, the status property reflects 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. Do not confuse status with defaultStatus. The defaultStatus property reflects the default message displayed in the status bar.

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

A string specifying the name of the window that responses go to after a form has been submitted.

Syntax

```
formName.target
```

formName is the name of any form or an element in the forms array.

Description

The target property is a reflection of the TARGET attribute of the HTML FORM tag. You cannot set this property after the Navigator has laid out the HTML source.

Applies to

form, link

Examples

The following example specifies that responses to the musicInfo form are displayed in the "msgWindow" window:

```
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

xxx

Syntax

xxx

Description

Current document title.

Applies to

document

Examples

xxx Examples to be supplied.

---

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

xxx

Syntax

xxx

Description

xxx Description to be supplied

Applies to

navigator

Examples

xxx Examples to be supplied.

See also

- appName, appVersion, appCodeName properties

---

value property

xxx

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, password, radio, reset, select, submit, text, textarea

Examples

xxx Examples to be supplied.

See also

For password, text, and textarea:
- defaultValue property

---

vlinkColor property

xxx

Syntax

xxx

Description

The color of visited links, 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.

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 disambiguate 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.

Applies to

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.

Applies to

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.

Applies to

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 an anchor to dynamically specify a value for the HREF attribute of the anchor, 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 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.

Applies to

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="...">`.

Applies to

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 when the mouse pointer is over an object. The onMouseOver event handler executes JavaScript code when a mouseOver event occurs.

You must return true if you want to set the window.status property with the onMouseOver event handler.

See the relevant objects for the onMouseOver syntax.

Applies to

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 anchor 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 must return true in the event handler to allow the form to be submitted; return false to prevent the form from being submitted.

See the relevant objects for the onSubmit syntax.

Applies to

form

Examples

In the following example, the onSubmit event handler evaluates the data being submitted to test if it is legal. If the data is legal, the form is submitted; otherwise, the form is not submitted.

```
form.onSubmit=
"if badFormData(this.form) {
    return false;
} else {
    return true;
}"
```

---

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 onLoad event handler within either the <BODY> or the <FRAMESET> tag, for example, `<BODY onLoad="...">`.

Applies to

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 semicolon.

Syntax conventions: All keywords in syntax statements are in bold. Words in italics represent user-defined names or statements. Any portions enclosed in square brackets, i.e. [and], 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
- return
- var
- while
- with

---

break statement

The **break** statement 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 statement

Comments are notations by the author to explain what the script does, and they 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 statement

The **continue** statement 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, it 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 statement

A **for** loop 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 option

ally declare new variables with the `var` keyword. This expression is optional.
- The *condition* that is 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 four spaces from the beginning of the for statement.

Syntax

```
for ([initial expression]; [condition]; [update expression]) {
    statements
}
initial expression = statement | variable declartion
```

Examples

This simple 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 statement

The **for** statement iterates 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 = "", i = "";
    for (i in obj)
        result += obj_name + "." + i + " = " + obj[i] + "\n";
    return result;
}
```

function statement

The **function** statement declares a JavaScript function *name* with the specified parameters *param.* 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 statement

The **if...else** statement is 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;
```

---

return statement

The **return** 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;
}
```

---

var statement

The **var** statement declares a variable *varname*, optionally initializing it to have *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 neccessary in functions if there is a global variable of the same name. So, in general, it is a good idea to always use **var**, but you should definitely use it when declaring a local variable in a function, to ensure that any global variable of the same name does not override it.

Syntax

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

Examples

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

---

while statement

The **while** statement is 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 next statement following the *statements*.

Although not required, it is good practice to indent the statements a **while** loop four spaces from the beginning of the for statement.

Syntax

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

Examples

The following simple **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 first pass: x = 1 and n = 1
- After second pass: x = 2 and n = 3
- After 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 statement

The **with** statement 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