Expressions and Operators

JavaScript expressions are formed by combining literal values and variables with JavaScript operators. Parentheses can be used in an expression to group subexpressions and alter the default order of evaluation of the expression. For example:

```
l+2
total /n
sum(o.x, a[3]) + +
(1+2)*3
```

JavaScript defines a complete set of operators, most of which should be familiar to all C, C++, and Java programmers. In the following table, the P column specifies operator precedence and the A column specifies operator associativity:

L means left-to-right associativity, and R means right-to-left associativity.

P	**A**	**operator**	**operation performed**
15	L	.	access an object property
	L	[]	access an array element
	L	()	invoke a function
14	R	++	unary pre- or post-increment
	R	- -	unary pre- or post-decrement
	R	-	unary minus (negation)
	R	~	numeric bitwise complement
	R	!	unary boolean complement
	R	delete	undefine a property
	R	new	create a new object
	R	typeof	return type of operand
	R	void	return undefined value
13	L	*, /, %	multiplication, division, modulo
12	L	+, -	addition, subtraction

	L	+	string concatenation
11	L	<<	integer shift left
	L	>>	shift right, sign extension
	L	>>>	shift right, zero extension
10	L	<, <=	less than, less than or equal
	L	>, >=	greater than, greater than or equal
9	L	==, !=	test for equality or inequality
	L	===, !==	test for identity or non-identity
8	L	&	integer bitwise AND
7	L	^	integer bitwise XOR
6	L	|	integer bitwise OR
5	L	&&	logical AND, evaluate 2nd operand only if 1st is true
4	L	||	logical OR, evaluate 2nd operand only if 1st is false
3	L	?:	conditional: *if ? then : else*
2	R	=	assignment
	R	*=, +=, -=, etc.	assignment with operation
1	L	,	multiple evaluation

Statements

A JavaScript program is a sequence of JavaScript statements. Most JavaScript statements have the same syntax as the corresponding C, C++, and Java statements:

Expression statements

Every JavaScript expression can stand alone as a statement. Assignments, method calls, increments, and decrements are expression statements. For example:

```
s = "hello world";
x = Math.sqrt(4);
```

X++

Compound statements

When a sequence of JavaScript statements is enclosed within curly braces, it counts as a single compound statement. For example, the body of a *while* loop consists of a single statement. If you want the loop to execute more than one statement, use a compound statement. This is a common technique with *if, for*, end other statements described later.

Empty statements

The empty statement is simply a semicolon by itself. It does nothing, and is occasionally useful for coding empty loop bodies.

Labeled statements

In JavaScript 1.2, any statement can be labeled with a name. Labeled loops can then be used with the labeled versions of the *break* and *continue* statements:

label : statement

break

The *break* statement terminates execution of the innermost enclosing loop, or, in JavaScript 1. 2, the named loop:

```
break ;
break     label ;        //  JavaScript 1.2
```

case

Case is not a true statement. Instead it is a keyword used to label statements within a JavaScript 1.2 *switch* statement:

```
case constant-expression:
statements
[break ;]
```

Because of the nature of the *switch* statement, a group of statements labeled by *case* should usually end with a *break* statement.

continue

The *continue* statement restarts the innermost enclosing loop, or, in JavaScript 1.2, restarts the named loop:

```
continue ;
```

continue label ; *//JavaScript 1.2*

default

like case , default is not a true statement, but instead a label that may appear within a JavaScript 1. 2 s w i t c h statement:

defau1t :
  statements
  [break ;]

do/while

The *do/whi1e* loop repeatedly executes a statement while an expression is *true* . It is like the *whi1e* loop, except that the loop condition appears (and is tested) at the bottom of the loop. This means that the body of the loop will be executed at least once:

do
statement
while (expression) ;

This statement is new in JavaScript 1.2. In Navigator 4, the *continue* statement does not work correctly within *do / while* loops.

export

The e x p o r t statement was introduced in Navigator 4. It intakes the specified functions and properties accessible to other windows or execution contexts:

export *expression L, expression* [, *expression* …] ;

for

The f o r statement is an easy-to-use loop that combines the initialization and increment expressions with the loop condition expression:

for *(initialize ; test ; increment)*
statement

The *for* loop repeatedly executes a statement as long is its test expression is t r u e . It evaluates the initialization expression once before starting the loop and evaluates the increment expression at the end of each iteration.

for/in

The f o r/ i n statement loops through the properties of a specified object:

for *(variable in object)*
statement
The f o r/ i n loop executes a statement once for each property of an object. Each nine through the loop, it assigns the name of the current property to the specified variable. Some properties of pre-defined JavaScript objects are not enumerated by the f o r / i n loop. Userdefined properties are always enumerated.

function

The f u n c t i o n statement defines a function in a JavaScript program:

function funcname(args) { *statements* }

This statement defines a function named *funcname,* with a body that consists of the specified statement, and arguments as specified by *args*. *args* is a comma-separated list of zero or more argument names. These arguments can be used in the body of the function to refer to the parameter values passed to the function.

if /else

The i f statement executes a statement if an expression is t r u e:

```
if  (expression)
   statement
```

When an e l se clause is added, the statement executes a different statement if the expression is f a 1 s e:

```
if ( expression)
   statement
e l se
   statements2
```

Any e l se clause may be combined with a nested i f / e l s e statement to produce an e l s e / i f statement:

```
if ( expression)
   statement
else if ( expressions2)
   statements2
e l se
   statements3
```

import

The i m p o r t statement was introduced in Navigator 4 along with export. It makes the named functions and variables available in the current window or execution context, or, in the second form of the statement, makes all properties and methods of the specified object available within the current context:

import *expression [, expression] ;*
import *expression.* ;*

return

The r e t u r n statement causes the currently executing function to stop executing and return to its caller. If followed by an expression, the value of that expression is used as the function return value.

return ;
return *expression* ;

switch

The s w i t c h statement is a multi-way branch. it evaluates an expression and then jumps to a statement that is labeled with a case clause that matches the value of the expression. If no matching case label is found, the s w i t c h statement jumps to the statement, if any, labeled with d e f a u l t:

```
switch ( expression) {
      case constant-expression: statements
    [ case constant-expression: statements]
    […]

    default: statements
```

var

The v a r statement declares and optionally initializes one or more variables. Variable declaration is optional in toplevel code, but is required to declare local variables within function bodies:

var name [=*value*] [, *name2* [*value2*] ...] ;

while

The w h i I e statement is a basic loop. it repeatedly executes a statement while an expression is true:

while *(expression)*
    statement ;

with

The w i t h statement adds an object to the scope chain, so that a statement is interpreted in the context of the object:

with *(object)*
    statement ;

The use of w i t h statements is discouraged.

Regular Expressions

JavaScript 1.2 supports regular expressions, using the same syntax is Perl 4. A regular expression is specified literally as a sequence of characters within forward slashes (/), or as a JavaScript string passed to the *RegExp*( ) constructor. The optional *g* (global search) and *i* (case-insensitive search) modifiers may follow the second / character, or may be passed to *RegExp*( ). The following table summarizes regular expression syntax:

Character	Meaning
\n,\r,\t	Match literal newline, carriage return, tab
\\, \ /, \ *, \+, \ ?, etc.	Match a special character literally, ignoring or escaping its special meaning
[…]	Match any one character between brackets
[^…]	Match any one character not between brackets
.	Match any character other than newline
\w, \W	Match any word/non-word character
\s, \S	Match any whitespace/non-whitespace
\d, \D	Match any digit/non-digit
^, $	Require match at beginning/end of a string, or in multi-lane mode, beginning/ end of a line
\b,\B	Require match at a word boundary non- boundary

?	Optional term; Match zero or one time
+	Match previous term one or more times
*	Match term zero or more times
{n}	Match previous term exactly n times
{n, }	Match previous term *n* or more times
{n,m}	Match at least *n* but no more than *m* times
a I b	Match either a *or b*
(sub)	Group sub-expression sub into a single term, and remember the text that it matched
\n	Match exactly the same characters that were matched by sub-expression number n
$n	In replacement strings, substitute the text that matched the nth sub-expression

JavaScript in HTML

Client-side JavaScript code may be embedded in HTML files in several ways:

<SCRIPT> tag

Most JavaScript code appears in HTML files between a <SCRIPT> tag and a </SCRIPT> tag. The <SCRIPT> tag can also be used to include an external file of JavaScript code into an HTML document. The <SCRIPT> tag supports a number of attributes, including these three important ones:

LANGUAGE

Specifies the scripting language in which the script is written. In most browsers, this attribute defaults to "*JavaScript*". You must set it if you are mixing scripting languages, such as IavaScript and VBScript.

Set this attribute to "JavaScript1.l" to specify that the code uses JavaScript 1.1 features, and that it should not be interpreted by JavaScript 1.0 browsers. Set this attribute to "JavaScript l.2" to specify that only Java Script 1.2 browsers should interpret the code. (Note, however, that Navigator 4 has some non-standard behaviors when "JavaScript 1.2" is specified.)

SRC

Specifies the URL, of an external script to be loaclecl and executed. Files of JavaScript code typically have a *.js* extension. Note that the </SCRIPT> tag is still required when this attribute is used. Supported in Iavascript 1.1 and later.

ARCHIVE

Specifies the URL of a JAR file that contains the script specified by the SRC attribute. Supported in JavaScript 1.2 and later. Archives are required to use Navigator 4 signed scripts.

Event handlers

JavaScript code may also appear as the Value of event handler attributes of HTML tags. Event handler attributes alwavs begin with "on". The code specified by one of these attributes is executed when the named event occurs. For example, the following HTML specifies a button that displays a clialog box when clicked:

```
<INPUT TYPE=button VALUE="Press Me"
        onClick="alert('hello world!');">
```

JavaScript URLs

JavaScript code may appear in a URL that uses the special *javascript*: pseudo-protocol. The JavaScript code is evaluated, and the resulting value (converted to a string, if necessary) is used as the contents of the URL. Use the *void* operator if you want a javascript URL, that executes JavaScript statements without overwriting the Current document:

```
<FORM ACTION="JavaScript:void validates">
```

JavaScript entities

In JavaScript 1.1, HTML attribute values may contain a javascript code in the form of javascript entities. An HTML entity is a string like & l t ; that represents some other character or string. A JavaScript entity is javascript code contained within &{ and } ;. Its value is the value of the JavaScript expression within:

```
<BODY BGCOLOR="& { getFavoriteColor() };">
```

Forms

One of the powerful features of JavaScript is its ability to manipulate HTML forms. HTML defines the following form elements:

Button (<INPUT TYPE=button>)
A graphical Push button; *o n C 1 i c k* events

Checkbox (<INPUT TYPE=checkbox>)
A toggle button without mutually-exclusive behavior;
o n C l i c k events

FileUpload (<INPUT TYPE=file>)
A file entry field and file browser; *onchange* events

Hidden (<INPUT TYPE=hidden>)
A non-visual datafield; no event handlers

Option (<OPTION>)
An item within a Select list; event handlers are on the Select object, not Option objects

Password (<INPUT TYPE=Password>)
An input field for sensitive data; *onChange* events

Radio (<INPUT TYPE=radio>)
A toggle button with mutually-exclusive "radio" behavior;
o n C l i c k events

Reset (<INPUT TYPF=reset>)
A button that resets a form; *o n C 1 i c k* events

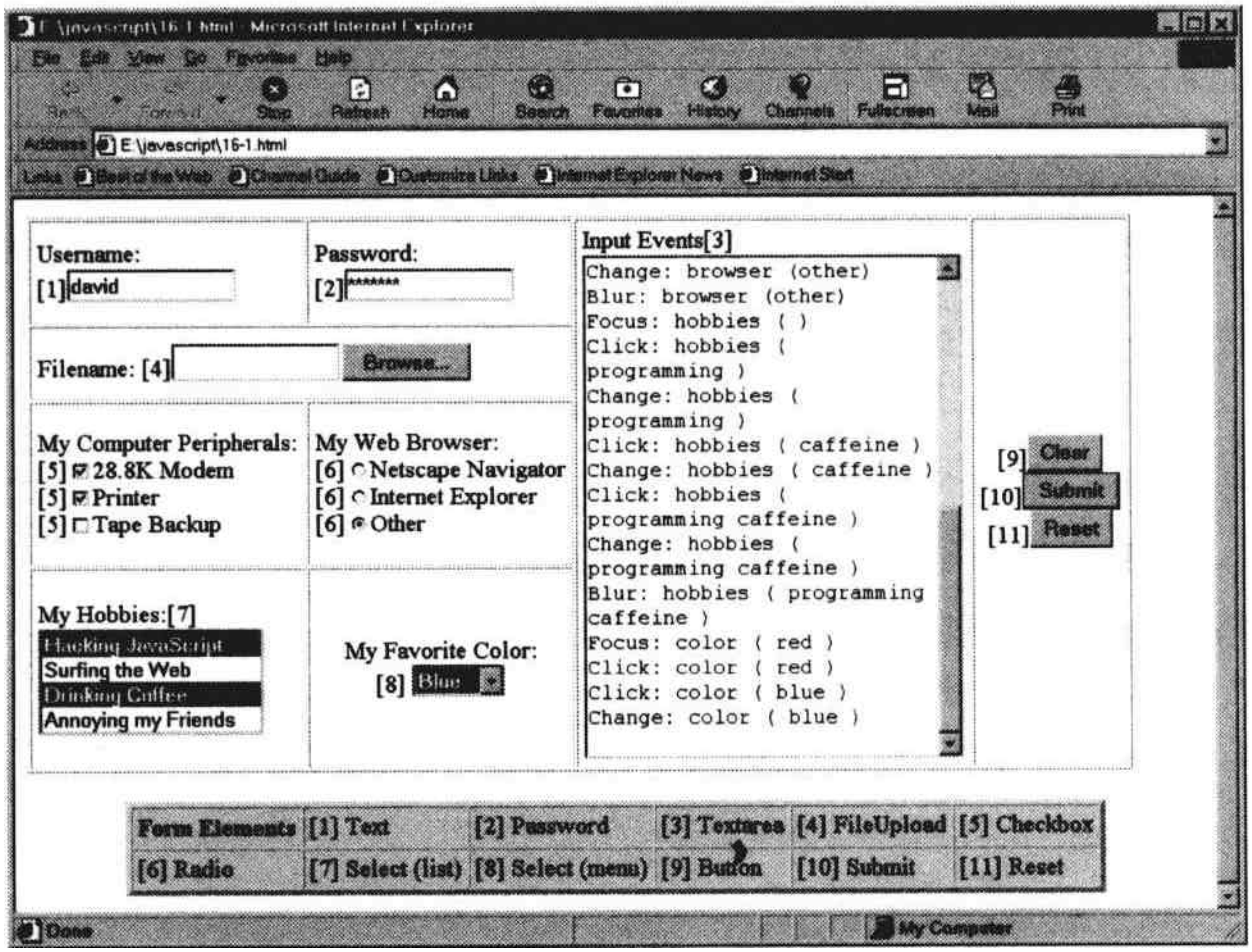
Select (< SELECT [MULTIPLE] > . . . </SELECT>)
A list or drop-down menu from which one or more Option items may be selected; *onChange* events

Submit (<INPUT TYPE=submit>)
A button that submits a form; *o n C 1 i c k* events

Text (<INPUT TYPE=text>)
A single-line text entry field; *o n C h a n g e* events

TextArea (<TEXTAREA> . . . </TEXTAREA>)
A multi-line text entry fields; *onChange* events

A web page containing each type of form element:


E:\javascript\16-1.html - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites History Channels Fullscreen Mail Print
Address E:\javascript\16-1.html
Links Best of the Web Channel Guide Customize Links Internet Explorer News Internet Start
Browse...
Clear
Submit
Reset
Done
My Computer

Username: [1] david

Password: [2] *******

Filename: [4]

My Computer Peripherals:
[5] ☑ 28.8K Modem
[5] ☑ Printer
[5] ☐ Tape Backup

My Web Browser:
[6] ○ Netscape Navigator
[6] ○ Internet Explorer
[6] ● Other

My Hobbies:[7]
Hacking JavaScript
Surfing the Web
Drinking Coffee
Annoying my Friends

My Favorite Color: [8] Blue

Input Events[3]

```
Change: browser (other)
Blur: browser (other)
Focus: hobbies ( )
Click: hobbies (
programming )
Change: hobbies (
programming )
Click: hobbies ( caffeine )
Change: hobbies ( caffeine )
Click: hobbies (
programming caffeine )
Change: hobbies (
programming caffeine )
Blur: hobbies ( programming
caffeine )
Focus: color ( red )
Click: color ( red )
Click: color ( blue )
Change: color ( blue )
```

[9] Clear
[10] Submit
[11] Reset

Form Elements	[1] Text	[2] Password	[3] Textarea	[4] FileUpload	[5] Checkbox
[6] Radio	[7] Select (list)	[8] Select (menu)	[9] Button	[10] Submit	[11] Reset

Events

Client-side JavaScript supports a number of event types. The following table lists the event handlers and the client-side objects that support the handlers. Note that some events, such as *onDblClick*, are not reliably generated on all platforms.

Event Handler **Supported** By

onAbort	Image (JavaScript 1.1)
onBlur, onFocus	Text elements; Window and all other form elements (1.1)
onChange	Select, text input elements
onClick	Button elements, Link. Return *false* to cancel default action.
onDblClick	Document, Link, Image, Button elements (1.2)
onError	Image, Window (1.1)

onKeyDown, onKeyPress, onKeyUp	Document, Image, Link, text elements (1. 2). Return *fa1se* to cancel.
onLoad, onUnload	Window; Image in 1.1
OnMouseDown, onMouseUp	Document, Link, Image, Button elements (I. 2). Return *fa1se* to cancel.
onMouseOver, onMouseOut	Link; Image and Layer (1. 2). Return true to prevent URL display.
onReset, onSubmit	Form (1. 1). Return f a I se to prevent reset or submission.

JavaScript Security Restrictions

For security reasons, there are restrictions on the tasks that untrusted JavaScript code can perform. In Navigator 4, signed scripts can circumvent these restrictions by requesting certain privileges:

Same origin policy

Scripts can only read properties of windows and documents that were loaded from the same web server unless they have *Universa1BrowserRead*.

User's browsing history

Scripts cannot read the array of URI,s from the History object without Universa1BrowserRead.

File upLoads

Scripts cannot set the value property of the FileUpload form element without UniversalBrowserRead.

Sending email and posting news

Scripts cannot submit forms to a mailto: or news: URL without user confirmation or UniversalSendMai1.

Closing windows

A script can only close browser windows that it created, unless it gets user confirmation or has UniversalBrowserWrite.

Snooping in the cache

A script cannot load any about: URls, such as about: cache, without UniversalBrowserRead.

Hidden windows and window decorations

A script cannot create small or offscreen windows or windows without i titlebar, and cannot show or hide window decorations without UniversalBrowserWrite.

Intercepting or spoofing events

A script cannot capture events from windows or documents from a different server and cannot set the fields of an Event object without UniversalBrowserWrite.

Reading and setting preferences

A script cannot read or write user preferences using Navigator.preferences without
UniversalPreferencesRead or UniversalPreferencesWrite.

Global Properties

Core JavaScript defines two global constants:

Infinity

A numeric constant that represents infinity. Internet Explorer 4; ECMA-262; not supported by Navigator 4.

NaN

The not-a-number constant. Internet Explorer 4, ECMA-262; not supported by Navigator 4.

In addition to these core global properties, the Window object defines a number of client-side global properties.

Global Functions

Core JavaScript defines a handful of global functions:

escape(s)

Encode a string for transmission. JavaScript 1.0; ECMA-262; Unicode support in Internet Explorer 4.

eval(code)

Execute JavaScript code from a string.

getClass(javaobj)

Return the JavaClass of a Java0biect. Navigator 3.

isFinite (n)

Determine whether a number is finite. JavaScript 1.2; ECMA-262.

isNaN (x)

Check for not-a-number. JavaScript 1. 1 ECMA-262.

parsefloat(s)

Convert a string to a number. JavaScript 1.0; enhanced in JavaScript 1.1; ECMA-262.

parseInt(s, radix)

Convert a string to an integer. JavaScript 1.0; enhanced in JavaScript 1.1; ECMA-262.

unescape(s)

Decode an escaped string. JavaScript 1.0; ECMA-262; Unicode support in Internet Explorer 4.

In addition to these core global functions the Window object defines a number of client-side global methods.