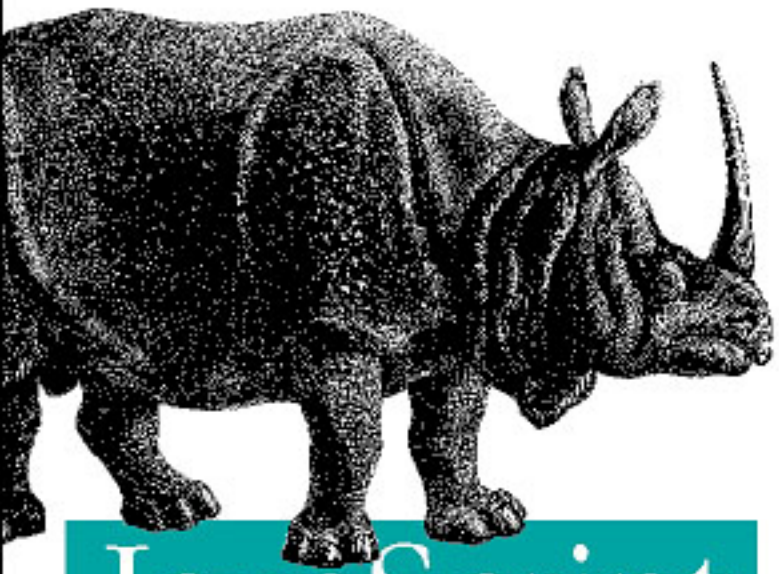
Activate Your Web Pages

2nd Edition
Covers JavaScript 1.5



JavaScript

Pocket Reference

easy computing

O'REILLY

David Flanagan

SECOND EDITION

JavaScript

Pocket Reference

David Flanagan

O'REILLY®

Beijing · Cambridge · Farnham · Köln · Paris · Sebastopol · Taipei · Tokyo

JavaScript Pocket Reference, Second Edition
by David Flanagan

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly Media, Inc. books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*safari.oreilly.com*). For more information contact our corporate/institutional sales department: (800) 998-9938 or *corporate@oreilly.com*.

Editor:	Paula Ferguson
Production Editor:	Philip Dangler
Cover Designer:	Edie Freedman
Interior Designer:	David Futato

Printing History:

October 1998:	First Edition
November 2002:	Second Edition

Nutshell Handbook, the Nutshell Handbook logo, and the O'Reilly logo are registered trademarks of O'Reilly Media, Inc. The *Pocket Reference* series designation, *JavaScript Pocket Reference, Second Edition*, the image of a rhinoceros, and related trade dress are trademarks of O'Reilly Media, Inc. Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and O'Reilly Media, Inc. was aware of a trademark claim, the designations have been printed in caps or initial caps.

While every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

0-596-00411-7
[C] [8/04]

Contents

JavaScript Pocket Reference

The JavaScript Language

JavaScript is a lightweight, object-based scripting language that can be embedded in HTML pages. This book starts with coverage of the core JavaScript language, followed by material on client-side JavaScript, as used in web browsers. The final portion of this book is a quick-reference for the core and client-side JavaScript APIs.

Syntax

JavaScript syntax is modeled on Java syntax, Java syntax, in turn, is modeled on C and C++ syntax. Therefore, C, C++, and Java programmers should find that JavaScript syntax is comfortably familiar.

Case sensitivity

JavaScript is a case-sensitive language. All keywords are in lowercase. All variables, function names, and other identifiers must be typed with a consistent capitalization.

Whitespace

JavaScript ignores whitespace between tokens. You may use spaces, tabs, and newlines to format and indent your code in a readable fashion.

Semicolons

JavaScript statements are terminated by semicolons. When a statement is followed by a newline, however, the terminating semicolon may be omitted. Note that this places a restriction on where you may legally break lines in your JavaScript programs: you may not break a statement across two lines if the first line can be a complete legal statement on its own.

Comments

JavaScript supports both C and C++ comments. Any amount of text, on one or more lines, between `/*` and `*/` is a comment, and is ignored by JavaScript. Also, any text between `//` and the end of the current line is a comment, and is ignored. Examples:

```
// This is a single-line, C++-style comment.
/*
 * This is a multi-line, C-style comment.
 * Here is the second line.
 */
/* Another comment. */ // This too.
```

Identifiers

Variable, function, and label names are JavaScript *identifiers*. Identifiers are composed of any number of letters and digits, and _ and $ characters. The following are legal identifiers:

```
i
my_variable_name
v13
$str
```

Keywords

The following keywords are part of the JavaScript language, and have special meaning to the JavaScript interpreter. Therefore, they may not be used as identifiers:

```
break     do        if         switch  typeof
case      else      in         this    var
catch     false     instanceof throw   void
continue  finally   new        true    while
default   for       null       try     with
delete    function  return
```

JavaScript also reserves the following words for possible future extensions. You may not use any of these words as identifiers either:

```
abstract  enum        int        short
boolean   export      interface  static
byte      extends     long       super
char      final       native     synchronized
class     float       package    throws
const     goto        private    transient
debugger  implements  protected  volatile
double    import      public
```

In addition, you should avoid creating variables that have the same name as global properties and methods: see the Global, Object, and Window reference pages. Within functions, do not use the identifier `arguments` as an argument name or local variable name.

Variables

Variables are declared and initialized with the `var` statement:

```
var i = 1+2+3;
var x = 3, message = 'hello world';
```

Variable declarations in top-level JavaScript code may be omitted, but they are required to declare local variables within the body of a function.

JavaScript variables are *untyped*: they can contain values of any data type.

Global variables in JavaScript are implemented as properties of a special Global object. Local variables within functions are implemented as properties of the Argument object for

that function. Global variables are visible throughout a JavaScript program. Variables declared within a function are only visible within that function. Unlike C, C++, and Java, JavaScript does not have block-level scope: variables declared within the curly braces of a compound statement are not restricted to that block and are visible outside of it.

Data Types

JavaScript supports three primitive data types: numbers, booleans, and strings; and two compound data types: objects and arrays. In addition, it defines specialized types of objects that represent functions, regular expressions, and dates.

Numbers

Numbers in JavaScript are represented in 64-bit floating-point format. JavaScript makes no distinction between integers and floating-point numbers. Numeric literals appear in JavaScript programs using the usual syntax: a sequence of digits, with an optional decimal point and an optional exponent. For example:

```
1
3.14
0001
6.02e23
```

Integers may also appear in hexadecimal notation. A hexadecimal literal begins with `0x`:

```
0xFF // The number 255 in hexadecimal
```

When a numeric operation overflows, it returns a special value that represents positive or negative infinity. When an operation underflows, it returns zero. When an operation such as taking the square root of a negative number yields an error or meaningless result, it returns the special value `NaN`, which represents a value that is not-a-number. Use the global function `isNaN()` to test for this value.

The Number object defines useful numeric constants. The Math object defines various mathematical functions such as `Math.sin()`, `Math.pow()`, and `Math.random()`.

Booleans

The boolean type has two possible values, represented by the JavaScript keywords `true` and `false`. These values represent truth or falsehood, on or off, yes or no, or anything else that can be represented with one bit of information.

Strings

A JavaScript string is a sequence of arbitrary letters, digits, and other characters from the 16-bit Unicode character set.

String literals appear in JavaScript programs between single or double quotes. One style of quotes may be nested within the other:

```
'testing'
"3.14"
'name="myform"'
"Wouldn't you prefer O'Reilly's book?"
```

When the backslash character (\) appears within a string literal, it changes, or escapes, the meaning of the character that follows it. The following table lists these special escape sequences:

Escape	Represents
`\b`	Backspace
`\f`	Form feed
`\n`	Newline
`\r`	Carriage return
`\t`	Tab
`\'`	Apostrophe or single quote that does not terminate the string
`\"`	Double-quote that does not terminate the string
`\\`	Single backslash character

Escape	Represents
\x*dd*	Character with Latin-1 encoding specified by two hexadecimal digits *dd*
\u*dddd*	Character with Unicode encoding specified by four hexadecimal digits *dddd*

The String class defines many methods that you can use to operate on strings. It also defines the `length` property, which specifies the number of characters in a string.

The addition (+) operator concatenates strings. The equality (==) operator compares two strings to see if they contain exactly the same sequences of characters. (This is compare-by-value, not compare-by-reference, as C, C++, or Java programmers might expect.) The inequality operator (`!=`) does the reverse. The relational operators(`<`, `<=`, `>`, and `>=`) compare strings using alphabetical order.

JavaScript strings are *immutable*, which means that there is no way to change the contents of a string. Methods that operate on strings typically return a modified copy of the string.

Objects

An *object* is a compound data type that contains any number of properties. Each property has a name and a value. The . operator is used to access a named property of an object. For example, you can read and write property values of an object `o` as follows:

```
o.x = 1;
o.y = 2;
o.total = o.x + o.y;
```

Object properties are not defined in advance as they are in C, C++, or Java; any object can be assigned any property. JavaScript objects are associative arrays: they associate arbitrary data values with arbitrary names. Because of this fact, object properties can also be accessed using array notation:

```
o["x"] = 1;
o["y"] = 2;
```

Objects are created with the new operator. You can create a new object with no properties as follows:

```
var o = new Object();
```

Typically, however, you use predefined constructors to create objects that are members of a class of objects and have suitable properties and methods automatically defined. For example, you can create a Date object that represents the current time with:

```
var now = new Date();
```

You can also define your own object classes and corresponding constructors; doing this is documented later in this section.

In JavaScript 1.2 and later, you can use object literal syntax to include objects literally in a program. An object literal is a comma-separated list of name:value pairs, contained within curly braces. For example:

```
var o = {x:1, y:2, total:3};
```

See Object (and Date) in the reference section.

Arrays

An array is a type of object that contains numbered values rather than named values. The [ ] operator is used to access the numbered values of an array:

```
a[0] = 1;
a[1] = a[0] + a[0];
```

The first element of a JavaScript array is element 0. Every array has a length property that specifies the number of elements in the array. The last element of an array is element length-1. Array elements can hold any type of value, including objects and other arrays, and the elements of an array need not all contain values of the same type.

You create an array with the `Array()` constructor:

```
var a = new Array();      // Empty array
var b = new Array(10);    // 10 elements
var c = new Array(1,2,3); // Elements 1,2,3
```

As of JavaScript 1.2, you can use array literal syntax to include arrays directly in a program. An array literal is a comma-separated list of values enclosed within square brackets. For example:

```
var a = [1,2,3];
var b = [1, true, [1,2], {x:1, y:2}, "Hello"];
```

See Array in the reference section for a number of useful array manipulation methods.

Functions and methods

A function is a piece of JavaScript code that is defined once and can be executed multiple times by a program. A function definition looks like this:

```
function sum(x, y) {
  return x + y;
}
```

Functions are invoked using the `()` operator and passing a list of argument values:

```
var total = sum(1,2);  // Total is now 3
```

In JavaScript 1.1, you can create functions using the `Function()` constructor:

```
var sum = new Function("x", "y", "return x+y;");
```

In JavaScript 1.2 and later, you can define functions using function literal syntax, which makes the `Function()` constructor obsolete:

```
var sum = function(x,y) { return x+y; }
```


When a function is assigned to a property of an object, it is called a *method* of that object. Within the body of a method,

Index

We'd like to hear your suggestions for improving our indexes. Send email to *index@oreilly.com*.

Q

R

T

U

W