Feature Information

JScript Features - ECMA
JScript Features - Non-ECMA
Microsoft Scripting Run-Time Features

Alphabetic Keyword List

\$1\$	n	Dro	nonti	00
$\Phi \perp \cdots \downarrow$	<u> </u>	<u> </u>	<u>per u</u>	<u>les</u>

abs Method

acos Method

ActiveXObject Object

Addition Operator (+)

anchor Method

arguments Property

Array Object

asin Method

Assignment Operator (=)

atan Method

atan2 Method

atEnd Method

big Method

Bitwise AND Operator (&)

Bitwise Left Shift Operator (<<)

Bitwise NOT Operator (~)

Bitwise OR Operator ()

Bitwise Right Shift Operator (>>)

Bitwise XOR Operator (^)

blink Method

bold Method

Boolean Object

break Statement

caller Property

catch Statement

@cc on Statement

ceil Method

charAt Method

charCodeAt Method

Comma Operator (,)

// (Single-line Comment Statement)

/*..*/ (Multiline Comment Statement)

Comparison Operators

compile Method

Compound Assignment Operators

concat Method (Array)

concat Method (String)

Conditional Compilation

Conditional Compilation Variables

Conditional (trinary) Operator (?:)

constructor Property

continue Statement

cos Method

Data Type Conversion

Date Object

Decrement Operator (--)

delete Operator

description Property

Dictionary Object

dimensions Method

Division Operator (/)

do...while Statement

E Property

Enumerator Object

Equality Operator (==)

Error Object

escape Method

eval Method

exec Method

exp Method

FileSystemObject Object

fixed Method

floor Method

fontcolor Method

fontsize Method

for Statement

for...in Statement

fromCharCode Method

Function Object

function Statement

getDate Method

getDay Method

getFullYear Method

getHours Method

getItem Method

getMilliseconds Method

getMinutes Method

getMonth Method

GetObject Function

getSeconds Method

getTime Method

getTimezoneOffset Method

getUTCDate Method

getUTCDay Method

getUTCFullYear Method

getUTCHours Method

getUTCMilliseconds Method

getUTCMinutes Method

getUTCMonth Method

getUTCSeconds Method

getVarDate Method

getYear Method

Global Object

Greater than Operator (>)

Greater than or equal to Operator (>=)

Identity Operator (===)

@if Statement

if...else Statement

Increment Operator (++)

index Property

indexOf Method

Inequality Operator (!=)

Infinity Property

input Property

instanceof Operator

isFinite Method

isNaN Method

italics Method

item Method

join Method

Labeled Statement

lastIndex Property

lastIndexOf Method

lbound Method

length Property (Array)

length Property (Function)

length Property (String)

Less than Operator (<)

Less than or equal to Operator (<=)

link Method

LN2 Property

LN10 Property

log Method

LOG2E Property

LOG10E Property

Logical AND Operator (&&)

Logical NOT Operator (!)

Logical OR Operator (||)

match Method

Math Object

max Method

MAX_VALUE Property

min Method

MIN_VALUE Property

Modulus Operator (%)

moveFirst Method

moveNext Method

Multiplication Operator (*)

NaN Property (Global)

NaN Property (Number)

NEGATIVE_INFINITY Property

new Operator

Nonidentity Operator (!==)

Number Object

number Property

Object Object

Operator Precedence

parse Method

parseFloat Method

parseInt Method

PI Property

POSITIVE_INFINITY Property

pow Method

prototype Property

random Method

RegExp Object

Regular Expression Object

Regular Expression Syntax

replace Method

return Statement

reverse Method

round Method

ScriptEngine Function

ScriptEngineBuildVersion Function

ScriptEngineMajorVersion Function

ScriptEngineMinorVersion Function

search Method

@set Statement

setDate Method

setFullYear Method

setHours Method

setMilliseconds Method

setMinutes Method

setMonth Method

setSeconds Method

setTime Method

setUTCDate Method

setUTCFullYear Method

setUTCHours Method

setUTCMilliseconds Method

setUTCMinutes Method

setUTCMonth Method

setUTCSeconds Method

setYear Method

sin Method

slice Method (Array)

slice Method (String)

small Method

sort Method

source Property

split Method

sqrt Method

SQRT1_2 Property

SQRT2 Property

strike Method

String Object

sub Method

substr Method

substring Method

Subtraction Operator (-)

sup Method

switch Statement

tan Method

test Method

this Statement

throw Statement

toArray Method

toGMTString Method

toLocaleString Method

toLowerCase Method

toString Method

toUpperCase Method

toUTCString Method

try Statement

typeof Operator

ubound Method

Unary Negation Operator (-)

unescape Method

<u>Unsigned Right Shift Operator >>>)</u>

UTC Method

valueOf Method

var Statement

VBArray Object

void Operator

while Statement

with Statement

Microsoft® JScript® JScript Errors

Run-Time Errors
Syntax Errors

Functions

GetObject Function
ScriptEngine Function
ScriptEngineBuildVersion Function
ScriptEngineMajorVersion Function
ScriptEngineMinorVersion Function

Methods

abs Method

acos Method

anchor Method

asin Method

atan Method

atan2 Method

AtEnd Method

big Method

blink Method

bold Method

ceil Method

charAt Method

charCodeAt Method

compile Method

concat Method (Array)

concat Method (String)

cos Method

dimensions Method

escape Method

eval Method

exec Method

exp Method

fixed Method

floor Method

fontcolor Method

fontsize Method

fromCharCode Method

getDate Method

getDay Method

getFullYear Method

getHours Method

getItem Method

getMilliseconds Method

getMinutes Method

getMonth Method

getSeconds Method

getTime Method

getTimezoneOffset Method

getUTCDate Method

getUTCDay Method

getUTCFullYear Method

getUTCHours Method

getUTCMilliseconds Method

getUTCMinutes Method

getUTCMonth Method

getUTCSeconds Method

getVarDate Method

getYear Method

indexOf Method

isFinite Method

isNaN Method

italics Method

item Method

join Method

lastIndexOf Method

lbound Method

link Method

log Method

match Method

max Method

min Method

moveFirst Method

moveNext Method

parse Method

parseFloat Method

parseInt Method

pow Method

random Method

replace Method

reverse Method

round Method

search Method

setDate Method

setFullYear Method

setHours Method

setMilliseconds Method

setMinutes Method

setMonth Method

setSeconds Method

setTime Method

setUTCDate Method

setUTCFullYear Method

setUTCHours Method

setUTCMilliseconds Method

setUTCMinutes Method

setUTCMonth Method

setUTCSeconds Method

setYear Method

sin Method

slice Method (Array)

slice Method (String)

small Method

sort Method

split Method

sqrt Method

strike Method

sub Method

substr Method

substring Method

sup Method

tan Method

test Method

toArray Method

toGMTString Method

toLocaleString Method

toLowerCase Method

toString Method

toUpperCase Method

toUTCString Method

ubound Method

unescape Method

UTC Method

valueOf Method

Objects

ActiveXObject Object

Array Object

Boolean Object

Date Object

Dictionary Object

Enumerator Object

Error Object

FileSystemObject Object

Function Object

Global Object

Math Object

Number Object

Object Object

RegExp Object

Regular Expression Object

String Object

VBArray Object

Operators

Addition Operator (+)
Assignment Operator (=)
Bitwise AND Operator (&)
Bitwise Left Shift Operator (<<)
Bitwise NOT Operator (~)
Bitwise OR Operator ()
Bitwise Right Shift Operator (>>)
Bitwise XOR Operator (^)
Comma Operator (,)
Comparison Operators
Compound Assignment Operators
Conditional (trinary) Operator (?:)
Decrement Operator ()
delete Operator
Division Operator (/)
Equality Operator (==)
Greater than Operator (>)
Greater than or equal to Operator (>=
<u>Identity Operator (===)</u>
Increment Operator (++)
<pre>Inequality Operator (!=)</pre>
instanceof Operator
Less than Operator(<)
<pre>Less than or equal to Operator (<=)</pre>
Logical AND Operator (&&)
Logical NOT Operator (!)
Logical OR Operator ()
Modulus Operator (%)

Multiplication Operator (*)

new Operator

Nonidentity Operator (!==)

Operator Precedence

Subtraction Operator (-)

typeof Operator

Unary Negation Operator (-)

Unsigned Right Shift Operator (>>>)

void Operator

Properties

\$1...\$9 Properties

arguments Property

caller Property

constructor Property

description Property

E Property

index Property

Infinity Property

input Property

lastIndex Property (RegExp)

length Property (Array)

length Property (Function)

length Property (String)

LN2 Property

LN10 Property

LOG2E Property

LOG10E Property

MAX_VALUE Property

MIN_VALUE Property

Nan Property (Global)

NaN Property (Number)

NEGATIVE_INFINITY Property

number Property

PI Property

POSITIVE_INFINITY Property

prototype Property

source Property

SQRT1_2 Property

SQRT2 Property

Statements

break Statement

catch Statement

@cc on Statement

Comment Statements

continue Statement

do...while Statement

for Statement

for...in Statement

function Statement

@if Statement

if...else Statement

Labeled Statement

return Statement

@set Statement

switch Statement

this Statement

throw Statement

try Statement

var Statement

while Statement

with Statement

JScript is the Microsoft implementation of the ECMA 262 language specification. It is a full implementation, plus some enhancements that take advantage of capabilities of Microsoft Internet Explorer. This tutorial is intended to help you get started with JScript.

Easy to Use, Easy to Learn

JScript is an interpreted, object-based scripting language. Although it has fewer capabilities than full-fledged object-oriented languages like C++ and Java, JScript is more than sufficiently powerful for its intended purposes.

JScript is not a cut-down version of any other language (it is only distantly and indirectly related to Java, for example), and it is not a simplification of anything. It is, however, limited. You cannot write standalone applications in it, for example, and it has little capability for reading or writing files. Moreover, JScript scripts can run only in the presence of an interpreter, either in a Web server or a Web browser.

JScript is a loosely typed language. That means you do not have to declare the data types of variables explicitly. In fact, you cannot explicitly declare data types in JScript. Moreover, in many cases JScript performs conversions automatically when they are needed. For instance, if you try to add a number to an item that consists of text (a string), the number is converted to text.

The rest of this tutorial is an overview of JScript features. For full details of the language implementation, consult the <u>language reference</u>.

Note The code in many of this tutorial's examples is somewhat more explicit and less dense than code you'll find in actual Web pages. Most of it is also fairly simple. The intent here is to clarify the concepts, not to express optimal coding conciseness and style. There is, in any case, no shame in writing code that you can read and easily understand, six months after you write it.

Microsoft® JScript® Writing JScript

Code

Like many other programming languages, Microsoft JScript is written in text format, and is organized into statements, blocks consisting of related sets of statements, and comments. Within a statement you can use <u>variables</u>, immediate data such as strings and numbers, and expressions.

Statements

A JScript code **statement** consists of one or more items and symbols on a line. A new line begins a new statement, but it is a good idea to terminate your statements explicitly. You can do this with the semicolon (;), which is the JScript termination character.

```
aBird = "Robin";
var today = new Date();
```

A group of JScript statements that is surrounded by braces ({}) is called a block. Blocks of statements are used, for example, in functions and conditionals.

In the following example, the first statement begins the definition of a function, which consists of a block of five statements. The last three statements, which are not surrounded by braces, are not a block and are not part of the function definition.

```
function convert(inches) {
  feet = inches / 12; // These five statements are in a
```

```
miles = feet / 5280;
nauticalMiles = feet / 6080;
cm = inches * 2.54;
meters = inches / 39.37;
}
km = meters / 1000; // These three statements are not kradius = km;
mradius = miles;
```

Comments

A single-line JScript comment begins with a pair of forward slashes (//). A multiline comment begins with a forward slash and asterisk in combination (/*), and ends with the reverse (*/).

aGoodIdea = "Comment your code thoroughly."; // T

/*

This is a multiline comment that explains the precedin

The statement assigns a value to the aGoodIdea variat is contained between the quote marks, is called a literal and directly contains information; it does not refer to to (The quote marks are not part of the literal.)

// This is another multiline comment, written as a seri
// After the statement is executed, you can refer to the

// variable by using its name, as in the next statement,
// appended to the aGoodIdea variable by concatenation

var extendedIdea = aGoodIdea + " You never know w

Assignments and Equality

The equal sign (=) is used in JScript to indicate the action of assigning a value. That is, a JScript code statement could say

```
anInteger = 3;
```

It means "Assign the value 3 to the variable anInteger," or "anInteger takes the value 3." When you want to compare two values to find out whether they are equal, use a pair of equal signs (==). This is discussed in detail in Controlling Program Flow.

Expressions

A JScript expression is something that a person can read as a Boolean or numeric expression. Expressions contain symbol characters like "+" rather than words like "added to". Any valid combination of values, variables, operators, and expressions constitutes an expression.

```
var anExpression = "3 * (4 / 5)";
var aSecondExpression = "Math.PI * radius * 2";
var aThirdExpression = aSecondExpression + "%" + a
var aFourthExpression = "(" + aSecondExpression + "
```

Microsoft® JScript Variables

Variables are used in Microsoft JScript to store values in your scripts. They are a way to retrieve and manipulate values using names. When used effectively then can help in understanding what a script does.

Declaring Variables

Although not required, it is considered good practice to declare variables before using them. You do this using the **var** statement. The only time you must use the **var** statement is when declaring variables that are local to a function. Local variables are those that are only within the function. At all other times, using the **var** statement to declare variables before their use is a recommended practice.

The following code examples are of variable declaration:

```
var mim = "A man, a plan, a canal, Panama!"; // The walle of which is assigned
```

```
var ror = 3;  // The value stored in ror has numeric
var nen = true;  // The value stored in nen has Boo
var fif = 2.718281828  // The value stored in fif ha
```

Naming Variables

JScript is a case-sensitive language, so naming a variable *myCounter* is different from naming it *MYCounter*. In addition, variable names, which can be of any length, must follow certain rules:

- The first character must be a letter (either uppercase or lowercase) or an underscore (_), or a dollar sign (\$).
- Subsequent characters can be letters, numbers, underscores, or dollar signs.
- The variable name can't be a <u>reserved word</u>.

Some examples of valid variable names:

- _pagecount
- Part9
- Number Items

Some invalid variable names:

- 99Balloons // Starts with a number.
- Smith&Wesson; // Ampersand (&) is not a valid character for variable names.

In instances in which you want to declare a variable and initialize it, but without giving it any particular value, you may assign it a special value, <u>null</u>.

```
var zaz = null;
var notalot = 3 * zaz;  // At this point, notalot beco
```

If you declare a variable without assigning any value to it, it

exists but is undefined.

```
var godot;
var waitingFor = 1 * godot; // Places the value NaN in
```

You can declare a variable implicitly (without using **var**) by assigning a value to it. You cannot, however, use a variable that has never been declared at all. To do so generates an error at runtime.

```
lel = ""; // The variable lel is declared implicitly.
var aMess = vyv + zez; // Generates an error because
```

Coercion

As JScript is a loosely-typed language, variables in JScript technically have no fixed type. Instead, they have a type equivalent to the type of the value they contain. It is possible, under some circumstances, to force the automatic conversion (or coercion) of a variable or a piece of data into a different type. Numbers can easily be included in strings, but strings cannot be included directly in numbers, so explicit conversion functions, parseInt() and parseFloat(), are provided.

```
var theFrom = 1;
var theTo = 10;
var doWhat = "Count from ";
doWhat += theFrom + " to " + theTo + ".";
```

After this code is executed, the *doWhat* variable contains "Count

from 1 to 10." The number data have been coerced into string form.

After this code is executed, the *nowWhat* variable contains "0110". The following steps are followed to arrive at this result:

- 1. Look at the types of 1 and "10". The "10" is a string, the 1 is a number, so the number is coerced into a string.
- 2. As the values on either side of the + operator are both strings, do a string concatenation. This results in "110"
- 3. Look at the types of the values on either side of the +=. *nowWhat* contains a number, and "110" is a string, so convert the number to a string.
- 4. As there are now strings on either side of the += operator, do a string concatentation. This results in "0110".
- 5. Store this result in *nowWhat*.

```
var nowThen = 0;
nowThen += 1 + parseInt("10");  // In this case, "+
```

After this code is executed, the *nowThen* variable contains the integer 11.

Microsoft® JScript® JScript Data Types

What Are the JScript Data Types?

Microsoft JScript has six types of data. The main types are numbers, strings, objects, and Booleans. The other two are **null** and **undefined**.

String Data Type

Strings are delineated by single or double quotation marks. (Use single quotes to type strings that contain quotation marks.) A string is also an object in JScript, but it is a special case, with special properties. The following are examples of strings:

```
"The cow jumped over the moon."
"Avast, ye lubbers!" roared the technician.'
"42"
```

A string can contain zero or more unicode characters. When it contains zero, it is called a zero-length string ("").

Number Data Type

JScript supports both integer and floating-point numbers. Integers can be positive, 0, or negative; a floating-point number

.

can contain either a decimal point, an "e" (uppercase or lowercase), which is used to represent "ten to the power of" in scientific notation, or both. These numbers follow the IEEE 754 standard for numerical representation. Last, there are certain number values that are special:

- NaN, or not a Number
- Positive Infinity
- Negative Infinity
- Positive 0
- Negative 0

Integers can be represented in base 10 (decimal), base 8 (octal), and base 16 (hexadecimal).

Octal integers are specified by a leading "0", and can contain digits 0 through 7. If a number has a leading "0" but contains the digits "8" and/or "9", it is a decimal number. A number that would otherwise be an octal number but contains the letter "e" (or "E") generates an error.

Hexadecimal ("hex") integers are specified by a leading "0x" (the "X" can be uppercase or lowercase) and can contain digits 0 through 9 and letters A through F (either uppercase or lowercase). The letter "e" is a permissible digit in hexadecimal notation and does not signify an exponential number. The letters A through F are used to represent, as single digits, the numbers that are 10 through 15 in base 10. That is, 0xF is equivalent to 15, and 0x10 is equivalent to 16.

Octal and hexadecimal numbers can be negative, but cannot be fractional. A number that begins with a single "0" and contains a decimal point is a decimal floating-point number; if a number that begins with "0x" or "00" contains a decimal point, anything to the right of the decimal point is ignored.

Some example numbers:

```
.0001, 0.0001, 1e-4, 1.0e-4 // Four floating-point num
                     // A floating-point number, equiv
3.45e2
42
                   // An integer number.
0377
                    // An octal integer, equivalent to 2
                     // As octal numbers cannot have
00.0001
0378
                    // An integer, equivalent to 378.
                    // A hexadecimal integer, equivale
0Xff
0x37CF
                      // A hexadecimal integer, equiva
                     // A hexadecimal integer, equival
0x3e7
                      // As hexadecimal numbers can
0x3.45e2
```

Booleans

The possible Boolean values are **true** and **false**. These are special values, and are not usable as 1 and 0.

Note In a comparison, any expression that evaluates to 0 is taken to be false, and any statement that evaluates to a number other than 0 is taken to be true. Thus the following expression evaluates to true:

For more information on comparisons, see **Controlling Program Flow**.

Undefined Data Type

A value that is undefined is simply a value given to a variable after it has been created, but before a value has been assigned to it.

Null Data Type

A **null** value is one that has no value and means nothing.

Microsoft® JScript Operators

JScript has a full range of operators, including arithmetic, logical, bitwise, and assignment operators. There are also a few miscellaneous operators.

Computational		Logical		Bitwise		Assig
Description	Symbol	Description	Symbol	Description	Symbol	Descripti
Unary negation	-	<u>Logical</u> <u>NOT</u>	!	<u>Bitwise</u> NOT	2	Assignme
Increment	++	Less than	<	<u>Bitwise</u> Left Shift	<<	Compoun Assignme
<u>Decrement</u>		<u>Greater</u> <u>than</u>	>	Bitwise Right Shift	>>	
Multiplication	*	<u>Less than or</u> <u>equal to</u>	<=	<u>Unsigned</u> Right Shift	>>>	
Division	/	<u>Greater</u> <u>than or</u> <u>equal to</u>	>=	Bitwise AND	&	
Modulus arithmetic	%	Equality	==	Bitwise XOR	٨	
Addition	+	<u>Inequality</u>	!=	Bitwise OR		
Subtraction	-	<u>Logical</u> <u>AND</u>	&&			
		Logical OR				
		Conditional (trinary)	?:			
		Comma	,			
		<u>Identity</u>	===			

Operator Precedence

Operators in JScript are evaluated in a particular order. This order is known as the operator precedence. The following table lists the operators in highest to lowest precedence order. Operators in the same row are evaluated in left to right order.

Operator	Description	
. [] ()	Field access, array indexing, and function calls	
++ ~! typeof new void delete	Unary operators, return data type, object creation, undefined values	
* / %	Multiplication, division, modulo division	
+ - +	Addition, subtraction, string concatenation	
<< >> >>>	Bit shifting	
< <= > >=	Less than, less than or equal to, greater than, greater than or equal to	
== != === !==	Equality, inequality, identity, nonidentity	
&	Bitwise AND	
Λ	Bitwise XOR	
	Bitwise OR	
&&	Logical AND	
	Logical OR	
?:	Conditional	
= OP=	Assignment, assignment with operation	
,	Multiple evaluation	

Parentheses are used to alter the order of evaluation. The expression within parentheses is fully evaluated before its value is used in the remainder of the statement.

An operator with higher precedence is evaluated before one with lower precedence. For example:

$$z = 78 * (96 + 3 + 45)$$

There are five operators in this expression: =, *, (), +, and +. According to precedence, they are evaluated in the following order: (), *, +, +, =.

- 1. Evaluation of the expression within the parentheses is first: There are two addition operators, and they have the same precedence: 96 and 3 are added together and 45 is added to that total, resulting in a value of 144.
- 2. Multiplication is next: 78 and 144 are multiplied, resulting in a value of 11232.
- 3. Assignment is last: 11232 is assigned into z.

Microsoft® JScript® Controlling Program Flow

Why Control the Flow of Execution?

Fairly often, you need a script to do different things under different conditions. For example, you might write a script that checks the time every hour, and changes some parameter appropriately during the course of the day. You might write a script that can accept some sort of input, and act accordingly. Or you might write a script that repeats a specified action.

There are several kinds of conditions that you can test. All conditional tests in Microsoft JScript are Boolean, so the result of any test is either **true** or **false**. You can freely test values that are of Boolean, numeric, or string type.

JScript provides control structures for a range of possibilities. The simplest control structures are the conditional statements.

Using Conditional Statements

JScript supports **if** and **if...else** conditional statements. In **if** statements a condition is tested, and if the condition meets the test, some JScript code you've written is executed. (In the **if...else** statement, different code is executed if the condition fails the test.) The simplest form of an **if** statement can be written entirely on one line, but multiline **if** and **if...else** statements are much more common.

The following examples demonstrate syntaxes you can use with if and

if...else statements. The first example shows the simplest kind of Boolean test. If (and only if) the item between the parentheses evaluates to **true**, the statement or block of statements after the **if** is executed.

```
// The smash() function is defined elsewhere in the cod
if (newShip)
  smash(champagneBottle,bow); // Boolean test of wl
// In this example, the test fails unless both conditions
if (rind.color == "deep yellow " && rind.texture == "l
{
     theResponse = ("Is it a Crenshaw melon? <br > ")
// In this example, the test succeeds if either condition
var theReaction = "";
if ((lbsWeight > 15) || (lbsWeight > 45))
  theReaction = ("Oh, what a cute kitty! <br>");
else
  theReaction = ("That's one huge cat you've got there
```

Conditional Operator

JScript also supports an implicit conditional form. It uses a question mark after the condition to be tested (rather than the word **if** before the condition), and specifies two alternatives, one to be used if the condition is met and one if it is not. The

alternatives are separated by a colon.

```
var hours = "";

// Code specifying that hours contains either the content
// theHour, or theHour - 12.
hours += (theHour >= 12) ? " PM" : " AM";
```

Tip If you have several conditions to be tested together, and you know that one is more likely to pass or fail than any of the others, depending on whether the tests are connected with OR (||) or AND (&&), you can speed execution of your script by putting that condition first in the conditional statement. For example, if three conditions must all be true (using && operators) and the second test fails, the third condition is not tested.

Similarly, if only one of several conditions must be true (using \parallel operators), testing stops as soon as any one condition passes the test. This is particularly effective if the conditions to be tested involve execution of function calls or other code.

An example of the side effect of short-circuiting is that runsecond will not be executed in the following example if runfirst() returns 0 or **false**.

Using Repetition, or Loops

There are several ways to execute a statement or block of statements repeatedly. In general, repetitive execution is called *looping*. It is typically controlled by a test of some variable, the value of which is changed each time the loop is executed.

Microsoft JScript supports many types of loops: **for** loops, **for...in** loops, **while** loops, **do...while** loops, and **switch** loops.

Using for Loops

The **for** statement specifies a counter variable, a test condition, and an action that updates the counter. Just before each time the loop is executed (this is called one pass or one iteration of the loop), the condition is tested. After the loop is executed, the counter variable is updated before the next iteration begins.

If the condition for looping is never met, the loop is never executed at all. If the test condition is always met, an infinite loop results. While the former may be desirable in certain cases, the latter rarely is, so take care when writing your loop conditions.

```
The update expression ("icount++" in the following exists executed at the end of the loop, after the block of stabody of the loop is executed, and before the condition */
```

```
var howFar = 11; // Sets a limit of 11 on the loop.

var sum = new Array(howFar); // Creates an array cal
var theSum = 0;
sum[0] = 0;

for(var icount = 1; icount < howFar; icount++) {
theSum += icount;</pre>
```

```
sum[icount] = theSum;
}

var newSum = 0;
for(var icount = 1; icount > howFar; icount++) {
  newSum += icount;
}

var sum = 0;
for(var icount = 1; icount > 0; icount++) { // This sum += icount;
}
```

Using for...in Loops

JScript provides a special kind of loop for stepping through all the properties of an <u>object</u>. The loop counter in a **for...in** loop steps through all indexes in the array. It is a string, not a number.

```
for (j in tagliatelleVerde) // tagliatelleVerde is an obje
{
  // Various JScript code statements.
}
```

Using while Loops

The **while** loop is very similar to a **for** loop. The difference is that a **while** loop does not have a built-in counter variable or update expression. If you already have some changing condition

that is reflected in the value assigned to a variable, and you want to use it to control repetitive execution of a statement or block of statements, use a **while** loop.

Note Because **while** loops do not have explicit built-in counter variables, they are even more vulnerable to infinite looping than the other types. Moreover, partly because it is not necessarily easy to discover where or when the loop condition is updated, it is only too easy to write a **while** loop in which the condition, in fact, never does get updated. You should be extremely careful when you design **while** loops.

Using break and continue Statements

Microsoft JScript provides a statement to stop the execution of a

loop. The **break** statement can be used to stop execution if some (presumably special) condition is met. The **continue** statement can be used to jump immediately to the next iteration, skipping the rest of the code block but updating the counter variable as usual if the loop is a **for** or **for...in** loop.

```
var theComment = "";
var the Remainder = 0;
var theEscape = 3;
var checkMe = 27;
for (kcount = 1; kcount <= 10; kcount++)
  theRemainder = checkMe % kcount;
  if (theRemainder == theEscape)
       break; // Exits from the loop at the first encou
theComment = checkMe + " divided by " + kcount + '
for (kcount = 1; kcount <= 10; kcount++)
{
 theRemainder = checkMe % kcount;
 if (theRemainder != theEscape)
   continue; // Selects only those remainders that equ
```

```
// JScript code that uses the selected remainders.
var theMoments = "";
var theCount = 42; // The counter is initialized.
while (theCount >= 1) {
// if (theCount < 10) { // Warning!
// This use of continue creates an infinite loop!
// continue;
// }
  if (theCount > 1) {
     theMoments = "Only " + theCount + " moments ]
  else {
     theMoments = "Only one moment left!";
theCount--; // The counter is updated.
theCount = "BLASTOFF!";
```

What Is a Function?

Microsoft JScript functions perform actions. They can also return results. Sometimes these are the results of calculations or comparisons.

Functions combine several operations under one name. This lets you streamline your code. You can write out a set of statements, name it, and then execute the entire set any time you want to, just by calling it and passing to it any information it needs.

You pass information to a function by enclosing the information in parentheses after the name of the function. Pieces of information that are being passed to a function are called arguments or parameters. Some functions don't take any arguments at all; some functions take one argument; some take several. There are even functions for which the number of arguments depends on how you are using the function.

JScript supports two kinds of functions: those that are built into the language, and those you create yourself.

Special Built-in Functions

The JScript language includes several built-in functions. Some of them let you handle expressions and special characters, and convert strings to numeric values.

For example, **escape()** and **unescape()** are used to convert characters that have special meanings in HTML code, characters that you cannot just put directly into text. For example, the angle brackets, "<" and ">", delineate

HTML tags.

The **escape** function takes as its argument any of these special characters, and returns the escape code for the character. Each escape code consists of a percent sign (%) followed by a two-digit number. The **unescape** function is the exact inverse. It takes as its argument a string consisting of a percent sign and a two-digit number, and returns a character.

Another useful built-in function is **eval()**, which evaluates any valid mathematical expression that is presented in string form. The **eval()** function takes one argument, the expression to be evaluated.

```
var anExpression = "6 * 9 % 7";
var total = eval(anExpression);  // Assigns the value
var yetAnotherExpression = "6 * (9 % 7)";
total = eval(yetAnotherExpression)  // Assigns the
var totality = eval("...surrounded by acres of clams.");
```

Consult the <u>language reference</u> for more information about these and other built-in functions.

Creating Your Own Functions

You can create your own functions and use them where you need them. A function definition consists of a function statement and a block of JScript statements.

The checkTriplet function in the following example takes as its arguments the lengths of the sides of a triangle, and calculates from them whether the triangle is a right triangle by checking whether the three numbers constitute a Pythagorean triplet. (The square of the length of the hypotenuse of a right triangle is equal to the sum of the squares of the lengths of the other two sides.) The checkTriplet function calls one of two other functions to make

the actual test.

Notice the use of a very small number ("epsilon") as a testing variable in the floating-point version of the test. Because of uncertainties and roundoff errors in floating-point calculations, it is not practical to make a direct test of whether the square of the hypotenuse is equal to the sum of the squares of the other two sides unless all three values in question are known to be integers. Because a direct test is more accurate, the code in this example determines whether it is appropriate and, if it is, uses it.

```
var epsilon = 0.0000000000001; // Some very small r
var triplet = false;
function integerCheck(a, b, c) { // The test function f
  if ((a*a) == ((b*b) + (c*c))) { // The test itself.
  triplet = true;
} // End of the integer checking function.
function floatCheck(a, b, c) { // The test function for
var the Check = ((a*a) - ((b*b) + (c*c))) // Make the te
  if (the Check < 0) { // The test requires the absolute
  theCheck *= -1;
  if (epsilon > theCheck) { // If it's as close as that, i
  triplet = true;
} // End of the floating-poing check function.
function checkTriplet(a, b, c) { // The triplet checker.
```

```
var d = 0; // Create a temporary holding bin.
  if (c > b) { // If c > b, swap them.
  d = c:
  c = b;
  b = d;
  } // If not, ignore them.
  if (b > a) { // If b > a, swap them.
  d = b;
  b = a;
  a = d;
  } // If not, ignore them.
// Side "a" is now the hypotenuse, if there is one.
  if (((a\%1) == 0) \&\& ((b\%1) == 0) \&\& ((c\%1) == 0)
  integerCheck(a, b, c); // If so, use the precise check
  else
     floatCheck(a, b, c); // If not, get as close as is rea
} // End of the triplet check function.
// The next three statements assign sample values for t
var sideA = 5;
var sideB = 5;
var sideC = Math.sqrt(50);
checkTriplet(sideA, sideB, sideC); // Call the function
```

What Are Objects?

In Microsoft JScript, objects are, essentially, collections of properties and methods. A method is a function that is a member of an object, and a property is a value or set of values (in the form of an array or object) that is a member of an object. JScript supports three kinds of objects: intrinsic objects, objects you create, and browser objects, which are covered elsewhere.

Objects as Arrays

In JScript, objects and arrays are handled identically. You can refer to any of the members of an object (its properties and methods) either by name (using the name of the object, followed by a period, followed by the name of the property) or by its array subscript index. Subscript numbering in JScript begins with 0. For convenience, the subscript can also be referred to by its name.

Thus, a property can be referred to in several ways. All of the following statements are equivalent.

```
theWidth = spaghetti.width;
theWidth = spaghetti[3]; // [3] is the "width" index.
theWidth = spaghetti["width"];
```

While it is possible to use brackets to refer to a property by its

numeric index, it is not possible to use the dot (.) convention with index numbers. The following statement generates an error.

```
theWidth = spaghetti.3;
```

When an object has another object as a property, the naming convention extends in a straightforward way.

```
var init4 = toDoToday.shoppingList[3].substring(0,1);
```

The fact that objects can have other objects as properties lets you generate arrays with more than one subscript, which are not directly supported. The following code creates a multiplication table for values from 0 times 0 through 16 times 16.

```
var multTable = new Array(17); // Make the shell that
for (var j = 0; j < multTable.length; j++) { // Prepare
  var aRow = new Array(17); // Create a row.
  for (var i = 0; i < aRow.length; i++) { // Prepare to
    aRow[i] = (i + " times " + j + " = " + i*j); // Make a
  }
multTable[j] = aRow; // Put the filled row into the tab
}</pre>
```

To refer to one of the elements of an array of this kind, use multiple sets of brackets.

```
var multiply3x7 = multTable[3][7];
```

The following statement generates an error.

```
var multiply3x7 = multTable[3, 7];
```

Microsoft® JScript® JScript Reserved Keywords

JScript has a number of reserved keywords. These words come in three types: JScript reserved keywords, future reserved words, and words to avoid.

JScript Keywords				
break	false	in	this	void
continue	for	new	true	while
delete	function	null	typeof	with
else	if	return	var	

JScript Future Keywords				
case	debugger	export	super	
catch	default	extends	switch	
class	do	finally	throw	
const	enum	import	try	

The words to avoid are any that are already the names of intrinsic JScript objects or functions. Words like *String* or *parseInt* are included in this.

Using any of the keywords from the first two categories causes a compilation error when your script is first loaded. Using a reserved word from the third set can cause odd behavior problems if you attempt to use both your variable and the original entity of the same name in the same script. For example, the following script does not do quite what you think it should:

var String;

var text = new String("This is a string object");

In this case, you get an error saying that String is not an object. Many cases of using a pre-existing identifier aren't this obvious.

Microsoft® JScript® Recursion

Recursion is an important programming technique. It's used to have a function call itself from within itself. One handy example is the calculation of factorials. The factorials of 0 and 1 are both defined specifically to be 1. The factorials of larger numbers are calculated by multiplying 1 * 2 * ..., incrementing by 1 until you reach the number for which you're calculating the factorial.

The following paragraph is a function, defined in words, that calculates a factorial.

"If the number is less than zero, reject it. If it isn't an integer, round it down to the next integer. If the number is zero or one, its factorial is one. If the number is larger than one, multiply it by the factorial of the next smaller number."

To calculate the factorial of any number that is larger than 1, you need to calculate the factorial of at least one other number. The function you use to do that is the function you're in the middle of already; the function must call itself for the next smaller number, before it can execute on the current number. This is an example of recursion.

Clearly, there is a way to get in trouble here. You can easily create a recursive function that doesn't ever get to a definite result, and cannot reach an endpoint. Such a recursion causes the computer to execute a so-called "infinite" loop. Here's an example: omit the first rule (the one about negative numbers) from the verbal description of calculating a factorial, and try to calculate the factorial of any negative number. This fails, because in order to calculate the factorial of, say, -24 you first have to calculate the factorial of -25; but in order to do that you first have to calculate the factorial of -26; and so on. Obviously, this never reaches a stopping place.

Thus, it is extremely important to design recursive functions with great care. If you even suspect that there's any chance of an infinite recursion, you can have the function count the number of times it calls itself. If the function calls itself too many times, however many you decide that should be, it automatically quits.

Here's the factorial function again, this time written in JScript code.

```
function factorial(aNumber) {
  aNumber = Math.floor(aNumber); // If the number is
  if (aNumber < 0) { // If the number is less than zero,
    return "not a defined quantity";
  }
  if ((anumber == 0) || (anumber == 1)) { // If the n
    return 1;
  }
   else return (anumber * factorial(anumber - 1)); //
}</pre>
```

Microsoft® JScript® Variable Scope

Microsoft JScript has two scopes: global and local. If you declare a variable outside of any function definition, it is a global variable, and its value is accessible and modifiable throughout your program. If you declare a variable inside of a function definition, that variable is local. It is created and destroyed every time the function is executed; it cannot be accessed by anything outside the function.

A local variable can have the same name as a global variable, but it is entirely distinct and separate. Consequently, changing the value of one variable has no effect on the other. Inside the function in which the local variable is declared, only the local version has meaning.

```
var aCentaur = "a horse with rider,"; // Global definiti
// JScript code, omitted for brevity.
function antiquities() // A local aCentaur variable is d
{
// JScript code, omitted for brevity.
var aCentaur = "A centaur is probably a mounted Scyt
// JScript code, omitted for brevity.
aCentaur += ", misreported; that is, "; // Adds to the
// JScript code, omitted for brevity.
} // End of the function.
```

```
var nothinginparticular = antiquities();
aCentaur += " as seen from a distance by a naive inno
```

/*

Within the function, the variable contains "A centaur i misreported; that is, "; outside the function, the variab "a horse with rider, as seen from a distance by a naive */

It's important to note that variables act as if they were declared at the beginning of whatever scope they exist in. Sometimes this results in unexpected behaviors.

```
var aNumber = 100;
var withAdditive = 0;
withAdditive += aNumber; // withAdditive is now 10
tweak();
withAdditive += aNumber; // withAdditive is now 20
function tweak() {
 var newThing = 0; // Explicit declaration of the newT
 // The next statement, if it were not commented out, w
 // newThing = aNumber;
 // The next statement assigns the value 42 to the local
 aNumber = 42;
 if (false) {
```

```
var aNumber; // This statement is never executed.
aNumber = "Hello!"; // This statement is never exe
} // End of the conditional.
} // End of the function definition.
```

The statement that is commented out attempts to assign the value of the local variable *aNumber* to the local variable *newThing*. It fails, despite the fact that a local *aNumber* variable is defined elsewhere in the function, and therefore exists throughout. The *aNumber* variable does not have any assigned value at the point where this statement occurs in the code, and is thus <u>undefined</u>.

Microsoft® JScript® Copying, Passing, and Comparing Data

In Microsoft JScript, how data is handled depends on its data type.

By Value vs. By Reference

Numbers and Boolean values (**true** and **false**) are copied, passed, and compared *by value*. When you copy or pass by value, you allocate a space in computer memory and put the value of the original into it. If you then change the original, the copy is not affected (and vice versa), because the two are separate entities.

Objects, arrays, and functions are copied, passed, and compared *by reference* under most circumstances. When you copy or pass by reference, you essentially create a pointer to the original item, and use the pointer as if it were a copy. If you then change the original, you change both the original and the copy (and vice versa). There is really only one entity; the "copy" is not actually a copy, it's just another reference to the data.

Note You can change this behavior for objects and arrays by specifying the **assign()** method for them.

Last, strings are copied and passed by reference, but are compared by value.

Note Because of the way the <u>ASCII</u> and ANSI character sets are constructed, capital letters precede lowercase ones in sequence order. For example, "Zoo" compares as *less* than

Passing Parameters to Functions

When you pass a parameter to a function by value, you are making a separate copy of that parameter, a copy that exists only inside the function. If, on the other hand, you pass a parameter by reference, and the function changes the value of that parameter, it is changed everywhere in the script.

Testing Data

When you perform a test by value, you compare two distinct items to see whether they are equal to each other. Usually, this comparison is performed on a byte-by-byte basis. When you test by reference, you are checking to see whether two items are pointers to a single original item. If they are, then they compare as equal; if not, even if they contain the exact same values, byte-for-byte, they compare as unequal.

Copying and passing strings by reference saves memory; but because you cannot change strings once they are created, it becomes possible to compare them by value. This lets you test whether two strings have the same content even if one was generated entirely separately from the other.

Array Indexing

Arrays in JScript are *sparse*. That is, if you have an array with three elements that are numbered 0, 1, and 2, you can create element 50 without worrying about elements 3 through 49. If the array has an automatic length variable (see <u>Intrinsic Objects</u> for an explanation of automatic monitoring of array length), the length variable is set to 51, rather than to 4. You can certainly create arrays in which there are no gaps in the numbering of elements, but you aren't required to. In fact, in JScript, your arrays don't have to have numbered subscripts at all.

In JScript, objects and arrays are essentially identical to each other. The real difference is not in the data, but rather in the way you address the members of an array or the <u>properties</u> and methods of an object.

Addressing Arrays

There are two main ways to address the members of an array. Ordinarily, you address arrays by using brackets. The brackets enclose either a numeric value or an <u>expression</u> that evaluates to a nonnegative integer. The following example assumes that the *entryNum* variable is defined and assigned a value elsewhere in the script.

This method of addressing is equivalent to the method for

addressing objects, though in object addressing, what follows the period must be the name of an actual property. If there is no such property, your code generates an error.

The second way to address an array is to make an object/array that contains properties that are numbered, and then generate the numbers in a loop. The following example generates two arrays, one for the name and one for the address, from a listing in *addressBook*. Each of these contains four properties. An instance of *theName*, for example, built from the [Name1] through [Name4] properties of *theListing*, might contain "G." "Edward" "Heatherington" "IV", or "George" "" "Sand" "".

```
theListing = addressBook[entryNu)
for (i = 1; i < 4; i++) {
theName[i] = theListing["Name" +
theAddress[i] = theListing["Addres
}</pre>
```

While this particular instance is short, and could easily have been written in the "dot" style of notation, (that is, addressing theListing, theName, and theAddress as objects rather than as arrays), that is not always possible. Sometimes the particular property may not exist until run time, or there may be no way to know which one it will be in advance. For example, if the addressBook array were arranged by last name instead of by numbered listings, the user would probably be entering names "on the fly," while the script is running, to look people up. The following example assumes the existence of appropriate function definitions elsewhere in the script.

theListing = addressBook[getName

theIndivListing = theListing[getFire

This is *associative* addressing of the array, that is, addressing by means of fully arbitrary strings. Objects in JScript are actually associative arrays. Although you can (and frequently do) use the "dot" style of addressing, you are not by any means required to. Because the members of any JScript object can be accessed using array notation, a JScript object can be used as an associative array.

The following code creates and initializes the most familiar form of an array:

var myArray = new Array("Athens

Each element of this array is addressed using its element number; in this case 0, 1, or 2. Using the **for...in** statement, the array can be iterated starting at 0 and ending at 2. For example:

for (key in myArray) response.write("Element value is '

The following code creates and initializes an associative array containing three elements:

In this array, elements are addressed using the key strings("a", "b", or "c") instead of an array element number (0, 1, or 2). This

allows you to create and use arrays with more intuitive addressing schemes. The same **for...in** statement code shown above can be used to iterate this array as well.

Microsoft® JScript® Advanced Object Creation

Using Constructors to Create Objects

In Microsoft JScript, you use constructors to create and build a class of <u>objects</u>. You invoke a constructor with the **new** statement. It returns whatever it constructs.

The special case Function constructor lets you create functions that are *anonymous*. An anonymous function is one that does not have a name. You can use the Function constructor to build a function "on the fly", for example, as one of the instructions inside another function. Such a function, which is only called from the one location, doesn't need a name.

In the following example, such an anonymous function generates one line of a "name-and-email-address" listing. It checks the value of the <code>firstNameFirst</code> variable to decide whether to put the first name or the last name first, and the value of the <code>emailNameFirst</code> variable to decide whether to put the name or the email address first. The example assumes that the values of <code>firstNameFirst</code> and <code>emailNameFirst</code> are set elsewhere.

```
for (j = 1; j < addressList[length]; j
{
  oneListingLine = new Function(em
  if(firstNameFirst)
      {
      theName=(addressList[j].firstN
      },);</pre>
```

```
if (emailNameFirst)
    {
    theListing = addressList[j].emailNa
    } else theListing = theName + '
    document.write(oneListingLine + ''
}
```

Writing Constructors

To write your own constructors, you use the **this** keyword within the constructor to refer to the newly-created object. The constructor initializes the object.

Though the constructor in the next example starts at an index of 0, this is not required. You can start with a first index of 1 if, for example, you want a parameter that indicates the actual number of indexes of the array or object. In the example, it's called *extent* to distinguish it from the automatically maintained length parameter of the built-in **Array()** object). If you write code that adds properties to the array, you have to update the *extent* parameter (or your equivalent) because this parameter is not maintained by JScript. Notice that even this extremely simple example uses both object (dot) and array (bracket) notation styles to refer to the current object.

```
function MakeStringArray(length)
this.extent = length;
for (iNum = 0; iNum < length; i++)
this[iNum] = "";</pre>
```

} }

// Use the constructor to create and myStringArray = new MakeString/

Using Prototypes to Create Objects

When you write an object definition, you can use *prototype* properties to create properties that are held in common by all objects that are generated by the definition. Prototype properties are copied by reference into each object of a class, so they have the same value for all objects in the class. However, you can change the value of a prototype property in one object, and the new value overrides the default, but only in that one instance. Other objects that are members of the class are not affected by the change.

Using this principle, you can define additional properties for objects that are part of the JScript language, which all have prototypes. For example, if you want a special constant for a calculation, and the constant is not among those provided in the **Math** and **Number** objects, you can define it yourself and then assign it their respective object prototypes, or the prototype property of your object class.

Math.prototype.Avogadro = 6.0232 function howManyMolecules(wtGreturn ((wtGrams/molWt)*Math.pr

```
document.write("There are " + how ("What's the molecular weight?",0) " molecules in that amount.");
```

Perhaps more to the point, you can define a function, assign it to String.prototype as a method, and use it on any string anywhere in your script. The following example assumes the existence of a Periodic Chart array called "theElements", defined elsewhere in the script, which contains symbols for the elements, their names, their atomic weights, and other relevant information about them.

```
function atomName(theSymbol) {
return(theElements[theSymbol].ful)
}
```

String.prototype.atomName = atom

function decodeFormula(theFormula)
var theCurrentPiece = "";

```
var theDecodedFormula = "";
for (i = 1; i = theFormula.length; i-
if (theFormtheCurrentPiece
// Code statements to separate the fe
// Loop through the formula array a
theDecodedFormula += formula[n]
theDecodedFormula += " ";
theDecodedFormula += formula[n]
theDecodedFormula += " "
// End of loop.
return theDecodedFormula;
```

decodeFormula(window.prompt("F

Microsoft® JScript® Special Characters

Special Characters

JScript provides special characters that allow you to include in strings some characters you can't type directly. Each of these characters begins with a backslash. The backslash is an *escape* character you use to inform the JScript interpreter that the next character is special.

Escape Sequence	Character
\b	Backspace
\f	Form feed
\n	Line feed (newline)
\r	Carriage return
\t	Horizontal tab (Ctrl-I)
\'	Single quotation mark
\''	Double quotation mark
\\	Backslash

Notice that because the backslash itself is used as the escape character, you cannot directly type one in your script. If you want to write a backslash, you must type two of them together (\\).

document.write('The image path is document.write('The caption reads,

You can use these escape sequences to control formatting of text inside <PRE> and <XMP> tags and, to some extent, inside alert, prompt, and confirm message boxes.

Microsoft® JScript® Troubleshooting Your Scripts

There are places in any programming language where you can get caught if you are not careful, and every language has specific surprises in it. Take, for example, the <u>null</u> value: The one in Microsoft JScript behaves differently than the **null** value in the C or C++ languages.

Here are some of the trouble areas that that you may run into as you write JScript scripts.

Syntax Errors

Because syntax is much more rigid in programming languages than in natural languages, it is important to pay strict attention to detail when you write scripts. If, for example, you mean for a particular parameter to be a string, you will run into trouble if you forget to enclose it in quotation marks when you type it.

Order of Script Interpretation

JScript interpretation is part of the your Web browser's HTML parsing process. So, if you place a script inside the <HEAD> tag in a document, it is interpreted before any of the <BODY> tag is examined. If you have objects that are created in the <BODY> tag, they do not exist at the time the <HEAD> is being parsed, and cannot be manipulated by the script.

Automatic Type Coercion

JScript is a loosely typed language with automatic coercion. Consequently, despite the fact that values having different types are not equal, the expressions in the following example evaluate to **true**.

Operator Precedence

When a particular operation is performed during the evaluation of an expression has more to do with <u>operator precedence</u> than with the location of the expression. Thus, in the following example, multiplication is performed before subtraction, even though the subtraction appears first in the expression.

theRadius = aPerimeterPoint - theCenterpoint *

Using for...in Loops with Objects

When you step through the properties of an object with a **for...in** loop, you cannot necessarily predict or control the order in which the fields of the object are assigned to the loop counter variable. Moreover, the order may be different in different implementations of the language.

with Keyword

The with statement is convenient for addressing properties that already exist in a specified object, but cannot be used to add properties to an object. To create new properties in an object, you must refer to the object specifically.

this Keyword

Although you use the **this** keyword inside the definition of an object, to refer to the object itself, you cannot ordinarily use **this** or similar keywords to refer to the currently executing function when that function is not an object definition. You can, if the function is to be assigned to an object as a method, use the **this** keyword within the function, to refer to the object.

Writing a Script That Writes a Script

The </SCRIPT> tag terminates the current script if the interpreter encounters it. To display "</SCRIPT>" itself, rewrite this as at least two strings, for example, "</SCR" and "IPT>", which you can then concatenate together in the statement that writes them out.

Implicit Window References

Because more than one window can be open at a time, any window reference that is implicit is taken to point to the current window. For other windows, you must use an explicit reference.

Microsoft® JScript® Displaying Information In the Browser

<u>JScript Tutorial</u> <u>Previous</u> Next

Microsoft JScript provides two ways to display data directly in your browser. You can use the **write()** and **writeln()**, which are methods of the **document** object. You can also display information in forms within the browser, and in <u>alert, prompt</u>, and <u>confirm</u> message boxes.

Using document.write() and document.writeln()

The most common way to display information is the **write()** method of the document object. It takes one argument, a string, which it displays in the browser. The string can be either plain text or HTML.

Strings can be enclosed in either single or double quotation marks. This lets you quote something that contains quote marks or apostrophes.

document.write("Pi is approximate document.write();

Tip The following simple function is a way around havin something to appear in the browser window. This functior write is undefined, but does let you issue the command "w

function w(m) { // Write function

The **writeln()** method is almost identical to the **write()** method, except that it appends a newline character to whatever string you provide. In HTML this ordinarily results only in a space after your item; but if you're using <PRE> and <XMP> tags, the newline character is interpreted literally and the browser displays it.

When you call the **write()** method, it opens and clears the document if the document is not in the process of being opened and parsed when the **write()** method is called, so it can be dangerous. The example shows a script that is intended to display the time once a minute, but fails to do so after the first time because it clears itself in the process.



```
<SCRIPT LANGUAGE="JScript">
function singOut() {
var theMoment = new Date();
var theHour = theMoment.getHour
var theMinute = theMoment.getMi
var theDisplacement = (theMoment
theHour -= theDisplacement;
if (theHour > 23) {
theHour -= 24
document.write(theHour + " hours,
window.setTimeout("singOut();", 6
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT>
```

```
singOut();
</SCRIPT>
</BODY>
</HTML>
```

If you use the **alert()** method of the window object instead of **document.write()**, the script works.

```
window.alert(theHour + " hours, " -
window.setTimeout("singOut();", 6
}
```

Clearing the Current Document

The **clear()** method of the **document** object empties the current document. This method also clears your script (along with the rest of the document), so be very careful how and when you use it.

document.clear();

Microsoft® JScript® Scripting Run-Time Library Reference

Welcome to the Scripting Run-Time Library Reference

These handy blocks of information will help you explore the many different parts of the Scripting Run-Time Library.

You'll find *all* the parts of the Scripting Run-Time Library listed alphabetically under the Alphabetic Keyword List. But if you want to examine just one category, say, objects, each language category has its own, more compact section.

How's it work? Click on one of the headings to the left to display a list of items contained in that category. From this list, select the topic that you want to view. Once you've opened that topic, you can easily link to other related sections.

So, go ahead and take a look! Study some statements, mull over the methods, or figure out a few functions. You'll see just how versatile the Scripting Run-Time Library can be!

- Feature Information
- Alphabetic Keyword List
- Methods
- Objects
- Properties

Features Description

Microsoft Scripting Run-Time Features

List of features currently in Microsoft Scripting Run-Time Library.

© 2000 Microsoft Corporation. All rights reserved.

FileSystemObject Object Model

When writing scripts for Active Server Pages, the Windows Scripting Host, or other applications where scripting can be used, it's often important to add, move, change, create, or delete folders (directories) and files on the Web server. It may also be necessary to get information about and manipulate drives attached to the Web server.

Scripting allows you to process drives, folders, and files using the **FileSystemObject** (FSO) object model, which is explained in the following sections:

- Introduction to the FileSystemObject and the Scripting Run-Time Library Reference
- FileSystemObject Objects
- Programming the FileSystemObject
- Working with Drives and Folders
- Working with Files
- FileSystemObject Sample Code

Introduction to the FileSystemObject and the Scripting Run-Time Library Reference

Previous Next

The **FileSystemObject** (FSO) object model allows you to use the familiar *object.method* syntax with a rich set of properties, methods, and events to process folders and files.

Use this object-based tool with:

- HTML to create Web pages
- Windows Scripting Host to create batch files for Microsoft Windows
- Script Control to provide a scripting capability to applications developed in other languages

Because use of the FSO on the client side raises serious security issues about providing potentially unwelcome access to a client's local file system, this documentation assumes use of the FSO object model to create scripts executed by Internet Web pages on the server side. Since the server side is used, the Internet Explorer default security settings do not allow client-side use of the **FileSystemObject** object. Overriding those defaults could subject a local computer to unwelcome access to the file system, which could result in total destruction of the file system's integrity, causing loss of data, or worse.

The FSO object model gives your server-side applications the ability to create, alter, move, and delete folders, or to detect if particular folders exist, and if so, where. You can also find out information about folders, such as their names, the date they were created or last modified, and so forth.

The FSO object model also makes it easy to process files. When processing files, the primary goal is to store data in a space- and resource-efficient, easy-to-access format. You need to be able to create files, insert and change the data, and output (read) the data. Since storing data in a database, such as Access or SQL Server, adds a significant amount of overhead to your application, storing your data in a binary or text file may be the most efficient solution. You may prefer not to have this overhead, or your data access requirements may not require all the extra features associated with a full-featured database.

The FSO object model, which is contained in the Scripting type library (Scrrun.dll), supports text file creation and manipulation through the **TextStream** object. Although it does not yet support the creation or manipulation of binary files, future support of binary files is planned.

FileSystemObject Objects

The **FileSystemObject** (FSO) object model contains the following objects and <u>collections</u>.

Object/Collection	Description
FileSystemObject	Main object. Contains methods and properties that allow you to create, delete, gain information about, and generally manipulate drives, folders, and files. Many of the methods associated with this object duplicate those in other FSO objects; they are provided for convenience.
Drive	Object. Contains methods and properties that allow you to gather information about a drive attached to the system, such as its share name and how much room is available. Note that a "drive" isn't necessarily a

	hard disk, but can be a CD-ROM drive, a RAM disk, and so forth. A drive doesn't need to be physically attached to the system; it can be also be logically connected through a network.
Drives	Collection. Provides a list of the drives attached to the system, either physically or logically. The Drives collection includes all drives, regardless of type. Removable-media drives need not have media inserted for them to appear in this collection.
File	Object. Contains methods and properties that allow you to create, delete, or move a file. Also allows you to query the system for a file name, path, and various other properties.
Files	Collection. Provides a list of all files contained within

	a folder.
	Object. Contains methods
	and properties that allow
	you to create, delete, or
Folder	move folders. Also allows
	you to query the system for
	folder names, paths, and
	various other properties.
	Collection. Provides a list
Folders	of all the folders within a
	Folder.
TextStream	Object. Allows you to read
	and write text files.

Programming the FileSystemObject

To program with the **FileSystemObject** (FSO) object model:

- Use the **CreateObject** method to create a **FileSystemObject** object.
- Use the appropriate method on the newly created object.
- Access the object's properties.

The FSO object model is contained in the Scripting type library, which is located in the Scrrun.dll file. Therefore, you must have Scrrun.dll in the appropriate system directory on your Web server to use the FSO object model.

Creating a FileSystemObject Object

First, create a **FileSystemObject** object by using the **CreateObject** method. In VBScript, use the following code to create an instance of the **FileSystemObject**:

Dim fso
Set fso = CreateObject("Scripting.FileSystemObject")

This <u>sample code</u> demonstrates how to create an instance of the **FileSystemObject**.

In JScript, use this code to do the same thing:

var fso;
fso = new ActiveXObject("Scripting.FileSystemObject")

In both of these examples, **Scripting** is the name of the type library and **FileSystemObject** is the name of the object that you want to create. You can create only one instance of the **FileSystemObject** object, regardless of how many times you try to create another.

Using the Appropriate Method

Second, use the appropriate method of the **FileSystemObject** object. For example, to create a new object, use either **CreateTextFile** or **CreateFolder** (the FSO object model doesn't support the creation or deletion of drives).

To delete objects, use the **DeleteFile** and **DeleteFolder** methods of the **FileSystemObject** object, or the **Delete** method of the **File** and **Folder** objects. You can also copy and move files and folders, by using the appropriate methods.

Note Some functionality in the **FileSystemObject** object model is redundant. For example, you can copy a file using either the **CopyFile** method of the **FileSystemObject** object, or you can use the **Copy** method of the **File** object. The methods work the same; both exist to offer programming flexibility.

Accessing Existing Drives, Files, and Folders

To gain access to an existing drive, file, or folder, use the appropriate "get" method of the **FileSystemObject** object:

- GetDrive
- GetFolder
- GetFile

To gain access to an existing file in VBScript:

```
Dim fso, f1
Set fso = CreateObject("Scripting.FileSystemObject")
Set f1 = fso.GetFile("c:\test.txt")
```

To do the same thing in JScript, use the following code:

```
var fso, f1;
fso = new ActiveXObject("Scripting.FileSystemObjec
f1 = fso.GetFile("c:\\test.txt");
```

Do not use the "get" methods for newly created objects, since the "create" functions already return a handle to that object. For example, if you create a new folder using the **CreateFolder** method, don't use the **GetFolder** method to access its properties, such as **Name**, **Path**, **Size**, and so forth. Just set a variable to the **CreateFolder** function to gain a handle to the newly created folder, then access its properties, methods, and events. To do this in VBScript, use the following code:

```
Sub CreateFolder

Dim fso, fldr

Set fso = CreateObject("Scripting.FileSystemObject'

Set fldr = fso.CreateFolder("C:\MyTest")

Response.Write "Created folder: " & fldr.Name
```

End Sub

To set a variable to the **CreateFolder** function in JScript, use this syntax:

```
function CreateFolder()
{
  var fso, fldr;
  fso = new ActiveXObject("Scripting.FileSystemObjetfldr = fso.CreateFolder("C:\\MyTest");
  Response.Write("Created folder: " + fldr.Name);
}
```

Accessing the Object's Properties

Once you have a handle to an object, you can access its properties. For example, to get the name of a particular folder, first create an instance of the object, then get a handle to it with the appropriate method (in this case, the **GetFolder** method, since the folder already exists).

Use this code to get a handle to the **GetFolder** method in VBScript:

```
Set fldr = fso.GetFolder("c:\")
```

To do the same thing in JScript, use the following code:

```
var fldr = fso.GetFolder("c:\\");
```

Now that you have a handle to a **Folder** object, you can check its **Name** property. Use the following code to check this in VBScript:

Response.Write "Folder name is: " & fldr.Name

To check a **Name** property in JScript, use this syntax:

```
Response.Write("Folder name is: " + fldr.Name);
```

To find out the last time a file was modified, use the following VBScript syntax:

```
Dim fso, f1
```

Set fso = CreateObject("Scripting.FileSystemObject")

' Get a File object to query.

Set f1 = fso.GetFile("c:\detlog.txt")

' Print information.

Response.Write "File last modified: " & f1.DateLastM

To find out the same thing in JScript, use this code:

```
var fso, f1;
fso = new ActiveXObject("Scripting.FileSystemObjec
// Get a File object to query.
f1 = fso.GetFile("c:\\detlog.txt");
// Print information.
Response.Write("File last modified: " + f1.DateLastM
```

Working with Drives and Folders

With the **FileSystemObject** (FSO) object model, you can work with drives and folders programmatically just as you can in the Windows Explorer interactively. You can copy and move folders, get information about drives and folders, and so forth.

Getting Information About Drives

The **Drive** object allows you to gain information about the various drives attached to a system, either physically or over a network. Its properties allow you to obtain information about:

- The total size of the drive in bytes (**TotalSize** property)
- How much space is available on the drive in bytes (AvailableSpace or FreeSpace properties)
- What letter is assigned to the drive (**DriveLetter** property)
- What type of drive it is, such as removable, fixed, network,
 CD-ROM, or RAM disk (**DriveType** property)
- The drive's serial number (**SerialNumber** property)
- The type of file system the drive uses, such as FAT, FAT32, NTFS, and so forth (**FileSystem** property)
- Whether a drive is available for use (**IsReady** property)
- The name of the share and/or volume (ShareName and VolumeName properties)

• The path or root folder of the drive (**Path** and **RootFolder** properties)

View the <u>sample code</u> to see how these properties are used in **FileSystemObject**.

Example Usage of the Drive Object

Use the **Drive** object to gather information about a drive. You won't see a reference to an actual **Drive** object in the following code; instead, use the **GetDrive** method to get a reference to an existing **Drive** object (in this case, drv).

The following example demonstrates how to use the **Drive** object in VBScript:

```
Sub ShowDriveInfo(drvPath)
Dim fso, drv, s
Set fso = CreateObject("Scripting.FileSystemObject'
Set drv = fso.GetDrive(fso.GetDriveName(drvPath))
s = "Drive " & UCase(drvPath) & " - "
s = s & drv.VolumeName & "<br/>s = s & "Total Space: " & FormatNumber(drv.TotalSis = s & "Kb" & "<br/>s = s & "Free Space: " & FormatNumber(drv.FreeSpace = s & "Kb" & "<br/>s = s & "K
```

The following code illustrates the same functionality in JScript:

```
function ShowDriveInfo1(drvPath)
{
  var fso, drv, s ="";
  fso = new ActiveXObject("Scripting.FileSystemObject drv = fso.GetDrive(fso.GetDriveName(drvPath));
  s += "Drive" + drvPath.toUpperCase()+" - ";
  s += drv.VolumeName + "<br/>
  s += "Total Space: " + drv.TotalSize / 1024;
  s += "Kb" + "<br/>
  s += "Free Space: " + drv.FreeSpace / 1024;
  s += "Kb" + "<br/>
  r Response.Write(s);
}
```

Working with Folders

Common folder tasks and the methods for performing them are described in the following table.

Task	Method
Create a folder.	FileSystemObject.CreateFolder
Delete a folder.	Folder.Delete or FileSystemObject.DeleteFolder
Move a folder.	Folder.Move or FileSystemObject.MoveFolder
Copy a folder.	Folder.Copy or FileSystemObject.CopyFolder
Retrieve the name of a folder.	Folder.Name
Find out if a folder	

exists on a drive.	FileSystemObject.FolderExists
Get an instance of an existing Folder object.	FileSystemObject.GetFolder
folder.	FileSystemObject.GetParentFolderName
Find out the path of system folders.	FileSystemObject.GetSpecialFolder

View the <u>sample code</u> to see how many of these methods and properties are used in **FileSystemObject**.

The following example demonstrates how to use the **Folder** and **FileSystemObject** objects to manipulate folders and gain information about them in VBScript:

Sub ShowFolderInfo()

Dim fso, fldr, s

'Get instance of FileSystemObject.

Set fso = CreateObject("Scripting.FileSystemObject"

' Get Drive object.

Set fldr = fso.GetFolder("c:")

' Print parent folder name.

Response.Write "Parent folder name is: " & fldr & " < ' Print drive name.

Response.Write "Contained on drive " & fldr.Drive & 'Print root file name.

If fldr.IsRootFolder = True Then

Response.Write "This is the root folder." & ""
Else

```
Response.Write "This folder isn't a root folder." & "
   End If
   'Create a new folder with the FileSystemObject obje
   fso.CreateFolder ("C:\Bogus")
   Response.Write "Created folder C:\Bogus" & "<br>"
   ' Print the base name of the folder.
   Response.Write "Basename = " & fso.GetBaseName
   ' Delete the newly created folder.
   fso.DeleteFolder ("C:\Bogus")
   Response.Write "Deleted folder C:\Bogus" & "<br>"
  End Sub
This example shows how to use the Folder and
FileSystemObject objects in JScript:
  function ShowFolderInfo()
   var fso, fldr, s = "";
   // Get instance of FileSystemObject.
   fso = new ActiveXObject("Scripting.FileSystemObje
   // Get Drive object.
   fldr = fso.GetFolder("c:");
   // Print parent folder name.
   Response.Write("Parent folder name is: " + fldr + "<
   // Print drive name.
   Response.Write("Contained on drive " + fldr.Drive +
   // Print root file name.
   if (fldr.IsRootFolder)
```

```
Response.Write("This is the root folder.");
else
Response.Write("This folder isn't a root folder.");
Response.Write("<br/>
'| Create a new folder with the FileSystemObject obj
fso.CreateFolder ("C:\\Bogus");
Response.Write("Created folder C:\\Bogus" + "<br/>
| Print the base name of the folder.
Response.Write("Basename = " + fso.GetBaseName()
| Delete the newly created folder.
fso.DeleteFolder ("C:\\Bogus");
Response.Write("Deleted folder C:\\Bogus" + "<br/>
| Response.Write("Deleted folder C:\\Bogus + "<br/>
```

Working with Files

There are two major categories of file manipulation:

- Creating, adding, or removing data, and reading files
- Moving, copying, and deleting files

Creating Files

There are three ways to create an empty text file (sometimes referred to as a "text stream").

The first way is to use the **CreateTextFile** method. The following example demonstrates how to create a text file using this method in VBScript:

```
Dim fso, f1
Set fso = CreateObject("Scripting.FileSystemObject")
Set f1 = fso.CreateTextFile("c:\testfile.txt", True)
```

To use this method in JScript, use this code:

```
var fso, f1;
fso = new ActiveXObject("Scripting.FileSystemObjectf1 = fso.CreateTextFile("c:\\testfile.txt", true);
```

View this <u>sample code</u> to see how the **CreateTextFile** method is used in **FileSystemObject**.

The second way to create a text file is to use the **OpenTextFile** method of the **FileSystemObject** object with the **ForWriting** flag set. In VBScript,

the code looks like this example:

```
Dim fso, ts

Const ForWriting = 2

Set fso = CreateObject("Scripting. FileSystemObject"

Set ts = fso.OpenTextFile("c:\test.txt", ForWriting, Tri
```

To create a text file using this method in JScript, use this code:

```
var fso, ts;
var ForWriting= 2;
fso = new ActiveXObject("Scripting.FileSystemObjects = fso.OpenTextFile("c:\\test.txt", ForWriting, true);
```

A third way to create a text file is to use the **OpenAsTextStream** method with the **ForWriting** flag set. For this method, use the following code in VBScript:

```
Dim fso, f1, ts

Const ForWriting = 2

Set fso = CreateObject("Scripting.FileSystemObject")

fso.CreateTextFile ("c:\test1.txt")

Set f1 = fso.GetFile("c:\test1.txt")

Set ts = f1.OpenAsTextStream(ForWriting, True)
```

In JScript, use the code in the following example:

```
var fso, f1, ts;
var ForWriting = 2;
fso = new ActiveXObject("Scripting.FileSystemObject")
```

```
fso.CreateTextFile ("c:\\test1.txt");
f1 = fso.GetFile("c:\\test1.txt");
ts = f1.OpenAsTextStream(ForWriting, true);
```

Adding Data to the File

Once the text file is created, add data to the file using the following three steps:

- 1. Open the text file.
- 2. Write the data.
- 3. Close the file.

To open an existing file, use either the **OpenTextFile** method of the **FileSystemObject** object or the **OpenAsTextStream** method of the **File** object.

To write data to the open text file, use the **Write**, **WriteLine**, or **WriteBlankLines** methods of the **TextStream** object, according to the tasks outlined in the following table.

Task	Method
Write data to an open text file without a trailing newline character.	Write
Write data to an open text file with a trailing newline character.	
Write one or more blank lines to an open text file.	WriteBlankLines

View this <u>sample code</u> to see how the **Write**, **WriteLine**, and **WriteBlankLines** methods are used in **FileSystemObject**.

To close an open file, use the **Close** method of the **TextStream** object.

View this <u>sample code</u> to see how the **Close** method is used in **FileSystemObject**.

Note The newline character contains a character or characters (depending on the operating system) to advance the cursor to the beginning of the next line (carriage return/line feed). Be aware that the end of some strings may already have such nonprinting characters.

The following VBScript example demonstrates how to open a file, use all three write methods to add data to the file, and then close the file:

```
Sub CreateFile()
Dim fso, tf
Set fso = CreateObject("Scripting.FileSystemObject'
Set tf = fso.CreateTextFile("c:\testfile.txt", True)
' Write a line with a newline character.
tf.WriteLine("Testing 1, 2, 3.")
' Write three newline characters to the file.
tf.WriteBlankLines(3)
' Write a line.
tf.Write ("This is a test.")
tf.Close
End Sub
```

This example demonstrates how to use the three methods in JScript:

```
function CreateFile()
{
  var fso, tf;
  fso = new ActiveXObject("Scripting.FileSystemObjetf = fso.CreateTextFile("c:\\testfile.txt", true);
  // Write a line with a newline character.
  tf.WriteLine("Testing 1, 2, 3.");
  // Write three newline characters to the file.
  tf.WriteBlankLines(3);
  // Write a line.
  tf.Write ("This is a test.");
  tf.Close();
}
```

Reading Files

To read data from a text file, use the **Read**, **ReadLine**, or **ReadAll** method of the **TextStream** object. The following table describes which method to use for various tasks.

Task	Method
Read a specified number of characters from a file.	Read
Read an entire line (up to, but not including, the newline character).	ReadLine
Read the entire contents of a text file.	ReadAll

View this <u>sample code</u> to see how the **ReadAll** and **ReadLine** methods are used in **FileSystemObject**.

If you use the **Read** or **ReadLine** method and want to skip to a particular

portion of data, use the **Skip** or **SkipLine** method. The resulting text of the read methods is stored in a string which can be displayed in a control, parsed by string functions (such as **Left**, **Right**, and **Mid**), concatenated, and so forth.

The following VBScript example demonstrates how to open a file, write to it, and then read from it:

```
Sub ReadFiles
   Dim fso, f1, ts, s
   Const ForReading = 1
   Set fso = CreateObject("Scripting.FileSystemObject"
   Set f1 = fso.CreateTextFile("c:\testfile.txt", True)
   ' Write a line.
   Response.Write "Writing file <br>"
   f1.WriteLine "Hello World"
   f1.WriteBlankLines(1)
   f1.Close
   ' Read the contents of the file.
   Response.Write "Reading file <br/> ''
   Set ts = fso.OpenTextFile("c:\testfile.txt", ForReadin
   s = ts.ReadLine
   Response.Write "File contents = " & s & ""
   ts.Close
  End Sub
This code demonstrates the same thing in JScript:
```

function ReadFiles()

```
var fso, f1, ts, s;
var ForReading = 1;
fso = new ActiveXObject("Scripting.FileSystemObje
f1 = fso.CreateTextFile("c:\\testfile.txt", true);
// Write a line.
Response.Write("Writing file <br>
Response.Write("Hello World");
f1.WriteBlankLines(1);
f1.Close();
// Read the contents of the file.
Response.Write("Reading file <br/>');
ts = fso.OpenTextFile("c:\\testfile.txt", ForReading);
s = ts.ReadLine();
Response.Write("File contents = "" + s + """);
ts.Close();
}
```

Moving, Copying, and Deleting Files

The FSO object model has two methods each for moving, copying, and deleting files, as described in the following table.

Task	Method
Move a file	File.Move or FileSystemObject.MoveFile
Copy a file	File.Copy or FileSystemObject.CopyFile
Delete a file	File.Delete or FileSystemObject.DeleteFile

View this <u>sample code</u> to see two ways to delete a file in **FileSystemObject**.

The following VBScript example creates a text file in the root directory of drive C, writes some information to it, moves it to a directory called \tmp, makes a copy of it in a directory called \temp, then deletes the copies from both directories.

To run the following example, create directories named \tmp and \temp in the root directory of drive C:

```
Sub ManipFiles
 Dim fso, f1, f2, s
 Set fso = CreateObject("Scripting.FileSystemObject"
 Set f1 = fso.CreateTextFile("c:\testfile.txt", True)
 Response.Write "Writing file <br/> ''
 ' Write a line.
 f1.Write ("This is a test.")
 'Close the file to writing.
 f1.Close
 Response.Write "Moving file to c:\tmp <br/> <br/> "
 'Get a handle to the file in root of C:\.
 Set f2 = fso.GetFile("c:\testfile.txt")
 ' Move the file to \tmp directory.
 f2.Move ("c:\tmp\testfile.txt")
 Response.Write "Copying file to c:\temp <br/> <br/>"
 'Copy the file to \temp.
 f2.Copy ("c:\temp\testfile.txt")
 Response.Write "Deleting files <br/> <br/> "
 'Get handles to files' current location.
 Set f2 = fso.GetFile("c:\tmp\testfile.txt")
 Set f3 = fso.GetFile("c:\temp\testfile.txt")
```

```
' Delete the files.
   f2.Delete
   f3.Delete
   Response.Write "All done!"
  End Sub
This code shows the same thing in JScript:
  function ManipFiles()
   var fso, f1, f2, s;
   fso = new ActiveXObject("Scripting.FileSystemObje
   f1 = fso.CreateTextFile("c:\\testfile.txt", true);
   Response.Write("Writing file <br/> '');
   // Write a line.
   f1.Write("This is a test.");
   // Close the file to writing.
   f1.Close();
   Response.Write("Moving file to c:\\tmp <br/>');
   // Get a handle to the file in root of C:\.
   f2 = fso.GetFile("c:\\testfile.txt");
   // Move the file to \tmp directory.
   f2.Move ("c:\\tmp\\testfile.txt");
   Response.Write("Copying file to c:\\temp <br/>br>");
   // Copy the file to \temp.
   f2.Copy ("c:\\temp\\testfile.txt");
   Response.Write("Deleting files <br/> '');
   // Get handles to files' current location.
```

```
f2 = fso.GetFile("c:\\tmp\\testfile.txt");
f3 = fso.GetFile("c:\\temp\\testfile.txt");
// Delete the files.
f2.Delete();
f3.Delete();
Response.Write("All done!");
}
```

FileSystemObject Sample Code

The sample code described in this section provides a real-world example that demonstrates many of the features available in the **FileSystemObject** object model. This code shows how all the features of the object model work together, and how to use those features effectively in your own code.

Note that since this code is fairly generic, some additional code and a little tweaking are needed to make this code actually run on your machine. These changes are necessary because of the different ways input and output to the user is handled between Active Server Pages and the Windows Scripting Host.

To run this code on an Active Server Page, use the following steps:

- 1. Create a standard Web page with an .asp extension.
- 2. Copy the following sample code into that file between the <BODY;>...</BODY> tags.
- 3. Enclose all the code within <%...%> tags.
- 4. Move the **Option Explicit** statement from its current position in the code to the very top of your HTML page, positioning it even before the opening <HTML> tag.
- 5. Place <%...%> tags around the **Option Explicit** statement to ensure that it's run on the server side.
- 6. Add the following code to the end of the sample code:

Sub Print(x)

```
Response.Write "<PRE><FONT; FACE=""Courier New"" SIZE=""1"">"
Response.Write x
Response.Write "</FONT></PRE>"
End Sub
Main
```

The previous code adds a print procedure that will run on the server side, but display results on the client side. To run this code on the Windows Scripting Host, add the following code to the end of the sample code:

```
Sub Print(x)
WScript.Echo x
End Sub
Main
```

The code is contained in the following section:

······································
' FileSystemObject Sample Code
Copyright 1998 Microsoft Corporation. All Rights Reserved.
Option Explicit
•
' Regarding code quality:

' 1) The following code does a lot of string manipulation by concatenating sho

is expensive, this is a very inefficient way to write code. However, it is a very maintainable way to write code, and is used here because this program perfection.

strings together with the "&" operator. Since string concatenation

- disk operations, and because the disk is much slower than the memory oper concatenate the strings. Keep in mind that this is demonstration code, not p
- ' 2) "Option Explicit" is used, because declared variable access is slightly faste undeclared variable access. It also prevents bugs from creeping into your cc when you misspell DriveTypeCDROM as DriveTypeCDORM.
- ' 3) Error handling is absent from this code, to make the code more readable. A precautions have been taken to ensure that the code will not error in commc systems can be unpredictable. In production code, use On Error Resume Ne Err object to trap possible errors.

'Some handy global variables

'Indiana TabStop
Dim TabStop
Dim NewLine

Const TestDrive = "C"
Const TestFilePath = "C:\Test"

'Constants returned by Drive.DriveType

'Indiana TabStop
Dim NewLine

Const TestDrive = "C"
Const TestFilePath = "C:\Test"

Const DriveTypeRemovable = 1 Const DriveTypeFixed = 2 Const DriveTypeNetwork = 3 Const DriveTypeCDROM = 4 Const DriveTypeRAMDisk = 5

```
......
'Constants returned by File.Attributes
Const FileAttrNormal = 0
Const FileAttrReadOnly = 1
Const FileAttrHidden = 2
Const FileAttrSystem = 4
Const FileAttrVolume = 8
Const FileAttrDirectory = 16
Const FileAttrArchive = 32
Const FileAttrAlias = 64
Const FileAttrCompressed = 128
......
'Constants for opening files
......
Const OpenFileForReading = 1
Const OpenFileForWriting = 2
Const OpenFileForAppending = 8
......
'ShowDriveType
' Purpose:
'Generates a string describing the drive type of a given Drive object.
' Demonstrates the following
```

```
' - Drive.DriveType
Function ShowDriveType(Drive)
    Dim S
    Select Case Drive.DriveType
    Case DriveTypeRemovable
        S = "Removable"
    Case DriveTypeFixed
        S = "Fixed"
    Case DriveTypeNetwork
        S = "Network"
    Case DriveTypeCDROM
        S = "CD-ROM"
    Case DriveTypeRAMDisk
        S = "RAM Disk"
    Case Else
        S = "Unknown"
    End Select
    ShowDriveType = S
End Function
......
'ShowFileAttr
' Purpose:
'Generates a string describing the attributes of a file or folder.
' Demonstrates the following
```

```
' - File. Attributes
' - Folder. Attributes
Function ShowFileAttr(File) 'File can be a file or folder
    Dim S
    Dim Attr
    Attr = File. Attributes
    If Attr = 0 Then
         ShowFileAttr = "Normal"
         Exit Function
    End If
    If Attr And FileAttrDirectory Then S = S & "Directory "
    If Attr And FileAttrReadOnly Then S = S & "Read-Only "
                               Then S = S \& "Hidden"
    If Attr And FileAttrHidden
                               Then S = S \& "System"
    If Attr And FileAttrSystem
                                Then S = S \& "Volume"
    If Attr And FileAttrVolume
    If Attr And FileAttrArchive Then S = S & "Archive"
                              Then S = S \& "Alias"
    If Attr And FileAttrAlias
    If Attr And FileAttrCompressed Then S = S & "Compressed "
    ShowFileAttr = S
End Function
'GenerateDriveInformation
```

' Purpose:

'Generates a string describing the current state of the available drives. ' Demonstrates the following ' - FileSystemObject.Drives ' - Iterating the Drives collection ' - Drives.Count ' - Drive. Available Space ' - Drive.DriveLetter ' - Drive.DriveType ' - Drive.FileSystem ' - Drive.FreeSpace ' - Drive.IsReady ' - Drive.Path ' - Drive.SerialNumber ' - Drive.ShareName ' - Drive.TotalSize ' - Drive.VolumeName Function GenerateDriveInformation(FSO) **Dim Drives** Dim Drive Dim S Set Drives = FSO.Drives S = "Number of drives:" & TabStop & Drives.Count & NewLine & Newl 'Construct 1st line of report. S = S & String(2, TabStop) & "Drive" S = S & String(3, TabStop) & "File" S = S & TabStop & "Total"

S = S & TabStop & "Free"

```
S = S & TabStop & "Available"
```

S = S & TabStop & "Serial" & NewLine

'Construct 2nd line of report.

S = S & "Letter"

S = S & TabStop & "Path"

S = S & TabStop & "Type"

S = S & TabStop & "Ready?"

S = S & TabStop & "Name"

S = S & TabStop & "System"

S = S & TabStop & "Space"

S = S & TabStop & "Space"

S = S & TabStop & "Space"

S = S & TabStop & "Number" & NewLine

' Separator line.

S = S & String(105, "-") & NewLine

For Each Drive In Drives

S = S & Drive.DriveLetter

S = S & TabStop & Drive.Path

S = S & TabStop & ShowDriveType(Drive)

S = S & TabStop & Drive.IsReady

If Drive.IsReady Then

If DriveTypeNetwork = Drive.DriveType Then

S = S & TabStop & Drive.ShareName

Else

S = S & TabStop & Drive.VolumeName End If

S = S & TabStop & Drive.FileSystem

S = S & TabStop & Drive.TotalSize

S = S & TabStop & Drive.FreeSpace

S = S & TabStop & Drive.AvailableSpace

S = S & TabStop & Hex(Drive.SerialNumber)

```
S = S & NewLine
    Next
    GenerateDriveInformation = S
End Function
......
' GenerateFileInformation
' Purpose:
'Generates a string describing the current state of a file.
' Demonstrates the following
' - File.Path
' - File.Name
' - File.Type
' - File.DateCreated
' - File.DateLastAccessed
' - File.DateLastModified
' - File.Size
Function GenerateFileInformation(File)
    Dim S
    S = NewLine & "Path:" & TabStop & File.Path
    S = S & NewLine & "Name:" & TabStop & File.Name
```

End If

```
S = S & NewLine & "Type:" & TabStop & File.Type
    S = S & NewLine & "Attribs:" & TabStop & ShowFileAttr(File)
    S = S & NewLine & "Created:" & TabStop & File.DateCreated
    S = S & NewLine & "Accessed:" & TabStop & File.DateLastAccessed
    S = S & NewLine & "Modified:" & TabStop & File.DateLastModified
    S = S & NewLine & "Size" & TabStop & File.Size & NewLine
    GenerateFileInformation = S
End Function
......
'GenerateFolderInformation
' Purpose:
'Generates a string describing the current state of a folder.
' Demonstrates the following
' - Folder.Path
' - Folder.Name
' - Folder.DateCreated
' - Folder.DateLastAccessed
' - Folder.DateLastModified
' - Folder.Size
......
Function GenerateFolderInformation(Folder)
    Dim S
    S = "Path:" & TabStop & Folder.Path
    S = S & NewLine & "Name:" & TabStop & Folder.Name
```

```
S = S & NewLine & "Attribs:" & TabStop & ShowFileAttr(Folder)
    S = S & NewLine & "Created:" & TabStop & Folder.DateCreated
    S = S & NewLine & "Accessed:" & TabStop & Folder.DateLastAccessed
    S = S & NewLine & "Modified:" & TabStop & Folder.DateLastModified
    S = S & NewLine & "Size:" & TabStop & Folder.Size & NewLine
    GenerateFolderInformation = S
End Function
......
'GenerateAllFolderInformation
' Purpose:
'Generates a string describing the current state of a
' folder and all files and subfolders.
' Demonstrates the following
' - Folder.Path
' - Folder.SubFolders
' - Folders.Count
```

$Function\ Generate All Folder Information (Folder)$

Dim S
Dim SubFolders
Dim SubFolder
Dim Files
Dim File

S = "Folder:" & TabStop & Folder.Path & NewLine & NewLine

```
Set Files = Folder.Files
If 1 = Files. Count Then
    S = S & "There is 1 file" & NewLine
Else
    S = S & "There are " & Files. Count & " files" & NewLine
End If
If Files.Count <> 0 Then
    For Each File In Files
         S = S & GenerateFileInformation(File)
    Next
End If
Set SubFolders = Folder.SubFolders
If 1 = SubFolders.Count Then
    S = S & NewLine & "There is 1 sub folder" & NewLine & NewLine
Else
    S = S & NewLine & "There are " & SubFolders.Count & " sub folde
End If
If SubFolders.Count <> 0 Then
    For Each SubFolder In SubFolders
         S = S & GenerateFolderInformation(SubFolder)
    Next
    S = S \& NewLine
    For Each SubFolder In SubFolders
         S = S & GenerateAllFolderInformation(SubFolder)
    Next
```

End If

GenerateAllFolderInformation = S

End Function ' GenerateTestInformation ' Purpose: 'Generates a string describing the current state of the C:\Test ' folder and all files and subfolders. ' Demonstrates the following ' - FileSystemObject.DriveExists ' - FileSystemObject.FolderExists ' - FileSystemObject.GetFolder Function GenerateTestInformation(FSO) Dim TestFolder Dim S If Not FSO.DriveExists(TestDrive) Then Exit Function If Not FSO.FolderExists(TestFilePath) Then Exit Function Set TestFolder = FSO.GetFolder(TestFilePath) GenerateTestInformation = GenerateAllFolderInformation(TestFolder)

End Function

```
......
' DeleteTestDirectory
' Purpose:
' Cleans up the test directory.
' Demonstrates the following
' - FileSystemObject.GetFolder
' - FileSystemObject.DeleteFile
' - FileSystemObject.DeleteFolder
' - Folder.Delete
' - File.Delete
......
Sub DeleteTestDirectory(FSO)
    Dim TestFolder
    Dim SubFolder
    Dim File
    'Two ways to delete a file:
    FSO.DeleteFile(TestFilePath & "\Beatles\OctopusGarden.txt")
    Set File = FSO.GetFile(TestFilePath & "\Beatles\BathroomWindow.txt")
    File.Delete
    'Two ways to delete a folder:
    FSO.DeleteFolder(TestFilePath & "\Beatles")
```

```
Set TestFolder = FSO.GetFolder(TestFilePath)
    TestFolder.Delete
End Sub
......
' CreateLyrics
' Purpose:
'Builds a couple of text files in a folder.
' Demonstrates the following
' - FileSystemObject.CreateTextFile
' - TextStream.WriteLine
' - TextStream.Write
' - TextStream.WriteBlankLines
' - TextStream.Close
......
Sub CreateLyrics(Folder)
    Dim TextStream
    Set TextStream = Folder.CreateTextFile("OctopusGarden.txt")
    TextStream.Write("Octopus' Garden") ' Note that this does not add a line
    TextStream.WriteLine("(by Ringo Starr)")
    TextStream.WriteBlankLines(1)
    TextStream.WriteLine("I'd like to be under the sea in an octopus' garden i
    TextStream.WriteLine("He'd let us in, knows where we've been -- in his c
```

FSO.DeleteFile(TestFilePath & "\ReadMe.txt")

TextStream.WriteBlankLines(2)

TextStream.Close

Set TextStream = Folder.CreateTextFile("BathroomWindow.txt")

TextStream.WriteLine("She Came In Through The Bathroom Window (by TextStream.WriteLine("")

TextStream.WriteLine("She came in through the bathroom window protectextStream.WriteLine("But now she sucks her thumb and wanders by the TextStream.WriteBlankLines(2)

TextStream.Close

End Sub

'GetLyrics
'Purpose:
'Displays the contents of the lyrics files.
'Demonstrates the following
'-FileSystemObject.OpenTextFile
'-FileSystemObject.GetFile
'-TextStream.ReadAll
'-TextStream.Close
'-File.OpenAsTextStream
'-TextStream.AtEndOfStream
'-TextStream.ReadLine

Function GetLyrics(FSO)

Dim TextStream Dim S Dim File

'There are several ways to open a text file, and several ways to read the

' data out of a file. Here's two ways to do each:

Set TextStream = FSO.OpenTextFile(TestFilePath & "\Beatles\OctopusG

S = TextStream.ReadAll & NewLine & NewLine TextStream.Close

Set File = FSO.GetFile(TestFilePath & "\Beatles\BathroomWindow.txt")
Set TextStream = File.OpenAsTextStream(OpenFileForReading)
Do While Not TextStream.AtEndOfStream

S = S & TextStream.ReadLine & NewLine

Loop

TextStream.Close

GetLyrics = S

End Function

.....

'BuildTestDirectory

' Purpose:

'Builds a directory hierarchy to demonstrate the FileSystemObject.

' We'll build a hierarchy in this order:

'C:\Test

'C:\Test\ReadMe.txt

'C:\Test\Beatles

```
'C:\Test\Beatles\OctopusGarden.txt
'C:\Test\Beatles\BathroomWindow.txt
' Demonstrates the following
' - FileSystemObject.DriveExists
' - FileSystemObject.FolderExists
' - FileSystemObject.CreateFolder
' - FileSystemObject.CreateTextFile
' - Folders.Add
' - Folder.CreateTextFile
' - TextStream.WriteLine
' - TextStream.Close
......
Function BuildTestDirectory(FSO)
    Dim TestFolder
    Dim SubFolders
    Dim SubFolder
    Dim TextStream
    'Bail out if (a) the drive does not exist, or if (b) the directory being built
    ' already exists.
    If Not FSO.DriveExists(TestDrive) Then
         BuildTestDirectory = False
         Exit Function
    End If
    If FSO.FolderExists(TestFilePath) Then
         BuildTestDirectory = False
         Exit Function
```

End If

```
Set TestFolder = FSO.CreateFolder(TestFilePath)
    Set TextStream = FSO.CreateTextFile(TestFilePath & "\ReadMe.txt")
    TextStream.WriteLine("My song lyrics collection")
    TextStream.Close
    Set SubFolders = TestFolder.SubFolders
    Set SubFolder = SubFolders.Add("Beatles")
    CreateLyrics SubFolder
    BuildTestDirectory = True
End Function
.......
'The main routine
' First, it creates a test directory, along with some subfolders and files.
'Then, it dumps some information about the available disk drives and
' about the test directory, and then cleans everything up again.
......
Sub Main
    Dim FSO
    ' Set up global data.
    TabStop = Chr(9)
    NewLine = Chr(10)
    Set FSO = CreateObject("Scripting.FileSystemObject")
```

If Not BuildTestDirectory(FSO) Then

Print "Test directory already exists or cannot be created. Cannot con Exit Sub

End If

Print GenerateDriveInformation(FSO) & NewLine & NewLine

Print GenerateTestInformation(FSO) & NewLine & NewLine

Print GetLyrics(FSO) & NewLine & NewLine

DeleteTestDirectory(FSO)

End Sub

${\tt Microsoft \& JScript \& } \ Microsoft \ JScript \\$

Features ECMA

Language Reference

Category	Feature/Keyword	
Array Handling	Array join, length, reverse, sort	
Assignments	Assign (=) Compound Assign (OP=)	
Booleans	<u>Boolean</u>	
Comments	<u>/**/ or //</u>	
Constants/Literals	NaN null true, false Infinity undefined	
Control flow	break continue for forin ifelse return while	
Dates and Time	Date getDate, getDay, getFullYear, getHours, getMilliseconds, getMinutes, getMonth, getSeconds, getTime, getTimezoneOffset, getYear, getUTCDate, getUTCDay, getUTCFullYear, getUTCHours, getUTCMilliseconds, getUTCMinutes, getUTCMonth, getUTCSeconds, setDate, setFullYear, setHours, setMilliseconds, setMinutes, setMonth, setSeconds, setTime, setYear, setUTCDate, setUTCFullYear, setUTCHours,	

	setUTCMonth, setUTCSeconds,		
	toGMTString, toLocaleString, toUTCString,		
	parse, <u>UTC</u>		
l li	<u>function</u>		
l II	new this		
l II	<u>this</u> var		
l II	with		
ļ <u>ļ</u>	Function		
iriinction Creation 🔠	arguments, length		
	<u> </u>		
l li	Global		
	<u>escape, unescape</u> eval		
l II	isFinite, isNaN		
	parseInt, parseFloat		
	Math		
l II	abs, acos, asin, atan, atan2, ceil, cos, exp, floor,		
	log, max, min, pow, random, round, sin, sqrt, tan,		
	E, LN2, LN10, LOG2E, LOG10E, PI, SQRT1_2,		
2	SQRT2		
	Number		
Numbers	MAX VALUE, <u>MIN VALUE</u>		
]	NaN		
]	NEGATIVE_INFINITY, POSITIVE_INFINITY		
<u> </u>	<u>Object</u>		
	<u>new</u>		
<u>(</u>	constructor, prototype, toString, valueOf		
l li	Addition (+), Subtraction (-)		
	Modulus arithmetic (%)		
	Multiplication (*), Division (/)		
	Negation (-)		
	Equality (==), Inequality (!=)		
	Less Than (<), Less Than or Equal To (<=)		
	Greater Than (>) Greater Than or Equal To (>=)		
10 1	Logical And(&&), Or (), Not (!)		
	Bitwise And (&), Or (), Not (~), Xor (^)		
	Bitwise Left Shift (<<), Shift Right (>>)		
	Unsigned Shift Right (>>>)		

	Conditional (?:)	
	Comma (,)	
	delete, typeof, void	
	Decrement (), Increment (++)	
	Array	
	<u>Boolean</u>	
	<u>Date</u>	
Objects	<u>Function</u>	
	<u>Global</u>	
	<u>Math</u>	
	<u>Number</u>	
	<u>Object</u>	
	<u>String</u>	
	String	
Strings	charAt, charCodeAt, fromCharCode	
	indexOf, lastIndexOf	
	<u>split</u>	
	toLowerCase, toUpperCase	
	<u>length</u>	

${\tt Microsoft \& JScript \& } \ Microsoft \ JScript \\$

Features Non-ECMA

Language Reference

Category	Feature/Keyword	
Array Handling	concat, slice VBArray dimensions, getItem, lbound, toArray, ubound	
Conditional Compilation	@cc_on @if Statement @set Statement Conditional Compilation Variables	
Control flow	dowhile Labeled switch	
Dates and Time	<u>getVarDate</u>	
Enumeration	Enumerator atEnd, item, moveFirst, moveNext	
Error Handling	Error description, number throw, trycatch	
Function Creation	<u>caller</u>	
Operators	Identity (===), Nonidentity (!==)	
Objects	Enumerator RegExp Regular Expression VBArray ActiveXObject GetObject	
Regular	RegExp	

Expressions and	index, input, lastIndex, \$1\$9, source,	
Pattern Matching	<u>compile, exec, test</u>	
	Regular Expression Syntax	
	<u>ScriptEngine</u>	
Script Engine	<u>ScriptEngineBuildVersion</u>	
Indentification	<u>ScriptEngineMajorVersion</u>	
	<u>ScriptEngineMinorVersion</u>	
	concat, slice	
Strings	<u>match, replace, search</u>	
Strings	anchor, big, blink, bold, fixed, fontcolor,	
	fontsize, italics, link, small, strike, sub, sup	

Microsoft® JScript® Microsoft

Scripting Run-Time Features

Category	Feature/Keyword
Collections	Drives Files Folders
Data Storage	Dictionary
Dictionary	Add Exists Items, Keys Remove, RemoveAll Count Item, Key
File System	Drive File FileSystemObject Folder TextStream
FileSystemObject	BuildPath CopyFile, CopyFolder CreateFolder, CreateTextFile DeleteFile, DeleteFolder DriveExists, FileExists, FolderExists GetAbsolutePathName, GetBaseName GetDrive, GetDriveName GetFile, GetExtensionName GetFileName GetFolder, GetParentFolderName GetSpecialFolder GetTempName MoveFile, MoveFolder OpenTextFile

	<u>Drives</u>
	<u>AvailableSpace</u>
	<u>Count</u>
	<u>DriveLetter</u>
	<u>DriveType</u>
	<u>FileSystem</u>
	<u>FreeSpace</u>
Drive, Drives	<u>IsReady</u>
	<u>Item</u>
	<u>RootFolder</u>
	<u>SerialNumber</u>
	<u>ShareName</u>
	<u>TotalSize</u>
	<u>VolumeName</u>
	Add
	<u>Attributes</u>
	Copy, Delete, Move
	Count
	<u>OpenAsTextStream</u>
File, Files	DateCreated, DateLastAccessed,
Folder, Folders	<u>DateLastModified</u>
roidei, roideis	<u>Drive</u>
	<u>Item</u>
	<u>ParentFolder</u>
	Name, Path
	<u>ShortName, ShortPath</u>
	<u>Size</u>
	<u>Close</u>
	Read, ReadAll, ReadLine
TextStream	<u>Skip, SkipLine</u>
TextStream	Write, WriteBlankLines, WriteLine
	AtEndOfLine, AtEndOfStream
	Column, Line

See Also

Applies To

Description

Returns the absolute value of a number.

Syntax

Math.abs(number)

The *number* argument is a <u>numeric expression</u> for which the absolute value is sought.

Remarks

The return value is the absolute value of the *number* argument.

The following example illustrates the use of the **abs** method:

```
function ComparePosNegVal(n)
{
  var s;
  var v1 = Math.abs(n);
  var v2 = Math.abs(-n);
  if (v1 == v2)
    s = "The absolute values of " + n
```

```
s += -n + " are identical.";
return(s);
}
```

Microsoft® JScript® acos Method

See Also Applies To

Description

Returns the arccosine of a number.

Syntax

Math.acos(number)

The *number* argument is a <u>numeric expression</u> for which the arccosine is sought.

Remarks

The return value is the arccosine of the *number* argument.

ActiveXObject Object

See Also

Description

Enables and returns a reference to an Automation object.

Syntax

var newObject = new ActiveXObject("servername.typename"[,
"location"])

The **ActiveXObject** object syntax and has these parts:

Part	Description
COMIONN AMO	Required. The name of the application providing
servername	the object.
	Required. The type or class of the object to create.
logation	Optional. The name of the network server where
location	the object is to be created.

Remarks

Automation servers provide at least one type of object. For example, a word-processing application may provide an application object, a document object, and a toolbar object.

To create an Automation object, assign the new **ActiveXObject** to an object variable:

var ExcelSheet; ExcelSheet = new ActiveXObject("Excel.Sheet

This code starts the application creating the object (in this case, a Microsoft Excel worksheet). Once an object is created, you refer to it in code using the object variable you defined. In the following example, you access properties and methods of the new object using the object variable ExcelSheet and other Excel objects, including the Application object and the ActiveSheet.Cells collection. For example:

```
// Make Excel visible through the Application c
ExcelSheet.Application.Visible = true;
// Place some text in the first cell of the sheet.
ExcelSheet.ActiveSheet.Cells(1,1).Value = "Th
// Save the sheet.
ExcelSheet.SaveAs("C:\\TEST.XLS");
// Close Excel with the Quit method on the Application.Quit();
// Release the object variable.
ExcelSheet = "";
```

Creating an object on a remote server can only be accomplished when Internet security is turned off. You can create an object on a remote networked computer by passing the name of the computer to the *servername* argument of **ActiveXObject**. That name is the same as the machine name portion of a sharename. For a network

share named "\myserver\public", the *servername* is "myserver". In addition, you can specify *servername* using DNS format or an IP address.

The following code returns the version number of an instance of Excel running on a remote network computer named "myserver":

```
Function GetVersion {
  var XLApp = CreateObject("Excel.Applicatio
  return(XLApp.Version);
}
```

An error occurs if the specified remote server does not exist or cannot be found.

See Also

Description

Used to sum two numbers or perform string concatenation.

Syntax

result = expression1 + expression2

The + operator syntax has these parts:

Part	Description	
result	Any <u>variable</u> .	
expression1	Any <u>expression</u> .	
expression2	Any expression.	

Remarks

The underlying subtype of the expressions determines the behavior of the + operator.

If	Then
Both expressions are numeric or Boolean	Add.
Both expressions are strings	Concatenate.
One expression is numeric and	

the other is a string

Concatenate.

For information on when a <u>run-time error</u> is generated by the + operator, see the <u>Operator Behavior</u> table.

Microsoft® JScript® anchor Method

See Also

Applies To

Description

Places an HTML anchor with a NAME attribute around specified text in the object.

Syntax

```
strVariable.anchor(anchorstring)
"String Literal".anchor(anchorstring)
```

The *anchorstring* argument is text you want to place in the NAME attribute of an HTML anchor.

Remarks

Call the **anchor** method to create a named anchor out of a **String** object. The following example demonstrates how the **anchor** method accomplishes this:

```
var strVariable = "This is an anchor";
strVariable = strVariable.anchor("Anchor1");
```

The value of *strVariable* after the last statement is:

This is an anchor

No checking is done to see if the tag has already been applied to the string.

Microsoft® JScript® Array Object

See Also Methods Properties

Description

Provides support for creation of arrays of any data type.

Syntax

new Array()
new Array(size)
new Array(element0, element1, ..., elementn)

The **Array** object creation syntax has these parts:

Part	Description
	The size of the array. As
	arrays are zero-based,
size	created elements will
	have indexes from zero to
	size -1.
	The elements to place in
	the array. This creates an
element0,,elementn	array with $n + 1$
	elements, and a length of
	n.

Remarks

After an array is created, the individual elements of the array can be accessed using [] notation, for example:

```
var my_array = new Array();
for (i = 0; i < 10; i++)
     {
      my_array[i] = i;
      }
x = my_array[4];</pre>
```

Since arrays in Microsoft JScript are zero-based, the last statement in the preceding example accesses the fifth element of the array. That element contains the value 4.

If only one argument is passed to the **Array** constructor, and the argument is a number, it is coerced into an unsigned integer, and the value is used as the size of the array. Otherwise, the parameter passed in is used as the only element of the array.

Microsoft® JScript® asin Method

See Also Applies To

Description

Returns the arcsine of a number.

Syntax

Math.asin(number)

The *number* argument is a <u>numeric expression</u> for which the arcsine is sought.

Remarks

The return value is the arcsine of its numeric argument.

Description

Assigns a value to a variable.

Syntax

result = expression

The = operator syntax has these parts:

Part	Description
result	Any <u>variable.</u>
expression	Any <u>numeric expression</u> .

Remarks

As the = operator behaves like other operators, expressions using it have a value in addition to assigning that value into *variable*. This means that you can chain assignment operators as follows:

$$j = k = l = 0;$$

 $j,\,k,\, {\sf and}\,\, l$ equal zero after the example statement is executed.

Microsoft® JScript® atan Method

See Also Applies To

Description

Returns the arctangent of a number.

Syntax

Math.atan(number)

The *number* argument is a <u>numeric expression</u> for which the arctangent is sought.

Remarks

The return value is the arctangent of its numeric argument.

See Also Applies To

Description

Returns the angle (in radians) from the X axis to a point (y,x).

Syntax

Math.atan2(y, x)

The **atan2** method syntax has these parts:

Part	Description
Math	Required. Invokes the intrinsic Math object.
11 V I	Required. A <u>numeric expression</u> representing the cartesian x-coordinate.
111 <i>7</i> 1	Required. A numeric expression representing the cartesian y-coordinate.

Remarks

The return value is between -pi and pi, representing the angle of the supplied (y,x) point.

Applies To

Description

Returns a Boolean value indicating if the enumerator is at the end of the collection.

Syntax

```
myEnum.atEnd()
```

The *myEnum* argument is any **Enumerator** object.

Return Value

The **atEnd** method returns **true** if the current item is the last one in the collection, the collection is empty, or the current item is undefined. Otherwise, it returns **false**.

Remarks

In following code, the **atEnd** method is used to determine if the end of a list of drives has been reached:

```
function ShowDriveList()
{
  var fso, s, n, e, x;
```

```
fso = new ActiveXObject("Scripting.FileSystemObje
e = new Enumerator(fso.Drives);
s = "";
for (; !e.atEnd(); e.moveNext())
 x = e.item();
 s = s + x.DriveLetter;
 s += " - ":
 if (x.DriveType == 3)
  n = x.ShareName;
 else if (x.IsReady)
  n = x.VolumeName;
 else
  n = "[Drive not ready]";
 s += n + " < br > ";
return(s);
```

Applies To

Description

Places HTML <BIG> tags around text in a **String** object.

Syntax

```
strVariable.big()
"String Literal".big()
```

Remarks

The example that follows shows how the **big** method works:

```
var strVariable = "This is a string object";
strVariable = strVariable.big( );
```

The value of *strVariable* after the last statement is:

```
<BIG>This is a string object</BIG>
```

No checking is done to see if the tag has already been applied to the string.

Description

Performs a bitwise AND on two expressions.

Syntax

result = expression1 & expression2

The & operator syntax has these parts:

Part	Description
result	Any <u>variable</u> .
expression1	Any <u>expression</u> .
expression2	Any expression.

Remarks

The & operator looks at the binary representation of the values of two expressions and does a bitwise AND operation on them. The result of this operation behaves as follows:

0101 (expression1)1100 (expression2)----0100 (result)

Any time both of the expressions have a 1 in a digit, the result has a 1 in that digit. Otherwise, the result has a 0 in that digit.

For information on when a <u>run-time error</u> is generated by the **&** operator, see the <u>Operator Behavior</u> table.

Description

Shifts the bits of an expression to the left.

Syntax

result = expression1 << expression2</pre>

The << operator syntax has these parts:

Part	Description
result	Any <u>variable</u> .
expression1	Any <u>expression</u> .
expression2	Any expression.

Remarks

The << operator shifts the bits of *expression1* left by the number of bits specified in *expression2*. For example:

The variable *temp* has a value of 56 because 14 (00001110 in binary) shifted left two bits equals 56 (00111000 in binary).

For information on when a <u>run-time error</u> is generated by the << operator, see the

Operator Behavior table.

Description

Performs a bitwise NOT (negation) on an expression.

Syntax

 $result = \sim expression$

The ~ operator syntax has these parts:

Part	Description
result	Any <u>variable.</u>
expression	Any <u>expression</u> .

Remarks

All unary operators, such as the ~ operator, evaluate expressions as follows:

- If applied to <u>undefined</u> or **null** expressions, a run-time error is raised.
- Objects are converted to strings.
- Strings are converted to numbers if possible. If not, a runtime error is raised.
- Boolean values are treated as numbers (0 if false, 1 if true).

The operator is applied to the resulting number.

The ~ operator looks at the binary representation of the values of the expression and does a bitwise negation operation on it. The result of this operation behaves as follows:

Any digit that is a 1 in the expression becomes a 0 in the result. Any digit that is a 0 in the expression becomes a 1 in the result.

Description

Performs a bitwise OR on two expressions.

Syntax

result = expression1 | expression2

The | operator syntax has these parts:

Part	Description
result	Any <u>variable</u> .
expression1	Any <u>expression</u> .
expression2	Any expression.

Remarks

The | operator looks at the binary representation of the values of two expressions and does a bitwise OR operation on them. The result of this operation behaves as follows:

0101 (expression1)1100 (expression2)----1101 (result)

Any time either of the expressions has a 1 in a digit, the result will have a 1 in that digit. Otherwise, the result will have a 0 in that digit.

For information on when a <u>run-time error</u> is generated by the | operator, see the <u>Operator Behavior</u> table.

Description

Shifts the bits of an expression to the right, maintaining sign.

Syntax

result = expression1 >> expression2

The >> operator syntax has these parts:

Part	Description
result	Any <u>variable</u> .
expression1	Any <u>expression</u> .
expression2	Any expression.

Remarks

The >> operator shifts the bits of *expression1* right by the number of bits specified in *expression2*. The sign bit of *expression1* is used to fill the digits from the left. Digits shifted off the right are discarded. For example, after the following code is evaluated, *temp* has a value of -4: 14 (11110010 in binary) shifted right two bits equals -4 (11111100 in binary).

For information on when a <u>run-time error</u> is generated by the >> operator, see the <u>Operator Behavior</u> table.

Applies To

Description

Places HTML <BLINK> tags around text in a **String** object.

Syntax

```
strVariable.blink()
"String Literal".blink()
```

Remarks

The following example demonstrates how the **blink** method works:

```
var strVariable = "This is a string object";
strVariable = strVariable.blink( );
```

The value of *strVariable* after the last statement is:

```
<BLINK>This is a string object</BLINK>
```

No checking is done to see if the tag has already been applied to the string.

The <BLINK> tag is not supported in Microsoft Internet Explorer.

Applies To

Description

Places HTML tags around text in a **String** object.

Syntax

```
strVariable.bold()
"String Literal".bold()
```

Remarks

The following example demonstrates how the **bold** method works:

```
var strVariable = "This is a string object";
strVariable = strVariable.bold( );
```

The value of *strVariable* after the last statement is:

```
<B>This is a string object</B>
```

No checking is done to see if the tag has already been applied to the string.

${\tt Microsoft @ JScript @} \ Boolean \ Object$

See Also Methods Properties

Description

Creates a new Boolean value.

Syntax

var variablename = new Boolean(boolvalue)

The optional *boolvalue* argument is the initial Boolean value for the new object. If this value is omitted, or is **false**, 0, **null**, **NaN**, or an empty string, the initial value of the **Boolean** object is **false**. Otherwise, the initial value is **true**.

Remarks

The **Boolean** object is a <u>wrapper</u> for the Boolean data type. JScript implicitly uses the **Boolean** object whenever a Boolean data type is converted to a **Boolean** object.

You rarely call the **Boolean** object explicitly.

Description

Terminates the current loop, or if in conjunction with a *label*, terminates the associated statement.

Syntax

break [label];

The optional *label* argument specifies the label of the statement you are breaking from.

Remarks

You typically use the **break** statement in **switch** statements and **while**, **for**, **for...in**, or **do...while** loops. You most commonly use the *label* argument in **switch** statements, but it can be used in any statement, whether simple or <u>compound</u>.

Executing the **break** statement exits from the current loop or statement, and begins script execution with the statement immediately following.

The following example illustrates the use of the **break** statement:

```
while (i < 100)
    {
    if (i == breakpoint)
        break;
        i++;
    }
    return(i);
}</pre>
```

Microsoft® JScript® try...catch Statement

See Also

Description

Implements error handling for JScript.

Syntax

tryStatement catch(exception) catchStatement

The **try...catch** statement syntax has these parts:

Part	Description
	Statement where an error can
tryStatement	occur. Can be a <u>compound</u>
	<u>statement</u> .
	Any <u>variable</u> name. The initial
exception	value of <i>exception</i> is the value
	of the thrown error.
	Statement to handle errors
catchStatement	occurring in the associated
	tryStatement. Can be a
	compound statement.

Remarks

The **try...catch** statement provides a way to handle some or all of the possible errors that may occur in a given block of code, while still running code. If errors occur that the programmer has not handled, JScript simply provides its normal error message to a user, as if there was no error handling.

The *tryStatement* argument contains code where an error can occur, while *catchStatement* contains the code to handle any error that does occur. If an error occurs in the *tryStatement*, program control is passed to *catchStatement* for disposition. The initial value of *exception* is the value of the error that occurred in *tryStatement*.

If the error cannot be handled in the *catchStatement* associated with the *tryStatement* where the error occurred, use the **throw** statement to propagate, or *rethrow*, the error to a higher-level error handler.

The following example throws an error based on a passed-in value. It then illustrates how that error is handled in a hierarchy of **try**...**catch** statements:

```
function TryCatchDemo(x)
{
   try {
    if (x == 0)
     throw "x equals zero";
    else
     throw "x does not equal zero";
}
```

```
catch(e) {
   if (e == "x equals zero")
    return(e + " handled locally.");
   else
                              // Cai
    throw e;
 catch(e) {
  return(e + " handled higher up.")
document.write(TryCatchDemo(0))
document.write(TryCatchDemo(1))
```



Description

Activates conditional compilation support.

Syntax

Remarks

The **@cc_on** statement activates conditional compilation in the scripting engine.

It is strongly recommended that you use the **@cc_on** statement in a comment, so that browsers that do not support conditional compilation will accept your script as valid syntax:

Alternatively, an **@if** or **@set** statement outside of a comment also activates conditional compilation.

See Also Applies To

Description

Returns the smallest integer greater than or equal to its numeric argument.

Syntax

Math.ceil(number)

The *number* argument is a <u>numeric expression</u>.

Remarks

The return value is an integer value equal to the smallest integer greater than or equal to its numeric argument.

Applies To

Description

Returns the character at the specified index.

Syntax

```
strVariable.charAt(index)
"String Literal".charAt(index)
```

The *index* argument is the zero-based index of the desired character. Valid values are between 0 and the length of the string minus 1.

Remarks

The **charAt** method returns a character value equal to the character at the specified index. The first character in a string is at index 0, the second is at index 1, and so forth. Values of index out of valid range return **undefined**.

The following example illustrates the use of the **charAt** method:

```
function charAtTest(n)
{
  var str = "ABCDEFGHIJKLMNC
  var s;
```

```
s = str.charAt(n - 1);
return(s);
}
```

Microsoft® JScript® charCodeAt

Method

See Also

Applies To

Description

Returns the Unicode encoding of the specified character.

Syntax

stringObj.charCodeAt(index)

The **charCodeAt** method syntax has these parts:

Part	Description
stringObj	Required. A String object or literal.
llindex	Required. The zero-based index of the specified character.

Remarks

If there is no character at the specified *index*, **NaN** is returned.

The following example illustrates the use of the **charCodeAt** method:

```
function charCodeAtTest(n)
{
  var str = "ABCDEFGHIJKLMNOPQRSTUV"
```

```
var s;
s = str.charCodeAt(n - 1);
// Return Unicode character code.
return(s);
}
```

Description

Causes two expressions to be executed sequentially.

Syntax

expression1, expression2

The , operator syntax has these parts:

Part	Description
expression1	Any <u>expression</u> .
expression2	Any expression.

Remarks

The , operator causes the expressions on either side of it to be executed in left-to-right order, and obtains the value of the expression on the right. The most common use for the , operator is in the increment expression of a **for** loop. For example:

```
for (i = 0; i < 10; i++, j++)
{
    k = i + j;
}
```

The **for** statement only allows a single expression to be executed at the end of every pass through a loop. The , operator is used to allow multiple expressions to be treated as a single expression, thereby getting around the restriction.

Microsoft® JScript® Comment Statements

Description

Causes comments to be ignored by the JScript parser.

Syntax 1 Single-line Comment: // comment Syntax 2 Multiline Comment: /* comment

The *comment* argument is the text of any comment you want to include in your script.

Syntax 3

*/

//@CondStatement

Syntax 4

/*@ CondStatement @*/

The *CondStatement* argument is conditional compilation code to be used if conditional compilation is activated. If Syntax 3 is used, there can be no space between the "//" and "@" characters.

Remarks

Use comments to keep parts of a script from being read by the JScript parser. You can use comments to include explanatory remarks in a program.

If Syntax 1 is used, the parser ignores any text between the comment marker and the end of the line. If Syntax 2 is used, it ignores any text between the beginning and end markers.

Syntaxes 3 and 4 are used to support conditional compilation while retaining compatibility with browsers that do not support that feature. These browsers treat those forms of comments as syntaxes 1 and 2 respectively.

The following example illustrates the most common uses of the **comment** statement:

```
function myfunction(arg1, arg2)
{
  /* This is a multiline comment tha
    can span as many lines as necess
  var r;
  // This is a single line comment.
  r = arg1 + arg2; // Sum the two arg
  return(r);
}
```

Microsoft® JScript® Comparison Operators

See Also

Description

Returns a Boolean value indicating the result of the comparison.

Syntax

expression1 comparisonoperator expression2

The Comparison operator syntax has these parts:

Part	Description
expression1	Any <u>expression</u> .
comparisonoporator	Any <u>comparison</u>
comparisonoperator	<u>operator</u> .
expression2	Any expression.

Remarks

When comparing strings, JScript uses the Unicode character value of the string expression.

The following describes how the different groups of operators behave depending on the types and values of *expression1* and *expression2*:

Relational (<, >, <=, >=)

• Attempt to convert both *expression1* and *expression2* into

numbers.

- If both expressions are strings, do a lexicographical string comparison.
- If either expression is **NaN**, return **false**.
- Negative zero equals Positive zero.
- Negative Infinity is less than everything including itself.
- Positive Infinity is greater than everything including itself.

Equality (==, !=)

- If the types of the two expressions are different, attempt to convert them to string, number, or Boolean.
- NaN is not equal to anything including itself.
- Negative zero equals positive zero.
- **null** equals both **null** and **undefined**.
- Values are considered equal if they are identical strings, numerically equivalent numbers, the same object, identical Boolean values, or (if different types) they can be coerced into one of these situations.
- Every other comparison is considered unequal.

Identity (===. !==)

These operators behave identically to the equality operators except no type conversion is done, and the types must be the same to be considered equal.

Microsoft® JScript® compile Method

See Also

Applies To

Description

Compiles a regular expression into an internal format.

Syntax

rgexp.compile(pattern)

The **compile** method syntax has these parts:

Part	Description
	Required. A Regular Expression object. Can be a variable name or a literal.
pattern	Required. A <u>string expression</u> containing a regular expression pattern to be compiled.

Remarks

The **compile** method converts *pattern* into an internal format for faster execution. This allows for more efficient use of regular expressions in loops, for example.

The following example illustrates the use of the **compile** method:

```
function CompileDemo()
{
```

Microsoft® JScript® Compound Assignment Operators

Language Reference

```
Addition (+=)
```

Bitwise AND (&=)

Bitwise OR (|=)

Bitwise XOr (^=)

Division (/=)

Left Shift (<<=)</pre>

Modulus (%=)

Multiplication (*=)

Right Shift (>>=)

Subtraction (-=)

Unsigned Right Shift (>>>=)

Microsoft® JScript® concat Method (Array)

See Also

Applies To

Description

Returns an new array consisting of a combination of two arrays.

Syntax

array1.concat(array2)

The **concat** method syntax has these parts:

Part	Description
array1	Required. An Array object to concatenate with <i>array2</i> .
array2	Required. An Array object to concatenate to the end of <i>array1</i> .

Remarks

The **concat** method returns an **Array** object containing the concatenation of *array1* and *array2*.

If an object reference is copied from either *array1* or *array2* to the result, the object reference in the result still points to the same object. Changes to that object are reflected in both arrays.

The following example illustrates the use of the **concat** method:

```
function ConcatArrayDemo()
{
  var a, b, c;
  a = new Array(0,1,2,3,4);
  b = new Array(5,6,7,8,9);
  c = a.concat(b);
  return(c);
}
```

Microsoft® JScript® concat Method (String)

See Also

Applies To

Description

Returns a **String** object containing the concatenation of two supplied strings.

Syntax

string1.concat(string2)

The **concat** method syntax has these parts:

Part	Description
string1	Required. The String object or literal to concatenate with <i>string2</i> .
string2	Required. A String object or literal to concatenate to the end of <i>string1</i> .

Remarks

The result of the **concat** method is equivalent to: *result* = *string1* + *string2*.

The following example illustrates the use of the **concat** method:

function concatDemo()

```
var str1 = "ABCDEFGHIJKLM"
var str2 = "NOPQRSTUVWXYZ";
var s = str1.concat(str2);
// Return concatenated string.
return(s);
}
```

Microsoft® JScript® Conditional Compilation

See Also

Description

Allows the use of new JScript language features without sacrificing compatibility with browsers that don't support the features.

Remarks

Conditional compilation is activated by using the **@cc_on** statement, or using an **@if** or **@set** statement outside of a comment. Some typical uses for conditional compilation are using new features in JScript, embedding debugging support into a script, and tracing code execution.

It is strongly recommended that conditional compilation code be placed in comments:

```
/*@cc_on @*/
/*@if (@_jscript_version == 4)
alert("JScript version 4");
@else @*/
alert("You need a more recent script engine.")
/*@end @*/
```

This example uses special comment delimiters that are only used if conditional compilation is activated by the **@cc_on** statement. Scripting engines that do not understand conditional compilation only see the message informing of the need for a new scripting engine.

Microsoft® JScript® Conditional Compilation Variables

See Also

The following predefined variables are available for conditional compilation. If a variable is not **true**, it is not defined and behaves as **NaN** when accessed.

Variable	Description
@_win32	true if running on a Win32 system.
@_win16	true if running on a Win16 system.
@_mac	true if running on a Apple Macintosh system.
@_alpha	true if running on a DEC Alpha processor.
@_x86	true if running on an Intel processor.
@_mc680x0	true if running on a Motorola 680x0 processor.
@_PowerPC	true if running on a Motorola PowerPC processor.
@_jscript	Always true .
@_jscript_build	Contains the build number of the JScript scripting engine.
@_jscript_version	Contains the JScript version number in major.minor format.

See Also

Description

Executes one of two expressions depending on a condition.

Syntax

test? expression1: expression2

The **?:** operator syntax has these parts:

Part	Description
test	Any Boolean expression.
expression1	An expression executed if <i>test</i> is
	true.
expression2	An expression executed if <i>test</i> is
	false.

Remarks

The **?:** operator is a shortcut for an **if...else** statement. It is typically used as part of a larger expression where an **if...else** statement would be awkward. For example:

The example creates a string containing "Good evening." if it is after 6pm. The equivalent code using an **if...else** statement would look as follows:

```
var now = new Date();
var greeting = "Good";
if (now.getHours() > 17)
  greeting += " evening.";
else
  greeting += " day.";
```

Microsoft® JScript® CONSTRUCTOR

Property

See Also

Applies To

Description

Specifies the function that creates an object.

Syntax

object.constructor

The required *object* argument is the name of an object or function.

Remarks

The **constructor** property is a member of the prototype of every object that has a prototype. This includes all <u>intrinsic JScript</u> <u>objects</u> except the **Global** and **Math** objects. The **constructor** property contains a reference to the function that constructs instances of that particular object. For example:

```
x = new String("Hi");
if (x.constructor == String)
    // Do something (the condition will be truε)
```

or

```
function MyFunc {
    // Body of function.
```

```
y = new MyFunc;
if (y.constructor == MyFunc)
// Do something (the condition will be truε
```

Microsoft® JScript® **continue Statement**

See Also

Description

Stops the current iteration of a loop, and starts a new iteration.

Syntax

continue [label];

The optional *label* argument specifies the statement to which **continue** applies.

Remarks

You can use the **continue** statement only inside a **while**, **do...while**, **for**, or **for...in** loop. Executing the **continue** statement stops the current iteration of the loop and continues program flow with the beginning of the loop. This has the following effects on the different types of loops:

- **while** and **do...while** loops test their condition, and if true, execute the loop again.
- **for** loops execute their increment expression, and if the test expression is true, execute the loop again.
- **for...in** loops proceed to the next field of the specified variable and execute the loop again.

The following example illustrates the use of the **continue** statement:

```
function skip5()
 var s = "", i=0;
 while (i < 10)
  i++;
  // Skip 5
  if (i==5)
    continue;
 s += i;
 return(s);
```

Microsoft® JScript® cos Method

See Also Applies To

Description

Returns the cosine of a number.

Syntax

Math.cos(number)

The *number* argument is a <u>numeric expression</u> for which the cosine is sought.

Remarks

The return value is the cosine of its numeric argument.

Microsoft® JScript® Date Object

See Also Methods Properties

Description

Enables basic storage and retrieval of dates and times.

Syntax

```
var newDateObj = new Date()
var newDateObj = new Date(dateVal)
var newDateObj = new Date(year, month, date[, hours[, minutes[, seconds[,ms]]]])
```

The **Date** object <u>constructor</u> syntax has these parts:

Part	Description
dateVal	If a numeric value, <i>dateVal</i> represents
	the number of milliseconds in
	<u>Universal Coordinated Time</u> between
	the specified date and midnight
	January 1, 1970. If a string, <i>dateVal</i> is
	parsed according to the rules in the
	parse method. The <i>dateVal</i> argument
	can also be a VT_DATE value as
	returned from some ActiveX® objects.
	Required. The full year, for example,
	1976 (and not 76).
	, in the second of the second

	!/
month	Required. The month as an integer between 0 and 11 (January to December).
date	Required. The date as an integer between 1 and 31.
hours	Optional. Must be supplied if <i>minutes</i> is supplied. An integer from 0 to 23 (midnight to 11pm) that specifies the hour.
	Optional. Must be supplied if <i>seconds</i> is supplied. An integer from 0 to 59 that specifies the minutes.
seconds	Optional. Must be supplied if milliseconds is supplied. An integer from 0 to 59 that specifies the seconds.
ms	Optional. An integer from 0 to 999 that specifies the milliseconds.

Remarks

A **Date** object contains a number representing a particular instant in time to within a millisecond. If the value of an argument is greater than its range or is a negative number, other stored values are modified accordingly. For example, if you specify 150 seconds, JScript redefines that number as two minutes and 30 seconds.

If the number is **NaN**, that indicates that the object does not represent a specific instant of time. If you pass no parameters to the **Date** object, it is initialized to the current time

(UTC). A value must be given to the object before you can use it.

The range of dates that can be represented in a **Date** object is approximately 285,616 years on either side of January 1, 1970.

The **Date** object has two static methods that are called without creating a **Date** object. They are **parse** and **UTC**.



See Also

Description

Used to increment or decrement a variable by one.

Syntax 1

```
result = ++variable
result = --variable
result = variable++
result = variable--
```

Syntax 2

```
++variable
--variable
variable++
variable--
```

The syntax of the ++ and -- operators has these parts:

Part	Description
result	Any <u>variable</u> .
variable	Any variable.

Remarks

The increment and decrement operators are used as a shortcut to

modify the value stored in a variable. The value of an expression containing one of these operators depends on whether the operator comes before or after the variable:

j is assigned the value 3, as the increment occurs before the expression is evaluated.

Contrast the following example:

Here, j is assigned the value 2, as the increment occurs after the expression is evaluated.

Language Reference Version 5

Microsoft® JScript® description Property

See Also

Applies to

Description

Returns or sets the descriptive string associated with a specific error.

Syntax

object.description [= stringexpression]

The **description** property syntax has these parts:

Part	Description
object	Any instance of an Error object.
stringexpression	A <u>string expression</u> containing a description of the error.

Remarks

The **description** property contains the error message string associated with a specific error. Use the value contained in this property to alert a user to an error that you can't or don't want to handle.

The following example illustrates the use of the **description** property:

try {

Microsoft® JScript® dimensions Method

See Also

Applies To

Description

Returns the number of dimensions in a VBArray.

Syntax

array.dimensions()

The *array* argument is a **VBArray** object.

Remarks

The **dimensions** method provides a way to retrieve the number of dimensions in a specified VBArray.

The following example consists of three parts. The first part is VBScript code to create a Visual Basic safe array. The second part is JScript code that determines the the number of dimensions in the safe array and the upper bound of each dimension. Both of these parts go into the <HEAD> section of an HTML page. The third part is the JScript code that goes in the <BODY> section to run the other two parts.

```
Function CreateVBArray()
 Dim i, j, k
 Dim a(2, 2)
 k = 1
 For i = 0 To 2
   For j = 0 To 2
    a(j, i) = k
    k = k + 1
   Next
 Next
 CreateVBArray = a
End Function
-->
</SCRIPT>
<SCRIPT LANGUAGE="JScript">
function VBArrayTest(vba)
var i, s;
var a = new VBArray(vba);
 for (i = 1; i \le a.dimensions(); i++)
 s = "The upper bound of dimension";
 s += i + " is ";
 s += a.ubound(i)+ ".<BR>";
return(s);
}
-->
</SCRIPT>
```

```
</HEAD>

<BODY>
<SCRIPT language="jscript">
document.write(VBArrayTest(CreateVBArray()));
</SCRIPT>
</BODY>
```

Microsoft® JScript® do...while Statement

See Also

Description

Executes a statement block once, and then repeats execution of the loop until a condition expression evaluates to **false**.

Syntax

```
do
  statement
while (expression);
```

The **do...while** statement syntax has these parts:

	Description
statement	The statement to be executed if <i>expression</i> is true .
	Can be a <u>compound statement</u> .
expression	An <u>expression</u> that can be coerced to Boolean true
	or false . If <i>expression</i> is true , the loop is executed
	again. If <i>expression</i> is false , the loop is terminated.

Remarks

The value of *expression* is not checked until after the first iteration of the loop, guaranteeing that the loop is executed at least once. Thereafter, it is checked after each succeeding iteration of the loop.

The following code uses the **do...while** statement to iterate the **Drives** collection:

```
function GetDriveList()
 var fso, s, n, e, x;
 fso = new ActiveXObject("Scripting.FileSyste
 e = new Enumerator(fso.Drives);
 s = "";
 do
  x = e.item();
  s = s + x.DriveLetter;
  s += " - ";
  if (x.DriveType == 3)
   n = x.ShareName;
  else if (x.IsReady)
   n = x.VolumeName;
  else
   n = "[Drive not ready]";
  s += n + " < br > ";
  e.moveNext();
 while (!e.atEnd());
```

```
return(s);
}
```

${\tt Microsoft @ JScript @} \ E \ Property$

See Also

Applies To

Description

Returns Euler's constant, the base of natural logarithms. The **E** property is approximately equal to 2.718.

Syntax

var numVar
numVar = Math.E

Microsoft® JScript® Enumerator

Object

See Also Methods Properties

Description

Enables enumeration of items in a collection.

Syntax

new Enumerator(collection)

The *collection* argument is any collection object.

Remarks

Collections differ from arrays in that the members of a collection are not directly accessible. Instead of using indexes, as you would with arrays, you can only move the current item pointer to the first or next element of a collection.

The **Enumerator** object provides a way to access any member of a collection and behaves similarly to the **For...Each** statement in VBScript.

The following code shows the usage of the **Enumerator** object:

```
function ShowDriveList()
{
  var fso, s, n, e, x;
```

```
fso = new ActiveXObject("Scripting.FileSyste
 e = new Enumerator(fso.Drives);
 s = "";
 for (;!e.atEnd();e.moveNext())
  {
   x = e.item();
   s = s + x.DriveLetter;
   s += " - ";
   if (x.DriveType == 3)
    n = x.ShareName;
   else if (x.IsReady)
    n = x.VolumeName;
   else
    n = "[Drive not ready]";
   s += n + " < br > ";
 return(s);
}
```

Microsoft® JScript® Error Object

See Also Properties

Description

Contains information about errors.

Syntax

```
var newErrorObj = new Error()
var newErrorObj = new Error(number)
var newErrorObj = new Error(number, description)
```

The **Error** object <u>constructor</u> syntax has these parts:

Part	Description
number	Numeric value assigned to an error. Zero if omitted.
description	Brief string that describes an error. Empty string if omitted.

Remarks

Whenever a <u>run-time error</u> occurs, an instance of the **Error** object is created to describe the error. This instance has two intrinsic properties that contain the description of the error (**description** property) and the error number (**number** property).

An error number is a 32-bit value. The upper 16-bit word is the facility code, while the lower word is the actual error code.

Error objects can also be explicitly created, using the syntax shown above, or thrown using the **throw** statement. In both cases, you can add any properties you choose, to expand the capability of the **Error** object.

Typically, the local <u>variable</u> that's created in a **try...catch** statement refers to the implicitly created **Error** object. As a result, you can use the error number and description in any way you choose.

The following example illustrates the use of the implicitly created **Error** object:

Microsoft® JScript® escape Method

See Also Applies To

Description

Encodes **String** objects so they can be read on all computers.

Syntax

escape(charstring)

The *charstring* argument is a **String** object to be encoded.

Remarks

The **escape** method returns a new **String** object (in Unicode format) that contains the contents of *charstring*. All spaces, punctuation, accented characters, and any other <u>non-ASCII</u> characters are replaced with *%xx* encoding, where *xx* is equivalent to the hexadecimal number representing the character. For example, a space is returned as "%20."

Characters with a value greater than 255 are stored using the **%u**xxxx format.

Applies To

Description

Evaluates JScript code and executes it.

Syntax

eval(codestring)

The *codestring* argument is a **String** object that contains valid JScript code. This string is parsed by the JScript parser and executed.

Remarks

The **eval** function allows dynamic execution of JScript source code. For example, the following code creates a new variable *mydate* that contains a **Date** object:

The code passed to the **eval** method is executed in the same context as the call to the **eval** method.

Applies To

Description

Executes a search for a match in a specified string.

Syntax

rgexp.exec(str)

The **exec** method syntax has these parts:

Part	Description	
rgexp	Required. A Regular Expression object. Can be a variable name or a literal.	
	Required. The string to perform a search on.	

Remarks

The results of an **exec** method search are placed into an array.

If the **exec** method does not find a match, it returns **null**. If it finds one or more matches, the **exec** method returns an array, and the **RegExp** object is updated to reflect the results of the search.

The following example illustrates the use of the **exec** method:

function ExecDemo()

```
var s = "AaBbCcDdEeFfGgHhIiJjKkLlMmNi
var r = new RegExp("g", "i");
var a = r.exec(s);
document.write(a);
r.compile("g");
var a = r.exec(s);
document.write(a);
}
```

See Also Applies To

Description

Returns e (the base of natural logarithms) raised to a power.

Syntax

Math.exp(number)

The *number* argument is a <u>numeric expression</u> representing the power of e.

Remarks

The return value is e^{number} . The constant e is Euler's constant, approximately equal to 2.178 and number is the supplied argument.

Applies To

Description

Places HTML <TT> tags around text in a **String** object.

Syntax

```
strVariable.fixed()
"String Literal".fixed()
```

Remarks

The following example demonstrates how the **fixed** method works:

```
var strVariable = "This is a string object";
strVariable = strVariable.fixed( );
```

The value of *strVariable* after the last statement is:

```
<TT>This is a string object</TT>
```

No checking is done to see if the tag has already been applied to the string.



See Also Applies To

Description

Returns the greatest integer less than or equal to its numeric argument.

Syntax

Math.floor(number)

The *number* argument is a <u>numeric expression</u>.

Remarks

The return value is an integer value equal to the greatest integer less than or equal to its numeric argument.

Microsoft® JScript® fontcolor Method

See Also

Applies To

Description

Places an HTML tag with the COLOR attribute around the text in a **String** object.

Syntax

```
strVariable.fontcolor(colorval)
"String Literal".fontcolor(colorval)
```

The *colorval* argument is a string containing a color value. This can either be the hexadecimal value for a color, or the predefined name for a color.

Remarks

The following example demonstrates the **fontcolor** method:

```
var strVariable = "This is a string";
strVariable = strVariable.fontcolor("red");
```

The value of *strVariable* after the last statement is:

Valid predefined color names depend on your JScript host (browser, server, and so forth). They may also vary from version to version of your host. Check your host documentation for more information.

No checking is done to see if the tag has already been applied to the string.

Applies To

Description

Places an HTML tag with the SIZE attribute around the text in a **String** object.

Syntax

```
strVariable.fontsize(intSize)
"String Literal".fontsize(intSize)
```

The *intSize* argument is an integer value that determines the size of the text.

Remarks

The following example demonstrates the **fontsize** method:

```
var strVariable = "This is a string";
strVariable = strVariable.fontsize(-1);
```

The value of *strVariable* after the last statement is:

Valid integer values depend on your Microsoft JScript host. See your host documentation for more information.

No checking is done to see if the tag has already been applied to the string.

Description

Executes a block of statements for as long as a specified condition is true.

Syntax

for (initialization; test; increment) statement

The **for** statement syntax has these parts:

Part	Description
	An <u>expression</u> . This expression is
initialization	executed only once, before the
	loop is executed.
	A <u>Boolean expression</u> . If <i>test</i> is
test	true , <i>statement</i> is executed. If <i>test</i>
	if false , the loop is terminated.
	An expression. The increment
increment	expression is executed at the end
	of every pass through the loop.
	The statement to be executed if
statement	<i>test</i> is true . Can be a <u>compound</u>
	<u>statement</u> .

Remarks

You usually use a **for** loop when the loop is to be executed a specific number of times. The following example demonstrates a **for** loop.

```
/* i is set to 0 at start, and is incremented by 1 a
of each iteration. Loop terminates when i is not
than 10 before a loop iteration. */
var myarray = new Array();
for (i = 0; i < 10; i++)
{
    myarray[i] = i;
}</pre>
```

Description

Executes one or more statements for each <u>property</u> of an object, or each element of an array.

Syntax

for (variable **in** [object | array]) statement

The **for** statement syntax has these parts:

Part	Description
variable	A <u>variable</u> that can be any property of <i>object</i> or any element of <i>array</i> .
object, array	An object or array over which to iterate.
	The statement or statements to be executed for each property of <i>object</i> or each element of <i>array</i> . Can be a compound statement.

Remarks

Before each iteration of a loop, *variable* is assigned the next property of *object* or the next element of *array*. You can then use it in any of the statements inside the loop, exactly as if you were

using the property of *object* or the element of *array*.

When iterating over an object, there is no way to determine or control the order in which the members of the object are assigned to *variable*. Iterating through an array will be performed in element order, that is, 0, 1, 2, ...

The following example illustrates the use of the **for ... in** statement with an object used as an associative array:

```
function ForInDemo()
{
    // Create some variables.
    var a, key, s = "";
    // Initialize object.
    a = {"a" : "Athens" , "b" : "Belgrade", "c" : "Cairo"}
    // Iterate the properties.
    for (key in a)
    {
        s += a[key] + "<BR;>";
    }
    return(s);
}
```

Note Use the **enumerator** object to iterate members of a collection.

Microsoft® JScript® from CharCode

Method

See Also

Applies To

Description

Returns a string from a number of Unicode character values.

Syntax

String.fromCharCode(code1, code2, ..., coden)

The *code* argument is the series of Unicode character values to convert into a string.

Remarks

A **String** object need not be created before calling **fromCharCode**.

In the following example, *test* contains the string "plain":

var test = String.fromCharCode(112, 108, 97, 1

${\tt Microsoft @ JScript @} \ Function \ Object$

See Also Methods Properties

Description

Creates a new function.

Syntax 1

```
function functionname( [argname1 [, ... argnameN]] )
{
   body
}
```

Syntax 2

var functionname = new Function([argname1, [... argnameN,]]
body);

The **Function** object syntax has these parts:

Part	Description
functionname	The name of the newly created function
argname1argnameN	An optional list of arguments that the function accepts.
body	A string that contains the block of JScript code to be executed when the function is called.

Remarks

The function is a basic data type in JScript. Syntax 1 creates a function value that JScript converts into a **Function** object when necessary. JScript converts **Function** objects created by Syntax 2 into function values at the time the function is called.

Syntax 1 is the standard way to create new functions in JScript. Syntax 2 is an alternative form used to create function objects explicitly.

For example, to create a function that adds the two arguments passed to it, you can do it in either of two ways:

Example 1

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

Example 2

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

In either case, you call the function with a line of code similar to the following:

```
add(2, 3);
```

Note When calling a function, ensure that you always include the parentheses and any required arguments. Calling a function

without parentheses causes the text of the function to be returned instead of the results of the function.

Microsoft® JScript® getItem Method

See Also Applies To

Description

Returns the item at the specified location.

Syntax

safeArray.getItem(dimension1[, dimension2, ...], dimensionn)

The **getItem** method syntax has these parts:

Part	Description
safeArray	Required. A VBArray object.
dimension1,	Specifies the exact location of the desired element
	of the VBArray. <i>n</i> equals the number of
dimensionn	dimensions in the VBArray.

The following example consists of three parts. The first part is VBScript code to create a Visual Basic safe array. The second part is JScript code that iterates the VB safe array and prints out the contents of each element. Both of these parts go into the <HEAD> section of an HTML page. The third part is the JScript code that goes in the <BODY> section to run the other two parts.

<HEAD>
<SCRIPT LANGUAGE="VBScript">
<!-Function CreateVBArray()

```
Dim i, j, k
 Dim a(2, 2)
 k = 1
 For i = 0 To 2
  For j = 0 To 2
   a(i, j) = k
   document.writeln(k)
   k = k + 1
  Next
  document.writeln("<BR>")
 Next
 CreateVBArray = a
End Function
-->
</SCRIPT>
<SCRIPT LANGUAGE="JScript">
<!--
function GetItemTest(vbarray)
{
 var i, j;
 var a = new VBArray(vbarray);
 for (i = 0; i \le 2; i++)
 {
  for (j = 0; j \le 2; j++)
  {
   document.writeln(a.getItem(i, j));
```

```
}
}-->
</SCRIPT>
</HEAD>
<BODY;>
<SCRIPT LANGUAGE="JScript">
<!--
   GetItemTest(CreateVBArray());
-->
</SCRIPT>
</BODY>
```

Microsoft® JScript® GetObject Function

See Also

Description

Returns a reference to an Automation object from a file.

Syntax

GetObject([pathname] [, class])

The **GetObject** function syntax has these parts:

Part	Description
pathname	Optional. Full path and name of the file containing the object to retrieve. If <i>pathname</i> is omitted, <i>class</i> is required.
class	Optional. <u>Class</u> of the object.

The *class* argument uses the syntax *appname.objectype* and has these parts:

Part	Description	
appname	Required. Name of the application providing the object.	
objectype	Required. Type or class of object to create.	

Remarks

Use the **GetObject** function to access an Automation object from

a file. Assign the object returned by **GetObject** to the object variable. For example:

var CADObject; CADObject = GetObject("C:\\CAD\\SCHEMA.CAD'

When this code is executed, the application associated with the specified *pathname* is started, and the object in the specified file is activated. If *pathname* is a zero-length string (""), **GetObject** returns a new object instance of the specified type. If the *pathname* argument is omitted, **GetObject** returns a currently active object of the specified type. If no object of the specified type exists, an error occurs.

Some applications allow you to activate part of a file. Add an exclamation point (!) to the end of the file name and follow it with a string that identifies the part of the file you want to activate. For information on how to create this string, see the documentation for the application that created the object.

For example, in a drawing application you might have multiple layers to a drawing stored in a file. You could use the following code to activate a layer within a drawing called SCHEMA.CAD:

var LayerObject = GetObject("C:\\CAD\\SCHE

If you don't specify the object's class, Automation determines the application to start and the object to activate, based on the file name you provide. Some files, however, may support more than one class of object. For example, a drawing might support three different types of objects: an Application object, a Drawing object, and a Toolbar object, all of which are part of the same file. To specify which object in a file you want to activate, use the optional *class* argument. For example:

var MyObject; MyObject = GetObject("C:\\DRAWINGS\\SAN

In the preceding example, FIGMENT is the name of a drawing application and DRAWING is one of the object types it supports. Once an object is activated, you reference it in code using the object variable you defined. In the preceding example, you access properties and methods of the new object using the object variable MyObject. For example:

MyObject.Line(9, 90); MyObject.InsertText(9, 100, "Hello, world."); MyObject.SaveAs("C:\\DRAWINGS\\SAMPL]

Note Use the **GetObject** function when there is a current instance of the object, or if you want to create the object with a file already loaded. If there is no current instance, and you don't want the object started with a file loaded, use the **ActiveXObject** object.

If an object has registered itself as a single-instance object, only one instance of the object is created, no matter how many times **ActiveXObject** is executed. With a single-instance object, **GetObject** always returns the same instance when called with the zero-length string ("") syntax, and it causes an error if the *pathname* argument is omitted.

Description

Conditionally executes a group of statements, depending on the value of an expression.

Syntax

```
@if (condition1)
    text1
[@elif (condition2)
    text2]
[@else
    text3]
@end
```

The **@if** statement syntax has these parts:

Part	Description
condition1,	An expression that can be coerced
condition2	into a <u>Boolean expression</u> .
text1	Text to be parsed if <i>condition1</i> is
lexu	true.
tovt?	Text to be parsed if <i>condition1</i> is
text2	false and <i>condition2</i> is true .
	Text to be parsed if both
	_

are

Remarks

When you write an **@if** statement, you don't have to place each clause on a separate line. You can use multiple **@elif** clauses, however, all **@elif** clauses must come before an **@else** clause.

You commonly use the **@if** statement to determine which text among several options should be used for text output. For example:

Description

Conditionally executes a group of statements, depending on the value of an expression.

Syntax

```
if (condition)
    statement1
[else
    statement2]
```

The **if...else** statement syntax has these parts:

Part	Description
	A Boolean expression. If condition
condition	is null or <u>undefined</u> , condition is
	treated as false .
statement1	The statement to be executed if
	condition is true . Can be a
	compound statement.
	The statement to be executed if
statement2	condition is false . Can be a
	compound statement.

Remarks

It is generally good practice to enclose *statement1* and *statement2* in braces ({}) for clarity and to avoid inadvertent errors. In the following example, you may intend that the **else** be used with the first **if** statement, but it is used with the second one.

```
if (x == 5)
  if (y == 6)
    z = 17;
else
  z = 20;
```

Changing the code in the following manner eliminates any ambiguities:

```
if (x == 5)
  {
  if (y == 6)
    z = 17;
  }
else
  z = 20;
```

Similarly, if you want to add a statement to *statement1*, and you don't use braces, you can accidentally create an error:

$$if (x == 5)$$

```
z = 7;
q = 42;
else
z = 19;
```

In this case, there is a syntax error, because there is more than one statement between the **if** and **else** statements. Braces are required around the statements between **if** and **else**.

See Also Applies To

Description

Returns the character position where the first occurrence a substring occurs within a **String** object.

Syntax

strVariable.indexOf(substring, startindex)
"String Literal".indexOf(substring, startindex)

The **indexOf** method syntax has these arguments:

Part	Description
uciinctrina	The substring to search for within the String object.
startindex	An optional integer value specifying the index to begin searching within the String object. If omitted, searching begins at the beginning of the string.

Remarks

The **indexOf** method returns an integer value indicating the beginning of the substring within the **String** object. If the substring is not found, a -1 is returned.

If *startindex* is negative, *startindex* is treated as zero. If it is larger than the greatest character position index, it is treated as the largest possible index.

Searching is performed from left to right. Otherwise, this method is identical to **lastIndexOf**.

The following example illustrates the use of the **indexOf** method:

```
function IndexDemo(str2)
{
  var str1 = "BABEBIBOBUBABE
  var s = str1.indexOf(str2);
  return(s);
}
```

Microsoft® JScript® isNaN Method

See Also Applies To

Description

Returns a Boolean value that indicates whether a value is the reserved value **NaN** (not a number).

Syntax

isNaN(numvalue)

The *numvalue* argument is the value to be tested against **NaN**.

Remarks

The **isNaN** function returns **true** if the value is **NaN**, and **false** otherwise. You typically use this function to test return values from the **parseInt** and **parseFloat** methods.

Alternatively, a variable could be compared to itself. If it compares as unequal, it is **NaN**. This is because **NaN** is the only value that is not equal to itself.

Applies To

Description

Places HTML <I> tags around text in a **String** object.

Syntax

```
strVariable.italics()
"String Literal".italics()
```

Remarks

The following example demonstrates how the **italics** method works:

```
var strVariable = "This is a string";
strVariable = strVariable.italics();
```

The value of *strVariable* after the last statement is:

```
<I>This is a string</I>
```

No checking is done to see if the tag has already been applied to the string.

Applies To

Description

Returns the current item in the collection.

Syntax

```
myEnum.item()
```

The *myEnum* argument is any **Enumerator** object.

Return Value

The **item** method returns the current item. If the collection is empty or the current item is undefined, it returns **undefined**.

Remarks

In following code, the **item** method is used to return a member of the **Drives** collection:

```
function ShowDriveList()
{
  var fso, s, n, e, x;
  fso = new ActiveXObject("Scripting.FileSystematics")
```

```
e = new Enumerator(fso.Drives);
 s = "";
 for (; !e.atEnd(); e.moveNext())
  x = e.item();
  s = s + x.DriveLetter;
  s += " - ":
  if (x.DriveType == 3)
   n = x.ShareName;
  else if (x.IsReady)
   n = x.VolumeName;
  else
   n = "[Drive not ready]";
  s += n + " < br > ";
 return(s);
}
```

Applies To

Description

Returns a **String** object consisting of all the elements of an array concatenated together.

Syntax

```
arrayobj.join(separator)
```

The *separator* argument is a **String** object that is used to separate one element of an array from the next in the resulting **String** object. If omitted, the array elements are separated with an empty string.

Remarks

The **join** method returns a **String** object that contains each element converted to a string and concatenated together.

The following example illustrates the use of the **join** method:

```
function JoinDemo()
{
  var a, b;
  a = new Array(0,1,2,3,4);
  b = a.join("-");
```

```
return(b);
}
```

Microsoft® JScript® Labeled Statement

See Also

Description

Provides an identifier for a statement.

Syntax

label:

statement

Labeled statement syntax has these parts:

Part	Description	
Mapei I	A unique identifier used when referring to the labeled statement.	
statement	The statement associated with <i>label</i> . May be a <u>compound statement</u> .	

Remarks

Labels are used by the **break** and **continue** statements to specify the statement to which the **break** and **continue** apply.

In the following statement the **continue** statement uses a **labeled** statement to create an array in which the third column of each row contains and undefined value:

```
function labelDemo()
{
  var a = new Array();
  var i, j, s = "", s1 = "";
 Outer:
  for (i = 0; i < 5; i++)
    Inner:
     for (j = 0; j < 5; j++)
      if (j == 2)
        continue Inner;
      else
        a[i,j] = j + 1;
  for (i = 0; i < 5; i++)
   S = ""
    for (j = 0; j < 5; j++)
    {
     s += a[i,j];
```

```
}
  s1 += s + "\n";
}
return(s1)
}
```

${\tt Microsoft @ JScript @ } last Index Of$

Method

See Also Applies To

Description

Returns the last occurrence of a substring within a **String** object.

Syntax

strVariable.lastIndexOf(substring, startindex)
"String Literal".lastIndexOf(substring, startindex)

The **lastIndexOf** method syntax has these arguments:

Part	Description	
substring	The substring to search for within the String object.	
startinde <i>x</i>	An optional integer value specifying the index to begin searching within the String object. If omitted, searching begins at the end of the string.	

Remarks

The **lastIndexOf** method returns an integer value indicating the beginning of the substring within the **String** object. If the substring is not found, a -1 is returned.

If *startindex* is negative, *startindex* is treated as zero. If it is larger than the greatest character position index, it is treated as the largest possible index.

Searching is performed right to left. Otherwise, this method is identical to **indexOf**.

The following example illustrates the use of the **lastIndexOf** method:

```
function lastIndexDemo(str2)
{
  var str1 = "BABEBIBOBUBABE
  var s = str1.lastIndexOf(str2);
  return(s);
}
```



See Also Applies To

Description

Returns the lowest index value used in the specified dimension of a VBArray.

Syntax

safeArray.lbound(dimension)

The **lbound** method syntax has these parts:

Part	Description	
safeArray	Required. A VBArray object.	
II .	Optional. The dimension of the VBArray for which the lower bound index is wanted. If omitted, lbound	
	behaves as if a 1 was passed.	

Remarks

If the VBArray is empty, the **lbound** method returns **undefined**. If *dimension* is greater than the number of dimensions in the VBArray, or is negative, the method generates a "Subscript out of range" error.

The following example consists of three parts. The first part is VBScript code to create a Visual Basic safe array. The second part is JScript code that determines the number of dimensions in the safe array and the lower

bound of each dimension. Since the safe array is created in VBScript rather than Visual Basic, the lower bound will always be zero. Both of these parts go into the <HEAD> section of an HTML page. The third part is the JScript code that goes in the <BODY> section to run the other two parts.

```
<HEAD>
<SCRIPT LANGUAGE="VBScript">
<!--
Function CreateVBArray()
 Dim i, j, k
 Dim a(2, 2)
 k = 1
 For i = 0 To 2
  For j = 0 To 2
   a(j, i) = k
   k = k + 1
  Next
 Next
 CreateVBArray = a
End Function
__>
</SCRIPT>
<SCRIPT LANGUAGE="JScript">
function VBArrayTest(vba)
```

```
var i, s;
 var a = new VBArray(vba);
 for (i = 1; i \le a.dimensions(); i++)
  s = "The lower bound of dimension";
  s += i + " is ";
  s += a.lbound(i)+ ".<BR>";
 return(s);
 }
}
-->
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT language="jscript">
 document.write(VBArrayTest(CreateVBArray()));
</SCRIPT>
</BODY>
```

Microsoft® JScript® length Property (Array)

See Also

Applies To

Description

Returns an integer value one higher than the highest element defined in an array.

Syntax

```
numVar = arrayObj.length
```

Remarks

As the elements in an array do not have to be contiguous, the **length** property is not necessarily the number of elements in the array. For example, in the following array definition,

my_array.length contains 7, not 2:

```
var my_array = new Array();
my_array[0] = "Test";
my_array[6] = "Another Test";
```

If a value smaller than its previous value is assigned to the **length** property, the array is truncated, and any elements with array indexes equal to or greater than the new value of the **length** property are lost.

If a value larger than its previous value is assigned to the **length** property, the array is expanded, and any new elements created have the value <u>undefined</u>.

The following example illustrates the use of the **length** property:

```
function LengthDemo()
{
  var a, l;
  a = new Array(0,1,2,3,4);
  l = a.length;
  return(l);
}
```

Language Reference Version 2

Microsoft® JScript® length Property (Function)

See Also

Applies To

Description

Returns the number of arguments defined for a function.

Syntax

functionname.length

The *functionname* argument is required and is the name of the function in question.

Remarks

The **length** property of a function is initialized by the scripting engine to the number of arguments in the function's definition when an instance of the function is created.

What happens when a function is called with a number of arguments different from the value of its **length** property depends on the function.

The following example illustrates the use of the **length** property:

```
function ArgTest(a, b)
{
  var i, s = "The ArgTest function expected ";
```

```
var numargs = ArgTest.arguments.length;
var expargs = ArgTest.length;
if (expargs < 2)
    s += expargs + " argument. ";
else
    s += expargs + " arguments. ";
if (numargs < 2)
    s += numargs + " was passed.";
else
    s += numargs + " were passed.";
return(s);
}</pre>
```

Microsoft® JScript® length Property (String)

Language Reference Version 1

See Also

Applies To

Description

Returns the length of a **String** object.

Syntax

strVariable.length
"String Literal".length

Remarks

The **length** property contains an integer that indicates the number of characters in the **String** object. The last character in the **String** object has an index of **length** - 1.

Applies To

Description

Places an HTML anchor with an HREF attribute around the text in a **String** object.

Syntax

```
strVariable.link(linkstring)
"String Literal".link(linkstring)
```

The *linkstring* argument is the text that you want to place in the HREF attribute of the HTML anchor.

Remarks

Call the **link** method to create a hyperlink out of a **String** object. The following is an example of how the method accomplishes this:

```
var strVariable = "This is a hyperlink";
strVariable = strVariable.link("http://www.micr
```

The value of *strVariable* after the last statement is:

No checking is done to see if the tag has already been applied to the string.

${\tt Microsoft @ JScript @ } LN2 \ Property$

See Also

Applies To

Description

Returns the natural logarithm of 2.

Syntax

```
var numVar
numVar = Math.LN2
```

Syntax

The **LN2** property is approximately equal to 0.693.

${\tt Microsoft @ JScript @} \ LN10 \ Property$

See Also

Applies To

Description

Returns the natural logarithm of 10.

Syntax

var numVar
numVar = Math.LN10

Remarks

The **LN10** property is approximately equal to 2.302.

Microsoft® JScript® log Method

See Also Applies To

Description

Returns the natural logarithm of a number.

Syntax

Math.log(number)

The *number* argument is a numeric expression for which the natural logarithm is sought.

Return Value

The return value is the natural logarithm of number. The base is e.

${\tt Microsoft @ JScript @ } LOG2E \ Property$

See Also

Applies To

Description

Returns the base-2 logarithm of *e*, Euler's constant.

Syntax

```
var varName
varName = objName.LOG2E
```

Remarks

The **LOG2E** property, a constant, is approximately equal to 1.442.

${\tt Microsoft @ JScript @ LOG10E}$

Property

See Also

Applies To

Description

Returns the base-10 logarithm of *e*, Euler's constant.

Syntax

var varName
varName = objName.LOG10E

Remarks

The **LOG10E** property, a constant, is approximately equal to 0.434.

Description

Performs a logical conjunction on two expressions.

Syntax

result = expression1 && expression2

The **&&** operator syntax has these parts:

Part	Description
result	Any <u>variable</u> .
expression1	Any <u>expression</u> .
expression2	Any expression.

Remarks

If, and only if, both expressions evaluate to **True**, *result* is **True**. If either expression evaluates to **False**, *result* is **False**.

For information on when a <u>run-time error</u> is generated by the **&&** operator, see the <u>Operator Behavior</u> table.

JScript uses the following rules for converting non-Boolean values to Boolean values:

• All objects are considered true.

- Strings are considered false if, and only if, they are empty.
- **null** and <u>undefined</u> are considered false.
- Numbers are false if, and only if, they are zero.

Description

Performs logical negation on an expression.

Syntax

result = !expression

The ! operator syntax has these parts:

Part	Description
result	Any <u>variable</u> .
expression	Any <u>expression</u> .

Remarks

The following table illustrates how *result* is determined.

If expression is	Then result is
True	False
False	True

All unary operators, such as the ! operator, evaluate expressions as follows:

• If applied to <u>undefined</u> or **null** expressions, a <u>run-time error</u> is raised.

- Objects are converted to strings.
- Strings are converted to numbers if possible. If not, a runtime error is raised.
- Boolean values are treated as numbers (0 if false, 1 if true).

The operator is applied to the resulting number.

For the ! operator, if *expression* is nonzero, *result* is zero. If *expression* is zero, *result* is 1.

Description

Performs a logical disjunction on two expressions.

Syntax

result = expression1 || expression2

The || operator syntax has these parts:

Part	Description
result	Any <u>variable</u> .
expression1	Any <u>expression</u> .
expression2	Any expression.

Remarks

If either or both expressions evaluate to **True**, *result* is **True**. The following table illustrates how *result* is determined:

If expression1 is	And expression2 is	The result is
True	True	True
True	False	True
False	True	True
False	False	False

For information on when a <u>run-time error</u> is generated by the **&&** operator, see the <u>Operator Behavior</u> table.

JScript uses the following rules for converting non-Boolean values to Boolean values:

- All objects are considered true.
- Strings are considered false if and only if they are empty.
- **null** and <u>undefined</u> are considered false.
- Numbers are false if, and only if, they are 0.

Microsoft® JScript® Math Object

See Also Methods Properties

Description

An intrinisic object that provides basic mathematics functionality and constants.

Syntax

Math[.{property | method}]

Remarks

The **Math** object cannot be created using the **new** operator, and gives an error if you attempt to do so. It is created by the scripting engine when the engine is loaded. All of its methods and properties are available to your script at all times.

See Also Applies To

Description

Returns the greater of two supplied numeric expressions.

Syntax

retVal = Math.max(number1, number2)

The **max** method syntax has these parts:

Part	Description	
retVal	The greater of <i>number1</i> or <i>number2</i> .	
number1	A <u>numeric expression</u> to be compared	
	to number2.	
number2	A numeric value to be compared to number1.	
	number1.	



See Also Applies To

Description

Returns the lesser of two supplied numbers.

Syntax

retVal = Math.min(number1, number2)

The **min** method syntax has these parts:

Part	Description	
retVal	The lesser of <i>number1</i> or <i>number2</i> .	
number1	A <u>numeric expression</u> to be compared	
	to number2.	
number2	A numeric value to be compared to	
	number1.	

Microsoft® JScript® moveFirst

Method

See Also

Applies To

Description

Resets the current item in the collection to the first item.

Syntax

```
myEnum.moveFirst()
```

The *myEnum* argument is any **Enumerator** object.

Remarks

If there are no items in the collection, the current item is set to **undefined**.

In following example, the **moveFirst** method is used to begin evaluating members of the **Drives** collection from the beginning of the list:

```
function ShowFirstAvailableDrive()
{
  var fso, s, e, x;
  fso = new ActiveXObject("Scripting.FileSyste
  e = new Enumerator(fso.Drives);
  e.moveFirst();
```

```
s = "";
do
 x = e.item();
  if (x.IsReady)
   s = x.DriveLetter + ":";
   break;
  else
   if (e.atEnd())
   {
    s = "No drives are available";
    break;
  e.moveNext();
while (!e.atEnd());
return(s);
```

Microsoft® JScript® moveNext

Method

See Also

Applies To

Description

Moves the current item to the next item in the collection.

Syntax

```
myEnum.moveNext()
```

The *myEnum* argument is any **Enumerator** object.

Remarks

If the enumerator is at the end of the collection or the collection is empty, the current item is set to **undefined**.

In following example, the **moveNext** method is used to move to the next drive in the **Drives** collection:

```
function ShowDriveList()
{
  var fso, s, n, e, x;
  fso = new ActiveXObject("Scripting.FileSyste
  e = new Enumerator(fso.Drives);
  s = "";
```

```
for (; !e.atEnd(); e.moveNext())
 {
  x = e.item();
  s = s + x.DriveLetter;
  s += " - ";
  if (x.DriveType == 3)
   n = x.ShareName;
  else if (x.IsReady)
   n = x.VolumeName;
  else
   n = "[Drive not ready]";
  s += n + " < br > ";
 return(s);
}
```

Microsoft® JScript® NaN Property

See Also Applies To

Description

A special value that indicates an arithmetic expression returned a value that was not a number.

Syntax

Number.NaN

The *number* argument is the **Number** object.

Remarks

The **Number** object does not have to be created before the **NaN** property can be accessed.

NaN does not compare equal to any value, including itself. To test if a value is equivalent to **NaN**, use the **isNaN** function.

See Also

Description

Creates a new object.

Syntax

new constructor[(arguments)]

The *constructor* argument calls object's <u>constructor</u>. The parentheses can be omitted if the constructor takes no arguments.

Remarks

The **new** operator performs the following tasks:

- 1. It creates an object with no members.
- 2. It calls the constructor for that object, passing a pointer to the newly created object as the **this** pointer.

The constructor then initializes the object according to the arguments passed to the constructor.

These are examples of valid uses of the **new** operator:

my_date = new Date("Jan 5 1996")

Microsoft® JScript® number Property

See Also

Applies to

Description

Returns or sets the numeric value associated with a specific error. The **Error** object's default property is **number**.

Syntax

object.number [= errornumber]

The **number** property syntax has these parts:

Part	Description
object	Any instance of the Error object.
errornumber	An integer representing an error.

Remarks

An error number is a 32-bit value. The upper 16-bit word is the facility code, while the lower word is the actual error code.

The following example illustrates the use of the **number** property:

```
try {
  x = y  // Cause an error.
}
```

```
catch(var e) { // Create local var. document.write(e) // Prints "[object document.write(e.number>>16 & 0x1FFF)// F document.write(e.number & 0xFFFF) // Prints document.write(e.description) // Prints "'y }
```

Microsoft® JScript® Object Object

See Also Methods Properties

Description

Provides functionality common to all JScript objects.

Syntax

new Object([value])

The optional *value* argument is used to convert a <u>primitive</u> data type (number, Boolean, string, or function) into an object. If omitted, an object with no contents is created.

Remarks

The **Object** object is contained in all other JScript objects--all of its methods and properties are available in all other objects. The methods can be redefined in user-defined objects, and are called by JScript at appropriate times. The **toString** method is an example of a frequently redefined **Object** method.

In this language reference, the description of each **Object** method includes both default and object-specific implementation information for the <u>intrinsic JScript objects</u>.

Microsoft® JScript® Operator Precedence

Operators in JScript are evaluated in a particular order. This order is known as the operator precedence. The following table lists the operators in highest to lowest precedence order. Operators with the same precedence are evaluated in left to right order in the expression.

Operator	Description
. [] ()	Field access, array indexing, and function calls
++ ~! delete new typeof void	Unary operators, return data type, object creation, undefined values
* / %	Multiplication, division, modulo division
+ - +	Addition, subtraction, string concatenation
<< >> >>>	Bit shifting
< <= > >= instanceof	Less than, less than or equal, greater than, greater than or equal, instanceof
== != === !==	Equality, inequality, identity, nonidentity
&	Bitwise AND
Λ	Bitwise XOR
	Bitwise OR
&&	Logical AND
	Logical OR
?:	Conditional
= OP=	Assignment, assignment with operation
,	Multiple evaluation

Parentheses are used to alter the order of evaluation. The expression within

parentheses is fully evaluated before its value is used in the remainder of the statement.

An operator with higher precedence is evaluated before one with lower precedence. For example:

$$z = 78 * (96 + 3 + 45)$$

There are five operators in this expression: =, *, (), +, and +. According to precedence, they are evaluated in the following order: (), *, +, +, =.

- 1. Evaluation of the expression within the parentheses is first: There are two addition operators, and they have the same precedence: 96 and 3 are added together and 45 is added to that total, resulting in a value of 144.
- 2. Multiplication is next: 78 and 144 are multiplied, resulting in a value of 10998.
- 3. Assignment is last: 11232 is assigned into z.

Microsoft® JScript® PI Property

See Also

Applies To

Description

Returns the ratio of the circumference of a circle to its diameter, approximately 3.141592653589793.

Syntax

var numVar
numVar = Math.PI

Syntax

The **PI** property, a constant, is approximately equal to 3.14159.

See Also Applies To

Description

Returns the value of a base expression taken to a specified power.

Syntax

Math.pow(base, exponent)

The **pow** method syntax has these parts:

Part	Description
base	The base value of the expression.
exponent	The exponent value of the expression.

Remarks

In the following example, a <u>numeric expression</u> equal to *base* exponent returns 1000.

Math.pow(10,3);

Microsoft® JScript® **prototype**Property

See Also

Applies To

Description

Returns a reference to the prototype for a class of objects.

Syntax

objectname.prototype

The *objectname* argument is the name of an object.

Remarks

Use the **prototype** property to provide a base set of functionality to a class of objects. New instances of an object "inherit" the behavior of the prototype assigned to that object.

For example, say you want to add a method to the **Array** object that returns the value of the largest element of the array. To do this, declare the function, add it to **Array.prototype**, and then use it.

```
function array_max()
{
  var i, max = this[0];
  for (i = 1; i < this.length; i++)</pre>
```

After this code is executed, y contains the largest value in the array x, or 6.

All <u>intrinsic JScript objects</u> have a **prototype** property that is read-only. Functionality may be added to the prototype, as in the example, but the object may not be assigned a different prototype. However, <u>user-defined objects</u> may be assigned a new prototype.

The method and property lists for each intrinsic object in this language reference indicate which ones are part of the object's prototype, and which are not.

Microsoft® JScript® random Method

See Also Applies To

Description

Returns a pseudorandom number between 0 and 1.

Syntax

Math.random()

Remarks

The pseudorandom number generated is between 0 and 1 inclusive. The random number generator is seeded automatically when JScript is first loaded.

Microsoft® JScript® RegExp Object

See Also Methods Properties

Description

Stores information on regular expression pattern searches.

Syntax

RegExp.*propertyname*

The *propertyname* argument is one of the **RegExp** object properties.

Remarks

The **RegExp** object cannot be created directly, but is always available for use. Its properties have <u>undefined</u> as their value until a successful regular expression search has been completed.

The following example illustrates the use of the **RegExp** object:

```
function matchDemo()
{
  var s;
  var re = new RegExp("d(b+)(d)","ig");
  var str = "cdbBdbsbdbdz";
  var arr = re.exec(str);
```

```
s = "$1 contains: " + RegExp.$1 + "<BR>";
s += "$2 contains: " + RegExp.$2 + "<BR>";
s += "$3 contains: " + RegExp.$3;
return(s);
}
```

Microsoft® JScript® Regular Expression Object

See Also Methods Properties

Description

Contains a regular expression pattern.

Syntax 1

var regularexpression = /pattern/[switch]

Syntax 2

var regularexpression = new RegExp("pattern",["switch"])

The regular expression object syntax has these parts:

Part	Description
pattern	Required. The regular expression pattern to use. If you use Syntax 1, delimit the pattern by "/" characters. If you use Syntax 2, enclose the pattern in quotation marks.
	 Optional. Enclose switch in quotation marks if you use Syntax 2. Available switches are: i (ignore case) g (global search for all occurrences of <i>pattern</i>) gi (global search, ignore case)

Remarks

Regular Expression objects store patterns used when searching strings for character combinations. After the **Regular Expression** object is created, it is either passed to a string method, or a string is passed to one of the regular expression methods. Information about the most recent search performed is stored in the **RegExp** object.

Use Syntax 1 when you know the search string ahead of time. Use Syntax 2 when the search string is changing frequently, or is unknown, such as strings taken from user input.

The *pattern* argument is compiled into an internal format before use. For Syntax 1, *pattern* is compiled as the script is loaded. For Syntax 2, *pattern* is compiled just before use, or when the **compile** method is called.

Microsoft® JScript® Regular Expression Syntax

See Also Applies To

Description

Special characters and sequences are used in writing patterns for regular expressions. The following table describes these characters and includes short examples showing how the characters are used.

Character	Description
\	Marks the next character as either a special character or a literal. For example, "n" matches the character "n". "\n" matches a newline character. The sequence "\\" matches "\" and "\(" matches "(".
٨	Matches the beginning of input.
\$	Matches the end of input.
*	Matches the preceding character zero or more times. For example, "zo*" matches either "z" or "zoo".
+	Matches the preceding character one or more times. For example, "zo+" matches "zoo" but not "z".
?	Matches the preceding character zero or one time. For example, "a?ve?" matches the "ve" in "never".
•	Matches any single character except a newline character.
(pattern)	Matches <i>pattern</i> and remembers the match. The matched substring can be retrieved from the resulting Matches collection, using Item [0][n] . To match parentheses characters (), use "\(" or "\)".

	<u> </u>
x y	Matches either <i>x</i> or <i>y</i> . For example, "z food" matches "z" or "food". "(z f)ood" matches "zoo" or "food".
{ n }	n is a nonnegative integer. Matches exactly n times. For example, "o{2}" does not match the "o" in "Bob," but matches the first two o's in "foooood".
{n,}	n is a nonnegative integer. Matches at least n times. For example, "o{2,}" does not match the "o" in "Bob" and matches all the o's in "foooood". "o{1,}" is equivalent to "o+". "o{0,}" is equivalent to "o*".
{n,m}	m and n are nonnegative integers. Matches at least n and at most m times. For example, "o{1,3}" matches the first three o's in "fooooood". "o{0,1}" is equivalent to "o?".
[xyz]	A character set. Matches any one of the enclosed characters. For example, "[abc]" matches the "a" in "plain".
[^ <i>xyz</i>]	A negative character set. Matches any character not enclosed. For example, "[^abc]" matches the "p" in "plain".
[a-z]	A range of characters. Matches any character in the specified range. For example, "[a-z]" matches any lowercase alphabetic character in the range "a" through "z".
[^m-z]	A negative range characters. Matches any character not in the specified range. For example, "[m-z]" matches any character not in the range "m" through "z".
\ b	Matches a word boundary, that is, the position between a word and a space. For example, "er\b" matches the "er" in "never" but not the "er" in "verb".
\ B	Matches a nonword boundary. "ea*r\B" matches the "ear" in "never early".

\ d	Matches a digit character. Equivalent to [0-9].
\D	Matches a nondigit character. Equivalent to [^0-9].
\ f	Matches a form-feed character.
\n	Matches a newline character.
\ r	Matches a carriage return character.
\s	Matches any white space including space, tab, form-feed, etc. Equivalent to "[\f\n\r\t\v]".
\S	Matches any nonwhite space character. Equivalent to "[^ \f\n\r\t\v]".
\t	Matches a tab character.
\ v	Matches a vertical tab character.
\w	Matches any word character including underscore. Equivalent to "[A-Za-z0-9_]".
\ W	Matches any nonword character. Equivalent to " [^A-Za-z0-9_]".
\num	Matches <i>num</i> , where <i>num</i> is a positive integer. A reference back to remembered matches. For example, "(.)\1" matches two consecutive identical characters.
\ n	Matches <i>n</i> , where <i>n</i> is an octal escape value. Octal escape values must be 1, 2, or 3 digits long. For example, "\11" and "\011" both match a tab character. "\0011" is the equivalent of "\001" & "1". Octal escape values must not exceed 256. If they do, only the first two digits comprise the expression. Allows ASCII codes to be used in regular expressions.
\ x n	Matches <i>n</i> , where <i>n</i> is a hexadecimal escape value. Hexadecimal escape values must be exactly two digits long. For example, "\x41" matches "A". "\x041" is equivalent to "\x04" & "1". Allows ASCII codes to be used in regular expressions.

See Also

Description

Exits from the current function and returns a value from that function.

Syntax

```
return [expression];
```

The *expression* argument is the value to be returned from the function. If omitted, the function does not return a value.

Remarks

You use the **return** statement to stop execution of a function and return the value of *expression*. If *expression* is omitted, or no **return** statement is executed from within the function, the <u>expression</u> that called the current function is assigned the value undefined.

The following example illustrates the use of the **return** statement:

```
function myfunction(arg1, arg2)
{
  var r;
  r = arg1 * arg2;
```

```
return(r);
}
```

See Also

Applies To

Description

Returns an **Array** object with the elements reversed.

Syntax

```
arrayobj.reverse()
```

Remarks

The **reverse** method reverses the elements of an **Array** object in place. It does not create a new **Array** object during execution.

If the array is not contiguous, the **reverse** method creates elements in the array that fill the gaps in the array. Each of these created elements has the value <u>undefined</u>.

The following example illustrates the use of the **reverse** method:

```
function ReverseDemo()
{
  var a, l;
  a = new Array(0,1,2,3,4);
  l = a.reverse();
  return(l);
```

Microsoft® JScript® round Method

See Also Applies To

Description

Returns a supplied <u>numeric expression</u> rounded to the nearest integer.

Syntax

Math.round(number)

The *number* argument is the value to be rounded to the nearest integer.

Remarks

If the decimal portion of *number* is 0.5 or greater, the return value is equal to the smallest integer greater than *number*. Otherwise, **round** returns the largest integer less than or equal to *number*.

Microsoft® JScript® ScriptEngine

Function

See Also

Description

Returns a string representing the scripting language in use.

Syntax

ScriptEngine()

Return Values

The **ScriptEngine** function can return any of the following strings:

String	Description
INCrint	Indicates that Microsoft JScript is the
	current scripting engine.
VBA	Indicates that Microsoft Visual
	Basic® for Applications is the
	current scripting engine.
	Indicates that Microsoft Visual Basic
	Scripting Edition is the current
	scripting engine.

Remarks

The following code illustrates the use of the **ScriptEngine** function:

```
function GetScriptEngineInfo()
{
   var s;
   s = ""; // Build string with necessary info.
   s += ScriptEngine() + " Version ";
   s += ScriptEngineMajorVersion() + ".";
   s += ScriptEngineMinorVersion() + ".";
   s += ScriptEngineBuildVersion();
   return(s);
}
```

ScriptEngineBuildVersion Function

See Also

Description

Returns the build version number of the scripting engine in use.

Syntax

ScriptEngineBuildVersion()

Return Values

The return value corresponds directly to the version information contained in the dynamic-link library (DLL) for the scripting language in use.

Remarks

The following code illustrates the use of the **ScriptEngineBuildVersion** function:

```
function GetScriptEngineInfo()
{
   var s;
   s = ""; // Build string with necessary info.
   s += ScriptEngine() + " Version ";
   s += ScriptEngineMajorVersion() + ".";
   s += ScriptEngineMinorVersion() + ".";
   s += ScriptEngineBuildVersion();
```

```
return(s);
```

ScriptEngineMajorVersion Function

See Also

Description

Returns the major version number of the scripting engine in use.

Syntax

ScriptEngineMajorVersion()

Return Values

The return value corresponds directly to the version information contained in the dynamic-link library(DLL) for the scripting language in use.

Remarks

The following code illustrates the use of the **ScriptEngineMajorVersion** function:

```
function GetScriptEngineInfo()
{
   var s;
   s = ""; // Build string with necessary info.
   s += ScriptEngine() + " Version ";
```

```
s += ScriptEngineMajorVersion() + ".";
s += ScriptEngineMinorVersion() + ".";
s += ScriptEngineBuildVersion();
return(s);
}
```

ScriptEngineMinorVersion Function

See Also

Description

Returns the minor version number of the scripting engine in use.

Syntax

ScriptEngineMinorVersion()

Return Values

The return value corresponds directly to the version information contained in the dynamic-link library (DLL) for the scripting language in use.

Remarks

The following code illustrates the use of the **ScriptEngineMinorVersion** function:

```
function GetScriptEngineInfo()
{
   var s;
   s = ""; // Build string with necessary info.
   s += ScriptEngine() + " Version ";
```

```
s += ScriptEngineMajorVersion() + ".";
s += ScriptEngineMinorVersion() + ".";
s += ScriptEngineBuildVersion();
return(s);
}
```

See Also

Description

Creates variables used with conditional compilation statements.

Syntax

The **@set** statement syntax has these parts:

Part	Description
varname	Valid JScript variable name. Must be preceded by an "@" character at all times.
term	Zero or more unary operators followed by a constant, conditional compilation variable, or parenthesized expression.

Remarks

Numeric and Boolean variables are supported for conditional compilation. Strings are not. Variables created using **@set** are generally used in conditional compilation statements, but can be used anywhere in JScript code.

Examples of variable declarations look like this:

The following operators are supported in parenthesized expressions:

- ! ~
- */%
- + -
- << >> >>>
- <<=>>=
- == != === !==
- & ^ |
- && ||

If a variable is used before it has been defined, its value is **NaN**. **NaN** can be checked for using the **@if** statement:

This works because **NaN** is the only value not equal to itself.



See Also Applies To

Description

Returns the sine of a number.

Syntax

Math.sin(number)

The *number* argument is a <u>numeric expression</u> for which the sine is sought.

Remarks

The return value is the sine of its numeric argument.

Microsoft® JScript® slice Method (Array)

See Also

Applies To

Description

Returns a section of an array.

Syntax

arrayObj.slice(start, [end])

The **slice** method syntax has these parts:

Part	Description	
arrayObj	Required. An Array object.	
lictart i	Required. The zero-based index of the beginning of the specified portion of <i>arrayObj</i> .	
	Optional. The zero-based index of the end of the specified portion of <i>arrayObj</i> .	

Remarks

The **slice** method returns an **Array** object containing the specified portion of *arrayObj*.

The **slice** method copies up to, but not including, the element indicated by *end*. If negative, *end* indicates an offset from the end of *arrayObj*. In addition, it is not zero-based. If omitted, extraction continues to the end of *arrayObj*.

In the following example, all but the last element of *myArray* is copied into *newArray*:

If an object reference is copied from *arrayObj* to the result, the object reference in the result still points to the same object. Changes to that object are reflected in both arrays.

Microsoft® JScript® slice Method (String)

See Also

Applies To

Description

Returns a section of a string.

Syntax

stringObj.slice(start, [end])

The **slice** method syntax has these parts:

Part	Description	
stringObj	Required. A String object or literal.	
HCTAYT I	Required. The zero-based index of the beginning of the specified portion of <i>stringObj</i> .	
IIPNA I	Optional. The zero-based index of the end of the specified portion of <i>stringObj</i> .	

Remarks

The **slice** method returns a **String** object containing the specified portion of *stringObj*.

If negative, *end* indicates an offset from the end of *stringObj*. In addition, it is not zero-based. If omitted, extraction continues to the end of *stringObj*.

In the example that follows, the two uses of the **slice** method return the

same thing. Negative one in the second example points to the last character in str1 as the ending point:

str1.slice(0)
str2.slice(0,-1)

Applies To

Description

Places HTML <SMALL> tags around text in a **String** object.

Syntax

```
strVariable.small()
"String Literal".small()
```

Remarks

The example that follows demonstrates how the **small** method works:

```
var strVariable = "This is a string";
strVariable = strVariable.small();
```

The value of *strVariable* after the last statement is:

```
<SMALL>This is a string</SMALL>
```

No checking is done to see if the tag has already been applied to the string.

Applies To

Description

Returns an **Array** objec with the elements sorted.

Syntax

arrayobj.sort(sortfunction)

The *sortfunction* argument is the name of the function used to determine the order of the elements. If omitted, the elements are sorted in ascending, ASCII character order.

Remarks

The **sort** method sorts the **Array** object in place; no new **Array** object is created during execution.

If you supply a function in the *sortfunction* argument, it must return one of the following values:

- A negative value if the first argument passed is less than the second argument.
- Zero if the two arguments are equivalent.
- A positive value if the first argument is greater than the second argument.

The following example illustrates the use of the **sort** method:

function SortDemo()

```
var a, l;
a = new Array("X","y","d", "Z",
l = a.sort();
return(l);
}
```

Applies To

Description

Returns a copy of the text of the regular expression pattern. Read-only.

Syntax

rgexp.source

The *rgexp* argument is a **Regular expression** object. It can be a variable name or a literal.

The following example illustrates the use of the **source** property:

```
function SourceDemo(re, s)
{
  var s1;
  // Test string for existence of regular expressic
  if (re.test(s))
    s1 = " contains ";
  else
    s1 = " does not contain ";
  // Get the text of the regular expression itself.
```

```
return(s + s1 + re.source);
}
```

${\tt Microsoft @ JScript @ } {\bf Sqrt \ Method}$

See Also Applies To

Description

Returns the square root of a number.

Syntax

Math.sqrt(number)

The *number* argument is a <u>numeric expression</u>.

Remarks

If *number* is negative, the return value is zero.

${\tt Microsoft @ JScript @ } SQRT1_2$

Property

See Also

Applies To

Description

Returns he square root of 0.5, or one divided by the square root of 2.

Syntax

```
var numVar
numVar = Math.SQRT1_2
```

Remarks

The **SQRT1_2** property, a constant, is approximately equal to 0.707.

${\tt Microsoft @ JScript @ } SQRT2 \ Property$

See Also

Applies To

Description

Returns the square root of 2.

Syntax

```
var numVar
numVar = Math.SQRT2
```

Syntax

The **SQRT2** property, a constant, is approximately equal to 1.414.

Applies To

Description

Places HTML <STRIKE> tags around text in a **String** object.

Syntax

```
strVariable.strike()
"String Literal".strike()
```

Remarks

The following example demonstrates how the **strike** method works:

```
var strVariable = "This is a string object";
strVariable = strVariable.strike( );
```

The value of *strVariable* after the last statement is:

```
<STRIKE>This is a string object</STRIKE>
```

No checking is done to see if the tag has already been applied to the string.

Microsoft® JScript® String Object

See Also Methods Properties

Description

Allows manipulation and formatting of text strings and determination and location of substrings within strings.

Syntax

```
StringObj[.method]
"String Literal"[.method]
```

Remarks

String objects can be created implicitly using string literals. **String** objects created in this fashion (referred to as standard strings) are treated differently than **String** objects created using the **new** operator. All string literals share a common, global string object. So, if a property is added to a string literal, it is available to all standard string objects:

```
var alpha, beta;
alpha = "This is a string";
beta = "This is also a string";
alpha.test = 10;
```

In this example, *test* is now defined for *beta* and all future string literals. In the following

example, however, added properties are treated differently:

var gamma, delta;
gamma = new String("This is a string delta = new String("This is also a string to the strin

gamma.test = 10;

In this case, *test* is not defined for *delta*. Each **String** object declared as a **new String** object has its own set of members. This is the only case where **String** objects and string literals are handled differently.

Applies To

Description

Places HTML <SUB> tags around text in a **String** object.

Syntax

```
strVariable.sub()
"String Literal".sub()
```

Remarks

The following example demonstrates how the **sub** method works:

```
var strVariable = "This is a string object"
strVariable = strVariable.sub( );
```

The value of *strVariable* after the last statement is:

```
<SUB>This is a string object</SUB>
```

No checking is done to see if the tag has already been applied to the string. See Also Applies To

Description

Returns a substring beginning at a specified location and having a specified length.

Syntax

stringvar.substr(start [, length])

The **substr** method syntax has these parts:

Part	Description	
stringvar	Required. A string literal or String object from which the substring is extracted.	
start	Required. The starting position of the desired substring. The index of the first character in the string is zero.	
II <i>e</i> nath	Optional. The number of characters to include in the returned substring.	

Remarks

If *length* is zero or negative, an empty string is returned. If not specified, the substring continues to the end of *stringvar*.

The following example illustrates the use of the **substr** method:

```
function SubstrDemo()
{
  var s, ss;
  var s = "The quick brown fox jumped over the ss = s.substr(16, 3);
  // Returns "fox".
  return(ss);
}
```

Microsoft® JScript® substring Method

See Also Applies To

Description

Returns the substring at the specified location within a **String** object.

Syntax

strVariable.substring(start, end)
"String Literal".substring(start, end)

The **substring** method syntax has these arguments:

Part	Description	
start	The zero-based index indicating the beginning of the substring.	
and	The zero-based index indicating the end of the substring.	

Remarks

The **substring** method returns a **String** object containing the substring derived from the original object.

The **substring** method uses the lower of *start* and *end* as the beginning point of the substring. For example, *strvar*.**substring**(0, 3) and *strvar*.**substring**(3, 0) return the same substring.

The only exception to this is for negative parameters. If the first parameter is less than zero, it is treated as zero. If the second parameter is negative, it is set to the value of the

first parameter.

The length of the substring is equal to the absolute value of the difference between *start* and *end*. For example, the length of the substring returned in *strvar*.**substring(**0, 3**)** and *strvar*.**substring(**3, 0**)** is three.

Finally, *start* and *end* can be strings. If so, these strings are coerced into integers if possible. If not, the value of the parameter is treated as zero.

The following example illustrates the use of the **substring** method:

```
function SubstringDemo()
{
  var s, ss;
  var s = "The quick brown fox jum
  ss = s.substring(16, 19);
  return(ss);
}
```

Description

Used to find the difference between two numbers or to indicate the negative value of a numeric expression.

Syntax 1

result = number1 - number2

Syntax 2

-number

The - operator syntax has these parts:

Part	Description
result	Any numeric <u>variable</u> .
number	Any <u>numeric expression</u> .
number1	Any numeric expression.
number2	Any numeric expression.

Remarks

In Syntax 1, the - operator is the arithmetic subtraction operator used to find the difference between two numbers. In Syntax 2, the - operator is used as the unary negation operator to indicate

the negative value of an expression.

For information on when a <u>run-time error</u> is generated by Syntax 1, see the <u>Operator</u> Behavior table.

For Syntax 2, as for all unary operators, expressions are evaluated as follows:

- If applied to <u>undefined</u> or **null** expressions, a run-time error is raised.
- Objects are converted to strings.
- Strings are converted to numbers if possible. If not, a runtime error is raised.
- Boolean values are treated as numbers (0 if false, 1 if true).

The operator is applied to the resulting number. In Syntax 2, if the resulting number is nonzero, *result* is equal to the resulting number with its sign reversed. If the resulting number is zero, *result* is zero.

Applies To

Description

Places HTML <SUP> tags around text in a **String** object.

Syntax

```
strVariable.sup()
"String Literal".sup()
```

Remarks

The following example demonstrates how the **sup** method works:

```
var strVariable = "This is a string object";
strVariable = strVariable.sup( );
```

The value of *strVariable* after the last statement is:

```
<SUP>This is a string object</SUP>
```

No checking is done to see if the tag has already been applied to the string.

Description

Enables the execution of one or more statements when a specified expression's value matches a label.

Syntax

```
switch (expression) {
   case label:
     statementlist
   case label:
     statementlist
...
   default:
     statementlist
}
```

The **switch** statement syntax has these parts:

An identifier to be matched against <i>expression</i> label === expression, execution starts with the statementlist immediately after the colon, and continues until it encounters either a break	
label === expression, execution starts with the statementlist immediately after the colon, and continues until it encounters either a break	
statement, which is optional, or the end of the switch statement.	

Remarks

Use the **default** clause to provide a statement to be executed if none of the label values matches *expression*. It can appear anywhere within the **switch** code block.

Zero or more *label* blocks may be specified. If no *label* matches the value of *expression*, and a **default** case is not supplied, no statements are executed.

Execution flows through a switch statement as follows:

- 1. Evaluate *expression* and look at *label* in order until a match is found.
- 2. If a *label* value equals *expression*, execute its accompanying *statementlist*.
 - Continue execution until a **break** statement is encountered, or the **switch** statement ends. This means that multiple *label* blocks are executed if a **break** statement is not used.
- 3. If no *label* equals *expression*, go to the **default** case. If there is no **default** case, go to last step.
- 4. Continue execution at the statement following the end of the **switch** code block.

The following example tests an object for its type:

```
function MyObject() {
...}
switch (object.constructor){
```

```
case Date:
...
case Number:
...
case String:
...
case MyObject:
...
default:
...
```

Microsoft® JScript® tan Method

See Also Applies To

Description

Returns the tangent of a number.

Syntax

Math.tan(number)

The *number* argument is a <u>numeric expression</u> for which the tangent is sought.

Remarks

The return value is the tangent of *number*.

Applies To

Description

Returns a Boolean value that indicates whether or not a pattern exists in a searched string.

Syntax

rgexp.test(str)

The **test** method syntax has these parts:

Part	Description
rgexp	Required. A Regular Expression object. Can be a variable name or a literal.
	Required. The string to test a search on.

Remarks

The **test** method checks to see if a pattern exists within a string and returns **true** if so, and **false** otherwise.

The **RegExp** object is not modified by the **test** method.

The following example illustrates the use of the **test** method:

function TestDemo(re, s)

```
var s1;
// Test string for existence of regular expressic
if (re.test(s))
   s1 = " contains ";
else
   s1 = " does not contain ";
// Get text of the regular expression itself.
return(s + s1 + re.source);
}
```

Description

Refers to the current object.

Syntax

this.property

Remarks

The **this** keyword is typically used in object <u>constructors</u> to refer to the current object. In the following example, **this** refers to the newly created Car object, and assigns values to three properties:

```
function Car(color, make, model)
{
  this.color = color;
  this.make = make;
  this.model = model;
}
```

For client versions of JScript, **this** refers to the **window** object if used outside of the context of any other object.

Applies To

Description

Returns a standard JScript array converted from a VBArray.

Syntax

safeArray.toArray()

The *safeArray* argument is a **VBArray** object.

Remarks

The conversion translates the multidimensional VBArray into a single dimensional JScript array. Each successive dimension is appended to the end of the previous one. For example, a VBArray with three dimensions and three elements in each dimension is converted into a JScript array as follows:

Suppose the VBArray contains: (1, 2, 3), (4, 5, 6), (7, 8, 9). After translation, the JScript array contains: 1, 2, 3, 4, 5, 6, 7, 8, 9.

There is currently no way to convert a JScript array into a VBArray.

The following example consists of three parts. The first part is VBScript code to create a Visual Basic safe array. The second part is JScript code that converts the VB safe array to a JScript array. Both of these parts go into the <HEAD> section of an HTML page. The third part is the JScript code that goes in the <BODY> section to run the other two parts.

```
<HEAD>
<SCRIPT LANGUAGE="VBScript">
<!--
Function CreateVBArray()
 Dim i, j, k
 Dim a(2, 2)
k = 1
 For i = 0 To 2
  For j = 0 To 2
   a(j, i) = k
   document.writeln(k)
   k = k + 1
  Next
  document.writeln("<BR>")
 Next
 CreateVBArray = a
End Function
-->
</SCRIPT>
<SCRIPT LANGUAGE="JScript">
<!--
function VBArrayTest(vbarray)
 var a = new VBArray(vbarray);
 var b = a.toArray();
 var i;
 for (i = 0; i < 9; i++)
```

```
{
   document.writeln(b[i]);
}
-->
</SCRIPT>
</HEAD>
<BODY;>
<SCRIPT LANGUAGE="JScript">
<!--
   VBArrayTest(CreateVBArray());
-->
</SCRIPT>
</BODY>
```

Microsoft® JScript® toLowerCase

Method

See Also

Applies To

Description

Returns a string where all alphabetic characters have been converted to lowercase.

Syntax

```
strVariable.toLowerCase()
"String Literal".toLowerCase()
```

Remarks

The **toLowerCase** method has no effect on nonalphabetic characters.

The following example demonstrates the effects of the **toLowerCase** method:

var strVariable = "This is a STRIN(strVariable = strVariable.toLowerC

The value of *strVariable* after the last statement is:

this is a string object

Microsoft® JScript® toString Method

See Also Applies To

Description

Returns a string representation of an object.

Syntax

objectname.toString([radix])

The **toString** method syntax has these parts:

Part	Description	
objectname	Required. An object for which a string	
	representation is sought.	
radiy	Optional. Specifies a radix for converting numeric	
	values to strings.	

Remarks

The **toString** method is a member of all built-in JScript objects. How it behaves depends on the object type:

Object	Behavior
	Elements of an Array are converted to strings. The resulting strings are concatenated, separated by commas.
Boolean	If the Boolean value is true , returns " true ". Otherwise, returns "false"

Function	Returns a string returned of the following form, where <i>functionname</i> is the name of the function whose toString method was called: function functionname() { [native code] }	
Number	Returns the textual representation of the number.	
String	Returns the value of the String object.	
Default	Returns "[object objectname]", where objectname is the name of the object type.	

The following example illustrates the use of the toString method with a radix argument:

```
function CreateRadixTable ()
{
  var s1, s2, s3, x;
  document.write("Hex Dec Bin<BR>");
  for (x = 0; x < 16; x++)
  {
    switch(x)
    {
    case 0:
        s1 = " ";
        s2 = " ";
        break;
    case 1:</pre>
```

```
s1 = " ";
 s2 = " ";
 s3 = " ";
 break;
case 2:
 s3 = " ";
 break;
case 3:
 s3 = " ";
 break;
case 4:
 s3 = " ";
 break;
case 5:
 s3 = "";
 break;
case 6:
 s3 = " ";
 break;
case 7:
 s3 = " ";
 break;
case 8:
```

Microsoft® JScript® toUpperCase

Method

See Also

Applies To

Description

Returns a string where all alphabetic characters have been converted to uppercase.

Syntax

```
strVariable.toUpperCase( )
"String Literal".toUpperCase( )
```

Remarks

The **toUpperCase** method has no effect on nonalphabetic characters.

The following example demonstrates the effects of the **toUpperCase** method:

var strVariable = "This is a STRIN(strVariable = strVariable.toUpperCa

The value of *strVariable* after the last statement is:

THIS IS A STRING OBJECT

Microsoft® JScript® typeof Operator

See Also

Description

Returns a string that identifies the data type of an expression.

Syntax

typeof [(] expression [)];

The *expression* argument is any <u>expression</u> for which type information is sought.

Remarks

The **typeof** operator returns type information as a string. There are six possible values that **typeof** returns: "number," "string," "boolean," "object," "function," and "undefined."

The parentheses are optional in the **typeof** syntax.

Microsoft® JScript® ubound Method

See Also Applies To

Description

Returns the highest index value used in the specified dimension of the VBArray.

Syntax

safeArray.ubound(dimension)

The **ubound** method syntax has these parts:

Part	Description
safeArray	Required. A VBArray object.
dimension	Optional. The dimension of the VBArray for which the higher bound index is wanted. If omitted, ubound behaves as if a 1 was passed.

Remarks

If the VBArray is empty, the **ubound** method returns **undefined**. If *dim* is greater than the number of dimensions in the VBArray, or is negative, the method generates a "Subscript out of range" error.

The following example consists of three parts. The first part is VBScript code to create a Visual Basic safe array. The second part is JScript code that determines the the number of dimensions in the safe array and the

upper bound of each dimension. Both of these parts go into the <HEAD> section of an HTML page. The third part is the JScript code that goes in the <BODY> section to run the other two parts.

```
<HEAD>
<SCRIPT LANGUAGE="VBScript">
<!--
Function CreateVBArray()
 Dim i, j, k
 Dim a(2, 2)
 k = 1
 For i = 0 To 2
  For j = 0 To 2
   a(j, i) = k
   k = k + 1
  Next
 Next
 CreateVBArray = a
End Function
-->
</SCRIPT>
<SCRIPT LANGUAGE="JScript">
function VBArrayTest(vba)
var i, s;
var a = new VBArray(vba);
```

```
for (i = 1; i <= a.dimensions(); i++)
{
    s = "The upper bound of dimension ";
    s += i + " is ";
    s += a.ubound(i)+ ".<BR>";
    return(s);
}
}
-->
</SCRIPT>
</HEAD>

</BODY>
<SCRIPT language="jscript">
    document.write(VBArrayTest(CreateVBArray()));
</SCRIPT>
</BODY>
```

Microsoft® JScript® unescape Method

See Also

Applies To

Description

Decodes **String** objects encoded with the **escape** method.

Syntax

unescape(charstring)

The *charstring* argument is a **String** object to be decoded.

Remarks

The **unescape** method returns a new **String** object that contains the contents of *charstring*. All characters encoded with the %xx hexadecimal form are replaced by their <u>ASCII</u> character set equivalents.

Characters encoded in **%u**xxxx format (Unicode characters) are replaced with the Unicode character with hexadecimal encoding xxxx.

See Also

Description

Performs an unsigned right shift of the bits in an expression.

Syntax

result = expression1 >>> expression2

The >>> operator syntax has these parts:

Part	Description
result	Any <u>variable</u> .
expression1	Any <u>expression</u> .
expression2	Any expression.

Remarks

The >>> operator shifts the bits of *expression1* right by the number of bits specified in *expression2*. Zeroes are filled in from the left. Digits shifted off the right are discarded. For example:

equals 1073741820 (00111111 11111111 11111111 11111100 in binary).

For information on when a <u>run-time error</u> is generated by the >>> operator, see the <u>Operator Behavior</u> table.

See Also

Applies To

Description

Returns the primitive value of the specified object.

Syntax

object.valueOf()

The *object* argument is any JScript object.

Remarks

The **valueOf** method is defined differently for each intrinsic JScript object.

Object	Return Value	
Array	The elements of the array are converted into strings, and the strings are concatenated together, separated by commas. This behaves the same as the Array.toString and Array.join methods.	
Boolean	The Boolean value.	
Date	The stored time value in milliseconds since midnight, January 1, 1970 <u>UTC</u> .	
Function	The function itself.	
Number	The numeric value.	
Object	The object itself. This is the default.	

String The string value.

The **Math** object does not have a **valueOf** method.

See Also

Description

Declares a variable.

Syntax

```
var variable [ = value ] [, variable2 [ = value2], ...]
```

The **var** statement syntax has the following parts:

Part	Description
variable,	The names of the variables
variable2	being declared.
value value?	The initial value assigned to the
value, value2	variable.

Remarks

Use the **var** statement to declare variables. These variables can be assigned values at declaration or later in your script. Examples of declaration follow:

```
var index;
var name = "Thomas Jefferson";
var answer = 42, counter, numpages = 10;
```

Microsoft® JScript® VBArray Object

See Also Methods Properties

Description

Provides access to Visual Basic safe arrays.

Syntax

new VBArray(safeArray)

The *safeArray* is a **VBArray** value.

Remarks

VBArrays are read-only, and cannot be created directly. The *safeArray* argument must have obtained a **VBArray** value before being passed to the **VBArray** constructor. This can only be done by retrieving the value from an existing ActiveX or other object.

VBArrays can have multiple dimensions. The indices of each dimension can be different. The **dimensions** method retrieves the number of dimensions in the array; the **lbound** and **ubound** methods retrieve the range of indices used by each dimension.

The following example consists of three parts. The first part is VBScript code to create a Visual Basic safe array. The second part is JScript code that converts the VB safe array to a JScript array. Both of these parts go into the <HEAD> section of an HTML page. The third part is the JScript code that goes in the <BODY> section to run the other two parts.

```
<HEAD>
<SCRIPT LANGUAGE="VBScript">
<!--
Function CreateVBArray()
 Dim i, j, k
 Dim a(2, 2)
k = 1
 For i = 0 To 2
  For j = 0 To 2
   a(j, i) = k
   document.writeln(k)
   k = k + 1
  Next
  document.writeln("<BR>")
 Next
 CreateVBArray = a
End Function
-->
</SCRIPT>
<SCRIPT LANGUAGE="JScript">
<!--
function VBArrayTest(vbarray)
 var a = new VBArray(vbarray);
 var b = a.toArray();
 var i;
 for (i = 0; i < 9; i++)
```

```
{
   document.writeln(b[i]);
}
-->
</SCRIPT>
</HEAD>
<BODY;>
<SCRIPT LANGUAGE="JScript">
<!--
   VBArrayTest(CreateVBArray());
-->
</SCRIPT>
</BODY>
```

Microsoft® JScript® void Operator

See Also

Description

Prevents an expression from returning a value..

Syntax

void expression

The *expression* argument is any valid JScript <u>expression</u>.

Remarks

The **void** operator evaluates its expression, and returns **undefined**. It is most useful in situations where you want an expression evaluated but do not want the results visible to the remainder of the script.



See Also

Description

Executes a statement until a specified condition is **false**.

Syntax

while (*expression*) *statement*

The **while** statement syntax has these parts:

Part	Description	
expression	A Boolean expression checked before each iteration of the loop. If expression is true , the loop is executed. If expression is false , the loop is terminated.	
	The statement to be executed if expression is true . Can be a compound statement.	

Remarks

The **while** statement checks *expression* before a loop is first executed. If *expression* is **false** at this time, the loop is never

executed.

The following example illustrates the use of the **while** statement:

```
function BreakTest(breakpoint)
{
  var i = 0;
  while (i < 100)
  {
  if (i == breakpoint)
    break;
    i++;
  }
  return(i);
}</pre>
```

See Also

Description

Establishes the default object for a statement.

Syntax

with (object) statement

The **with** statement syntax has these parts:

Part	Description
object	The new default object.
	The statement for which <i>object</i> is the default object. Can be a <u>compound</u> <u>statement</u> .

Remarks

The **with** statement is commonly used to shorten the amount of code that you have to write in certain situations. In the example that follows, notice the repeated use of **Math**:

x = Math.cos(3 * Math.PI) + Math.sin(Math.LI)
y = Math.tan(14 * Math.E)

When you use the **with** statement, your code becomes shorter and easier to read:

```
with (Math)
{
    x = cos(3 * PI) + sin (LN10)
    y = tan(14 * E)
}
```

${\tt Microsoft @ JScript @ JScript Run-time}$

Errors

JScript Syntax Errors

Error Number	Description
5	Invalid procedure call or argument
6	Overflow
7	Out of memory
9	Subscript out of range
10	This array is fixed or temporarily locked
11	Division by zero
13	Type mismatch
14	Out of string space
17	Can't perform requested operation
28	Out of stack space
35	Sub or Function not defined
48	Error in loading DLL
51	Internal error
52	Bad file name or number
53	File not found
54	Bad file mode
55	File already open
57	Device I/O error
58	File already exists
61	Disk full
62	Input past end of file
67	Too many files
68	Device unavailable
70	Permission denied
71	Disk not ready

74	Can't rename with different drive
75	Path/File access error
76	Path not found
91	Object variable or With block variable not set
92	For loop not initialized
94	Invalid use of Null
322	Can't create necessary temporary file
424	Object required
429	Automation server can't create object
430	Class doesn't support Automation
432	File name or class name not found during Automation operation
438	Object doesn't support this property or method
440	Automation error
445	Object doesn't support this action
446	Object doesn't support named arguments
447	Object doesn't support current locale setting
448	Named argument not found
449	Argument not optional
450	Wrong number of arguments or invalid property assignment
451	Object not a collection
453	Specified DLL function not found
458	Variable uses an Automation type not supported in JScript
462	The remote server machine does not exist or is unavailable
501	Cannot assign to variable
502	Object not safe for scripting
503	Object not safe for initializing
504	Object not safe for creating
507	An exception occurred
5000	Cannot assign to 'this'
5001	Number expected
	-

5002	Function expected
5003	Cannot assign to a function result
5004	Cannot index object
5005	String expected
5006	Date object expected
5007	Object expected
5008	Illegal assignment
5009	Undefined identifier
5010	Boolean expected
5011	Can't execute code from a freed script
5012	Object member expected
5013	VBArray expected
5014	JScript object expected
5015	Enumerator object expected
5016	Regular Expression object expected
5017	Syntax error in regular expression
5018	Unexpected quantifier
5019	Expected ']' in regular expression
5020	Expected ')' in regular expression
5021	Invalid range in character set
5022	Exception thrown and not caught
5023	Function does not have a valid prototype object

${\tt Microsoft \& JScript \& } JScript \ Syntax$

Errors

JScript Run-time Errors

Error Number	Description
1001	Out of memory
1002	Syntax error
1003	Expected ':'
1004	Expected ';'
1005	Expected '('
1006	Expected ')'
1007	Expected ']'
1008	Expected '{'
1009	Expected '}'
1010	Expected identifier
1011	Expected '='
1012	Expected '/'
1013	Invalid number
1014	Invalide character
1015	Unterminated string constant
1016	Unterminated comment
1018	'return' statement outside of function
1019	Can't have 'break' outside of loop
1020	Can't have 'continue' outside of loop
1023	Expected hexadecimal digit
1024	Expected 'while'
1025	Label redefined
1026	Label not found
1027	'default' can only appear in a 'switch' statement
1028	Expected identifier or string

1029	Expected '@end'
1030	Conditional compilation turned off
1031	Expected constant
1032	Expected '@'
1033	Expected 'catch'
1034	Expected 'var'
1035	'throw' must be followed by an expression on the same source line

Microsoft® JScript® isFinite Method

See Also Applies To

Description

Returns a Boolean value that indicates if a supplied number is finite.

Syntax

isFinite(number)

The *number* argument is a required numeric value.

Remarks

The **isFinite** method returns **true** if *number* is any value other than **NaN**, negative infinity, or positive infinity. In those three cases, it returns **false**.

Microsoft® JScript® search Method

See Also Applies To

Description

Returns the position of the first substring match in a regular expression search.

Syntax

stringObj.search(rgexp)

The **search** method syntax has these parts:

Part	Description
stringObj	Required. The String object or literal to search.
uraexn I	Required. A Regular Expression object containing the pattern to search for.

Remarks

The **search** method indicates if a match is present or not. If a match is found, the **search** method returns an integer value that indicates the offset from the beginning of the string where the match occurred. If no match is found, it returns -1. To get further information, use the **match** method.

The following example illustrates the use of the **search** method:

```
function SearchDemo()
{
  var r, re;
  var s = "The quick brown fox jumped over the re = /fox/i;
  r = s.search(re);
  return(r);
}
```

Microsoft® JScript® delete Operator

See Also

Description

Deletes a property from an object, or removes an element from an array.

Syntax

delete expression

Where *expression* is a valid JScript expression that usually (but does not have to) result in a property name or array element.

Remarks

If the result of *expression* is an object, the property specified in *expression* exists, and the object will not allow it to be deleted, **false** is returned.

In all other cases, **true** is returned.

What Is JScript?

JScript Basics

Writing JScript Code

JScript Variables

JScript Data Types

JScript Operators

Controlling Program Flow

JScript Functions

JScript Objects

JScript Reserved Keywords

Advanced JScript

Recursion

Variable Scope

Copying, Passing, and Comparing Data

Using Arrays

Advanced Object Creation

Special Characters

Troubleshooting Your Scripts

Using JScript In Internet Explorer

Displaying Information in the Browser Using Message Boxes

ASCII Character Set

American Standard Code for Information Interchange (ASCII) 7-bit character set widely used to represent letters and symbols found on a standard U.S. keyboard. The ASCII character set is the same as the first 128 characters (0–127) in the ANSI character set.

Automation object

An object that is exposed to other applications or programming tools through Automation interfaces.

bitwise comparison

A bit-by-bit comparison of identically positioned bits in two numeric expressions.

Boolean expression

An expression that evaluates to either **true** or **false**. Non-Boolean expressions are converted to Boolean values, when necessary, according to the following rules:

- All objects are considered true.
- Strings are considered false if and only if they are empty.
- **null** and **undefined** are considered false.
- Numbers are considered false if and only if they are zero.

character code

A number that represents a particular character in a set, such as the ASCII character set.

class

The formal definition of an object. The class acts as the template from which an instance of an object is created at run time. The class defines the properties of the object and the methods used to control the object's behavior.

comment

Text added to code by a programmer that explains how the code works. In JScript, a comment line generally starts with //. Use the /* and */ delimiters to create a multiline comment.

comparison operator

A character or symbol indicating a relationship between two or more values or expressions. These operators include less than (<), less than or equal to (<=), greater than (>), greater than or equal to (>=), not equal (!=), and equal (==).

compound statement

A sequence of statements enclosed in braces ({}). Can be used to perform multiple tasks any time a single statement is expected.

constructor

A JScript function that has two special features:

- It is invoked by the **new** operator.
- It is passed the address of a newly created object through the **this** keyword.

Use constructors to initialize new objects.

expression

A combination of keywords, operators, variables, and literals that yield a string, number, or object. An expression can perform a calculation, manipulate characters, call a function, or test data.

intrinsic object

An object that is part of the standard JScript language. These objects are available to all scripts. The intrinsic objects in JScript are **Array**, **Boolean**, **Date**, **Function**, **Global**, **Math**, **Number**, **Object**, **RegExp**, **Regular Expression**, and **String**.

local time

The time on a computer, either a client or server, from where a script is executed.

locale

The set of information that corresponds to a given language and country. A locale affects the language of predefined programming terms and locale-specific settings. There are two contexts where locale information is

important:

- The code locale affects the language of terms such as keywords and defines locale-specific settings such as the decimal and list separators, date formats, and character sorting order.
- The system locale affects the way locale-aware functionality behaves, for example, when you display numbers or convert strings to dates.
 You set the system locale using the Control Panel utilities provided by the operating system.

null

A value indicating that a variable contains no valid data. **null** is the result of:

- An explicit assignment of **null** to a variable.
- Any operation between expressions that contain **null**.

numeric expression

Any expression that can be evaluated as a number. Elements of the expression can include any combination of keywords, variables, literals, and operators that result in a number. In certain circumstances, strings are also converted to numbers if possible.

primitive

A data type that is part of the JScript language and manipulated by value. The data types in JScript considered to be primitive are number, Boolean, string, and function. Objects and arrays are not primitive data types.

property

A named attribute of an object. Properties define object characteristics such as size, color, and screen location, or the state of an object, such as enabled or disabled.

run-time error

An error that occurs when code is running. A run-time error results when a statement attempts an invalid operation.

scope

Defines the visibility of a variable, procedure, or object. Variables declared in functions are visible only within the function and lose their value between calls.

string comparison

A comparison of two sequences of characters. Unless specified in the function making the comparison, all string comparisons are binary. In English, binary comparisons are case-sensitive; text comparisons are not.

string expression

Any expression that evaluates to a sequence of continuguous characters. Elements of a string expression can include a function that returns a string, a string literal, a **String** object, or a string variable.

undefined

A special value given to variables after they are created and before a value

has been assigned to them.

Universal Coordinated Time (UTC)

Universal Coordinated Time, which refers to the time as set by the World Time Standard. Previously referred to as Greenwich Mean time or GMT.

user-defined object

An object is one that is created by a user in source code.

variable

A location used for storing and manipulating values by name. As JScript is loosely typed, a single variable can hold different types of data over the course of a script.

wrapper

An object that is created to provide an object-style interface to some other type of data. The **Number** and **Boolean** objects are examples of wrapper objects.

Microsoft JScript provides nine intrinsic (or "built-in") objects. They are the **Array**, **Boolean**, **Date**, b>Function, **Global**, **Math**, **Number**, **Object**, and **String** objects. Each of the intrinsic objects has associated methods and properties that are described in detail in the <u>language reference</u>. Certain of the objects are also described here.

Array Object

In JScript, objects are handled as arrays and arrays are handled as objects. The subscripts of an array, which are entirely equivalent to the properties of an object, can be referred to by number (or by name, if you assign names to them). To create a new array, use the **new** operator and the **Array()** constructor, as in the following example.

```
var theMonths = new Array(12) {
theMonths[0] = "Jan";
theMonths[1] = "Feb";
theMonths[2] = "Mar";
theMonths[3] = "Apr";
theMonths[4] = "May";
theMonths[5] = "Jun";
theMonths[6] = "Jul";
theMonths[7] = "Aug";
theMonths[8] = "Sep";
```

```
theMonths[9] = "Oct";
theMonths[10] = "Nov";
theMonths[11] = "Dec";
}
```

When you create an array by using the **Array** keyword, JScript includes in the array a write-only <u>length</u> property, which records the number of entries in the array. If you do not specify a number, the length is set to 0, and the array has no entries. If you specify a number, the length is set to that number. If you specify more than one parameter, the parameters are used as entries in the array, and the number of parameters is assigned to the length property, as in the following example, which is equivalent to the preceding one.

```
var theMonths = new Array("Jan", "Feb", "Mar", "Ap
"Jul", "Aug", "Sep", "Oct", "Nov", "Dec");
```

JScript automatically changes the value of **length** if you add elements to an array that you created with the **Array** keyword.

String Object

In JScript, strings are objects. This means that any time you declare a string variable or use a string literal, what you're actually doing is creating a new string object. The **String** object has certain built-in methods, which you can use with your strings. One of these is the **substring** method, which returns part of the string. It takes two numbers as its arguments.

```
aString = "0123456789";
var aChunk = aString.substring(4, 7); // Sets aChunk
```

var aNotherChunk = aString.substring(7, 4); // Sets al

// Using the preceding Array creation example:
firstLetter = theMonths [5].substring(0,1); // Sets the

Another property of the **String** object is the **length** property. This property contains the number of characters in the string, which is 0 for an empty string. This a numeric value, and can be used directly in calculations.

var howLong = "Hello World".length // Sets the howI

Math Object

The Math object has a number of properties and methods, all predefined. The properties are specific numbers. One of these is the value of pi (approximately 3.14159...). This is the Math.PI property, shown in the following example.

// A radius variable is declared and assigned a numeric var circleArea = Math.PI * radius * radius; // Note ca

One of the built-in methods of the **Math** object is the exponentiation method, or **pow**, which raises a number to a specified power. The following example makes use of both pi and exponentiation.

// This formula calculates the volume of a sphere with volume = (4/3)*(Math.PI*Math.pow(radius,3));

Date Object

Use the **Date** object to capture today's date, and to calculate differences between dates. It has a number of properties and methods, all predefined. In general, the **Date** object provides the day of the week; the month, day, and year; and the time in hours, minutes, and seconds. This information is based on the number of milliseconds since January 1, 1970, 00:00:00.000 GMT. GMT stands for "Greenwich Mean Time"; the preferred term is UTC, or "Universal Coordinated Time," which refers to signals issued by the World Time Standard.

Note As far as JScript is concerned, time begins at midnight on January 1, 1970; you cannot ask JScript to create a **Date** object that represents an earlier time than that. If you need to deal with earlier times you must write your own code to do so, a formidable task.

To create a new **Date** object you use the <u>new</u> operator. The following example calculates, for the current year, the number of days that have passed and the number of days that are left.

```
/*
This example uses the array of month names defined process. The first statement assigns today's date, in "Day Mont format, to the thisIsToday variable.
*/
var thisIsToday = new Date();

var toDay = new Date(); // Capture today's date.

// Extract the year, the month, and the day.
```

```
var thisYear = toDay.getYear() + 1900;
var thisMonth = theMonths[toDay.getMonth()];
var thisDay = thisMonth + " " + toDay.getDate() + ",'
// Determine the # of days since the start.
thisDay = Math.round(Date.parse(thisDay)/8.64e7);
// Do the same for the beginning of the year.
var firstDay = "Jan 1, " + thisYear;
firstDay = Math.floor(Date.parse(firstDay)/8.64e7);
// Do it again for the end of the year, in case it's a leap
var lastDay = "Dec 31, " + thisYear;
lastDay = Math.floor(Date.parse(lastDay)/8.64e7);
// Compute the number of days in the year.
var daysInYear = (lastDay - firstDay) + 1;
// Determine how many days have elapsed, and how n
var daysElapsed = thisDay - firstDay;
var daysLeft = daysInYear - daysElapsed;
// Set up comments for most of the year.
var comment1 = daysElapsed+ " days have elapsed in
var comment2 = "That means there are " + daysLeft +
// Cover the special cases: beginning & end of year, ar
```

```
if (daysElapsed == 0) {
comment1 = "It's January first, " + thisYear + ".";
if (daysElapsed == 1) {
comment1 = "Only one day gone so far.";
if(daysElapsed == daysInYear) {
comment1 = thisYear + " is just about over.";
if (daysLeft == 0) {
comment2 = "Best wishes for the New Year!";
if (daysLeft == 1) {
comment2 = "There's only one day left in " + this Year
if (daysLeft == daysInYear) {
comment2 = "Happy New Year!";
```

Number Object

In addition to the special numeric properties (**PI**, for example) that are available in the **Math** object, several other properties are available in Microsoft JScript through the **Number** object.

Property	Description
	Largest possible number, about

MAX_VALUE	1.79E+308; can be positive or negative. (Value varies slightly from system to system.)
MIN_VALUE	Smallest possible number, about 2.22E-308; can be positive or negative. (Value varies slightly from system to system.)
NaN	Special nonnumeric value, "not a number."
	Any positive value larger than Number.MAX_VALUE is automatically converted to this value; represented as "Inf".
NEGATIVE_INFINITY	Any negative value larger than - Number.MAX_VALUE is automatically converted to this value; represented as "-Inf".

Number.NaN is a special property that is defined as "not a number." Division by zero, for example, returns **NaN**. An attempt to parse a string that cannot be parsed as a number also returns **Number.NaN**. **NaN** compares unequal to any number and also to itself. To test for a **NaN** result, do not compare against **Number.NaN**; use the **isNaN()** function instead.

Microsoft® JScript® Creating Your Own Objects

To create instances of an object, you must first define it by giving it properties and, if appropriate, methods. For instance, the following example defines a pasta object. Notice the keyword this, which you use to refer to the current object.

```
function pasta( grain, grain2, width, shape, shapenum, {
    this.length = 7; // Number of properties in the objec
    this.grain = grain; // What grain is it made of? (strin
    this.grain2 = grain2; // Any other flour in it? (string
    this.width = width; // How wide is it? (number)
    this.shape = shape; // What is the cross-section? (str
    this.shapenum = shapenum; // Is it one of the registe
    this.extent = extent; // How long is it? (number)
    this.egg = egg; // Does it have egg yolk as a binder?
}
```

Once you define an object, you create instances of it with the **new** operator.

```
var spaghetti = new pasta("wheat", "", 0.2, "circle", 9,
var linguine = new pasta("wheat", "", 0.3, "oval", 17, 1
```

You can add properties to one instance of an object, to change that instance, but those properties do not become part of the definition of the object, and do not show up in other instances unless you specifically add them. If you want the extra properties to show up in all instances of the object, you must add them to the object definition.

```
// Additional properties for spaghetti.
spaghetti.color = "pale straw";
spaghetti.drycook = 7;
spaghetti.freshcook = 0.5;

var chowFun = new pasta("rice", "", 3, "flat", , 12, fals/*
Neither the chowFun object, the linguine object, nor tl has the three extra properties given to the spaghetti ob */
```

Including Methods in the Definition

It is possible to include methods in the definition of an object. The following example builds an object that consists of an array of strings, and a method. The method adds a string to the array, increasing its size in order to do so. Notice that this makes each instance of the object indefinitely extensible.

```
function addItem(newItem) // Define a function to ex
{
   this.length += 1; // Increment the length of the arra;
   this[(this.length-1)] = newItem; // Add the new iter
}
```

```
function shoppingList(firstItem) // Define a "shopping
{
    this.length = 2; // Number of properties in the object this.addItem = addItem; // Include the addItem funthis[(this.length-1)] = firstItem; // The first item is 1
}
```

var myList = new shoppingList("Milk");
myList.addItem("Eggs"); // Use the method to add Eg
myList.addItem("Breadfruit"); // Breadfruit becomes

At this point, the contents of the array are as follows:

- myList[length] is 4
- myList[addItem] is the addItem function
- myList[1] is Milk
- myList[2] is Eggs
- myList[3] is Breadfruit

Note that the indexing is not exactly as you might expect it to be if it were handled in a strictly numeric way. If you execute a **for...in** loop on this array, the loop iterates in the order given here, and the loop variable has the initial value "length" rather than 0.

${\tt Microsoft @ JScript @ } Using \ Message$

Boxes

Using alert, prompt, and confirm

Use alert, confirm, and prompt message boxes to obtain input from your user. The boxes are methods of the interface **window** object. Because the **window** object is at the top of the object hierarchy, you do not actually have to use the full name (for example, "window.alert()") of any of these message boxes, but it is a good idea to do so, because it helps you remember to which object they belong.

Alert Message Box

The **alert** method has one argument, the string of text you want to display to the user. The string is not HTML. The message box provides an OK button so the user can close it and is modal, that is, the user must close the message box before continuing.

window.alert("Welcome! Press OK to continue.

Confirm Message Box

The confirm message box lets you ask the user a "yes-or-no" question, and gives the user the option of clicking either an OK button or a Cancel button. The **confirm** method returns either **true** or **false**. This message box is also modal: the user must respond to it (click a button), and thereby close it, before proceeding.

```
var truthBeTold = window.confirm("Click OK i
if (truthBeTold) {
  window.alert("Welcome to our Web page!");
} else window.alert("Bye for now!");
```

Prompt Message Box

The prompt message box provides a text field in which the user can type an answer in response to your prompt. This box has an OK button and a Cancel button. If you provide a second string argument, the prompt message box displays that second string in the text field, as the default response. Otherwise, the default text is "<undefined>".

Like the **alert()** and **confirm()** methods, **prompt** displays a modal message box. The user must close it before continuing.

var theResponse = window.prompt(

Reference

- Feature Information
- Alphabetic Keyword
- Errors
- Functions
- Methods
- Objects
- Operators
- Properties
- Statements

Welcome to the JScript Language Reference

These handy blocks of information will help you explore the many different parts of JScript.

You'll find *all* the parts of the JScript language listed alphabetically under the Alphabetic Keyword List. But if you want to examine just one category, say, objects, each language category has its own, more compact section.

How's it work? Click on one of the headings to the left to display a list of items contained in that category. From this list, select the topic that you want to view. Once you've opened that topic, you can easily link to other related sections.

So, go ahead and take a look! Study some statements, mull over the methods, or figure out a few functions. You'll see just how versatile the JScript language can be!

<u>JScript</u> (Non-ECMA)

List of non-**ECMA** Features features currently in JScript.

Microsoft Scripting List of scripting run-time features currently in JScript.

© 2000 Microsoft Corporation. All rights reserved.

Scripting Run-Time Reference Version 2

Microsoft® JScript® Add Method (Dictionary)

See Also

Applies To

Description

Adds a key and item pair to a **Dictionary** object.

Syntax

object.Add (key, item)

The **Add** method has the following parts:

Part	Description
object	Required. Always the name of a Dictionary object.
key	Required. The <i>key</i> associated with the <i>item</i> being added.
item	Required. The <i>item</i> associated with the <i>key</i> being added.

Remarks

An error occurs if the *key* already exists.

The following example illustrates the use of the **Add** method:

```
var d;
d = new ActiveXObject("Scripting.Dictionary"
d.Add("a", "Athens");
```

```
d.Add("b", "Belgrade");
d.Add("c", "Cairo");
```

Microsoft® JScript® Add Method (Folders)

See Also

Applies To

Description

Adds a new **Folder** to a **Folders** collection.

Syntax

object.Add (folderName)

The **Add** method has the following parts:

Part	Description	
IIODI <i>PC</i> T	Required. Always the name of a Folders collection.	
folderName	Required. The name of the new Folder being added.	

Remarks

The following example illustrates the use of the **Add** method to create a new folder:

```
function AddNewFolder(path,folderName)
{
  var fso, f, fc, nf;
  fso = new ActiveXObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSyste
```

```
f = fso.GetFolder(path);
fc = f.SubFolders;
if (folderName != "" )
   nf = fc.Add(folderName);
else
   nf = fc.Add("New Folder");
}
```

An error occurs if the *folderName* already exists.

Microsoft® JScript® AtEndOfLine Property

See Also

Applies To

Description

Returns **true** if the file pointer is positioned immediately before the end-of-line marker in a **TextStream** file; **false** if it is not. Read-only.

Syntax

object.AtEndOfLine

The *object* is always the name of a **TextStream** object.

Remarks

The **AtEndOfLine** property applies only to **TextStream** files that are open for reading; otherwise, an error occurs.

The following code illustrates the use of the **AtEndOfLine** property:

```
function GetALine(filespec)
{
  var fso, a, s, ForReading;
  ForReading = 1, s = "";
  fso = new ActiveXObject("Scripting.FileSystems)
```

```
a = fso.OpenTextFile(filespec, ForReading, fa
while (!a.AtEndOfLine)
{
   s += a.Read(1);
}
a.Close();
return(s);
}
```

Microsoft® JScript® AtEndOfStream

Property

See Also

Applies To

Description

Returns **true** if the file pointer is at the end of a **TextStream** file; **false** if it is not. Read-only.

Syntax

object.AtEndOfStream

The *object* is always the name of a **TextStream** object.

Remarks

The **AtEndOfStream** property applies only to **TextStream** files that are open for reading, otherwise, an error occurs.

The following code illustrates the use of the **AtEndOfStream** property:

```
function GetALine(filespec)
{
  var fso, f, s, ForReading;
  ForReading = 1, s = "";
```

```
fso = new ActiveXObject("Scripti
f = fso.OpenTextFile(filespec, For
while (!f.AtEndOfStream)
   s += f.ReadLine();
f.Close();
return(s);
```

Microsoft® JScript® Attributes

Property

See Also Applies To

Description

Sets or returns the attributes of files or folders. Read/write or read-only, depending on the attribute.

Syntax

object.Attributes [= newattributes]

The **Attributes** property has these parts:

Part	Description	
IIOD1eCT	Required. Always the name of a File or Folder object.	
newattributes	Optional. If provided, <i>newattributes</i> is the new value for the attributes of the specified <i>object</i> .	

Settings

The *newattributes* argument can have any of the following values or any logical combination of the following values:

Constant	Value	Description
Normal	0	Normal file. No attributes are set.
ReadOnly	1	Read-only file. Attribute is read/write.

Hidden	2	Hidden file. Attribute is read/write.
System	4	System file. Attribute is read/write.
Volume	8	Disk drive volume label. Attribute is readonly.
Directory	16	Folder or directory. Attribute is read-only.
Archive	32	File has changed since last backup. Attribute is read/write.
Alias	64	Link or shortcut. Attribute is read-only.
Compressed	128	Compressed file. Attribute is read-only.

Remarks

The following code illustrates the use of the **Attributes** property with a file:

```
function ToggleArchiveBit(filespec)
{
  var fso, f, r, s;
  fso = new ActiveXObject("Scripting.FileSystemObjetf = fso.GetFile(filespec)
  if (f.attributes && 32)
  {
    f.attributes = f.attributes - 32;
    s = "Archive bit is cleared.";
  }
  else
  {
    f.attributes = f.attributes + 32;
    s = "Archive bit is set.";
}
```

```
}
return(s);
}
```

Property

See Also

Applies To

Description

Returns the amount of space available to a user on the specified drive or network share.

Syntax

object.AvailableSpace

The *object* is always a **Drive** object.

Remarks

The value returned by the **AvailableSpace** property is typically the same as that returned by the **FreeSpace** property. Differences may occur between the two for computer systems that support quotas.

The following code illustrates the use of the **AvailableSpace** property:

```
function ShowAvailableSpace(drvPath)
{
  var fso, d, s;
  fso = new ActiveXObject("Scripting.FileSystems)
```

```
d = fso.GetDrive(fso.GetDriveName(drvPath)
s = "Drive " + drvPath.toUpperCase() + " - ";
s += d.VolumeName + "<br>";
s += "Available Space: " + d.AvailableSpace/1
return(s);
}
```

Microsoft® JScript® BuildPath Method

See Also

Applies To

Description

Appends a name to an existing path.

Syntax

object.BuildPath(path, name)

The **BuildPath** method syntax has these parts:

Part	Description
object	Required. Always the name of a FileSystemObject .
path	Required. Existing path to which <i>name</i> is appended. Path can be absolute or relative and need not specify an existing folder.
name	Required. Name being appended to the existing <i>path</i> .

Remarks

The **BuildPath** method inserts an additional path separator between the existing path and the name, only if necessary.

The following example illustrates use of the **BuildPath** method:

function GetBuildPath(path)

```
var fso, newpath;
fso = new ActiveXObject("Scripting.FileSyste
newpath = fso.BuildPath(path, "New Folder")
return(newpath);
}
```

See Also

Applies To

Description

Closes an open **TextStream** file.

Syntax

object.Close();

The *object* is always the name of a **TextStream** object.

Remarks

The following example illustrates use of the **Close** method:

var fso; fso = new ActiveXObject("Scriptin a = fso.CreateTextFile("c:\\testfile.t a.WriteLine("This is a test."); a.Close(); See Also

Applies To

Description

Read-only property that returns the column number of the current character position in a **TextStream** file.

Syntax

object.Column

The *object* is always the name of a **TextStream** object.

Remarks

After a newline character has been written, but before any other character is written, **Column** is equal to 1.

The following examples illustrates the use of the **Column** property:

```
function GetColumn()
{
  var fso, f, m;
  var ForReading = 1, ForWriting =
  fso = new ActiveXObject("Scripti")
```

```
f = fso.OpenTextFile("c:\\testfile.t
f.Write("Hello World!");
f.Close();
f = fso.OpenTextFile("c:\\testfile.t
m = f.ReadLine();
return(f.Column);
}
```

See Also Applies To

Description

Copies a specified file or folder from one location to another.

Syntax

object.Copy(destination[, overwrite]);

The **Copy** method syntax has these parts:

Part	Description
μωρ <i>ιρετ</i> ι	Required. Always the name of a File or Folder object.
destination	Required. Destination where the file or folder is to be copied. Wildcard characters are not allowed.
overwrite	Optional. Boolean value that is True (default) if existing files or folders are to be overwritten; False if they are not.

Remarks

The results of the **Copy** method on a **File** or **Folder** are identical to operations performed using **FileSystemObject.CopyFile** or **FileSystemObject.CopyFolder** where the file or folder referred to by *object* is passed as an argument. You should note, however, that the alternative methods are capable of copying multiple files

or folders.

The following example illustrates the use of the **Copy** method:

```
var fso, f;
fso = new ActiveXObject("Scripting.FileSyster
f = fso.CreateTextFile("c:\\testfile.txt", true);
f.WriteLine("This is a test.");
f.Close();
f = fso.GetFile("c:\\testfile.txt");
f.Copy("c:\\windows\\desktop\\test2.txt");
```

See Also Applies To

Description

Copies one or more files from one location to another.

Syntax

object.CopyFile (source, destination[, overwrite])

The **CopyFile** method syntax has these parts:

Part	Description		
object	Required. The <i>object</i> is always the name of a FileSystemObject .		
source	Required. Character string file specification, which can include wildcard characters, for one or more files to be copied.		
destination	Required. Character string destination where the file or files from <i>source</i> are to be copied. Wildcard characters are not allowed.		
overwrite	Optional. Boolean value that indicates if existing files are to be overwritten. If true , files are overwritten; if false , they are not. The default is true . Note that CopyFile will fail if <i>destination</i> has the read-only attribute set, regardless of the value of <i>overwrite</i> .		

Remarks

Wildcard characters can only be used in the last path component of the *source* argument. For example, you can use:

fso = new ActiveXObject("Scripting.FileSystemObjectfso.CopyFile ("c:\mydocuments\\letters*.doc", "c:\\t

But you can't use:

fso = new ActiveXObject("Scripting.FileSystemObjectfso.CopyFile("c:\\mydocuments*\\R1???97.xls", "c:

If *source* contains wildcard characters or *destination* ends with a path separator (\), it is assumed that *destination* is an existing folder in which to copy matching files. Otherwise, *destination* is assumed to be the name of a file to create. In either case, three things can happen when an individual file is copied.

- If *destination* does not exist, *source* gets copied. This is the usual case.
- If *destination* is an existing file, an error occurs if *overwrite* is **false**. Otherwise, an attempt is made to copy *source* over the existing file.
- If *destination* is a directory, an error occurs.

An error also occurs if a *source* using wildcard characters doesn't match any files. The **CopyFile** method stops on the first error it encounters. No attempt is made to roll back or undo any changes made before an error occurs.

Microsoft® JScript® CopyFolder Method

See Also

Applies To

Description

Recursively copies a folder from one location to another.

Syntax

object.CopyFolder (source, destination[, overwrite]);

The **CopyFolder** method syntax has these parts:

Part	Description		
object	Required. Always the name of a FileSystemObject.		
source	Required. Character string folder specification, which can include wildcard characters, for one or more folders to be copied.		
	Required. Character string destination where the folder and subfolders from <i>source</i> are to be copied. Wildcard characters are not allowed.		
overwrite	Optional. Boolean value that indicates if existing folders are to be overwritten. If true , files are overwritten; if false , they are not. The default is true .		

Remarks

Wildcard characters can only be used in the last path component

of the *source* argument. For example, you can use:

fso = new ActiveXObject("Scripting.FileSystemObjectso.CopyFolder ("c:\\mydocuments\\letters*", "c:\\tell

But you can't use:

fso = new ActiveXObject("Scripting.FileSystemObjectfso.CopyFolder("c:\\mydocuments**", "c:\\tempfo

If *source* contains wildcard characters or *destination* ends with a path separator (\), it is assumed that *destination* is an existing folder in which to copy matching folders and subfolders. Otherwise, *destination* is assumed to be the name of a folder to create. In either case, four things can happen when an individual folder is copied.

- If *destination* does not exist, the *source* folder and all its contents gets copied. This is the usual case.
- If *destination* is an existing file, an error occurs.
- If *destination* is a directory, an attempt is made to copy the folder and all its contents. If a file contained in *source* already exists in *destination*, an error occurs if *overwrite* is **false**. Otherwise, it will attempt to copy the file over the existing file.
- If *destination* is a read-only directory, an error occurs if an attempt is made to copy an existing read-only file into that directory and *overwrite* is **false**.

An error also occurs if a *source* using wildcard characters doesn't match any folders.

The **CopyFolder** method stops on the first error it encounters. No attempt is made to roll back any changes made before an error occurs.

See Also

Applies To

Description

Returns the number of items in a collection or **Dictionary** object. Read-only.

Syntax

object.Count

The *object* is always the name of one of the items in the Applies To list.

Remarks

The following code illustrates use of the **Count** property:

```
for (i = 0; i < d.Count; i++) //Iterate the dictionary.
{
   s += a.getItem(i) + " - " + d(a.getItem(i)) + " < br > ";
}
return(s); // Return the results.
}
```

Microsoft® JScript® CreateFolder Method

See Also

Applies To

Description

Creates a folder.

Syntax

object.CreateFolder(foldername)

The **CreateFolder** method has these parts:

Part	Description	
llani <i>ect</i> I	Required. Always the name of a FileSystemObject .	
foldername	Required. <u>String expression</u> that identifies the folder to create.	

Remarks

An error occurs if the specified folder already exists.

The following code illustrates how to use the **CreateFolder** method to create a folder:

var fso = new ActiveXObject("Scripting.FileSy
var a = fso.CreateFolder("c:\\new folder");

Microsoft® JScript® CreateTextFile Method

See Also

Applies To

Description

Creates a specified file name and returns a **TextStream** object that can be used to read from or write to the file.

Syntax

object.CreateTextFile(filename[, overwrite[, unicode]])

The **CreateTextFile** method has these parts:

Part	Description		
ahiaat	Required. Always the name of a FileSystemObject		
object	or Folder object.		
filename	Required. String expression that identifies the file to		
jiienume	create.		
	Optional. Boolean value that indicates whether you		
II ALLANIA LINITA	can overwrite an existing file. The value is true if the		
over write	file can be overwritten, false if it can't be overwritten.		
	If omitted, existing files are not overwritten.		
	Optional. Boolean value that indicates whether the		
	file is created as a Unicode or ASCII file. The value		
	is true if the file is created as a Unicode file, false if		
	it's created as an ASCII file. If omitted, an ASCII file		
	is assumed.		

Remarks

The following code illustrates how to use the **CreateTextFile** method to create and open a text file:

```
var fso = new ActiveXObject("Scripting.FileSystemO
var a = fso.CreateTextFile("c:\\testfile.txt", true);
a.WriteLine("This is a test.");
a.Close();
```

If the *overwrite* argument is **false**, or is not provided, for a *filename* that already exists, an error occurs.

Microsoft® JScript® DateCreated Property

See Also

Applies To

Description

Returns the date and time that the specified file or folder was created. Read-only.

Syntax

object. Date Created

The *object* is always a **File** or **Folder** object.

Remarks

The following code illustrates the use of the **DateCreated** property with a file:

```
function ShowFileInfo(filespec)
{
  var fso, f, s;
  fso = new ActiveXObject("Scripting.FileSystemObjetf = fso.GetFile(filespec);
  s = "Created: " + f.DateCreated;
  return(s);
}
```

DateLastAccessed Property

See Also

Applies To

Description

Returns the date and time that the specified file or folder was last accessed. Read-only.

Syntax

object.DateLastAccessed

The *object* is always a **File** or **Folder** object.

Remarks

The following code illustrates the use of the **DateLastAccessed** property with a file:

```
s += "Last Accessed: " + f.DateLastAccessed + "<br/>s += "Last Modified: " + f.DateLastModified;
return(s);
}
```

Important This method depends on the underlying operating system for its behavior. If the operating system does not support providing time information, none will be returned.

DateLastModified Property

See Also

Applies To

Description

Returns the date and time that the specified file or folder was last modified. Read-only.

Syntax

object.DateLastModified

The *object* is always a **File** or **Folder** object.

Remarks

The following code illustrates the use of the **DateLastModified** property with a file:

```
s += "Last Accessed: " + f.DateLastAccessed + "<br/>s += "Last Modified: " + f.DateLastModified;
return(s);
}
```

Microsoft® JScript® Delete Method

See Also

Applies To

Description

Deletes a specified file or folder.

Syntax

object. Delete(force);

The **Delete** method syntax has these parts:

Part	Description
object	Required. Always the name of a File or Folder object.
force	Optional. Boolean value that is True if files or folders with the read-only attribute set are to be deleted; False (default) if they are not.

Remarks

An error occurs if the specified file or folder does not exist.

The results of the **Delete** method on a **File** or **Folder** are identical to operations performed using **FileSystemObject.DeleteFile** or **FileSystemObject.DeleteFolder**.

The **Delete** method does not distinguish between folders that have contents and those that do not. The specified folder is deleted regardless of whether or not it has contents.

The following example illustrates the use of the **Delete** method:

```
var fso, f;
fso = new ActiveXObject("Scripting.FileSyster
f = fso.CreateTextFile("c:\\testfile.txt", true);
f.WriteLine("This is a test.");
f.Close();
f = fso.GetFile("c:\\testfile.txt");
f.Delete();
```

Microsoft® JScript® DeleteFile Method

See Also

Applies To

Description

Deletes a specified file.

Syntax

object.DeleteFile (filespec[, force]);

The **DeleteFile** method syntax has these parts:

Part	Description	
object	Required. Always the name of a FileSystemObject .	
	Required. The name of the file to delete. The <i>filespec</i> can contain wildcard characters in the last path component.	
	Optional. Boolean value that is true if files with the read-only attribute set are to be deleted; false (default) if they are not.	

Remarks

An error occurs if no matching files are found. The **DeleteFile** method stops on the first error it encounters. No attempt is made to roll back or undo any changes that were made before an error occurred.

The following example illustrates the use of the **DeleteFile** method:

```
function DeleteFile(filespec)
{
  var fso;
  fso = new ActiveXObject("Scripting.FileSystefso.DeleteFile(filespec);
}
```

Microsoft® JScript® DeleteFolder Method

See Also

Applies To

Description

Deletes a specified folder and its contents.

Syntax

object.**DeleteFolder (** folderspec[, force] **)**;

The **DeleteFolder** method syntax has these parts:

Part	Description		
object	Required. Always the name of a FileSystemObject .		
	Required. The name of the folder to delete. The folderspec can contain wildcard characters in the last path component.		
force	Optional. Boolean value that is true if folders with the read-only attribute set are to be deleted; false (default) if they are not.		

Remarks

The **DeleteFolder** method does not distinguish between folders that have contents and those that do not. The specified folder is deleted regardless of whether or not it has contents.

An error occurs if no matching folders are found. The **DeleteFolder**

method stops on the first error it encounters. No attempt is made to roll back or undo any changes that were made before an error occurred.

The following example illustrates the use of the **DeleteFolder** method:

```
function DeleteFolder(folderspec)
{
  var fso;
  fso = new ActiveXObject("Scripting.FileSystefso.DeleteFolder(folderspec);
}
```

Microsoft® JScript® Dictionary Object

See Also

Methods

Properties

Description

Object that stores data key, item pairs.

Syntax

```
y = new ActiveXObject("Scripting.Dictionary")
```

Remarks

A **Dictionary** object is the equivalent of a PERL associative array. Items can be any form of data, and are stored in the array. Each item is associated with a unique key. The key is used to retrieve an individual item and is usually a integer or a string, but can be anything except an array.

The following code illustrates how to create a **Dictionary** object:

```
var y = new ActiveXObject("Scripting.Dictional
y.add ("a", "test");
if (y.Exists("a"))
  document.write("true");
```

• • •

Microsoft® JScript® Drive Object

See Also Properties Methods

Description

Provides access to the properties of a particular disk drive or network share.

Remarks

The following code illustrates the use of the **Drive** object to access drive properties:

```
function ShowFreeSpace(drvPath)
{
   var fso, d, s;
   fso = new ActiveXObject("Scripting.FileSystemObde d = fso.GetDrive(fso.GetDriveName(drvPath));
   s = "Drive " + drvPath + " - ";
   s += d.VolumeName + "<br/>
   s += "Free Space: " + d.FreeSpace/1024 + " Kbytes return(s);
}
```

See Also

Applies To

Description

Returns the drive letter of the drive on which the specified file or folder resides. Read-only.

Syntax

```
object.Drive
```

The *object* is always a **File** or **Folder** object.

Remarks

The following code illustrates the use of the **Drive** property:

```
function ShowFileAccessInfo(filespec)
{
  var fso, f, s;
  fso = new ActiveXObject("Scripting.FileSystemObjetf = fso.GetFile(filespec);
  s = f.Name + " on Drive " + f.Drive + "<br>
  s += "Created: " + f.DateCreated + "<br>
  s += "Last Accessed: " + f.DateLastAccessed + "<br/>
  s += "Last Modified: " + f.DateLastModified;
```

```
return(s);
}
```

Microsoft® JScript® **DriveExists**

Method

See Also Applies To

Description

Returns **True** if the specified drive exists; **False** if it does not.

Syntax

object.**DriveExists**(drivespec)

The **DriveExists** method syntax has these parts:

Part	Description	
object	Required. Always the name of a FileSystemObject .	
drivespec	Required. A drive letter or a complete path specification.	

Remarks

For drives with removable media, the **DriveExists** method returns **true** even if there are no media present. Use the **IsReady** property of the **Drive** object to determine if a drive is ready.

The following example illustrates the use of the **DriveExists** method:

```
function ReportDriveStatus(drv)
{
```

```
var fso, s = "";
fso = new ActiveXObject("Scripting.FileSyste
if (fso.DriveExists(drv))
    s += "Drive " + drv + " exists.";
else
    s += "Drive " + drv + " doesn't exist.";
return(s);
}
```

Microsoft® JScript® DriveLetter Property

See Also

Applies To

Description

Returns the drive letter of a physical local drive or a network share. Read-only.

Syntax

object.DriveLetter

The *object* is always a **Drive** object.

Remarks

The **DriveLetter** property returns a zero-length string ("") if the specified drive is not associated with a drive letter, for example, a network share that has not been mapped to a drive letter.

The following code illustrates the use of the **DriveLetter** property:

```
function ShowDriveLetter(drvPath)
{
  var fso, d, s;
  fso = new ActiveXObject("Scripting.FileSysted = fso.GetDrive(fso.GetDriveName(drvPath))
```

```
s = "Drive " + d.DriveLetter.toUpperCase() +
s += d.VolumeName + "<br>";
s += "Available Space: " + d.AvailableSpace/1
return(s);
}
```

Microsoft® JScript® Drives Collection

<u>See Also</u> <u>Properties</u> <u>Methods</u>

Description

Read-only collection of all available drives.

Remarks

Removable-media drives need not have media inserted for them to appear in the **Drives** collection.

The following example illustrates how to get the **Drives** collection using the **Drives** property and iterate the collection using the **Enumerator** object:

```
function ShowDriveList()
{
  var fso, s, n, e, x;
  fso = new ActiveXObject("Scripting.FileSyste
  e = new Enumerator(fso.Drives);
  s = "";
  for (; !e.atEnd(); e.moveNext())
  {
    x = e.item();
    s = s + x.DriveLetter;
```

```
s += " - ";
if (x.DriveType == 3)
    n = x.ShareName;
else if (x.IsReady)
    n = x.VolumeName;
else
    n = "[Drive not ready]";
    s += n + " < br > ";
}
return(s);
}
```

Microsoft® JScript® **Drives Property**

See Also

Applies To

Description

Returns a **Drives** collection consisting of all **Drive** objects available on the local machine.

Syntax

```
object.Drives
```

The *object* is always a **FileSystemObject**.

Remarks

Removable-media drives need not have media inserted for them to appear in the **Drives** collection.

You can iterate the members of the **Drives** collection using the **Enumerator** object and the **for** statement:

```
function ShowDriveList()
{
  var fso, s, n, e, x;
  fso = new ActiveXObject("Scripting.FileSyste
  e = new Enumerator(fso.Drives);
```

```
s = "";
for (; !e.atEnd(); e.moveNext())
 x = e.item();
 s = s + x.DriveLetter;
 s += " - ";
 if (x.DriveType == 3)
  n = x.ShareName;
 else if (x.IsReady)
  n = x.VolumeName;
 else
  n = "[Drive not ready]";
 s += n + " < br > ";
return(s);
```

Microsoft® JScript® DriveType Property

See Also

Applies To

Description

Returns a value indicating the type of a specified drive.

Syntax

```
object.DriveType
```

The *object* is always a **Drive** object.

Remarks

The following code illustrates the use of the **DriveType** property:

```
function ShowDriveType(drvpath)
{
  var fso, d, s, t;
  fso = new ActiveXObject("Scripting.FileSystemObjet d = fso.GetDrive(drvpath);
  switch (d.DriveType)
  {
    case 0: t = "Unknown"; break;
    case 1: t = "Removable"; break;
```

```
case 2: t = "Fixed"; break;
  case 3: t = "Network"; break;
  case 4: t = "CD-ROM"; break;
  case 5: t = "RAM Disk"; break;
}
s = "Drive " + d.DriveLetter + ": - " + t;
return(s);
}
```

Applies To

Description

Returns **true** if a specified key exists in the **Dictionary** object, **false** if it does not.

Syntax

```
object.Exists(key)
```

The **Exists** method syntax has these parts:

Part	Description	
object	Required. Always the name of a Dictionary object.	
$\Pi K \rho V = I$	Required. <i>Key</i> value being searched for in the Dictionary object.	

The following example illustrates the use of the \boldsymbol{Exists} method:

```
function keyExists(k)
{
  var fso, s = "";
  d = new ActiveXObject("Scripting.Dictionary
  d.Add("a", "Athens");
  d.Add("b", "Belgrade");
```

```
d.Add("c", "Cairo");
if (d.Exists(k))
  s += "Specified key exists.";
else
  s += "Specified key doesn't exist.";
return(s);
}
```

Microsoft® JScript® File Object

See Also Properties Methods

Description

Provides access to all the properties of a file.

Remarks

The following code illustrates how to obtain a **File** object and how to view one of its properties.

```
function ShowFileInfo(filespec)
{
  var fso, f, s;
  fso = new ActiveXObject("Scripting.FileSystemObjetf = fso.GetFile(filespec);
  s = f.DateCreated;
  return(s);
}
```



Method

See Also

Applies To

Description

Returns **True** if a specified file exists; **False** if it does not.

Syntax

object.FileExists(filespec)

The **FileExists** method syntax has these parts:

Part	Description	
object	Required. Always the name of a FileSystemObject .	
filespec	Required. The name of the file whose existence is to be determined. A complete path specification (either absolute or relative) must be provided if the file isn't expected to exist in the current folder.	

The following example illustrates the use of the **FileExists** method:

```
function ReportFileStatus(filespec)
{
  var fso, s = filespec;
  fso = new ActiveXObject("Scripting.FileSyste
  if (fso.FileExists(filespec))
```

```
s += " exists.";
else
s += " doesn't exist.";
return(s);
}
```

Microsoft® JScript® Files Collection

See Also Properties Methods

Description

Collection of all **File** objects within a folder.

Remarks

The following example illustrates how to get a **Files** collection and iterate the collection using the **Enumerator** object and the **for** statement:

```
function ShowFolderFileList(folderspec)
{
  var fso, f, f1, fc, s;
  fso = new ActiveXObject("Scripting.FileSystemObjetf = fso.GetFolder(folderspec);
  fc = new Enumerator(f.files);
  s = "";
  for (; !fc.atEnd(); fc.moveNext())
  {
    s += fc.item();
    s += "<br/>  return(s);
```

Applies To

Description

Returns a **Files** collection consisting of all **File** objects contained in the specified folder, including those with hidden and system file attributes set.

Syntax

```
object.Files
```

The *object* is always a **Folder** object.

Remarks

The following code illustrates the use of the **Files** property:

```
function ShowFolderFileList(folderspec)
{
  var fso, f, fc, s;
  fso = new ActiveXObject("Scripting.FileSystemObjetf = fso.GetFolder(folderspec);
  fc = new Enumerator(f.files);
  s = "";
  for (; !fc.atEnd(); fc.moveNext())
```

```
{
    s += fc.item();
    s += "<br>";
}
return(s);
}
```

Microsoft® JScript® FileSystemObject Object

Scripting Run-Time Reference
Version 2

See Also Methods Properties

Description

Provides access to a computer's file system.

Syntax

```
y = new ActiveXObject("Scripting.FileSystemObject")
```

Remarks

The following code illustrates how the **FileSystemObject** is used to return a **TextStream** object that can be read from or written to:

```
var fso = new ActiveXObject("Scripting.FileSystemO
var a = fso.CreateTextFile("c:\\testfile.txt", true);
a.WriteLine("This is a test.");
a.Close();
```

In the example code, the **ActiveXObject** object is assigned to the **FileSystemObject** (fso). The **CreateTextFile** method then creates the file as a **TextStream** object (a), and the **WriteLine** method writes a line of text to the created text file. The **Close** method flushes the buffer and closes the file.

Microsoft® JScript® FileSystem Property

See Also

Applies To

Description

Returns the type of file system in use for the specified drive.

Syntax

```
object.FileSystem
```

The *object* is always a **Drive** object.

Remarks

Available return types include FAT, NTFS, and CDFS.

The following code illustrates the use of the **FileSystem** property:

```
function ShowFileSystemType(drvPath)
{
  var fso,d, s;
  fso = new ActiveXObject("Scripting.FileSysted = fso.GetDrive(drvPath);
  s = d.FileSystem;
  return(s);
```

Microsoft® JScript® Folder Object

<u>See Also</u> <u>Properties</u> <u>Methods</u>

Description

Provides access to all the properties of a folder.

Remarks

The following code illustrates how to obtain a **Folder** object and how to return one of its properties:

```
function ShowFolderInfo(folderspec)
{
  var fso, folder, s;
  fso = new ActiveXObject("Scripting.FileSystemObjetolder = fso.GetFolder(folderspec);
  s = folder.DateCreated;
  return(s);
}
```

Microsoft® JScript® Folders Collection

Properties

Methods

Description

See Also

Collection of all **Folder** objects contained within a **Folder** object.

Remarks

The following example illustrates how to get a **Folders** collection and how to iterate the collection using the **Enumerator** object and the **for** statement:

```
function ShowFolderList(folderspec)
{
  var fso, f, fc, s;
  fso = new ActiveXObject("Scripting.FileSystemObjetf = fso.GetFolder(folderspec);
  fc = new Enumerator(f.SubFolders);
  s = "";
  for (; !fc.atEnd(); fc.moveNext())
  {
    s += fc.item();
    s += "<br/>);
}
```

```
return(s);
}
```

${\tt Microsoft @ JScript @ } Folder Exists$

Method

See Also Applies To

Description

Returns **True** if a specified folder exists; **False** if it does not.

Syntax

object.FolderExists(folderspec)

The **FolderExists** method syntax has these parts:

Part	Description	
object	Required. Always the name of a FileSystemObject.	
folderspec	Required. The name of the folder whose existence is to be determined. A complete path specification (either absolute or relative) must be provided if the folder isn't expected to exist in the current folder.	

The following example illustrates the use of the **FileExists** method:

```
function ReportFolderStatus(fldr)
{
  var fso, s = fldr;
  fso = new ActiveXObject("Scripting.FileSyste
  if (fso.FolderExists(fldr))
```

```
s += " exists.";
else
s += " doesn't exist.";
return(s);
}
```

Microsoft® JScript® FreeSpace Property

See Also

Applies To

Description

Returns the amount of free space available to a user on the specified drive or network share. Read-only.

Syntax

object.FreeSpace

The *object* is always a **Drive** object.

Remarks

The value returned by the **FreeSpace** property is typically the same as that returned by the **AvailableSpace** property. Differences may occur between the two for computer systems that support quotas.

The following code illustrates the use of the **FreeSpace** property:

```
function ShowFreeSpace(drvPath)
{
  var fso, d, s;
  fso = new ActiveXObject("Scripting.FileSystematics")
```

```
d = fso.GetDrive(fso.GetDriveName(drvPath)
s = "Drive " + drvPath.toUpperCase() + " - ";
s += d.VolumeName + "<br>";
s += "Free Space: " + d.FreeSpace/1024 + " K
return(s);
}
```

<u>Scripting Run-Time Reference</u> Version 3

GetAbsolutePathName Method

See Also Applies To

Description

Returns a complete and unambiguous path from a provided path specification.

Syntax

object. Get Absolute Path Name (path spec)

The **GetAbsolutePathName** method syntax has these parts:

Part	Description
object	Required. Always the name of a FileSystemObject .
pathspec	Required. Path specification to change to a complete and unambiguous path.
	and unambiguous path.

Remarks

A path is complete and unambiguous if it provides a complete reference from the root of the specified drive. A complete path can only end with a path separator character (\) if it specifies the root folder of a mapped drive.

Assuming the current directory is c:\mydocuments\reports, the following table illustrates the behavior of the **GetAbsolutePathName** method.

pathspec	Returned path
"c:"	"c:\mydocuments\reports"
"c:"	"c:\mydocuments"
"c:\\"	"c:\"
"c:*.*\\may97"	"c:\mydocuments\reports*.*\may97"
"region1"	"c:\mydocuments\reports\region1"
"c:\\\\mydocuments"	"c:\mydocuments"

The following example illustrates the use of the **GetAbsolutePathName** method:

```
function ShowAbsolutePath(path)
{
  var fso, s= "";
  fso = new ActiveXObject("Scripting.FileSyste
  s += fso.GetAbsolutePathName(path);
  return(s);
}
```

Microsoft® JScript® GetBaseName Method

See Also

Applies To

Description

Returns a string containing the base name of the last component, less any file extension, in a path.

Syntax

object.GetBaseName(path)

The **GetBaseName** method syntax has these parts:

Part	Description
object	Required. Always the name of a FileSystemObject .
	Required. The path specification for the component whose base name is to be returned.

Remarks

The **GetBaseName** method returns a zero-length string ("") if no component matches the *path* argument.

Note The **GetBaseName** method works only on the provided *path* string. It does not attempt to resolve the path, nor does it check for the existence of the specified path.

The following example illustrates the use of the **GetBaseName** method:

```
function ShowBaseName(filespec)
{
  var fso, s = "";
  fso = new ActiveXObject("Scripting.FileSystes s += fso.GetBaseName(filespec);
  return(s);
}
```

Applies To

Description

Returns a **Drive** object corresponding to the drive in a specified path.

Syntax

object. GetDrive (drivespec);

The **GetDrive** method syntax has these parts:

Part	Description
object	Required. Always the name of a FileSystemObject .
	Required. The <i>drivespec</i> argument can be a drive letter (c), a drive letter with a colon appended (c:), a drive letter with a colon and path separator appended (c:\), or any network share specification (\\computer2\share1).

Remarks

For network shares, a check is made to ensure that the share exists.

An error occurs if *drivespec* does not conform to one of the accepted forms or does not exist.

To call the **GetDrive** method on a normal path string, use the following sequence to get a string that is suitable for use as *drivespec*:

DriveSpec = GetDriveName(GetAbsolutePath)

The following example illustrates the use of the **GetDrive** method:

```
function ShowFreeSpace(drvPath)
{
  var fso, d, s ="";
  fso = new ActiveXObject("Scripting.FileSyste
  d = fso.GetDrive(fso.GetDriveName(drvPath)
  s = "Drive " + drvPath.toUpperCase() + " - ";
  s += d.VolumeName + "<br>
  rs += "Free Space: " + d.FreeSpace/1024 + " K
  return(s);
}
```

Microsoft® JScript® GetDriveName Method

See Also

Applies To

Description

Returns a string containing the name of the drive for a specified path.

Syntax

object.GetDriveName(path)

The **GetDriveName** method syntax has these parts:

Part	Description
object	Required. Always the name of a FileSystemObject .
	Required. The path specification for the component whose drive name is to be returned.

Remarks

The **GetDriveName** method returns a zero-length string ("") if the drive can't be determined.

Note The **GetDriveName** method works only on the provided *path* string. It does not attempt to resolve the path, nor does it check for the existence of the specified path.

The following example illustrates the use of the **GetDriveName** method:

```
function GetDriveLetter(path)
{
  var fso, s ="";
  fso = new ActiveXObject("Scripting.FileSyste
  s += fso.GetDrive(fso.GetDriveName(fso.Get
  return(s);
}
```

Scripting Run-Time Reference Version 3

GetExtensionName Method

See Also Applies To

Description

Returns a string containing the extension name for the last component in a path.

Syntax

object.GetExtensionName(path)

The **GetExtensionName** method syntax has these parts:

Part	Description	
object	Required. Always the name of a FileSystemObject .	
	Required. The path specification for the component whose extension name is to be returned.	

Remarks

For network drives, the root directory (\) is considered to be a component.

The **GetExtensionName** method returns a zero-length string ("") if no component matches the *path* argument.

The following example illustrates the use of the **GetExtensionName** method:

```
function ShowExtensionName(filespec)
{
  var fso, s = "";
  fso = new ActiveXObject("Scripting.FileSystes s += fso.GetExtensionName(filespec);
  return(s);
}
```

Applies To

Description

Returns a **File** object corresponding to the file in a specified path.

Syntax

```
object.GetFile(filespec)
```

The **GetFile** method syntax has these parts:

Part	Description	
object	Required. Always the name of a FileSystemObject .	
filespec	Required. The <i>filespec</i> is the path (absolute or relative) to a specific file.	

Remarks

An error occurs if the specified file does not exist.

The following example illustrates the use of the **GetFile** method:

```
function ShowFileAccessInfo(filespec)
{
  var fso, f, s;
  fso = new ActiveXObject("Scripting.FileSystematics")
```

```
f = fso.GetFile(filespec);
s = f.Path.toUpperCase() + "<br/>s += "Created: " + f.DateCreated + "<br/>s += "Last Accessed: " + f.DateLastAccessed
s += "Last Modified: " + f.DateLastModified
return(s);
}
```



Applies To

Description

Returns the last component of specified path that is not part of the drive specification.

Syntax

object.GetFileName(pathspec)

The **GetFileName** method syntax has these parts:

Part	Description	
object	Required. Always the name of a FileSystemObject .	
pathspec	Required. The path (absolute or relative) to a specific file.	

Remarks

The **GetFileName** method returns a zero-length string ("") if *pathspec* does not end with the named component.

Note The **GetFileName** method works only on the provided path string. It does not attempt to resolve the path, nor does it check for the existence of the specified path.

The following example illustrates the use of the **GetFileName** method:

```
function ShowFileName(filespec)
{
  var fso, s = "";
  fso = new ActiveXObject("Scripting.FileSystes s += fso.GetFileName(filespec);
  return(s);
}
```

Microsoft® JScript® GetFileVersion Method

See Also

Applies To

Description

Returns the version number of a specified file.

Syntax

object.GetFileVersion(pathspec)

The **GetFileVersion** method syntax has these parts:

Part	Description
object	Required. Always the name of a FileSystemObject .
pathspec	Required. The path (absolute or relative) to a specific file.

Remarks

The **GetFileVersion** method returns a zero-length string ("") if *pathspec* does not end with the named file or if the file does not contain version information.

Note The **GetFileVersion** method works only on the provided path string. It does not attempt to resolve the path, nor does it check for the existence of the specified path.

The following example illustrates the use of the **GetFileVersion** method:

```
function ShowFileVersion(pathspec)
{
  var fso, s = "";
  fso = new ActiveXObject("Scripting.FileSyste
  s += fso.GetFileVersion(pathspec);
  if (s == "")
    s = "No version information available.";
  return(s);
}
```

Microsoft® JScript® GetFolder

Method

See Also

Applies To

Description

Returns a **Folder** object corresponding to the folder in a specified path.

Syntax

object.GetFolder(folderspec)

The **GetFolder** method syntax has these parts:

Part	Description	
object	Required. Always the name of a FileSystemObject .	
folderspec	Required. The <i>folderspec</i> is the path (absolute or relative) to a specific folder.	

Remarks

An error occurs if the specified folder does not exist.

The following example illustrates the use of the **GetFolder** method:

```
function ShowFolderList(folderspec)
{
  var fso, f, fc, s;
```

```
fso = new ActiveXObject("Scripting.FileSyste
f = fso.GetFolder(folderspec);
fc = new Enumerator(f.SubFolders);
s = "";
for (; !fc.atEnd(); fc.moveNext())
{
    s += fc.item();
    s += "<br/>
    return(s);
}
```

Scripting Run-Time Reference Version 3

GetParentFolderName Method

See Also

Applies To

Description

Returns a string containing the name of the parent folder of the last component in a specified path.

Syntax

object.GetParentFolderName(path)

The **GetParentFolderName** method syntax has these parts:

Part	Description
object	Required. Always the name of a FileSystemObject .
	Required. The path specification for the component
	whose parent folder name is to be returned.

Remarks

The **GetParentFolderName** method returns a zero-length string ("") if there is no parent folder for the component specified in the *path* argument.

Note The **GetParentFolderName** method works only on the provided *path* string. It does not attempt to resolve the path, nor does it check for the existence of the specified path.

The following example illustrates the use of the **GetParentFolderName** method:

```
function ShowParentFolderName(filespec)
{
  var fso, s = "";
  fso = new ActiveXObject("Scripting.FileSyste
  s += fso.GetParentFolderName(filespec);
  return(s);
}
```

${\tt Microsoft \& JScript \& } \ \, \textbf{GetSpecialFolder} \quad \, {\tt Scripting \ Run-Time \ Reference} \, \,$

Version 3

Method

See Also **Applies To**

Description

Returns the special folder object specified.

Syntax

object. GetSpecialFolder(folderspec)

The **GetSpecialFolder** method syntax has these parts:

Part	Description	
object	Required. Always the name of a FileSystemObject .	
	Required. The name of the special folder to be returned. Can be any of the constants shown in the	
II' - I	Settings section.	

Settings

The *folderspec* argument can have any of the following values:

Constant	Value	Description
WindowsFolder	0	The Windows folder contains files installed by the Windows operating system.
SystemFolder		The System folder contains libraries, fonts, and device drivers.

TemporaryFolder	2	The Temp folder is used to store temporary files. Its path is found in the TMP environment variable.
-----------------	---	--

The following example illustrates the use of the **GetSpecialFolder** method:

```
var fso, tempfile;
fso = new ActiveXObject("Scripting.FileSyster

function CreateTempFile()
{
   var tfolder, tfile, tname, fname, TemporaryFol
   tfolder = fso.GetSpecialFolder(TemporaryFole
   tname = fso.GetTempName();
   tfile = tfolder.CreateTextFile(tname);
   return(tfile);
}

tempfile = CreateTempFile();
tempfile.writeline("Hello World");
tempfile.close();
```

Microsoft® JScript® GetTempName Method

See Also

Applies To

Description

Returns a randomly generated temporary file or folder name that is useful for performing operations that require a temporary file or folder.

Syntax

object.GetTempName ();

The optional *object* is always the name of a **FileSystemObject**.

Remarks

The **GetTempName** method does not create a file. It provides only a temporary file name that can be used with **CreateTextFile** to create a file.

The following example illustrates the use of the **GetTempName** method:

var fso, tempfile; fso = new ActiveXObject("Scripting.FileSyster

function CreateTempFile()

```
var tfolder, tfile, tname, fname, TemporaryFol
tfolder = fso.GetSpecialFolder(TemporaryFole
tname = fso.GetTempName();
tfile = tfolder.CreateTextFile(tname);
return(tfile);
}
tempfile = CreateTempFile();
tempfile.writeline("Hello World");
tempfile.close();
```

Microsoft® JScript® IsReady Property

See Also

Applies To

Description

Returns **True** if the specified drive is ready; **False** if it is not.

Syntax

object.**IsReady**

The *object* is always a **Drive** object.

Remarks

For removable-media drives and CD-ROM drives, **IsReady** returns **True** only when the appropriate media is inserted and ready for access.

The following code illustrates the use of the **IsReady** property:

```
function ShowDriveInfo(drvpath)
{
  var fso, d, s, t;
  fso = new ActiveXObject("Scripting.FileSysted = fso.GetDrive(drvpath)
  switch (d.DriveType)
```

```
case 0: t = "Unknown"; break;
  case 1: t = "Removable"; break;
  case 2: t = "Fixed"; break;
  case 3: t = "Network"; break;
  case 4: t = "CD-ROM"; break;
  case 5: t = "RAM Disk"; break;
 s = "Drive " + d.DriveLetter + ": - " + t;
 if (d.IsReady)
  s += "<br/>br>" + "Drive is Ready.";
 else
  s += "<br/>br>" + "Drive is not Ready.";
 return(s);
}
```

Microsoft® JScript® IsRootFolder

Property

See Also

Applies To

Description

Returns **True** if the specified folder is the root folder; **False** if it is not.

Syntax

object.IsRootFolder

The *object* is always a **Folder** object.

Remarks

The following code illustrates the use of the **IsRootFolder** property:

```
function DisplayLevelDepth(pathspec)
{
  var fso, f, n, s = "";
  fso = new ActiveXObject("Scripting.FileSystemObjetf = fso.GetFolder(pathspec);
  n = 0;
  if (f.IsRootFolder)
    s = "The specified folder is the root folder."
```

```
else
{
    do
    {
        f = f.ParentFolder;
        n++;
    }
    while (!f.IsRootFolder)
    s = "The specified folder is nested " + n + " levels d
    }
    return(s);
}
```

Applies To

Description

Returns an array containing all the items in a **Dictionary** object.

Syntax

```
object.Items()
```

The *object* is always the name of a **Dictionary** object.

Remarks

The following code illustrates use of the **Items** method:

```
{
    s += a[i] + "<br>";
}
return(s);  // Return the r
}
```

Applies To

Description

Returns an array containing all existing keys in a **Dictionary** object.

Syntax

```
object.Keys()
```

The *object* is always the name of a **Dictionary** object.

Remarks

The following code illustrates use of the **Keys** method:

Applies To

Description

Read-only property that returns the current line number in a **TextStream** file.

Syntax

object.Line

The *object* is always the name of a **TextStream** object.

Remarks

After a file is initially opened and before anything is written, **Line** is equal to 1.

The following example illustrates the use of the **Line** property:

```
function GetLine()
{
  var fso, f, r
  var ForReading = 1, ForWriting =
  fso = new ActiveXObject("Scripti")
```

```
f = fso.OpenTextFile("c:\\textfile.t
f.WriteLine("Hello world!");
f.WriteLine("JScript is fun");
f.Close();
f = fso.OpenTextFile("c:\\textfile.t
r = f.ReadAll();
return(f.Line);
}
```

Microsoft® JScript® Move Method

See Also Applies To

Description

Moves a specified file or folder from one location to another.

Syntax

object.Move(destination);

The **Move** method syntax has these parts:

Part	Description		
imnieci i	Required. Always the name of a File or Folder object.		
destination	Required. Destination where the file or folder is to be moved. Wildcard characters are not allowed.		

Remarks

The results of the **Move** method on a **File** or **Folder** are identical to operations performed using **FileSystemObject.MoveFile** or **FileSystemObject.MoveFolder**. You should note, however, that the alternative methods are capable of moving multiple files or folders.

Applies To

Description

Moves one or more files from one location to another.

Syntax

object. **MoveFile** (source, destination);

The **MoveFile** method syntax has these parts:

Part	Description		
object	Required. Always the name of a FileSystemObject .		
source	Required. The path to the file or files to be moved. The <i>source</i> argument string can contain wildcard characters in the last path component only.		
	Required. The path where the file or files are to be moved. The <i>destination</i> argument can't contain wildcard characters.		

Remarks

If *source* contains wildcards or *destination* ends with a path separator (\), it is assumed that *destination* specifies an existing folder in which to move the matching files. Otherwise, *destination* is assumed to be the name of a destination file to create. In either case, three things can happen when an individual

file is moved:

- If *destination* does not exist, the file gets moved. This is the usual case.
- If *destination* is an existing file, an error occurs.
- If *destination* is a directory, an error occurs.

An error also occurs if a wildcard character that is used in *source* doesn't match any files. The **MoveFile** method stops on the first error it encounters. No attempt is made to roll back any changes made before the error occurs.

Important This method allows moving files between volumes only if supported by the operating system.

The following example illustrates the use of the **MoveFile** method:

```
function MoveFile2Desktop(filespec)
{
  var fso;
  fso = new ActiveXObject("Scripting.FileSyste
  fso.MoveFile(filespec, "c:\\windows\\desktop\)
}
```

Microsoft® JScript® MoveFolder

Method

See Also Applies To

Description

Moves one or more folders from one location to another.

Syntax

object. MoveFolder (source, destination);

The **MoveFolder** method syntax has these parts:

Part	Description		
object	Required. Always the name of a FileSystemObject .		
source	Required. The path to the folder or folders to be moved. The <i>source</i> argument string can contain wildcard characters in the last path component only.		
	Required. The path where the folder or folders are to be moved. The <i>destination</i> argument can't contain wildcard characters.		

Remarks

If *source* contains wildcards or *destination* ends with a path separator (\), it is assumed that *destination* specifies an existing folder in which to move the matching files. Otherwise, *destination* is assumed to be the name of a destination folder to create. In either case, three things can happen when an individual

folder is moved:

- If *destination* does not exist, the folder gets moved. This is the usual case.
- If *destination* is an existing file, an error occurs.
- If *destination* is a directory, an error occurs.

An error also occurs if a wildcard character that is used in *source* doesn't match any folders. The **MoveFolder** method stops on the first error it encounters. No attempt is made to roll back any changes made before the error occurs.

Important This method allows moving folders between volumes only if supported by the operating system.

The following example illustrates the use of the **MoveFolder** method:

```
function MoveFldr2Desktop(fldrspec)
{
  var fso;
  fso = new ActiveXObject("Scripting.FileSyste
  fso.MoveFolder(fldrspec, "c:\\windows\\deskt
}
```

Applies To

Description

Sets or returns the name of a specified file or folder. Read/write.

Syntax

```
object.Name [= newname]
```

The **Name** property has these parts:

Part	Description		
NODIPCT I	Required. Always the name of a File or Folder object.		
newname	Optional. If provided, <i>newname</i> is the new name of the specified <i>object</i> .		

Remarks

The following code illustrates the use of the ${\bf Name}$ property:

```
function ShowFileAccessInfo(filespec)
{
  var fso, f, s;
  fso = new ActiveXObject("Scripting.FileSystemObfe f = fso.GetFile(filespec);
```

```
s = f.Name + " on Drive " + f.Drive + "<br>";
s += "Created: " + f.DateCreated + "<br>";
s += "Last Accessed: " + f.DateLastAccessed + "<b
s += "Last Modified: " + f.DateLastModified;
return(s);
}</pre>
```

Microsoft® JScript® OpenTextFile Method

See Also

Applies To

Description

Opens a specified file and returns a **TextStream** object that can be used to read from, write to, or append to the file.

Syntax

object.OpenTextFile(filename[, iomode[, create[, format]]])

The **OpenTextFile** method has these parts:

Part	Description		
object	Required. <i>Object</i> is always the name of a		
	FileSystemObject.		
filename	Required. String expression that identifies the file to		
	open.		
iomode	Optional. Can be one of three constants: ForReading ,		
lomoue	ForWriting, or ForAppending.		
	Optional. Boolean value that indicates whether a new		
create	file can be created if the specified <i>filename</i> doesn't		
Credie	exist. The value is True if a new file is created, False		
	if it isn't created. If omitted, a new file isn't created.		
format	Optional. One of three Tristate values used to indicate		
	the format of the opened file. If omitted, the file is		
	opened as ASCII.		

Settings

The *iomode* argument can have any of the following settings:

Constant	Value	Description
ForReading		Open a file for reading only. You can't write to this file.
ForWriting	2	Open a file for writing.
ForAppending		Open a file and write to the end of the file.

The *format* argument can have any of the following settings:

Value	Description
TristateTrue	Open the file as Unicode.
TristateFalse	Open the file as ASCII.
TristateUseDefault	Open the file using the system default.

Remarks

The following code illustrates the use of the **OpenTextFile** method to open a file for appending text:

```
var fs, a, ForAppending;
ForAppending = 8;
fs = new ActiveXObject("Scripting.FileSystemObject
a = fs.OpenTextFile("c:\\testfile.txt", ForAppending, f
...
a.Close();
```

Microsoft® JScript® ParentFolder Property

See Also

Applies To

Description

Returns the folder object for the parent of the specified file or folder. Read-only.

Syntax

object.ParentFolder

The *object* is always a **File** or **Folder** object.

Remarks

The following code illustrates the use of the **ParentFolder** property with a file:

```
function ShowFileAccessInfo(filespec)
{
  var fso, f, s;
  fso = new ActiveXObject("Scripting.FileSystemObfer f = fso.GetFile(filespec);
  s = f.Name + " in " + f.ParentFolder + "<br/>
  s += "Created: " + f.DateCreated + "<br/>
  s += "Last Accessed: " + f.DateLastAccessed + "<br/>
  s += "Last Accessed: " + f.DateLastAccessed + "<br/>
  s += "Last Accessed: " + f.DateLastAccessed + "<br/>
  s += "Last Accessed: " + f.DateLastAccessed + "<br/>
  s += "Last Accessed: " + f.DateLastAccessed + "<br/>
  s += "Last Accessed: " + f.DateLastAccessed + "<br/>
  s += "Last Accessed: " + f.DateLastAccessed + "<br/>
  s += "Last Accessed: " + f.DateLastAccessed + "<br/>
  s += "Last Accessed: " + f.DateLastAccessed + "<br/>
  s += "Last Accessed: " + f.DateLastAccessed + "<br/>
  s += "Last Accessed: " + f.DateLastAccessed + "<br/>
  s += "Last Accessed: " + f.DateLastAccessed + "<br/>
  s += "Last Accessed: " + f.DateLastAccessed + "<br/>
  s += "Last Accessed: " + f.DateLastAccessed + "<br/>
  s += "Last Accessed: " + f.DateLastAccessed + "<br/>
  s += "Last Accessed: " + f.DateLastAccessed: " +
```

```
s += "Last Modified: " + f.DateLastModified;
return(s);
}
```

Applies To

Description

Returns the path for a specified file, folder, or drive.

Syntax

object.Path

The *object* is always a **File**, **Folder**, or **Drive** object.

Remarks

For drive letters, the root drive is not included. For example, the path for the C drive is C:, not C:\.

The following code illustrates the use of the **Path** property with a **File** object:

```
function ShowFileAccessInfo(filespec)
{
  var fso, d, f, s;
  fso = new ActiveXObject("Scripting.FileSyste
  f = fso.GetFile(filespec);
  s = f.Path.toUpperCase() + "<br>";
```

```
s += "Created: " + f.DateCreated + "<br>";
s += "Last Accessed: " + f.DateLastAccessed
s += "Last Modified: " + f.DateLastModified
return(s);
}
```

Applies To

Description

Reads a specified number of characters from a **TextStream** file and returns the resulting string.

Syntax

object.Read(characters)

The **Read** method syntax has these parts:

Part	Description
IIAN <i>IOCT</i> I	Required. Always the name of a
	TextStream object.
characters	Required. Number of characters you
	Required. Number of characters you want to read from the file.

The following example illustrates how to use the **Read** method to read a six character header from a file and return the resulting string:

```
function GetHeader()
{
 var fso, f;
```

```
var ForReading = 1, ForWriting =
fso = new ActiveXObject("Scripti
f = fso.OpenTextFile("c:\\testfile.t
f.Write("Header");
f.Write("1234567890987654321")
f.Close();
f = fso.OpenTextFile("c:\\testfile.t
return(f.Read(6));
}
```

Microsoft® JScript® ReadLine

See Also

Method

Applies To

Description

Reads an entire line (up to, but not including, the newline character) from a **TextStream** file and returns the resulting string.

Syntax

```
object.ReadLine()
```

The *object* argument is always the name of a **TextStream** object.

Remarks

The following example illustrates the use of the **Line** property:

```
function GetLine()
{
  var fso, f, r;
  var ForReading = 1, ForWriting =
  fso = new ActiveXObject("Scripti
  f = fso.OpenTextFile("c:\\testfile.t
```

```
f.WriteLine("Hello world!");
f.WriteLine("JScript is fun");
f.Close();
f = fso.OpenTextFile("c:\\testfile.t
r = f.ReadLine();
return(r);
}
```

Applies To

Description

Removes a key, item pair from a **Dictionary** object.

Syntax

object.Remove(key)

The **Remove** method syntax has these parts:

Part	Description
object	Required. Always the name of a Dictionary object.
	Required. <i>Key</i> associated with the key, item pair you want to remove from the Dictionary object.

Remarks

An error occurs if the specified key, item pair does not exist.

The following code illustrates use of the **Remove** method:

```
d.Add ("b", "Belgrade");
d.Add ("c", "Cairo");
...
d.Remove("b"); // Remove
```

Microsoft® JScript® RootFolder Property

See Also

Applies To

Description

Returns a **Folder** object representing the root folder of a specified drive. Read-only.

Syntax

object.RootFolder

The *object* is always a **Drive** object.

Remarks

All the files and folders contained on the drive can be accessed using the returned **Folder** object.

The following example illustrates the use of the **RootFolder** property:

```
function GetRootFolder(drv)
{
  var fso,d;
  fso = new ActiveXObject("Scripting.FileSyste
  if (fso.DriveExists(drv))
   {
```

```
d = fso.GetDrive(drv);
  return(d.RootFolder);
}
else
  return(false);
}
```

Microsoft® JScript® SerialNumber Property

See Also

Applies To

Description

Returns the decimal serial number used to uniquely identify a disk volume.

Syntax

object.SerialNumber

The *object* is always a **Drive** object.

Remarks

You can use the **SerialNumber** property to ensure that the correct disk is inserted in a drive with removable media.

The following code illustrates the use of the **SerialNumber** property:

```
function ShowDriveInfo(drvpath)
{
  var fso, d, s, t;
  fso = new ActiveXObject("Scripting.FileSyste
  d = fso.GetDrive(fso.GetDriveName(fso.GetA
  switch (d.DriveType)
```

```
{
    case 0: t = "Unknown"; break;
    case 1: t = "Removable"; break;
    case 2: t = "Fixed"; break;
    case 3: t = "Network"; break;
    case 4: t = "CD-ROM"; break;
    case 5: t = "RAM Disk"; break;
}

s = "Drive " + d.DriveLetter + ": - " + t;
s += "<br/>" + "SN: " + d.SerialNumber;
return(s);
}
```

Microsoft® JScript® ShareName Property

See Also

Applies To

Description

Returns the network share name for a specified drive.

Syntax

object.ShareName

The *object* is always a **Drive** object.

Remarks

If *object* is not a network drive, the **ShareName** property returns a zero-length string ("").

The following code illustrates the use of the **ShareName** property:

```
function ShowDriveInfo(drvpath)
{
  var fso, d, s;
  fso = new ActiveXObject("Scripting.FileSysted = fso.GetDrive(fso.GetDriveName(fso.GetAls = "Drive" + d.DriveLetter + ": - " + d.Share")
```

```
return(s);
}
```

Microsoft® JScript® ShortName

Property

See Also

Applies To

Description

Returns the short name used by programs that require the earlier 8.3 naming convention.

Syntax

object.ShortName

The *object* is always a **File** or **Folder** object.

Remarks

The following code illustrates the use of the **ShortName** property with a **File** object:

```
function ShowShortName(filespec)
{
  var fso, f, s;
  fso = new ActiveXObject("Scripting.FileSystemObfer f = fso.GetFile(filespec);
  s = "The short name for " + "" + f.Name;
  s += "" + "<br>";
  s += "is: " + "" + f.ShortName + "";
```

```
return(s);
}
```

Microsoft® JScript® ShortPath

Property

See Also

Applies To

Description

Returns the short path used by programs that require the earlier 8.3 file naming convention.

Syntax

object.ShortPath

The *object* is always a **File** or **Folder** object.

Remarks

The following code illustrates the use of the **ShortName** property with a **File** object:

```
function ShowShortPath(filespec)
{
  var fso, f, s;
  fso = new ActiveXObject("Scripting.FileSystemObfer f = fso.GetFile(filespec);
  s = "The short path for " + "" + f.Name;
  s += "" + "<br>";
  s += "is: " + "" + f.ShortPath + "";
```

```
return(s);
}
```

Microsoft® JScript® Size Property

See Also

Applies To

Description

For files, returns the size, in bytes, of the specified file. For folders, returns the size, in bytes, of all files and subfolders contained in the folder.

Syntax

```
object.Size
```

The *object* is always a **File** or **Folder** object.

Remarks

The following code illustrates the use of the **Size** property with a **Folder** object:

```
function ShowFolderSize(filespec)
{
  var fso, f, s;
  fso = new ActiveXObject("Scripting.FileSystemObfer f = fso.GetFolder(filespec);
  s = f.Name + " uses " + f.size + " bytes.";
  return(s);
```

Applies To

Description

Skips a specified number of characters when reading a **TextStream** file.

Syntax

object.Skip(characters)

The **Skip** method syntax has these parts:

Part	Description
object	Required. Always the name of a
object	TextStream object.
charactors	Required. Number of characters to
characters	Required. Number of characters to skip when reading a file.

Remarks

Skipped characters are discarded.

The following example illustrates the use of the **Skip** method:

```
function SkipDemo()
{
```

```
var fso, f, r;
var ForReading = 1, ForWriting =
fso = new ActiveXObject("Scripti
f = fso.OpenTextFile("c:\\testfile.t
f.WriteLine("Hello world!");
f.WriteLine("JScript is fun");
f.Close();
f = fso.OpenTextFile("c:\\testfile.t
f.Skip(6);
r = f.ReadLine();
return(r);
```

Microsoft® JScript® SubFolders

Property

See Also

Applies To

Description

Returns a **Folders** collection consisting of all folders contained in a specified folder, including those with hidden and system file attributes set.

Syntax

object.SubFolders

The *object* is always a **Folder** object.

Remarks

The following code illustrates the use of the **SubFolders** property:

```
function ShowFolderList(folderspec)
{
  var fso, f, fc, s;
  fso = new ActiveXObject("Scripting.FileSystemObjetf = fso.GetFolder(folderspec);
  fc = new Enumerator(f.SubFolders);
  s = "";
```

```
for (;!fc.atEnd(); fc.moveNext())
    {
        s += fc.item();
        s += "<br>";
     }
    return(s);
}
```

Microsoft® JScript® **TextStream**

Object

See Also Methods Properties

Description

Facilitates sequential access to file.

Syntax

TextStream.{property | method()}

The *property* and *method* arguments can be any of the properties and methods associated with the **TextStream** object. Note that in actual usage, **TextStream** is replaced by a variable placeholder representing the **TextStream** object returned from the **FileSystemObject**.

Remarks

In the following code, a is the **TextStream** object returned by the **CreateTextFile** method on the **FileSystemObject**:

```
var fso = new ActiveXObject("Scripting.FileSystemO
var a = fso.CreateTextFile("c:\\testfile.txt", true);
a.WriteLine("This is a test.");
a.Close();
```

WriteLine and **Close** are two methods of the **TextStream** object.

Microsoft® JScript® TotalSize

Property

See Also

Applies To

Description

Returns the total space, in bytes, of a drive or network share.

Syntax

object.TotalSize

The *object* is always a **Drive** object.

Remarks

The following code illustrates the use of the **TotalSize** property:

```
function SpaceReport(drvPath)
{
  var fso, d, s;
  fso = new ActiveXObject("Scripting.FileSystemObjet d = fso.GetDrive(fso.GetDriveName(drvPath));
  s = "Drive " + drvPath + " - ";
  s += d.VolumeName + "<br/>
  s += "Total Space: "+ d.TotalSize/1024 + " Kbytes <|
  s += "Free Space: " + d.FreeSpace/1024 + " Kbytes"</pre>
```

```
return(s);
}
```

Applies To

Description

Returns information about the type of a file or folder. For example, for files ending in .TXT, "Text Document" is returned.

Syntax

```
object.Type
```

The *object* is always a **File** or **Folder** object.

Remarks

The following code illustrates the use of the **Type** property to return a folder type. In this example, try providing the path of the Recycle Bin or other unique folder to the procedure.

```
function ShowFileType(filespec)
{
  var fso, f, s;
  fso = new ActiveXObject("Scripting.FileSystemObjeti (fso.FolderExists(filespec))
    f = fso.GetFolder(filespec);
  else if (fso.FileExists(filespec))
```

```
f = fso.GetFile(filespec);
else
  s = "File or Folder does not exist.";
s = f.Name + " is a " + f.Type;
return(s);
}
```

Microsoft® JScript® VolumeName Property

See Also Applies To

Description

Sets or returns the volume name of the specified drive. Read/write.

Syntax

object.**VolumeName** [= newname]

The **VolumeName** property has these parts:

Part	Description
object	Required. Always the name of a Drive object.
newname	Optional. If provided, <i>newname</i> is the new name of the specified <i>object</i> .

Remarks

The following code illustrates the use of the **VolumeName** property:

```
function SpaceReport(drvPath)
{
  var fso, d, s;
  fso = new ActiveXObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.FileSystemObject("Scripting.
```

```
d = fso.GetDrive(fso.GetDriveName(drvPath));
s = "Drive " + drvPath + " - ";
s += d.VolumeName + "<br/>s += "Total Space: "+ d.TotalSize/1024 + " Kbytes <|
s += "Free Space: " + d.FreeSpace/1024 + " Kbytes"
return(s);
}</pre>
```

Applies To

Description

Writes a specified string to a **TextStream** file.

Syntax

object.Write(string)

The **Write** method syntax has these parts:

Part	Description
object	Required. Always the name of a TextStream object.
string	Required. The text you want to write to the file.

Remarks

Specified strings are written to the file with no intervening spaces or characters between each string. Use the **WriteLine** method to write a newline character or a string that ends with a newline character.

The following example illustrates the use of the **Write** method:

function WriteDemo()

```
var fso, f, r
var ForReading = 1, ForWriting =
fso = new ActiveXObject("Scripti
f = fso.OpenTextFile("c:\\testfile.t
f.Write("Hello world!");
f.Close();
f = fso.OpenTextFile("c:\\testfile.t
r = f.ReadLine();
return(r);
}
```

Method

See Also

Applies To

Description

Writes a specified number of newline characters to a **TextStream** file.

Syntax

object.WriteBlankLines(lines)

The **WriteBlankLines** method syntax has these parts:

Part	Description
ahiaat	Required. Always the name of a
object	Required. Always the name of a TextStream object.
lines	Required. Number of newline characters
	you want to write to the file.

Remarks

The following example illustrates the use of the WriteBlankLines method:

function WriteBlanksDemo()

```
var fso, f, r;
var ForReading = 1, ForWriting =
fso = new ActiveXObject("Scripti
f = fso.OpenTextFile("c:\\testfile.t
f.Write("Hello world!");
f.WriteBlankLines(2);
f.Write("JScript is fun!");
f.Close();
f = fso.OpenTextFile("c:\\testfile.t
r = f.ReadAll();
return(r);
```

Microsoft® JScript® WriteLine Method

See Also

Applies To

Description

Writes a specified string and newline character to a **TextStream** file.

Syntax

object.WriteLine([string])

The **WriteLine** method syntax has these parts:

Part	Description
object	Required. Always the name of a
	Required. Always the name of a TextStream object.
string	Optional. The text you want to write to
	the file. If omitted, a newline character
	is written to the file.

Remarks

The following example illustrates use of the **WriteLine** method:

```
f = fso.CreateTextFile("c:\\testfile.t
f.WriteLine("This is a test.");
f.Close();
```

Copyright

Microsoft® JScript®

Information in this document is subject to change without notice. The names of companies, products, people, characters, and/or data mentioned herein are fictitious and are in no way intended to represent any real individual, company, product, or event, unless otherwise noted. Complying with all applicable copyright laws is the responsibility of the user. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

© 1991-2000 Microsoft Corporation. All rights reserved.

Microsoft, MS, MS-DOS, ActiveX, JScript, Microsoft Press, Visual Basic, Windows, Windows NT, Win32, and Win32s are either registered trademarks or trademarks of Microsoft Corporation in the U.S.A. and/or other countries.

Other product and company names mentioned herein may be the trademarks of their respective owners.

Microsoft® JScript® Infinity Property

See Also Applies To

Description

Returns an initial value of **Number.POSITIVE_INFINITY**.

Syntax

Infinity

Remarks

The **Infinity** property is a member of the **Global** object, and is made available when the scripting engine is initialized.

Applies To

Description

Returns the day of the month value in a **Date** object using <u>local time</u>.

Syntax

```
objDate.getDate()
```

Remarks

To get the date value using <u>Universal Coordinated Time (UTC)</u>, use the **getUTCDate** method.

The return value is an integer between 1 and 31 that represents the date value in the **Date** object.

The following example illustrates the use of the **getDate** method:

```
function DateDemo()
{
  var d, s = "Today's date is: ";
  d = new Date();
  s += (d.getMonth() + 1) + "/";
  s += d.getDate() + "/";
```

```
s += d.getYear();
return(s);
}
```

Applies To

Description

Returns the day of the week value in a **Date** object using <u>local</u> time.

Syntax

objDate.getDay()

Remarks

To get the day using <u>Universal Coordinated Time (UTC)</u>, use the **getUTCDay** method.

The value returned from the **getDay** method is an integer between 0 and 6 representing the day of the week and corresponds to a day of the week as follows:

- 0 = Sunday
- 1 = Monday
- 2 = Tuesday
- 3 = Wednesday
- 4 = Thursday
- 5 = Friday
- 6 = Saturday

The following example illustrates the use of the **getDay** method:

function DateDemo()

```
var d, day, x, s = "Today is: ";
var x = new Array("Sunday", "Mo
var x = x.concat("Wednesday","T]
var x = x.concat("Saturday");
d = new Date();
day = d.getDay();
return(s += x[day]);
}
```

See Also

Applies To

Description

Returns the hours value in a **Date** object using <u>local time</u>.

Syntax

objDate.getHours()

Remarks

To get the hours value using <u>Universal Coordinated Time (UTC)</u>, use the **getUTCHours** method.

The **getHours** method returns an integer between 0 and 23 indicating the number of hours since midnight. A zero occurs in two situations: the time is before 1:00:00 am, or the time was not stored in the **Date** object when the object was created. The only way to determine which situation you have is to also check the minutes and seconds for zero values. If they are all zeroes, it is nearly certain that the time was not stored in the **Date** object.

The following example illustrates the use of the **getHours** method:

```
function TimeDemo()
{
  var d, s = "The current local time recover c = ":";
```

```
d = new Date();
s += d.getHours() + c;
s += d.getMinutes() + c;
s += d.getSeconds() + c;
s += d.getMilliseconds();
return(s);
}
```

Microsoft® JScript® getMinutes Method

See Also

Applies To

Description

Returns the minutes value in a **Date** object using <u>local time</u>.

Syntax

objDate.getMinutes()

Remarks

To get the minutes value using <u>Universal Coordinated Time</u> (<u>UTC</u>), use the **getUTCMinutes** method.

The **getMinutes** method returns an integer between 0 and 59 equal to the minutes value stored in the **Date** object. A zero is returned in two situations: one occurs when the time is less than one minute after the hour. The other occurs when the time was not stored in the **Date** object when the object was created. The only way to determine which situation you have is to also check the hours and seconds for zero values. If they are all zeroes, it is nearly certain that the time was not stored in the **Date** object.

The following example illustrates the use of the **getMinutes** method:

```
function TimeDemo()
{
  var d, s = "The current local time recover c = ":";
```

```
d = new Date();
s += d.getHours() + c;
s += d.getMinutes() + c;
s += d.getSeconds() + c;
s += d.getMilliseconds();
return(s);
}
```

Microsoft® JScript® getMonth

Method

See Also

Applies To

Description

Returns the month value in the **Date** object using <u>local time</u>.

Syntax

objDate.getMonth()

Remarks

To get the month value using <u>Universal Coordinated Time</u> (<u>UTC</u>), use the **getUTCMonth** method.

The **getMonth** method returns an integer between 0 and 11 indicating the month value in the **Date** object. The integer returned is not the traditional number used to indicate the month. It is one less. If "Jan 5, 1996 08:47:00" is stored in a **Date** object, **getMonth** returns 0.

The following example illustrates the use of the **getMonth** method:

```
function DateDemo()
{
  var d, s = "Today's date is: ";
  d = new Date();
  s += (d.getMonth() + 1) + "/";
```

```
s += d.getDate() + "/";
s += d.getYear();
return(s);
}
```

Microsoft® JScript® getSeconds Method

See Also

Applies To

Description

Returns the seconds value in a **Date** object using <u>local time</u>.

Syntax

objDate.getSeconds()

Remarks

To get the seconds value using <u>Universal Coordinated Time</u> (<u>UTC</u>), use the **getUTCSeconds** method.

The **getSeconds** method returns an integer between 0 and 59 indicating the seconds value of the indicated **Date** object. A zero is returned in two situations. One occurs when the time is less than one second into the current minute. The other occurs when the time was not stored in the **Date** object when the object was created. The only way to determine which situation you have is to also check the hours and minutes for zero values. If they are all zeroes, it is nearly certain that the time was not stored in the **Date** object.

The following example illustrates the use of the **getSeconds** method:

```
function TimeDemo()
{
  var d, s = "The current local time :
```

```
var c = ":";
d = new Date();
s += d.getHours() + c;
s += d.getMinutes() + c;
s += d.getSeconds() + c;
s += d.getMilliseconds();
return(s);
}
```

See Also

Applies To

Description

Returns the time value in a **Date** object.

Syntax

objDate.getTime()

Remarks

The **getTime** method returns an integer value representing the number of milliseconds between midnight, January 1, 1970 and the time value in the **Date** object. The range of dates is approximately 285,616 years from either side of midnight, January 1, 1970. Negative numbers indicate dates prior to 1970.

When doing multiple date and time calculations, it is frequently useful to define variables equal to the number of milliseconds in a day, hour, or minute. For example:

The following example illustrates the use of the **getTime**

method:

```
function GetTimeTest()
 var d, s, t;
 var MinMilli = 1000 * 60;
 var HrMilli = MinMilli * 60;
 var DyMilli = HrMilli * 24;
 d = new Date();
 t = d.getTime();
 s = "It's been "
 s += Math.round(t / DyMilli) + " (
 return(s);
}
```

Microsoft® JScript®

getTimezoneOffset Method

Language Reference Version 1

See Also

Applies To

Description

Returns the difference in minutes between the time on the host computer and <u>Universal</u> <u>Coordinated Time (UTC)</u>.

Syntax

objDate.getTimezoneOffset()

Remarks

The **getTimezoneOffset** method returns an integer value representing the number of minutes between the time on the current machine and UTC. These values are appropriate to the computer the script is executed on. If it is called from a server script, the return value is appropriate to the server. If it is called from a client script, the return value is appropriate to the client.

This number will be positive if you are behind UTC (e.g., Pacific Daylight Time), and negative if you are ahead of UTC (e.g., Japan).

For example, suppose a server in New York City is contacted by a client in Los Angeles on December 1. **getTimezoneOffset** returns 480 if executed on the client, or 300 if executed on the server.

The following example illustrates the use of the **getTimezoneOffset** method:

function TZDemo()

```
var d, tz, s = "The current local tir
d = new Date();
tz = d.getTimezoneOffset();
if (tz < 0)
 s += tz / 60 + "hours before GM
else if (tz == 0)
 s += "GMT";
else
 s += tz / 60 + "hours after GMT
return(s);
```

Microsoft® JScript® getYear Method

See Also

Applies To

Description

Returns the year value in a **Date** object.

Syntax

objDate.getYear()

Remarks

This method is obsolete, and is provided for backwards compatibility only. Use the **getFullYear** method instead.

For years from 1900 through 1999, the year is a 2-digit integer value returned as the difference between the stored year and 1900. For other dates, the 4-digit year is returned. For example, 1996 is returned as 96, but 1825 and 2025 are returned as-is.

Note For JScript version 1.0, **get Year** returns a value that is the result of the subtraction of 1900 from the year value in the provided **Date** object, regardless of the value of the year. For example, the year 1899 is returned as -1 and the year to 2000 is returned as 100.

The following example illustrates the use of the **getYear** method:

```
function DateDemo()
{
```

```
var d, s = "Today's date is: ";
d = new Date();
s += (d.getMonth() + 1) + "/";
s += d.getDate() + "/";
s += d.getYear();
return(s);
}
```

Microsoft® JScript® getUTCFullYear

Method

See Also

Applies To

Description

Returns the year value in a **Date** object using <u>Universal</u> <u>Coordinated Time (UTC)</u>.

Syntax

objDate.getUTCFullYear()

Remarks

To get the year using <u>local time</u>, use the **getFullYear** method.

The **getUTCFullYear** method returns the year as an absolute number. This avoids the classic year 2000 problem where dates beginning with January 1, 2000 are confused with those beginning January 1, 1900.

The following example illustrates the use of the **getUTCFullYear** method:

```
function UTCDateDemo()
{
  var d, s = "Today's UTC date is: ";
  d = new Date();
  s += (d.getUTCMonth() + 1) + "/";
```

```
s += d.getUTCDate() + "/";
s += d.getUTCFullYear();
return(s);
}
```

Language Reference Version 3

Microsoft® JScript® getUTCHours

Method

See Also

Applies To

Description

Returns the hours value in a **Date** object using <u>Universal</u> <u>Coordinated Time (UTC)</u>.

Syntax

objDate.getUTCHours()

Remarks

To get the number of hours elapsed since midnight using <u>local</u> <u>time</u>, use the **getHours** method.

The **getUTCHours** method returns an integer between 0 and 23 indicating the number of hours since midnight. A zero occurs in two situations: the time is before 1:00:00 A.M., or a time was not stored in the **Date** object when the object was created. The only way to determine which situation you have is to also check the minutes and seconds for zero values. If they are all zeroes, it is nearly certain that the time was not stored in the **Date** object.

The following example illustrates the use of the **getUTCHours** method:

```
function UTCTimeDemo()
{
```

```
var d, s = "Current Universal Coordinated Tim
var c = ":";
d = new Date();
s += d.getUTCHours() + c;
s += d.getUTCMinutes() + c;
s += d.getUTCSeconds() + c;
s += d.getUTCMilliseconds();
return(s);
}
```

Language Reference Version 3

Microsoft® JScript® getUTCMinutes Method

See Also

Applies To

Description

Returns the minutes value in a **Date** object using <u>Universal</u> <u>Coordinated Time (UTC)</u>.

Syntax

objDate.getUTCMinutes()

Remarks

To get the number of minutes stored using <u>local time</u>, use the **getMinutes** method.

The **getUTCMinutes** method returns an integer between 0 and 59 equal to the number of minutes value in the **Date** object. A zero occurs in two situations: the time is less than one minute after the hour, or a time was not stored in the **Date** object when the object was created. The only way to determine which situation you have is to also check the hours and seconds for zero values. If they are all zeroes, it is nearly certain that the time was not stored in the **Date** object.

The following example illustrates the use of the **getUTCMinutes** method:

```
function UTCTimeDemo()
{
```

```
var d, s = "Current Universal Coordinated Tim
var c = ":";
d = new Date();
s += d.getUTCHours() + c;
s += d.getUTCMinutes() + c;
s += d.getUTCSeconds() + c;
s += d.getUTCMilliseconds();
return(s);
}
```

Microsoft® JScript® getUTCMonth Method

See Also

Applies To

Description

Returns the month value value in a **Date** object using <u>Universal</u> <u>Coordinated Time (UTC)</u>.

Syntax

objDate.getUTCMonth()

Remarks

To get the month in <u>local time</u>, use the **getMonth** method.

The **getUTCMonth** method returns an integer between 0 and 11 indicating the month value in the Date object. The integer returned is not the traditional number used to indicate the month. It is one less. If "Jan 5, 1996 08:47:00.0" is stored in a **Date** object, **getUTCMonth** returns 0.

The following example illustrates the use of the **getUTCMonth** method:

```
function UTCDateDemo()
{
  var d, s = "Today's UTC date is: ";
  d = new Date();
```

```
s += (d.getUTCMonth() + 1) + "/";
s += d.getUTCDate() + "/";
s += d.getUTCFullYear();
return(s);
}
```

Language Reference Version 3

Microsoft® JScript® getUTCSeconds

Method

See Also Applies To

Description

Returns the seconds value in a **Date** object using <u>Universal</u> <u>Coordinated Time (UTC)</u>.

Syntax

objDate.getUTCSeconds()

Remarks

To get the number of seconds in <u>local time</u>, use the **getSeconds** method.

The **getUTCSeconds** method returns an integer between 0 and 59 indicating the seconds value of the indicated **Date** object. A zero occurs in two situations: the time is less than one second into the current minute, or a time was not stored in the **Date** object when the object was created. The only way to determine which situation you have is to also check the minutes and hours for zero values. If they are all zeroes, it is nearly certain that the time was not stored in the **Date** object.

The following example illustrates the use of the **getUTCSeconds** method:

```
function UTCTimeDemo()
{
```

```
var d, s = "Current Universal Coordinated Tim
var c = ":";
d = new Date();
s += d.getUTCHours() + c;
s += d.getUTCMinutes() + c;
s += d.getUTCSeconds() + c;
s += d.getUTCMilliseconds();
return(s);
}
```

See Also

Applies To

Description

Sets the numeric date of the **Date** object using <u>local time</u>.

Syntax

```
objDate.setDate(numDate)
```

The *numDate* argument is a numeric value equal to the numeric date.

Remarks

To set the date value using <u>Universal Coordinated Time (UTC)</u>, use the **setUTCDate** method.

If the value of *numDate* is greater than the number of days in the month stored in the **Date** object or is a negative number, the date is set to a date equal to *numDate* minus the number of days in the stored month. For example, if the stored date is January 5, 1996, and **setDate(32)** is called, the date changes to February 1, 1996. Negative numbers have a similar behavior.

The following example illustrates the use of the **setDate** method:

```
function SetDateDemo(newdate)
{
  var d, s;
  d = new Date();
```

```
d.setDate(newdate);
s = "Current setting is ";
s += d.toLocaleString();
return(s);
}
```

See Also Applies To

Description

Sets the hour value in the **Date** object using <u>local time</u>.

Syntax

objDate.setHours(numHours[, numMin[, numSec[, numMilli]]])

The **setHours** method syntax has these parts:

Part	Description
numHours	Required. A numeric value equal to the hours value.
∥niim \/iin	Optional. A numeric value equal to the minutes value. Must be supplied if either of the following arguments are used.
	Optional. A numeric value equal to the seconds value. Must be supplied if the following argument is used.
numMilli	Optional. A numeric value equal to the milliseconds value.

Remarks

All **set** methods taking optional arguments use the value returned from corresponding **get** methods, if you do not specify an optional argument. For example, if the *numMonth* argument is optional, but not specified, JScript uses the value returned from the **getMonth** method.

To set the hours value using <u>Universal Coordinated Time (UTC</u>), use the **setUTCHours** method.

If the value of an argument is greater than its range or is a negative number, other stored values are modified accordingly. For example, if the stored date is "Jan 5, 1996 00:00:00", and **setHours(30)** is called, the date is changed to "Jan 6, 1996 06:00:00." Negative numbers have a similar behavior.

The following example illustrates the use of the **setHours** method:

```
function SetHoursDemo(nhr, nmin,
{
  var d, s;
  var sep = ":";
  d = new Date();
  d.setHours(nhr, nmin, nsec);
  s = "Current setting is " + d.toLoc
  return(s);
}
```

See Also Applies To

Description

Sets the month value in the **Date** object using <u>local time</u>.

Syntax

objDate.setMonth(numMonth[, dateVal])

The **setMonth** method syntax has these parts:

Part	Description
numMonth	Required. A numeric value equal to
	the month.
dateVal	Optional. A numeric value
	representing the date. If not
	supplied, the value from a call to
	the getDate method is used.

Remarks

To set the month value using <u>Universal Coordinated Time</u> (<u>UTC</u>), use the **setUTCMonth** method.

If the value of *numMonth* is greater than 11 (January is month 0) or is a negative number, the stored year is modified accordingly. For example, if the stored date is "Jan 5, 1996" and **setMonth(14)** is called, the date is changed to "Mar 5, 1997."

The following example illustrates the use of the **setMonth** method:

```
function SetMonthDemo(newmont
{
  var d, s;
  d = new Date();
  d.setMonth(newmonth);
  s = "Current setting is ";
  s += d.toLocaleString();
  return(s);
}
```

Microsoft® JScript® setSeconds

Method

See Also Applies To

Description

Sets the seconds value in the **Date** object using <u>local time</u>.

Syntax

objDate.setSeconds(numSeconds[, numMilli])

The **setSeconds** method syntax has these parts:

Part	Description
numSeconds	Required. A numeric value equal to the seconds value.
numMilli	Optional. A numeric value equal to the milliseconds value.

Remarks

All **set** methods taking optional arguments use the value returned from corresponding **get** methods, if you do not specify an optional argument. For example, if the *numMonth* argument is optional, but not specified, JScript uses the value returned from the **getMonth** method.

To set the seconds value using <u>Universal Coordinated Time (UTC)</u>, use the **setUTCSeconds** method.

If the value of an argument is greater than its range or is a negative number, other stored

values are modified accordingly. For example, if the stored date is "Jan 5, 1996 00:00:00" and **setSeconds(150)** is called, the date is changed to "Jan 5, 1996 00:02:30."

The following example illustrates the use of the **setSeconds** method:

```
function SetSecondsDemo(nsec, nr
{
  var d, s;
  var sep = ":";
  d = new Date();
  d.setSeconds(nsec, nmsec);
  s = "Current setting is ";
  s += d.toLocaleString() + sep + d.
  return(s);
}
```

See Also

Applies To

Description

Sets the date and time value in the **Date** object.

Syntax

objDate.setTime(milliseconds)

The *milliseconds* argument is an integer value representing the number of elapsed seconds since midnight, January 1, 1970 GMT.

Remarks

If *milliseconds* is negative, it indicates a date before 1970. The range of available dates is approximately 285,616 years from either side of 1970.

Setting the date and time with the **setTime** method is independent of the time zone.

The following example illustrates the use of the **setTime** method:

```
function SetTimeTest(newtime)
{
  var d, s;
  d = new Date();
```

```
d.setTime(newtime);
s = "Current setting is ";
s += d.toUTCString();
return(s);
}
```

Microsoft® JScript® setYear Method

See Also

Applies To

Description

Sets the year value in the **Date** object.

Syntax

objDate.setYear(numYear)

The *numYear* argument is a numeric value equal to the year minus 1900.

Remarks

This method is obsolete, and is maintained for backwards compatibility only. Use the **setFullYear** method instead.

To set the year of a **Date** object to 1997, call **setYear(97)**. To set the year to 2010, call **setYear(2010)**. Finally, to set the year to a year in the range 0-99, use the **setFullYear** method.

Note For JScript version 1.0, **setYear** uses a value that is the result of the addition of 1900 to the year value provided by the *numYear*, regardless of the value of the year. For example, to set the year to 1899 *numYear* is -1 and to set the year to 2000 *numYear* is 100.

The following example illustrates the use of the **setYear** method:

function SetYearDemo(newyear)

```
var d, s;
d = new Date();
d.setYear(newyear);
s = "Current setting is ";
s += d.toLocaleString();
return(s);
}
```

Microsoft® JScript® setUTCFullYear

Method

See Also Applies To

Description

Sets the year value in the **Date** object using <u>Universal</u> <u>Coordinated Time (UTC)</u>.

Syntax

objDate.setUTCFullYear(numYear[, numMonth[, numDate]])

The **setUTCFullYear** method syntax has these parts:

Part	Description	
numYear	Required. A numeric value equal to the year.	
numMonth	Optional. A numeric value equal to the month. Musbe supplied if <i>numDate</i> is supplied.	
numDate	Optional. A numeric value equal to the date.	

Remarks

All **set** methods taking optional arguments use the value returned from corresponding **get** methods, if you do not specify an optional argument. For example, if the *numMonth* argument is optional, but not specified, JScript uses the value returned from the **getMonth** method.

In addition, if the value of an argument is greater that its range or is a

negative number, other stored values are modified accordingly.

To set the year using <u>local time</u>, use the **setFullYear** method.

The range of years supported in the **Date** object is approximately 285,616 years from either side of 1970.

The following example illustrates the use of the **setUTCFullYear** method:

```
function SetUTCFullYearDemo(newyear)
{
  var d, s;
  d = new Date();
  d.setUTCFullYear(newyear);
  s = "Current setting is ";
  s += d.toUTCString();
  return(s);
}
```

Microsoft® JScript® setUTCHours

Method

See Also Applies To

Description

Sets the hours value in the **Date** object using <u>Universal</u> <u>Coordinated Time (UTC)</u>.

Syntax

objDate.setUTCHours(numHours[, numMin[, numSec[,
numMilli]]])

The **setUTCHours** method syntax has these parts:

Part	Description
numHours	Required. A numeric value equal to the hours value.
numMin	Optional. A numeric value equal to the minutes value. Must be supplied if either <i>numSec</i> or <i>numMilli</i> are used.
	Optional. A numeric value equal to the seconds value. Must be supplied if <i>numMilli</i> argument is used.
numMilli	Optional. A numeric value equal to the milliseconds value.

Remarks

All **set** methods taking optional arguments use the value returned

from corresponding **get** methods, if you do not specify an optional argument. For example, if the *numMonth* argument is optional, but not specified, JScript uses the value returned from the **getMonth** method.

To set the hours value using <u>local time</u>, use the **setHours** method.

If the value of an argument is greater than its range or is a negative number, other stored values are modified accordingly. For example, if the stored date is "Jan 5, 1996 00:00:00.00", and **setUTCHours(30)** is called, the date is changed to "Jan 6, 1996 06:00:00.00."

The following example illustrates the use of the **setUTCHours** method:

```
function SetUTCHoursDemo(nhr, nmin, nsec)
{
  var d, s;
  var sep = ":";
  d = new Date();
  d.setUTCHours(nhr, nmin, nsec);
  s = "Current setting is " + d.toUTCString()
  return(s);
}
```

Microsoft® JScript® **SetUTCMinutes**

Method

See Also

Applies To

Description

Sets the minutes value in the **Date** object using <u>Universal</u> <u>Coordinated Time (UTC)</u>.

Syntax

objDate.setUTCMinutes(numMinutes[, numSeconds[,
numMilli]])

The **setUTCMinutes** method syntax has these parts:

Part	Description	
numMinutes	Required. A numeric value equal to the minutes value.	
numSeconds	Optional. A numeric value equal to the seconds value. Must be supplied if <i>numMilli</i> is used.	
IIniim Miiiii I	Optional. A numeric value equal to the milliseconds value.	

Remarks

All **set** methods taking optional arguments use the value returned from corresponding **get** methods, if you do not specify an optional argument. For example, if the *numMonth* argument is optional, but not specified, JScript uses the value returned from

the **getMonth** method.

To modify the minutes value using <u>local time</u>, use the **setMinutes** method.

If the value of an argument is greater than its range or is a negative number, other stored values are modified accordingly. For example, if the stored date is "Jan 5, 1996 00:00:00.00", and **setUTCMinutes(70)** is called, the date is changed to "Jan 5, 1996 01:10:00.00."

The following example illustrates the use of the **setUTCMinutes** method:

```
function SetUTCMinutesDemo(nmin, nsec)
{
  var d, s;
  var sep = ":";
  d = new Date();
  d.setUTCMinutes(nmin,nsec);
  s = "Current setting is " + d.toUTCString()
  return(s);
}
```

Microsoft® JScript® setUTCMonth

Method

See Also Applies To

Description

Sets the month value in the **Date** object using <u>Universal</u> <u>Coordinated Time (UTC)</u>.

Syntax

objDate.setUTCMonth(numMonth[, dateVal])

The **setUTCMonth** method syntax has these parts:

Part	Description
numMonth	Required. A numeric value equal to the month.
dateVal	Optional. A numeric value representing the date. If not supplied, the value from a call to the getUTCDate method is used.

Remarks

To set the month value using <u>local time</u>, use the **setMonth** method.

If the value of *numMonth* is greater than 11 (January is month 0) or is a negative number, the stored year is incremented or decremented appropriately. For example, if the stored date is "Jan 5, 1996 00:00:00.00", and **setUTCMonth(14)** is called, the date is changed to "Mar 5, 1997

```
00:00:00.00."
```

The following example illustrates the use of the **setUTCMonth** method:

```
function SetUTCMonthDemo(newmonth)
{
  var d, s;
  d = new Date();
  d.setUTCMonth(newmonth);
  s = "Current setting is ";
  s += d.toUTCString();
  return(s);
}
```

${\tt Microsoft \& JScript \& } \textbf{SetUTCS} econds$

Method

See Also Applies To

Description

Sets the seconds value in the **Date** object using <u>Universal</u> <u>Coordinated Time (UTC)</u>.

Syntax

objDate.setUTCSeconds(numSeconds[, numMilli])

The **setUTCSeconds** method syntax has these parts:

Part	Description	
numSeconds	Required. A numeric value equal to the seconds value.	
llniim Millii	Optional. A numeric value equal to the milliseconds value.	

Remarks

All **set** methods taking optional arguments use the value returned from corresponding **get** methods, if you do not specify an optional argument. For example, if the *numMonth* argument is optional, but not specified, JScript uses the value returned from the **getMonth** method.

To set the seconds value using <u>local time</u>, use the **setSeconds** method.

If the value of an argument is greater than its range or is a negative number, other stored values are modified accordingly. For example, if the stored date is "Jan 5, 1996 00:00:00.00" and **setSeconds(150)** is called, the date is changed to "Jan 5, 1996 00:02:30.00."

The following example illustrates the use of the **setSeconds** method:

```
function SetUTCSecondsDemo(nsec, nmsec)
{
  var d, s;
  var sep = ":";
  d = new Date();
  d.setUTCSeconds(nsec, nmsec);
  s = "Current setting is ";
  s += d.toUTCString() + sep + d.getUTCMillis return(s);
}
```

Microsoft® JScript® toLocaleString

Method

See Also

Applies To

Description

Returns a date converted to a string using the current <u>locale</u>.

Syntax

```
dateObj.toLocaleString( )
```

Remarks

The **toLocaleString** method returns a **String** object that contains the date written in the current locale's default format. The format of the return value depends on the current locale. For example, in the United States, **toLocaleString** may return "01/05/96 00:00:00" for January 5, but in Europe, it may return "05/01/96 00:00:00" for the same date, as European convention puts the day before the month.

The following example illustrates the use of the **toLocaleString** method:

```
function toLocaleStrDemo()
{
  var d, s;
  d = new Date();
```

```
s = "Current setting is ";
s += d.toLocaleString();
return(s);
}
```

Microsoft® JScript® parse Method

See Also

Applies To

Description

Parses a string containing a date, and returns the number of milliseconds between that date and midnight, January 1, 1970.

Syntax

Date.parse(dateVal)

The *dateVal* argument is either a string containing a date in a format such as "Jan 5, 1996 08:47:00" or a VT_DATE value retrieved from an ActiveX® object or other object.

Remarks

The **parse** method returns an integer value representing the number of milliseconds between midnight, January 1, 1970 and the date supplied in *dateVal*.

The **parse** method is a static method of the **Date** object. Because it is a static method, it is invoked as shown in the following example rather than invoked as a method of a created **Date** object.

var datestring = "November 1, 199" Date.parse(datestring)

The following rules govern what the **parse** method can successfully parse:

• Short dates can use either a "/" or "-" date separator, but must

follow the month/day/year format, for example "7/20/96".

- Long dates of the form "July 10 1995" can be given with the year, month, and day in any order, and the year in 2-digit or 4-digit form. If you use the 2-digit form, the year must be greater than or equal to 70.
- Any text inside parentheses is treated as a comment. These parentheses may be nested.
- Both commas and spaces are treated as delimiters. Multiple delimiters are permitted.
- Month and day names must have two or more characters. Two character names that are not unique are resolved as the last match. For example, "Ju" is resolved as July, not June.
- The stated day of the week is ignored if it is incorrect given the remainder of the supplied date. For example, "Tuesday November 9 1996" is accepted and parsed even though that date actually falls on a Friday. The resulting **Date** object contains "Friday November 9 1996".
- JScript handles all standard time zones, as well as Universal Coordinated Time (UTC) and Greenwich Mean Time (GMT).
- Hours, minutes, and seconds are separated by colons, although all need not be specified. "10:", "10:11", and "10:11:12" are all valid.
- If the 24-hour clock is used, it is an error to specify "PM" for times later than 12 noon. For example, "23:15 PM" is an error.
- A string containing an invalid date is an error. For example, a string containing two years or two months is an error.

The following example illustrates the use of the **parse** method:

```
function GetTimeTest(testdate)
{
  var d, s, t;
  var MinMilli = 1000 * 60;
  var HrMilli = MinMilli * 60;
  var DyMilli = HrMilli * 24;
  d = new Date();
  t = Date.parse(testdate);
  s = "There are "
  s += Math.round(Math.abs(t / DyMilli)) + " day
  s += "between " + testdate + " and 1/1/70";
  return(s);
}
```

See Also Applies To

Description

Returns the number of milliseconds between midnight, January 1, 1970 <u>Universal</u> <u>Coordinated Time (UTC)</u> (or GMT) and the supplied date.

Syntax

Date.UTC(year, month, day[, hours[, minutes[, seconds[,ms]]]])

The **UTC** method syntax has these parts:

Part	Description	
	Required. The full year designation is	
	required for cross-century date	
year	accuracy. If <i>year</i> is between 0 and 99	
	is used, then <i>year</i> is assumed to be	
	1900 + year.	
	Required. The month as an integer	
month	between 0 and 11 (January to	
	December).	
data	Required. The date as an integer	
date	between 1 and 31.	
	Optional. Must be supplied if <i>minutes</i>	
_	is supplied. An integer from 0 to 23	
hours	(midnight to 11pm) that specifies the	

	hour.
	Optional. Must be supplied if <i>seconds</i>
	is supplied. An integer from 0 to 59
	that specifies the minutes.
seconds	Optional. Must be supplied if
	milliseconds is supplied. An integer
	from 0 to 59 that specifies the seconds.
IIMC I	Optional. An integer from 0 to 999
	that specifies the milliseconds.

Remarks

The **UTC** method returns the number of milliseconds between midnight, January 1, 1970 UTC and the supplied date. This return value can be used in the **setTime** method and in the **Date** object <u>constructor</u>. If the value of an argument is greater than its range or is a negative number, other stored values are modified accordingly. For example, if you specify 150 seconds, JScript redefines that number as two minutes and 30 seconds.

The difference between the **UTC** method and the **Date** object constructor that accepts a date is that the **UTC** method assumes UTC, and the **Date** object constructor assumes local time.

The **UTC** method is a static method. Therefore, a **Date** object does not have to be created before it can be used. The **UTC** method is invoked as follows:

var datestring = "November 1, 199". Date.UTC(datestring)

Note If *year* is between 0 and 99, use *1900* + *year* for the year.

```
function DaysBetweenDateAndNo<sup>1</sup>
{
 var d, r, t1, t2, t3;
 var MinMilli = 1000 * 60
 var HrMilli = MinMilli * 60
 var DyMilli = HrMilli * 24
 t1 = Date.UTC(yr, mo, dy)
 d = new Date();
 t2 = d.getTime();
 if (t2 >= t1)
  t3 = t2 - t1;
 else
  t3 = t1 - t2;
 r = Math.round(t3 / DyMilli);
 return(r);
```

Microsoft® JScript® function Statement

See Also

Description

Declares a new function.

Syntax

```
function functionname([argument1 [, argument2 [,
...argumentn]]])
{
   statements
}
```

The **function** statement syntax has the following parts:

Part	Description
functionname	The name of the
functionnume	function.
	An optional, comma-
argument1argumentn	separated list of
argamentiargamentir	arguments the function
	understands.
statoments	One or more JScript
statements	statements.

Remarks

Use the **function** statement to declare a function for later use. The code contained in *statements* is not executed until the function is called from elsewhere in the script.

The following example illustrates the use of the **function** statement:

```
function myfunction(arg1, arg2)
{
  var r;
  r = arg1 * arg2;
  return(r);
}
```

Note When calling a function, ensure that you always include the parentheses and any required arguments. Calling a function without parentheses causes the text of the function to be returned instead of the results of the function.

${\tt Microsoft @ JScript @ } parse Int \ Method$

See Also Applies To

Description

Returns an integer converted from a string.

Syntax

parseInt(numstring, [radix])

The **parseInt** method syntax has these parts:

Part	Description
numstring	Required. A string to convert into a
	number.
radix	Optional. A value between 2 and 36
	indicating the base of the number
	contained in <i>numstring</i> . If not
	supplied, strings with a prefix of '0x'
	are considered hexidecimal and
	strings with a prefix of '0' are
	considered octal. All other strings
	are considered decimal.

Remarks

The **parseInt** method returns an integer value equal to the

number contained in *numstring*. If no prefix of *numstring* can be successfully parsed into an integer, **NaN** (not a number) is returned.

```
parseInt("abc") // Returns NaN.
parseInt("12abc") // Returns 12.
```

You can test for NaN using the isNaN method.

Microsoft® JScript® parseFloat Method

See Also

Applies To

Description

Returns a floating-point number converted from a string.

Syntax

parseFloat(numstring)

The *numstring* argument is a string that contains a floating-point number.

Remarks

The **parseFloat** method returns an numerical value equal to the number contained in *numstring*. If no prefix of *numstring* can be successfully parsed into a floating-point number, **NaN** (not a number) is returned.

```
parseFloat("abc") // Returns NaN.
parseFloat("1.2abc") // Returns 1.2.
```

You can test for **NaN** using the **isNaN** method.

Description

Divides two numbers and returns the remainder.

Syntax

result = number1 % number2

The % operator syntax has these parts:

Part	Description
result	Any <u>variable.</u>
number1	Any <u>numeric expression</u> .
number2	Any numeric expression.

Remarks

The modulus, or remainder, operator divides *number1* by *number2* (rounding floating-point numbers to integers) and returns only the remainder as *result*. For example, in the following expression, A (which is *result*) equals 5.

$$A = 19 \% 6.7$$

For information on when a <u>run-time error</u> is generated by the % operator, see the <u>Operator Behavior</u> table.



Description

Multiplies two numbers.

Syntax

result = number1*number2

The * operator syntax has these parts:

Part	Description
result	Any <u>variable</u> .
number1	Any expression.
number2	Any expression.

Remarks

For information on when a <u>run-time error</u> is generated by the * operator, see the <u>Operator Behavior</u> table.

Description

Used to divide two numbers and return a numeric result.

Syntax

result = number1 / number2

The / operator syntax has these parts:

Part	Description
result	Any numeric <u>variable</u> .
number1	Any <u>numeric expression</u> .
number2	Any numeric expression.

Remarks

For information on when a <u>run-time error</u> is generated by the / operator, see the <u>Operator Behavior</u> table.

Microsoft® JScript® Global Object

See Also Methods Properties

Description

An <u>intrinsic object</u> whose purpose is to collect global methods into one object.

Syntax

The **Global** object has no syntax. You call its methods directly.

Remarks

The **Global** object is never used directly, and cannot be created using the **new** operator. It is created when the scripting engine is initialized, thus making its methods and properties available immediately.

See Also Applies To

Description

Returns the array of strings that results when a string is separated into substrings.

Syntax

stringObj.split(str)

The **split** method syntax has these parts:

Part	Description	
stringObj	Required. The String object or literal to be split. This object is not modified by the split method.	
str	Required. A string or Regular Expression object describing what character is used to define where the splits take place.	

Remarks

The result of the **split** method is an array of strings split at each point where *str* occurred in *stingObj*.

The following example illustrates the use of the **split** method:

function SplitDemo()

```
var s, ss;
var s = "The quick brown fox jumped over the
// Split at each space character.
ss = s.split(" ");
return(ss);
}
```

Microsoft® JScript® getVarDate Method

See Also

Applies To

Description

Returns the VT_DATE value in a **Date** object.

Syntax

dateobj.getVarDate()

The *dateobj* argument is any **Date** object.

Remarks

The **getVarDate** method is used when interacting with ActiveX® objects or other objects that accept and return date values in VT_DATE format.

Applies To

Description

Returns, as an array, the results of a search on a string using a supplied **Regular Expression** object.

Syntax

stringObj.match(rgExp)

The **match** method syntax has these parts:

Part	Description	
stringObj	Required. The String object or literal on which to perform the search.	
	Required. The regular expression to use in the search.	

Remarks

The **match** method, which behaves like the **exec** method, returns an array of values. Element zero of the array contains the last matched characters. Elements 1...*n* contain matches to any parenthesized substrings in the regular expression.

The method updates the contents of the **RegExp** object.

The following example illustrates the use of the **match** method:

```
function MatchDemo()
{
  var r, re;
  var s = "The quick brown fox jumped over the re = /fox/i;
  r = s.match(re);
  return(r);
}
```

See Also Applies To

Description

Returns a copy of a string with text replaced using a regular expression.

Syntax

stringObj.replace(rgExp, replaceText)

The **replace** method syntax has these parts:

Part	Description	
II I	Required. The String object or literal on which to perform the replace. This object is not modified by the replace method.	
IIra + yn 🔠	Required. A Regular Expression object describing what to search for.	
	Required. A String object or literal containing the eplaceText text to replace for every successful match of rgEx in stringObj.	

Remarks

The result of the **replace** method is a copy of *stringObj* after all replacements have been made.

The method updates the contents of the **RegExp** object.

The following example illustrates the use of the **replace** method:

```
function ReplaceDemo()
{
  var r, re;
  var s = "The quick brown fox jumped over the re = /fox/i;
  r = s.replace(re, "pig");
  return(r);
}
```

In addition, the **replace** method can also replace subexpressions in the pattern. The following example swaps each pair of words in the string:

```
function ReplaceDemo()
{
  var r, re;
  var s = "The quick brown fox jumped over the re = /(\S+)(\s+)(\S+)/g;
  r = s.replace(re, "$3$2$1"); // Swap each pair return(r);
}
```

Scripting Run-Time Reference Version 3

OpenAsTextStream Method

See Also Applies To

Description

Opens a specified file and returns a **TextStream** object that can be used to read from, write to, or append to the file.

Syntax

object.OpenAsTextStream([iomode, [format]])

The **OpenAsTextStream** method syntax has these parts:

Part	Description	
object	Required. Always the name of a File object.	
iomode	Optional. Indicates input/output mode. Can be one of three constants: ForReading, ForWriting, or ForAppending.	
format	Optional. One of three Tristate values used to indicate the format of the opened file. If omitted, the file is opened as ASCII.	

Settings

The *iomode* argument can have any of the following settings:

Constant	Value	Description

ForReading		Open a file for reading only. You can't write to this file.			
ForWriting	2	Open a file for writing. If a file with the same name exists, its previous contents are overwritten.			
ForAppending	8	Open a file and write to the end of the file.			

The *format* argument can have any of the following settings:

Constant	Value	Description		
TristateUseDefault	I -/ I	Opens the file using the system default.		
TristateTrue	-1	Opens the file as Unicode.		
TristateFalse	0	Opens the file as ASCII.		

Remarks

The **OpenAsTextStream** method provides the same functionality as the **OpenTextFile** method of the **FileSystemObject**. In addition, the **OpenAsTextStream** method can be used to write to a file.

The following code illustrates the use of the **OpenAsTextStream** method:

```
function TextStreamTest( )
{
   var fso, f, ts, s;
   var ForReading = 1, ForWriting = 2, ForApp
   var TristateUseDefault = -2, TristateTrue = -1
   fso = new ActiveXObject("Scripting.FileSys")
```

Microsoft® JScript® **abs Method**

Language Reference

Applies To

Math Object

Microsoft® JScript® acos Method

Language Reference

Applies To

Math Object

${\tt Microsoft @ JScript @} \ Active XObject$

Object See Also

Language Reference

GetObject Function

Microsoft® JScript® Operator

Behavior

The following table describes the behavior of most Microsoft JScript operators. The columns and rows represent the different types of expressions possible on either side of an operator in JScript, and the entries in the table describe the behavior.

An **E** indicates a <u>run-time error</u>. An **N** indicates a numeric result, or a Boolean result in the case of logical operators.

	obj	as	ns	num	bool	undef	null
obj	N	E	N	N	N	E	E
as	E	E	E	E	E	E	E
ns	N	E	N	N	N	E	E
num	N	E	N	N	N	E	E
bool	N	E	N	N	N	E	E
undef	E	E	E	E	E	E	E
null	E	E	E	E	E	E	E

obj = object, as = alphanumeric string, ns = numeric string, num = number, bool = Boolean, undef = undefined, null = null value.

Microsoft® JScript® anchor Method

Language Reference

See Also

link Method
String Object Methods
String Object Properties

Microsoft® JScript® anchor Method

Language Reference

Applies To

String Object

Microsoft® JScript® asin Method

Language Reference

Applies To

Math Object

Microsoft® JScript® atan Method

Language Reference

Applies To

Math Object

Microsoft® JScript® atan2 Method

Language Reference

See Also

atan MethodMath Object Methodstan Method

Microsoft® JScript® atan2 Method

Language Reference

Applies To

Math Object

See Also

<u>item Method</u> <u>moveFirst Method</u> <u>moveNext Method</u>

Microsoft® JScript® atEnd Method

Language Reference

Applies To

Enumerator Object

${\tt Microsoft @ JScript @ } big \ Method$

See Also

small MethodString Object MethodsString Object Properties

${\tt Microsoft \& JScript \& } \ big \ Method$

Applies To

String Object

Microsoft® JScript® & Operator

See Also

&= Operator

Operator Behavior

Operator Precedence

$_{\tt Microsoft \& JScript \&} << Operator$

See Also

<<= Operator

>> Operator

>>> Operator

Operator Behavior

Operator Precedence

$Microsoft @ JScript @ \sim Operator$

See Also

! Operator

Operator Behavior

Operator Precedence

See Also

= Operator

Operator Behavior

Operator Precedence

$_{Microsoft @ JScript @} >> Operator$

See Also

<< Operator

>>= Operator

>>> Operator

Operator Behavior

Operator Precedence

Language Reference

See Also

String Object Methods
String Object Properties

Microsoft® JScript® blink Method

Language Reference

Applies To

String Object

Language Reference

See Also

<u>italics Method</u>
<u>String Object Methods</u>
<u>String Object Properties</u>

Microsoft® JScript® bold Method

Language Reference

Applies To

String Object

${\tt Microsoft @ JScript @} \ Boolean \ Object$

Language Reference

See Also

new Operator var Statement

${\tt Microsoft \& JScript \& } Boolean \ Object$

Methods

Members of Boolean.prototype

toString Method valueOf Method

Nonmembers of Boolean.prototype

The **Boolean** object has no methods that are not part of the prototype.

${\tt Microsoft @ JScript @} \ Boolean \ Object$

Properties

Members of Boolean.prototype

constructor Property

Nonmembers of Boolean.prototype

prototype Property

See Also

continue Statement do...while Statement for Statement for...in Statement Labeled Statement while Statement



Statement See Also

Language Reference

Conditional Compilation
Conditional Compilation Variables
@if Statement
@set Statement

Microsoft® JScript® ceil Method

Language Reference

Applies To

Math Object

Microsoft® JScript® charAt Method

Language Reference

See Also

String Object Methods
String Object Properties

Microsoft® JScript® charAt Method

Language Reference

Applies To

String Object

Microsoft® JScript® charCodeAt

Method See Also

Language Reference

fromCharCode Method String Object Methods

Microsoft® JScript® charCodeAt

Method Applies To

Language Reference

String Object

${\tt Microsoft @ JScript @}, \ Operator$

See Also

<u>for Statement</u>

Operator Behavior

Operator Precedence

${\tt Microsoft @ JScript @ } Comparison$

Operators See Also

Language Reference

Operator Behavior
Operator Precedence
Operator Summary

${\tt Microsoft @ JScript @ } \boldsymbol{Compile \ Method}$

Language Reference

See Also

Regular Expression Object Methods

Regular Expression Object Properties

Regular Expression Syntax

${\tt Microsoft @ JScript @ } \boldsymbol{Compile \ Method}$

Language Reference

Applies To

Regular Expression Object

Description

Used to increment a variable by a specified amount.

Syntax

result += expression

The += operator syntax has these parts:

Part	Description
result	Any <u>variable.</u>
expression	Any <u>expression</u> .

Remarks

Using this operator is exactly the same as specifying:

The underlying subtype of the expressions determines the behavior of the += operator.

If	Then
Both expressions are numeric or Boolean	Add.

Both expressions are strings	Concatenate.
One expression is numeric and	Concatenate.
the other is a string	Concatenate.

For information on when a <u>run-time error</u> is generated by the += operator, see the <u>Operator Behavior</u> table.

Description

Used to perform a bitwise AND on an expression.

Syntax

result **&=** expression

The **&=** operator syntax has these parts:

Part	Description
result	Any <u>variable</u> .
expression	Any <u>expression</u> .

Remarks

Using this operator is exactly the same as specifying:

The **&=** operator looks at the binary representation of the values of *result* and *expression* and does a bitwise AND operation on them. The output of this operation behaves like this:

0101 (result)1100 (expression)

0100 (output)

Any time both of the expressions have a 1 in a digit, the result has a 1 in that digit. Otherwise, the result has a 0 in that digit.

For information on when a <u>run-time error</u> is generated by the **&=** operator, see the <u>Operator Behavior</u> table.

Description

Used to perform a bitwise OR on an expression.

Syntax

result |= expression

The |= operator syntax has these parts:

Part	Description
result	Any <u>variable</u> .
expression	Any <u>expression</u> .

Remarks

Using this operator is exactly the same as specifying:

The |= operator looks at the binary representation of the values of *result* and *expression* and does a bitwise OR operation on them. The result of this operation behaves like this:

0101 (result)1100 (expression)

1101 (output)

Any time either of the expressions has a 1 in a digit, the result has a 1 in that digit. Otherwise, the result has a 0 in that digit.

For information on when a <u>run-time error</u> is generated by the |= operator, see the <u>Operator Behavior</u> table.

Description

Used to perform a bitwise exclusive OR on an expression.

Syntax

result ^= expression

The ^= operator syntax has these parts:

Part	Description
result	Any <u>variable</u> .
expression	Any <u>expression</u> .

Remarks

Using the ^= operator is exactly the same as specifying:

The ^= operator looks at the binary representation of the values of two expressions and does a bitwise exclusive OR operation on them. The result of this operation behaves as follows:

0101 (result)1100 (expression)

1001 (result)

When one, and only one, of the expressions has a 1 in a digit, the result has a 1 in that digit. Otherwise, the result has a 0 in that digit.

For information on when a <u>run-time error</u> is generated by the ^= operator, see the <u>Operator Behavior</u> table.

Description

Used to divide a variable by an expression.

Syntax

result /= expression

The /= operator syntax has these parts:

Part	Description
result	Any numeric <u>variable</u> .
expression	Any numeric <u>expression</u> .

Remarks

Using the /= operator is exactly the same as specifying:

For information on when a <u>run-time error</u> is generated by the /= operator, see the <u>Operator Behavior</u> table.

<= operator"> <<= operator; Left Shift operator; compound assignment operators; operators; shifting bits">

Language Reference Version 1

See Also

Description

Used to shift the bits of an expression to the left.

Syntax

result <<= expression

The <<= operator syntax has these parts:

Part	Description
result	Any <u>variable</u> .
expression	Any <u>expression</u> .

Remarks

Using the <<= operator is exactly the same as specifying:

The <<= operator shifts the bits of *result* left by the number of bits specified in *expression*. For example:

var temp

The variable *temp* has a value of 56 because 14 (00001110 in binary) shifted left two bits equals 56 (00111000 in binary). Bits are filled in with zeroes when shifting.

For information on when a <u>run-time error</u> is generated by the <<= operator, see the <u>Operator Behavior</u> table.

Description

Used to divide two numbers and return only the remainder.

Syntax

result %= expression

The **%=** operator syntax has these parts:

Part	Description
result	Any <u>variable</u> .
expression	Any <u>numeric expression</u> .

Remarks

Using the **%=** operator is exactly the same as specifying:

For information on when a <u>run-time error</u> is generated by the **%=** operator, see the <u>Operator Behavior</u> table.

Description

Used to multiply a number by another number.

Syntax

result ***=** expression

The ***=** operator syntax has these parts:

Part	Description
result	Any <u>variable</u> .
expression	Any <u>expression</u> .

Remarks

Using the ***=** operator is exactly the same as specifying:

For information on when a <u>run-time error</u> is generated by the *= operator, see the <u>Operator Behavior</u> table.

Description

Used to shift the bits of an expression to the right, preserving sign.

Syntax

result >>= expression

The >>= operator syntax has these parts:

Part	Description
result	Any <u>variable</u> .
expression	Any <u>expression</u> .

Remarks

Using the >>= operator is exactly the same as specifying:

The >>= operator shifts the bits of *result* right by the number of bits specified in *expression*. The sign bit of *result* is used to fill the digits from the left. Digits shifted off the right are discarded. For example, after the following code is evaluated, *temp* has a value of -4: 14 (11110010 in binary) shifted right two bits equals -4 (11111100 in binary).

```
var temp
temp = -14
temp >>= 2
```

For information on when a <u>run-time error</u> is generated by the >>= operator, see the <u>Operator Behavior</u> table.

Description

Used to subtract the value of an expression from a variable.

Syntax

result -= expression

The -= operator syntax has these parts:

Part	Description
result	Any numeric <u>variable</u> .
expression	Any <u>numeric expression</u> .

Remarks

Using the -= operator is exactly the same as doing the following:

For information on when a <u>run-time error</u> is generated by the - operator, see the <u>Operator Behavior</u> table.

Description

Used to make an unsigned right shift of the bits in a variable.

Syntax

The >>>= operator syntax has these parts:

Part	Description
result	Any <u>variable</u> .
expression	Any <u>expression</u> .

Remarks

Using the >>>= operator is exactly the same as doing the following:

The >>>= operator shifts the bits of *result* right by the number of bits specified in *expression*. Zeroes are filled in from the left. Digits shifted off the right are discarded. For example:

var temp

temp =
$$-14$$

temp >>>= 2

For information on when a <u>run-time error</u> is generated by the >>>= operator, see the <u>Operator Behavior</u> table.

(Array) See Also Language Reference

concat Method (String) join Method

(Array) Applies To

Language Reference

Array Object String Object

(String) See Also

Language Reference

Addition Operator (+)
concat Method (Array)
String Object Methods

(String) Applies To

Language Reference

Array Object String Object

Microsoft® JScript® Conditional

Compilation See Also

Language Reference

Conditional Compilation Variables

@cc_on Statement

@if Statement

@set Statement

Microsoft® JScript® Conditional Compilation Variables See Also

Language Reference

Conditional Compilation

@cc_on Statement

@if Statement

@set Statement

Microsoft® JScript® **?: Operator**

Language Reference

See Also

if...else Statement

Operator Behavior

Operator Precedence

Operator Summary

Microsoft® JScript® CONSTRUCTOR

Property

See Also

Language Reference

prototype Property

Microsoft® JScript® CONSTRUCTOT

Property

Applies To

Language Reference

Array Object

Boolean Object

Date Object

Function Object

Math Object

Number Object

Object Object

String Object

Microsoft® JScript® **Continue**

Statement See Also

Language Reference

break Statement
do...while Statement
for Statement
for...in Statement
Labeled Statement
while Statement

Language Reference

Applies To

Math Object

${\tt Microsoft @ JScript @ } \pmb{Date Object}$

Language Reference

See Also

new Operator var Statement

Properties

constructor Property prototype Property

$_{\mbox{\scriptsize Microsoft}\mbox{\scriptsize \$}\mbox{\scriptsize JScript}\mbox{\scriptsize \$}}$ ++ and --

Operators See Also

Language Reference

Operator Behavior
Operator Precedence
Operator Summary

${\tt Microsoft @ JScript @ } \ description$

Property See Also

Language Reference

<u>number Property</u>

${\tt Microsoft @ JScript @ } \ description$

Property

Applies To

Error Object

Language Reference

Microsoft® JScript® dimensions

Method See Also

Language Reference

getItem Method lbound Method toArray Method ubound Method

Microsoft® JScript® dimensions

Method Applies To

Language Reference

VBArray Object

Microsoft® JScript® do...while

Statement See Also

Language Reference

break Statement
continue Statement
for Statement
for...in Statement
while Statement
Labeled Statement

${\tt Microsoft @ JScript @ } E \ Property$

Applies To

Math Object

Microsoft® JScript® Enumerator

Object See Also

Language Reference

Drives Collection
Files Collection
Folders Collection

Microsoft® JScript® Enumerator

Object Methods

Language Reference

atEnd Method item Method moveFirst Method moveNext Method

Microsoft® JScript® Enumerator

Object Properties

Language Reference

The **Enumerator** object has no properties.

${\tt Microsoft @ JScript @ } Error \ Object$

Language Reference

See Also

new Operator throw Statement try...catch Statement var Statement

Properties

description Property number Property

${\tt Microsoft @ JScript @} \ escape \ Method$

Language Reference

See Also

String Object unescape Method

${\tt Microsoft \& JScript \& } escape \ Method$

Language Reference

Applies To

Global Object

See Also

String Object

Microsoft® JScript® eval Method

Language Reference

Applies To

Global Object

Microsoft® JScript® exec Method

Language Reference

See Also

RegExp Object

Regular Expression Object Methods

Regular Expression Object Properties

Regular Expression Syntax

Microsoft® JScript® exec Method

Language Reference

Applies To

Regular Expression Object

${\tt Microsoft \& JScript \& } exp\ Method$

Language Reference

Applies To

Math Object

Microsoft® JScript® fixed Method

See Also

String Object Methods
String Object Properties

Applies To

String Object

Microsoft® JScript® floor Method

Applies To

Math Object

Microsoft® JScript® fontcolor Method

Language Reference

See Also

fontsize Method
String Object Methods
String Object Properties

Microsoft® JScript® fontcolor Method

Language Reference

Applies To

String Object

Microsoft® JScript® fontsize Method

Language Reference

See Also

fontcolor Method String Object Methods String Object Properties

Microsoft® JScript® fontsize Method

Language Reference

Applies To

String Object

Microsoft® JScript® for Statement

See Also

for...in Statement while Statement

See Also

for Statement while Statement

Microsoft® JScript® from CharCode

Method See Also

Language Reference

charCodeAt Method
String Object Methods

Microsoft® JScript® from CharCode

Method Applies To

Language Reference

String Object

${\tt Microsoft @ JScript @ } Function \ Object$

Language Reference

See Also

function Statement new Operator var Statement

${\tt Microsoft @ JScript @} \ Function \ Object$

Methods

Members of Function.prototype

toString Method valueOf Method

Nonmembers of Function.prototype

The **Function** object has no methods that are not part of the prototype.

${\tt Microsoft \& JScript \& } Function \ Object$

Properties

Members of Function.prototype

arguments Property caller Property

constructor Property

Nonmembers of Function.prototype

prototype Property

Language Reference

See Also

dimensions Method lbound Method toArray Method ubound Method

${\tt Microsoft @ JScript @ } \ get Item \ Method$

Language Reference

Applies To

VBArray Object

${\tt Microsoft \& JScript \& } \ GetObject$

Function See Also

Language Reference

ActiveXObject Object



See Also

Conditional Compilation
Conditional Compilation Variables
@cc_on Statement
@set Statement

See Also

Conditional Operator (?:)

Microsoft® JScript® indexOf Method

Language Reference

Applies To

String Object

Language Reference

See Also

isFinite Method
NaN Property (Global)
parseFloat Method
parseInt Method

Microsoft® JScript® isNaN Method

Language Reference

Applies To

Global Object

Microsoft® JScript® italics Method

Language Reference

See Also

bold Method String Object Methods String Object Properties

Microsoft® JScript® italics Method

Language Reference

Applies To

String Object

See Also

atEnd Method moveFirst Method moveNext Method

Microsoft® JScript® item Method

Language Reference

Applies To

Enumerator Object

See Also

Array Object Methods
String Object

Applies To

Array Object

Microsoft® JScript® Labeled

Statement See Also

Language Reference

break Statement continue Statement

Microsoft® JScript® lastIndexOf

Method See Also

Language Reference

indexOf Method String Object Methods String Object Properties

Microsoft® JScript® lastIndexOf

Method

Language Reference

Applies To

String Object

Microsoft® JScript® lbound Method

See Also

dimensions Method getItem Method toArray Method ubound Method

Applies To

VBArray Object

(Array) See Also

Language Reference

length Property (Function)
length Property (String)

(Array)Applies To

Language Reference

Array Object

(Function) See Also

Language Reference

arguments Property length Property (Array) length Property (String)

(Function) Applies To

Language Reference

Function Object

(String) See Also

Language Reference

length Property (Array)
length Property (Function)
String Object Methods
String Object Properties

(String)

Applies To

Language Reference

String Object

Microsoft® JScript® link Method

See Also

anchor MethodString Object MethodsString Object Properties

Microsoft® JScript® link Method

Applies To

String Object

Microsoft® JScript® LN2 Property

Language Reference

Applies To

Microsoft® JScript® LN10 Property

Language Reference

Applies To

${\tt Microsoft \& JScript \& } \ log \ Method$

Applies To

Microsoft® JScript® LOG2E Property

Language Reference

Applies To

Microsoft® JScript® Log10E Property

Language Reference

Applies To

See Also

Number Object

Microsoft® JScript® max Method

Language Reference

Applies To

Applies To

Microsoft® JScript® moveFirst

Method See Also

Language Reference

atEnd Method item Method moveNext Method

Microsoft® JScript® moveFirst

Method Applies To

Language Reference

Enumerator Object

Microsoft® JScript® moveNext

Method See Also

Language Reference

atEnd Method item Method moveFirst Method

Microsoft® JScript® moveNext

Method Applies To

Language Reference

Enumerator Object

${\tt Microsoft @ JScript @} \ \, \pmb{NaN \ Property}$

Language Reference

Applies To

Number Object

See Also

function Statement

Microsoft® JScript® number Property

Language Reference

See Also

description Property

Microsoft® JScript® number Property

Language Reference

Applies To

Error Object

Language Reference

See Also

Function Object Global Object

Microsoft® JScript® Object Object

Methods

Members of Object.prototype

toString Method valueOf Method

Non-members of Object.prototype

The **Object** object has no methods that are not part of the prototype.

${\tt Microsoft \& JScript \& } \begin{picture}(2000)(2000) \put(0.000)(0.000) \put(0.000)(0.000)(0.000) \put(0.000)(0.000)(0.000) \put(0.000)(0.000)(0.000)(0.000) \put(0.000)(0.000)(0.000)(0.000) \put(0.000)(0.000)(0.000)(0.000) \put(0.000)(0.000)(0.000)(0.000)(0.000) \put(0.000)(0.000)(0.000)(0.000)(0.000) \put(0.000)(0.000)(0.000)(0.000)(0.000) \put(0.000)(0.000)(0.000)(0.000)(0.000) \put(0.000)(0.000)(0.000)(0.000)(0.000) \put(0.000)(0.000)(0.000)(0.000)(0.000) \put(0.000)(0.000)(0.000)(0.000)(0.000) \put(0.000)(0.000)(0.000)(0.000)(0.000) \put(0.000)(0.000)(0.000)(0.000)(0.000) \put(0.000)(0.000)(0.000)(0.000)(0.000)(0.000) \put(0.000)(0.000)(0.000)(0.000)(0.000) \put(0.000)(0.000)(0.000)(0.000)(0.000) \put(0.000)(0.000)(0.000)(0.000)(0.000) \put(0.000)(0.000)(0.000)(0.000)(0.000) \put(0.000)(0.000)(0.000)(0.000)(0.000) \put(0.000)(0.000)(0.000)(0.000)(0.000)(0.000)(0.000) \put(0.000)(0.000)(0.000)(0.000)(0.000)(0.000) \put(0.000)(0.000)(0.000)(0.000)(0.000)(0.000)(0.000) \put(0.000)(0.0$

Properties

Members of Object.prototype

prototype Property
constructor Property

Non-members of Object.prototype

The **Object** object has no properties that are not part of the prototype.

Microsoft® JScript® PI Property

Applies To

${\tt Microsoft \& JScript \& } \ pow \ Method$

Language Reference

Applies To

${\tt Microsoft @ JScript @ } {\color{red} prototype}$

Property

See Also

Language Reference

constructor Property

${\tt Microsoft @ JScript @ } {\color{red} prototype}$

Property

Applies To

Language Reference

Array Object

Boolean Object

Date Object

Function Object

Number Object

Object Object

String Object

Microsoft® JScript® random Method

Language Reference

Applies To

Microsoft® JScript® return Statement

Language Reference

See Also

function Statement

Microsoft® JScript® reverse Method

Language Reference

See Also

Array Object Methods

Microsoft® JScript® reverse Method

Language Reference

Applies To

Array Object

Microsoft® JScript® round Method

Language Reference

Applies To

${\tt Microsoft \& JScript \& ScriptEngine}$

Function See Also

Language Reference

ScriptEngineBuildVersion Function ScriptEngineMajorVersion Function ScriptEngineMinorVersion Function Microsoft® JScript®

ScriptEngineBuildVersion Function

Language Reference

See Also

ScriptEngine Function ScriptEngineMajorVersion Function ScriptEngineMinorVersion Function Microsoft® JScript®

ScriptEngineMajorVersion Function

Language Reference

See Also

ScriptEngine Function
ScriptEngineBuildVersion Function
ScriptEngineMinorVersion Function

Microsoft® JScript®

ScriptEngineMinorVersion Function

Language Reference

See Also

ScriptEngine Function
ScriptEngineBuildVersion Function
ScriptEngineMajorVersion Function

Microsoft® JScript® sin Method

Applies To

See Also

slice Method (String)

Microsoft® JScript® slice Method

Language Reference

Applies To

Array Object String Object

Microsoft® JScript® slice Method

(String) See Also

Language Reference

slice Method (Array)String Object Methods

Microsoft® JScript® slice Method

(String) Applies To

Language Reference

Array Object String Object

Language Reference

See Also

big Method
String Object Methods
String Object Properties

Microsoft® JScript® small Method

Language Reference

Applies To

String Object

Microsoft® JScript® sort Method

Language Reference

See Also

Array Object Methods

Microsoft® JScript® sort Method

Language Reference

Applies To

Array Object

${\tt Microsoft @ JScript @ } \textbf{Source Property}$

Language Reference

See Also

Regular Expression Object Methods

Regular Expression Object Properties

Regular Expression Syntax

Microsoft® JScript® Source Property

Language Reference

Applies To

Regular Expression Object

${\tt Microsoft \& JScript \& } \ sqrt \ Method$

Language Reference

Applies To

${\tt Microsoft @ JScript @ } SQRT1_2$

Property

Applies To

Language Reference

Microsoft® JScript® SQRT2 Property

Language Reference

Applies To

Microsoft® JScript® strike Method

Language Reference

See Also

String Object Methods
String Object Properties

Microsoft® JScript® strike Method

Language Reference

Applies To

String Object

See Also

new Operator

${\tt Microsoft \& JScript \& } String \ Object$

Properties

Members of String.prototype

constructor Property

Nonmembers of String.prototype

length Property prototype Property

See Also

String Object Methods
String Object Properties
sup Method

Microsoft® JScript® sub Method

Language Reference

Applies To

Microsoft® JScript® substr Method

Language Reference

See Also

String Object Methods
String Object Properties
substring Method

Microsoft® JScript® substr Method

Language Reference

Applies To

${\tt Microsoft \& JScript \& } \ substring \ Method$

Language Reference

See Also

String Object Methods
String Object Properties
substr Method

${\tt Microsoft @ JScript @ } {\color{red} {\bf Substring \ Method}}$

Language Reference

Applies To

See Also

String Object Methods
String Object Properties
sub Method

${\tt Microsoft \& JScript \& } sup \ Method$

Language Reference

Applies To

Microsoft® JScript® switch Statement

Language Reference

See Also

break Statement if...else Statement

Applies To

Math Object

Microsoft® JScript® test Method

Language Reference

See Also

RegExp Object

Regular Expression Object Methods

Regular Expression Object Properties

Regular Expression Syntax

Microsoft® JScript® test Method

Language Reference

Applies To

Regular Expression Object

See Also

new Operator

Language Reference

See Also

dimensions Method getItem Method lbound Method ubound Method

${\tt Microsoft @ JScript @ } to Array\ Method$

Language Reference

Applies To

VBArray Object

Microsoft® JScript® toLowerCase

Method See Also

Language Reference

String Object Methods
String Object Properties
toUpperCase Method

Microsoft® JScript® toLowerCase

Method

Language Reference

Applies To

See Also

<u>function Statement</u>

${\tt Microsoft \& JScript \& } \ to String \ Method$

Language Reference

Applies To

Array Object

Boolean Object

Function Object

Number Object

Object Object

${\tt Microsoft @ JScript @ } to Upper Case$

Method See Also

Language Reference

String Object Methods
String Object Properties
toLowerCase Method

${\tt Microsoft @ JScript @ } to Upper Case$

Method

Language Reference

Applies To

Language Reference

See Also

Operator Behavior Operator Precedence

Operator Summary

Language Reference

See Also

dimensions Method getItem Method lbound Method toArray Method

Microsoft® JScript® ubound Method

Language Reference

Applies To

VBArray Object

${\tt Microsoft @ JScript @ } unescape \ Method$

Language Reference

See Also

escape Method String Object

${\tt Microsoft @ JScript @ } unescape \ Method$

Language Reference

Applies To

Global Object

Microsoft® JScript® >>> Operator

See Also

>>>= <u>Operator</u>

<< Operator

>> Operator

Operator Behavior

Operator Precedence

Operator Summary

Microsoft® JScript® valueOf Method

Language Reference

See Also

toString Method

Microsoft® JScript® valueOf Method

Language Reference

Applies To

Array Object

Boolean Object

Date Object

Function Object

Number Object

Object Object

Microsoft® JScript® var Statement

Language Reference

See Also

function Statement new Operator

See Also

Array Object

Language Reference

Methods

dimensions Method getItem Method lbound Method toArray Method ubound Method

${\tt Microsoft \& JScript \& } \ VBArray \ Object$

Properties

The **VBArray** object has no properties.

Language Reference

See Also

Operator Behavior Operator Precedence

Operator Summary

Microsoft® JScript® while Statement

See Also

break Statement
continue Statement
do...while Statement
for Statement
for...in Statement

See Also

this Statement

Microsoft® JScript® isFinite Method

Language Reference

See Also

isNaN Method

Microsoft® JScript® isFinite Method

Language Reference

Applies To

Global Object

Microsoft® JScript® search Method

Language Reference

See Also

exec Method
match Method
replace Method
String Object Methods
test Method

Microsoft® JScript® search Method

Language Reference

Applies To

String Object

${\tt Microsoft @ JScript @} \ \, \boldsymbol{delete \ Operator}$

Language Reference

See Also

Operator Behavior Operator Precedence

Operator Summary

Microsoft® JScript® Add Method

(Dictionary) See Also

Scripting Run-Time Reference

Add Method (Folders)

Exists Method

Items Method

Keys Method

Remove Method

RemoveAll Method

${\tt Microsoft @ JScript @} \ Add \ Method$

(Dictionary)

Applies To

Scripting Run-Time Reference

Dictionary Object

Microsoft® JScript® Add Method

(Folders)

See Also

Scripting Run-Time Reference

Add Method (Dictionary)

Microsoft® JScript® Add Method

(Folders)

Applies To

Scripting Run-Time Reference

Folders Collection

Microsoft® JScript® AtEndOfLine

Property See Also

Scripting Run-Time Reference

AtEndOfStream Property

Microsoft® JScript® AtEndOfLine

Property

Applies To

Scripting Run-Time Reference

TextStream Object

Microsoft® JScript® AtEndOfStream

Property See Also

Scripting Run-Time Reference

AtEndOfLine Property

Microsoft® JScript® AtEndOfStream

Property

Applies To

Scripting Run-Time Reference

TextStream Object

Microsoft® JScript® Attributes

Scripting Run-Time Reference

Property See Also

DateCreated Property

DateLastAccessed Property

DateLastModified Property

Drive Property

Files Property

IsRootFolder Property

Name Property

ParentFolder Property

Path Property

ShortName Property

ShortPath Property

Size Property

SubFolders Property

Type Property

Microsoft® JScript® Attributes

Scripting Run-Time Reference

Property Applies To

<u>File Object</u> <u>Folder Object</u>

${\tt Microsoft @ JScript @} \ Available Space$

Scripting Run-Time Reference

Property See Also

DriveLetter Property

DriveType Property

FileSystem Property

FreeSpace Property

IsReady Property

Path Property

RootFolder Property

SerialNumber Property

ShareName Property

TotalSize Property

VolumeName Property

${\tt Microsoft @ JScript @} \ Available Space$

Scripting Run-Time Reference

Property Applies To

Drive Object

Microsoft® JScript® BuildPath

Method See Also

Scripting Run-Time Reference

GetAbsolutePathName Method

GetBaseName Method

GetDriveName Method

GetExtensionName Method

GetFileName Method

GetParentFolderName Method

GetTempName Method

${\tt Microsoft @ JScript @ } BuildPath$

Method Applies To

Scripting Run-Time Reference

FileSystemObject Object

Microsoft® JScript® Close Method

Scripting Run-Time Reference

See Also

Read Method Write Method

${\tt Microsoft \& JScript \& } \boldsymbol{Close} \,\, \boldsymbol{Method}$

Scripting Run-Time Reference

Applies To

TextStream Object

Microsoft® JScript® Column Property

Scripting Run-Time Reference

See Also

Line Property

Microsoft® JScript® Column Property

Scripting Run-Time Reference

Applies To

TextStream Object

${\tt Microsoft @ JScript @ } Copy \ Method$

Scripting Run-Time Reference

See Also

CopyFile Method
CopyFolder Method
Delete Method
Move Method
OpenAsTextStream Method

Microsoft® JScript® Copy Method

Scripting Run-Time Reference

Applies To

<u>File Object</u> <u>Folder Object</u>

${\tt Microsoft @ JScript @ } CopyFile\ Method$

Scripting Run-Time Reference

See Also

Copy Method
CopyFolder Method
CreateTextFile Method
DeleteFile Method
MoveFile Method

${\tt Microsoft \& JScript \& } \ CopyFile \ Method$

Scripting Run-Time Reference

Applies To

FileSystemObject Object

${\tt Microsoft @ JScript @ } CopyFolder$

Method See Also

Scripting Run-Time Reference

CopyFile Method

Copy Method

CreateFolder Method

DeleteFolder Method

MoveFolder Method

${\tt Microsoft @ JScript @ } CopyFolder$

Method Applies To

Scripting Run-Time Reference

FileSystemObject Object

Microsoft® JScript® Count Property

Scripting Run-Time Reference

See Also

CompareMode Property
Item Property
Key Property

${\tt Microsoft @ JScript @} \ \ \pmb{Count\ Property}$

Scripting Run-Time Reference

Applies To

Dictionary Object

Drives Collection

Files Collection

Folders Collection

Microsoft® JScript® CreateFolder

Method See Also

Scripting Run-Time Reference

CopyFolder Method
DeleteFolder Method
MoveFolder Method

Microsoft® JScript® CreateFolder

Method Applies To

Scripting Run-Time Reference

FileSystemObject Object

Microsoft® JScript® CreateTextFile

Method See Also

Scripting Run-Time Reference

CreateFolder Method
OpenAsTextStream Method
OpenTextFile Method

Microsoft® JScript® CreateTextFile

Method

Scripting Run-Time Reference

Applies To

FileSystemObject Object Folder Object

Microsoft® JScript® DateCreated

Scripting Run-Time Reference

Property See Also

Attributes Property

DateLastAccessed Property

DateLastModified Property

Drive Property

Files Property

IsRootFolder Property

Name Property

ParentFolder Property

Path Property

ShortName Property

ShortPath Property

Size Property

SubFolders Property

Type Property

Microsoft® JScript® DateCreated

Scripting Run-Time Reference

Property Applies To

<u>File Object</u> <u>Folder Object</u>

DateLastAccessed Property See Also

Scripting Run-Time Reference

Attributes Property

DateCreated Property

DateLastModified Property

Drive Property

Files Property

IsRootFolder Property

Name Property

ParentFolder Property

Path Property

ShortName Property

ShortPath Property

Size Property

SubFolders Property

Type Property

DateLastAccessed Property Applies To

Scripting Run-Time Reference

File Object Folder Object

DateLastModified Property See Also

Scripting Run-Time Reference

Attributes Property

DateCreated Property

DateLastAccessed Property

Drive Property

Files Property

IsRootFolder Property

Name Property

ParentFolder Property

Path Property

ShortName Property

ShortPath Property

Size Property

SubFolders Property

Type Property

DateLastModified Property Applies To

Scripting Run-Time Reference

File Object Folder Object

Microsoft® JScript® Delete Method

Scripting Run-Time Reference

See Also

Copy Method
DeleteFile Method
DeleteFolder Method
Move Method
OpenAsTextStream Method

Microsoft® JScript® Delete Method

Scripting Run-Time Reference

Applies To

<u>File Object</u> <u>Folder Object</u>

Microsoft® JScript® DeleteFile

Method See Also

Scripting Run-Time Reference

CopyFile Method
CreateTextFile Method
Delete Method
DeleteFolder Method
MoveFile Method

Microsoft® JScript® DeleteFile

Method Applies To

Scripting Run-Time Reference

FileSystemObject Object

Microsoft® JScript® DeleteFolder

Method See Also

Scripting Run-Time Reference

CopyFolder Method CreateFolder Method Delete Method DeleteFile Method MoveFolder Method

Microsoft® JScript® DeleteFolder

Method Applies To

Scripting Run-Time Reference

FileSystemObject Object

${\tt Microsoft @ JScript @ } Dictionary$

Object

Methods

Scripting Run-Time Reference

Add Method (Dictionary)

Exists Method

Items Method

Keys Method

Remove Method

RemoveAll Method

${\tt Microsoft @ JScript @ } \ \, Drive \ \, Object$

Scripting Run-Time Reference

See Also

Drives Collection

File Object

Files Collection

Folder Object

Folders Collection

GetDrive Method

Microsoft® JScript® Drive Object

Scripting Run-Time Reference

Properties

AvailableSpace Property

DriveLetter Property

DriveType Property

FileSystem Property

FreeSpace Property

IsReady Property

Path Property

RootFolder Property

SerialNumber Property

ShareName Property

TotalSize Property

VolumeName Property

Methods

The **Drive** object has no methods.

Microsoft® JScript® **Drive Property**

Scripting Run-Time Reference

See Also

Attributes Property

DateCreated Property

DateLastAccessed Property

DateLastModified Property

Files Property

IsRootFolder Property

Name Property

ParentFolder Property

Path Property

ShortName Property

ShortPath Property

Size Property

SubFolders Property

Type Property

Microsoft® JScript® **Drive Property**

Scripting Run-Time Reference

Applies To

<u>File Object</u> <u>Folder Object</u>

Microsoft® JScript® DriveExists

Method See Also

Scripting Run-Time Reference

Drive Object

Drives Collection

FileExists Method

FolderExists Method

GetDrive Method

GetDriveName Method

IsReady Property

Microsoft® JScript® DriveExists

Method Applies To

Scripting Run-Time Reference

FileSystemObject Object

Microsoft® JScript® DriveLetter

Scripting Run-Time Reference

Property See Also

AvailableSpace Property

DriveType Property

FileSystem Property

FreeSpace Property

IsReady Property

Path Property

RootFolder Property

SerialNumber Property

ShareName Property

TotalSize Property

VolumeName Property

Microsoft® JScript® DriveLetter

Scripting Run-Time Reference

Property Applies To

Drive Object

Microsoft® JScript® **Drives Collection**

Scripting Run-Time Reference

See Also

Drive Object

Drives Property

File Object

Files Collection

Folder Object

Folders Collection

Microsoft® JScript® Drives Collection

Scripting Run-Time Reference

Properties

Count Property
Item Property

Microsoft® JScript® Drives Collection

Scripting Run-Time Reference

Methods

The **Drives** collection has no methods.

Microsoft® JScript® **Drives Property**

Scripting Run-Time Reference

See Also

Drives Collection
Files Property
SubFolders Property

Microsoft® JScript® **Drives Property**

Scripting Run-Time Reference

Applies To

FileSystemObject Object

Microsoft® JScript® DriveType

Scripting Run-Time Reference

Property See Also

AvailableSpace Property

DriveLetter Property

FileSystem Property

FreeSpace Property

IsReady Property

Path Property

RootFolder Property

SerialNumber Property

ShareName Property

TotalSize Property

VolumeName Property

Scripting Run-Time Reference

Property Applies To

Drive Object

Microsoft® JScript® Exists Method

Scripting Run-Time Reference

See Also

Add Method (Dictionary)

Items Method

Keys Method

Remove Method

RemoveAll Method

Microsoft® JScript® Exists Method

Scripting Run-Time Reference

Applies To

Dictionary Object

${\tt Microsoft @ JScript @ } File \ Object$

Scripting Run-Time Reference

See Also

Drive Object

Drives Collection

Files Collection

Folder Object

Folders Collection

Scripting Run-Time Reference

Microsoft® JScript® File Object

Properties

Attributes Property

DateCreated Property

DateLastAccessed Property

DateLastModified Property

Drive Property

Name Property

ParentFolder Property

Path Property

ShortName Property

ShortPath Property

Size Property

Type Property

${\tt Microsoft @ JScript @ } File \ Object$

Scripting Run-Time Reference

Methods

Copy Method
Delete Method
Move Method
OpenAsTextStream Method

Microsoft® JScript® FileExists

Method See Also

Scripting Run-Time Reference

DriveExists Method
FolderExists Method
GetFile Method
GetFileName Method

Microsoft® JScript® FileExists

Method Applies To

Scripting Run-Time Reference

FileSystemObject Object

Microsoft® JScript® Files Collection

Scripting Run-Time Reference

See Also

Drive Object
Drives Collection
File Object
Folder Object
Folders Collection

Microsoft® JScript® Files Collection

Scripting Run-Time Reference

Properties

Count Property
Item Property

Microsoft® JScript® Files Collection

Scripting Run-Time Reference

Methods

The **Files** collection has no methods.

Microsoft® JScript® Version Information

The following table lists the version of Microsoft JScript implemented by host applications.

Host Application		JScript Version						
Host Application	1.0	2.0	3.0	4.0	5.0			
Microsoft Internet Explorer 3.0	X							
Microsoft Internet Information Server 1.0		X						
Microsoft Internet Explorer 4.0			X					
Microsoft Internet Information Server 4.0			x					
Microsoft Windows Scripting Host 1.0			X					
Microsoft Visual Studio 6.0				X				
Microsoft Internet Explorer 5.0					$oxed{x}$			
Microsoft Internet Information Services 5.0					X			

The following table lists JScript language features and the version when first introduced.

Language Element	Version First Introduced						
Language Element	1.0	2.0	3.0	4.0	5.0		
\$1\$9 Properties			X				
abs Method	X						
acos Method	X						
ActiveXObject Object			X				
Addition Operator (+)	X						
anchor Method	X						
			ĺ	ĺ			

arguments Property		x		
Array Object		Х		
asin Method	Х			
Assignment Operator (=)	Х			
atan Method	Х			
atan2 Method	х			
atEnd Method			X	
big Method	х			
Bitwise AND Operator (&)	X			
Bitwise Left Shift Operator (<<)	х			
Bitwise NOT Operator (~)	х			
Bitwise OR Operator ()	Х			
Bitwise Right Shift Operator (>>)	х			
Bitwise XOR Operator (^)	Х			
blink Method	Х			
bold Method	Х			
Boolean Object		X		
break Statement	Х			
caller Property		X		
catch Statement				X
@cc_on Statement			X	
<u>ceil Method</u>	X			
<u>charAt Method</u>	X			
<u>charCodeAt Method</u>			X	
Comma Operator (,)	Х			
// (Single-line Comment Statement)	x			
/**/ (Multiline Comment	X			

Statement)				
Comparison Operators	х			
compile Method			X	
concat Method (Array)			X	
concat Method (String)			X	
Conditional Compilation			X	
Conditional Compilation Variables			X	
Conditional (trinary) Operator (?:)	X			
constructor Property		X		
continue Statement	X			
<u>cos Method</u>	X			
Data Type Conversion			X	
Date Object	X			
Decrement Operator ()	X			
delete Operator			X	
description Property				X
dimensions Method			X	
Division Operator (/)	X			
dowhile Statement			X	
E Property	Х			
Enumerator Object			X	
Equality Operator (==)	Х			
Error Object				X
escape Method	Х			
eval Method	Х			
exec Method			X	
exp Method	х			
fixed Method	х			

<u>floor Method</u>	X			
fontcolor Method	X			
fontsize Method	X			
for Statement	X			
forin Statement				Х
fromCharCode Method			X	
Function Object		X		
function Statement	X			
getDate Method	X			
getDay Method	X			
getFullYear Method			X	
getHours Method	X			
getItem Method			X	
getMilliseconds Method			X	
getMinutes Method	X			
getMonth Method	X			
GetObject Function			X	
getSeconds Method	X			
getTime Method	X			
getTimezoneOffset Method	X			
getUTCDate Method			X	
getUTCDay Method			X	
getUTCFullYear Method			X	
getUTCHours Method			X	
getUTCMilliseconds Method			X	
getUTCMinutes Method			X	
getUTCMonth Method			X	
getUTCSeconds Method			X	
getVarDate Method			X	

getYear Method	x			
Global Object			X	
Greater than Operator (>)	X			
Greater than or equal to Operator (>=)	X			
Identity Operator (===)	X			
@if Statement			X	
<u>ifelse Statement</u>	X			
Increment Operator (++)	X			
index Property			X	
indexOf Method	X			
Inequality Operator (!=)	X			
Infinity Property			X	
input Property			X	
instanceof Operator				X
<u>isFinite Method</u>			X	
<u>isNaN Method</u>	X			
<u>italics Method</u>	X			
<u>item Method</u>			X	
<u>join Method</u>		X		
Labeled Statement			X	
lastIndex Property			X	
<u>lastIndexOf Method</u>	X			
<u>lbound Method</u>			X	
length Property (Array)		X		
length Property (Function)		X		
length Property (String)	X			
Less than Operator (<)	X			
Less than or equal to Operator	X			

(<=)				
<u>link Method</u>	X			
LN2 Property	X			
LN10 Property	X			
log Method	X			
LOG2E Property	X			
LOG10E Property	X			
Logical AND Operator (&&)	X			
Logical NOT Operator (!)	X			
Logical OR Operator ()	X			
match Method			X	
<u>Math Object</u>	X			
max Method	X			
MAX_VALUE Property		X		
min Method	X			
MIN_VALUE Property		X		
Modulus Operator (%)	X			
moveFirst Method			X	
moveNext Method			X	
Multiplication Operator (*)	X			
NaN Property (Global)			X	
NaN Property (Number)		X		
NEGATIVE_INFINITY		X		
<u>Property</u>		, A		
<u>new Operator</u>	X			
Nonidentity Operator (!==)	X			
Number Object		X		
<u>number Property</u>				X
Object Object			X	
Operator Precedence	X			

parse Method	X			
parseFloat Method	X			
parseInt Method	X			
PI Property	X			
POSITIVE_INFINITY		***		
<u>Property</u>		X		
pow Method	X			
prototype Property		X		
random Method	X			
RegExp Object			X	
Regular Expression Object			X	
Regular Expression Syntax			X	
replace Method	X			
<u>return Statement</u>	X			
reverse Method		X		
round Method	X			
ScriptEngine Function		х		
<u>ScriptEngineBuildVersion</u>		x		
<u>Function</u>	1			
ScriptEngineMajorVersion Experien		x		
Function ScriptEngineMinerVersion				
ScriptEngineMinorVersion Function		x		
search Method			X	
@set Statement	1		X	
setDate Method	X			
setFullYear Method			Х	
setHours Method	X			
setMilliseconds Method			X	
setMinutes Method	X			
	ĺ			

setMonth Method	x			
setSeconds Method	X			
setTime Method	X			
setUTCDate Method			X	
setUTCFullYear Method			X	
setUTCHours Method			X	
setUTCMilliseconds Method			X	
setUTCMinutes Method			X	
setUTCMonth Method			X	
setUTCSeconds Method			X	
setYear Method	X			
sin Method	X			
slice Method (Array)			X	
slice Method (String)			X	
small Method	X			
sort Method		X		
source Property			X	
split Method			X	
<u>sqrt Method</u>	X			
SQRT1_2 Property	X			
SQRT2 Property	X			
strike Method	X			
String Object	X			
<u>sub Method</u>	X			
substr Method			X	
substring Method	X			
Subtraction Operator (-)	X			
sup Method	X			
switch Statement			X	
tan Method	X			

test Method			X	
this Statement	X			
throw Statement				X
toArray Method			X	
toGMTString Method	X			
toLocaleString Method	X			
toLowerCase Method	X			
toString Method		X		
toUpperCase Method	X			
toUTCString Method			X	
try Statement				X
typeof Operator	X			
ubound Method			X	
Unary Negation Operator (-)	X			
unescape Method	X			
Unsigned Right Shift Operator (>>>)	X			
UTC Method	Х			
valueOf Method		X		
var Statement	Х			
VBArray Object			X	
void Operator		X		
while Statement	Х			
with Statement	X			

Microsoft® JScript® Files Property

Scripting Run-Time Reference

See Also

Attributes Property

DateCreated Property

DateLastAccessed Property

DateLastModified Property

Drive Property

IsRootFolder Property

Name Property

ParentFolder Property

Path Property

ShortName Property

ShortPath Property

Size Property

SubFolders Property

Type Property

Microsoft® JScript® Files Property

Scripting Run-Time Reference

Applies To

Folder Object

Microsoft® JScript® FileSystemObject

Scripting Run-Time Reference

Object

Methods

BuildPath Method

CopyFile Method

CopyFolder Method

CreateFolder Method

CreateTextFile Method

DeleteFile Method

DeleteFolder Method

DriveExists Method

FileExists Method

FolderExists Method

GetAbsolutePathName Method

GetBaseName Method

GetDrive Method

GetDriveName Method

GetExtensionName Method

GetFile Method

GetFileName Method

GetFolder Method

GetParentFolderName Method

GetSpecialFolder Method

GetTempName Method

MoveFile Method

MoveFolder Method

OpenTextFile Method

${\tt Microsoft \& JScript \& } File System Object$

Object

Scripting Run-Time Reference

Properties

Drives Property

${\tt Microsoft @ JScript @ } File System$

Scripting Run-Time Reference

Property See Also

AvailableSpace Property

DriveLetter Property

DriveType Property

FreeSpace Property

IsReady Property

Path Property

RootFolder Property

SerialNumber Property

ShareName Property

TotalSize Property

VolumeName Property

Microsoft® JScript® FileSystem

Scripting Run-Time Reference

Property Applies To

Drive Object

${\tt Microsoft @ JScript @} \ Folder \ Object$

Scripting Run-Time Reference

See Also

Drive Object
Drives Collection
File Object
Files Collection
Folders Collection

Microsoft® JScript® Folder Object

Scripting Run-Time Reference

Properties

Attributes Property

DateCreated Property

DateLastAccessed Property

DateLastModified Property

Drive Property

Files Property

IsRootFolder Property

Name Property

ParentFolder Property

Path Property

ShortName Property

ShortPath Property

Size Property

SubFolders Property

Type Property

${\tt Microsoft @ JScript @} \ Folder \ Object$

Scripting Run-Time Reference

Methods

Copy Method
Delete Method
Move Method
OpenAsTextStream Method

${\tt Microsoft @ JScript @ } Folders$

Collection See Also

Scripting Run-Time Reference

Drive Object

Drives Collection

File Object

Files Collection

Folder Object

SubFolders Property

${\tt Microsoft @ JScript @ } Folders$

Collection Properties

Scripting Run-Time Reference

Count Property
Item Property

${\tt Microsoft @ JScript @ } Folders$

Collection Methods

Scripting Run-Time Reference

Add Method (Folders)

Microsoft® JScript® FolderExists

Method See Also

Scripting Run-Time Reference

DriveExists Method
FileExists Method
GetFolder Method
GetParentFolderName Method

Microsoft® JScript® FolderExists

Method Applies To

Scripting Run-Time Reference

FileSystemObject Object

${\tt Microsoft \& JScript \& FreeSpace}$

Scripting Run-Time Reference

Property See Also

AvailableSpace Property

DriveLetter Property

DriveType Property

FileSystem Property

IsReady Property

Path Property

RootFolder Property

SerialNumber Property

ShareName Property

TotalSize Property

VolumeName Property

${\tt Microsoft @ JScript @ } Free Space$

Scripting Run-Time Reference

Property Applies To

Drive Object

Microsoft® JScript®

GetAbsolutePathName Method See Also

Scripting Run-Time Reference

GetBaseName Method

GetDrive Method

GetDriveName Method

GetExtensionName Method

GetFile Method

GetFileName Method

GetFileVersion Method

GetFolder Method

GetParentFolderName Method

GetSpecialFolder Method

GetTempName Method

Microsoft® JScript®

GetAbsolutePathName Method Applies To

Scripting Run-Time Reference

FileSystemObject Object

Microsoft® JScript® GetBaseName

Method See Also

Scripting Run-Time Reference

GetAbsolutePathName Method

GetDrive Method

GetDriveName Method

GetExtensionName Method

GetFile Method

GetFileName Method

GetFileVersion Method

GetFolder Method

GetParentFolderName Method

GetSpecialFolder Method

GetTempName Method

Microsoft® JScript® GetBaseName

Method Applies To

Scripting Run-Time Reference

FileSystemObject Object

Microsoft® JScript® GetDrive Method

Scripting Run-Time Reference

See Also

GetAbsolutePathName Method

GetBaseName Method

GetDriveName Method

GetExtensionName Method

GetFile Method

GetFileName Method

GetFileVersion Method

GetFolder Method

GetParentFolderName Method

GetSpecialFolder Method

GetTempName Method

${\tt Microsoft @ JScript @} \ \, \boldsymbol{GetDrive} \,\, \boldsymbol{Method}$

Scripting Run-Time Reference

Applies To

FileSystemObject Object

Microsoft® JScript® GetDriveName

Method See Also

Scripting Run-Time Reference

GetAbsolutePathName Method

GetBaseName Method

GetDrive Method

GetExtensionName Method

GetFile Method

GetFileName Method

GetFileVersion Method

GetFolder Method

GetParentFolderName Method

GetSpecialFolder Method

GetTempName Method

Microsoft® JScript® GetDriveName

Method Applies To

Scripting Run-Time Reference

FileSystemObject Object

Microsoft® JScript®

GetExtensionName Method See Also

Scripting Run-Time Reference

GetAbsolutePathName Method

GetBaseName Method

GetDrive Method

GetDriveName Method

GetFile Method

GetFileName Method

GetFileVersion Method

GetFolder Method

GetParentFolderName Method

GetSpecialFolder Method

GetTempName Method

Microsoft® JScript®

GetExtensionName Method Applies To

Scripting Run-Time Reference

Microsoft® JScript® GetFile Method

Scripting Run-Time Reference

See Also

GetAbsolutePathName Method

GetBaseName Method

GetDrive Method

GetDriveName Method

GetExtensionName Method

GetFileName Method

GetFileVersion Method

GetFolder Method

GetParentFolderName Method

GetSpecialFolder Method

GetTempName Method

${\tt Microsoft @ JScript @ } \boldsymbol{GetFile \ Method}$

Scripting Run-Time Reference

Applies To

Microsoft® JScript® GetFileName

Method See Also

Scripting Run-Time Reference

GetAbsolutePathName Method

GetBaseName Method

GetDrive Method

GetDriveName Method

GetExtensionName Method

GetFile Method

GetFileVersion Method

GetFolder Method

GetParentFolderName Method

GetSpecialFolder Method

GetTempName Method

Microsoft® JScript® GetFileName

Method Applies To

Scripting Run-Time Reference

Microsoft® JScript® GetFileVersion

Method See Also

Scripting Run-Time Reference

GetAbsolutePathName Method

GetBaseName Method

GetDrive Method

GetDriveName Method

GetExtensionName Method

GetFile Method

GetFileName Method

GetFolder Method

GetParentFolderName Method

GetSpecialFolder Method

GetTempName Method

Microsoft® JScript® GetFileVersion

Method Applies To

Scripting Run-Time Reference

Microsoft® JScript® GetFolder

Method See Also

Scripting Run-Time Reference

GetAbsolutePathName Method

GetBaseName Method

GetDrive Method

GetDriveName Method

GetExtensionName Method

GetFile Method

GetFileName Method

GetFileVersion Method

GetParentFolderName Method

GetSpecialFolder Method

GetTempName Method

Microsoft® JScript® GetFolder

Method Applies To

Scripting Run-Time Reference

Microsoft® JScript®

GetParentFolderName Method See Also

Scripting Run-Time Reference

GetAbsolutePathName Method

GetBaseName Method

GetDrive Method

GetDriveName Method

GetExtensionName Method

GetFile Method

GetFileName Method

GetFileVersion Method

GetFolder Method

GetSpecialFolder Method

GetTempName Method

Microsoft® JScript®

GetParentFolderName Method Applies To

Scripting Run-Time Reference

${\tt Microsoft \& JScript \& } \ GetSpecial Folder$

Method See Also

Scripting Run-Time Reference

GetAbsolutePathName Method

GetBaseName Method

GetDrive Method

GetDriveName Method

GetExtensionName Method

GetFile Method

GetFileName Method

GetFileVersion Method

GetFolder Method

GetParentFolderName Method

GetTempName Method

${\tt Microsoft @ JScript @} \ GetSpecial Folder$

Method Applies To

Scripting Run-Time Reference

${\tt Microsoft @ JScript @} \ \, \boldsymbol{GetTempName}$

Method See Also

Scripting Run-Time Reference

GetAbsolutePathName Method

GetBaseName Method

GetDrive Method

GetDriveName Method

GetExtensionName Method

GetFile Method

GetFileName Method

GetFileVersion Method

GetFolder Method

GetParentFolderName Method

GetSpecialFolder Method

${\tt Microsoft @ JScript @} \ \, \boldsymbol{GetTempName}$

Method Applies To

Scripting Run-Time Reference

Microsoft® JScript® IsReady Property

Scripting Run-Time Reference

See Also

AvailableSpace Property

DriveLetter Property

DriveType Property

FileSystem Property

FreeSpace Property

Path Property

RootFolder Property

SerialNumber Property

ShareName Property

TotalSize Property

VolumeName Property

Microsoft® JScript® IsReady Property Secript®

Scripting Run-Time Reference

Applies To

Drive Object

Microsoft® JScript® IsRootFolder

Scripting Run-Time Reference

Property See Also

Attributes Property

DateCreated Property

DateLastAccessed Property

DateLastModified Property

Drive Property

Files Property

Name Property

ParentFolder Property

Path Property

ShortName Property

ShortPath Property

Size Property

SubFolders Property

Type Property

${\tt Microsoft @ JScript @ } Is Root Folder$

Scripting Run-Time Reference

Property Applies To

Folder Object

Microsoft® JScript® Items Method

Scripting Run-Time Reference

See Also

Add Method (Dictionary)

Exists Method

Keys Method

Remove Method

RemoveAll Method

${\tt Microsoft @ JScript @} \ \ \pmb{Items Method}$

Scripting Run-Time Reference

Applies To

Dictionary Object

${\tt Microsoft @ JScript @ } Keys \ Method$

Scripting Run-Time Reference

See Also

Add Method (Dictionary)

Exists Method

Items Method

Remove Method

RemoveAll Method

${\tt Microsoft \& JScript \& Keys \ Method}$

Scripting Run-Time Reference

Applies To

Dictionary Object

Microsoft® JScript® Line Property

Scripting Run-Time Reference

See Also

Column Property

Microsoft® JScript® Line Property

Scripting Run-Time Reference

Applies To

TextStream Object

Microsoft® JScript® Move Method

Scripting Run-Time Reference

See Also

Copy Method
Delete Method
MoveFile Method
MoveFolder Method
OpenAsTextStream Method

Microsoft® JScript® Move Method

Scripting Run-Time Reference

Applies To

<u>File Object</u> <u>Folder Object</u>

Microsoft® JScript® MoveFile Method

Scripting Run-Time Reference

See Also

CopyFile Method

DeleteFile Method

GetFile Method

GetFileName Method

Move Method

MoveFolder Method

OpenTextFile Method

${\tt Microsoft @ JScript @}~ MoveFile~ Method$

Scripting Run-Time Reference

Applies To

Microsoft® JScript® MoveFolder

Method See Also

Scripting Run-Time Reference

CopyFile Method

DeleteFile Method

GetFile Method

GetFileName Method

Move Method

MoveFile Method

OpenTextFile Method

Microsoft® JScript® MoveFolder

Method Applies To

Scripting Run-Time Reference

${\tt Microsoft \& JScript \& } \begin{picture}(20,20) \put(0,0){\line(1,0){100}} \put(0,0){\line(1,0){$

Scripting Run-Time Reference

See Also

Attributes Property

DateCreated Property

DateLastAccessed Property

DateLastModified Property

Drive Property

Files Property

IsRootFolder Property

ParentFolder Property

Path Property

ShortName Property

ShortPath Property

Size Property

SubFolders Property

Type Property

Microsoft® JScript® Name Property

Scripting Run-Time Reference

Applies To

<u>File Object</u> <u>Folder Object</u>

${\tt Microsoft \& JScript \& } \ Open TextFile$

Method See Also

Scripting Run-Time Reference

<u>CreateTextFile Method</u> <u>OpenAsTextStream Method</u>

${\tt Microsoft \& JScript \& } \ Open TextFile$

Method

Applies To

Scripting Run-Time Reference

Microsoft® JScript® ParentFolder

Scripting Run-Time Reference

Property See Also

Attributes Property

DateCreated Property

DateLastAccessed Property

DateLastModified Property

Drive Property

Files Property

IsRootFolder Property

Name Property

Path Property

ShortName Property

ShortPath Property

Size Property

SubFolders Property

Type Property

Microsoft® JScript® ParentFolder

Scripting Run-Time Reference

Property Applies To

<u>File Object</u> <u>Folder Object</u>

Scripting Run-Time Reference

Microsoft® JScript® Path Property

See Also

Attributes Property

AvailableSpace Property

DateCreated Property

DateLastAccessed Property

DateLastModified Property

Drive Property

DriveLetter Property

DriveType Property

Files Property

FileSystem Property

FreeSpace Property

IsReady Property

IsRootFolder Property

Name Property

ParentFolder Property

RootFolder Property

SerialNumber Property

ShareName Property

ShortName Property

ShortPath Property

Size Property

SubFolders Property

TotalSize Property

Type Property

VolumeName Property

Microsoft® JScript® Path Property

Scripting Run-Time Reference

Applies To

Drive Object File Object Folder Object

Microsoft® JScript® Read Method

Scripting Run-Time Reference

See Also

ReadAll Method
ReadLine Method
Skip Method
SkipLine Method

${\tt Microsoft @ JScript @} \ Read\ Method$

Scripting Run-Time Reference

Applies To

TextStream Object

Microsoft® JScript® ReadLine

Method See Also

Scripting Run-Time Reference

Read Method
ReadAll Method
Skip Method
SkipLine Method

Microsoft® JScript® ReadLine

Method

Applies To

Scripting Run-Time Reference

TextStream Object

Microsoft® JScript® Remove Method

Scripting Run-Time Reference

See Also

Add Method (Dictionary)

Exists Method

Items Method

Keys Method

RemoveAll Method

Microsoft® JScript® Remove Method

Scripting Run-Time Reference

Applies To

Dictionary Object

Microsoft® JScript® RootFolder

Scripting Run-Time Reference

Property See Also

AvailableSpace Property

DriveLetter Property

DriveType Property

FileSystem Property

FreeSpace Property

IsReady Property

Path Property

SerialNumber Property

ShareName Property

TotalSize Property

VolumeName Property

Microsoft® JScript® RootFolder

Scripting Run-Time Reference

Property Applies To

Drive Object

Microsoft® JScript® SerialNumber

Scripting Run-Time Reference

Property See Also

AvailableSpace Property

DriveLetter Property

DriveType Property

FileSystem Property

FreeSpace Property

IsReady Property

Path Property

RootFolder Property

ShareName Property

TotalSize Property

VolumeName Property

Microsoft® JScript® SerialNumber

Scripting Run-Time Reference

Property Applies To

Drive Object

Microsoft® JScript® ShareName

Scripting Run-Time Reference

Property See Also

AvailableSpace Property

DriveLetter Property

DriveType Property

FileSystem Property

FreeSpace Property

IsReady Property

Path Property

RootFolder Property

SerialNumber Property

TotalSize Property

VolumeName Property

Microsoft® JScript® ShareName

Scripting Run-Time Reference

Property Applies To

Drive Object

Microsoft® JScript® ShortName

Scripting Run-Time Reference

Property See Also

Attributes Property

DateCreated Property

DateLastAccessed Property

DateLastModified Property

Drive Property

Files Property

IsRootFolder Property

Name Property

ParentFolder Property

Path Property

ShortPath Property

Size Property

SubFolders Property

Type Property

Microsoft® JScript® ShortName

Scripting Run-Time Reference

Property Applies To

<u>File Object</u> <u>Folder Object</u>

Microsoft® JScript® ShortPath

Scripting Run-Time Reference

Property See Also

Attributes Property

DateCreated Property

DateLastAccessed Property

DateLastModified Property

Drive Property

Files Property

IsRootFolder Property

Name Property

ParentFolder Property

Path Property

ShortName Property

Size Property

SubFolders Property

Type Property

Microsoft® JScript® ShortPath

Scripting Run-Time Reference

Property Applies To

<u>File Object</u> <u>Folder Object</u>

Microsoft® JScript® Size Property

Scripting Run-Time Reference

See Also

Attributes Property

DateCreated Property

DateLastAccessed Property

DateLastModified Property

Drive Property

Files Property

IsRootFolder Property

Name Property

ParentFolder Property

Path Property

ShortName Property

ShortPath Property

SubFolders Property

Type Property

Microsoft® JScript® Size Property

Scripting Run-Time Reference

Applies To

<u>File Object</u> <u>Folder Object</u>

${\tt Microsoft @ JScript @} \ Skip \ Method$

Scripting Run-Time Reference

See Also

Close Method

Read Method

ReadAll Method

ReadLine Method

SkipLine Method

Write Method

WriteLine Method

WriteBlankLines Method

${\tt Microsoft @ JScript @} \ Skip \ Method$

Scripting Run-Time Reference

Applies To

TextStream Object

Microsoft® JScript® SubFolders

Scripting Run-Time Reference

Property See Also

Attributes Property

DateCreated Property

DateLastAccessed Property

DateLastModified Property

Drive Property

Files Property

IsRootFolder Property

Name Property

ParentFolder Property

Path Property

ShortName Property

ShortPath Property

Size Property

Type Property

Microsoft® JScript® SubFolders

Scripting Run-Time Reference

Property Applies To

Folder Object

Microsoft® JScript® **TextStream**

Object

Methods

Scripting Run-Time Reference

Close Method

Read Method

ReadAll Method

ReadLine Method

Skip Method

SkipLine Method

Write Method

WriteBlankLines Method

WriteLine Method

Microsoft® JScript® **TextStream**

Object

Properties

Scripting Run-Time Reference

AtEndOfLine Property
AtEndOfStream Property
Column Property
Line Property

Microsoft® JScript® TotalSize

Scripting Run-Time Reference

Property See Also

AvailableSpace Property

DriveLetter Property

DriveType Property

FileSystem Property

FreeSpace Property

IsReady Property

Path Property

RootFolder Property

SerialNumber Property

ShareName Property

VolumeName Property

Microsoft® JScript® TotalSize

Scripting Run-Time Reference

Property Applies To

Drive Object

Microsoft® JScript® Type Property

Scripting Run-Time Reference

See Also

Attributes Property

DateCreated Property

DateLastAccessed Property

DateLastModified Property

Drive Property

Files Property

IsRootFolder Property

Name Property

ParentFolder Property

Path Property

ShortName Property

ShortPath Property

Size Property

SubFolders Property

Microsoft® JScript® **Type Property**

Scripting Run-Time Reference

Applies To

<u>File Object</u> <u>Folder Object</u>

Microsoft® JScript® VolumeName

Scripting Run-Time Reference

PropertySee Also

AvailableSpace Property

DriveLetter Property

DriveType Property

FileSystem Property

FreeSpace Property

IsReady Property

Path Property

RootFolder Property

SerialNumber Property

ShareName Property

TotalSize Property

Microsoft® JScript® VolumeName

Scripting Run-Time Reference

Property Applies To

Drive Object

Microsoft® JScript® Write Method

Scripting Run-Time Reference

See Also

WriteBlankLines Method WriteLine Method

Microsoft® JScript® Write Method

Scripting Run-Time Reference

Applies To

TextStream Object

Microsoft® JScript® WriteBlankLines

Method See Also

Scripting Run-Time Reference

Write Method
WriteLine Method

Microsoft® JScript® WriteBlankLines

Method

Scripting Run-Time Reference

Applies To

TextStream Object

Microsoft® JScript® WriteLine

Method See Also

Scripting Run-Time Reference

Write Method
WriteBlankLines Method

Microsoft® JScript® WriteLine

Method

Scripting Run-Time Reference

Applies To

TextStream Object

${\tt Microsoft @ JScript @ } \ getDate \ Method$

Language Reference

Applies To

Applies To

${\tt Microsoft \& JScript \& } \ getHours \ Method$

Language Reference

See Also

Date Object Methods getUTCHours Method setHours Method setUTCHours Method

${\tt Microsoft @ JScript @ } \boldsymbol{getHours \ Method}$

Language Reference

Applies To

${\tt Microsoft \& JScript \& } \ getMinutes$

Method See Also

Language Reference

Date Object Methods
getUTCMinutes Method
setMinutes Method
setUTCMinutes Method

Microsoft® JScript® **getMinutes**

Method

Language Reference

Applies To

${\tt Microsoft @ JScript @ } \ getMonth$

Method See Also

Language Reference

Date Object Methods
getUTCMonth Method
setMonth Method
setUTCMonth Method

${\tt Microsoft \& JScript \& } \ getMonth$

Method

Language Reference

Applies To

${\tt Microsoft \& JScript \& } \ getSeconds$

Method See Also

Language Reference

Date Object Methods
getUTCSeconds Method
setSeconds Method
setUTCSeconds Method

${\tt Microsoft \& JScript \& } \ getSeconds$

Method

Language Reference

Applies To

Language Reference

See Also

Date Object Methods setTime Method

${\tt Microsoft \& JScript \& } \ getTime \ Method$

Language Reference

Applies To

Microsoft® JScript®

getTimezoneOffset Method

Language Reference

Applies To

Language Reference

Microsoft® JScript® getYear Method

See Also

Date Object Methods
getFullYear Method
getUTCFullYear Method
setFullYear Method
setUTCFullYear Method
setUTCFullYear Method

${\tt Microsoft @ JScript @ } \ getYear \ Method$

Language Reference

Applies To

${\tt Microsoft \& JScript \& } \ getUTCFullYear$

Method See Also

Language Reference

Date Object Methods
getFullYear Method
setFullYear Method
setUTCFullYear Method

${\tt Microsoft @ JScript @} \ \, \boldsymbol{getUTCFullYear}$

Method Applies To

Language Reference

Method See Also

Language Reference

Date Object Methods
getHours Method
setHours Method
setUTCHours Method

${\tt Microsoft @ JScript @ } \ getUTCHours$

Method Applies To

Language Reference

${\tt Microsoft @ JScript @ } \ get UTCM in utes$

Method See Also

Language Reference

Date Object Methods
getMinutes Method
setMinutes Method
setUTCMinutes Method

Microsoft® JScript® **getUTCMinutes**

Method Applies To

Language Reference

${\tt Microsoft \& JScript \& } \ getUTCMonth$

Method See Also

Language Reference

Date Object Methods
getMonth Method
setMonth Method
setUTCMonth Method

${\tt Microsoft \& JScript \& } \ getUTCMonth$

Method Applies To

Language Reference

${\tt Microsoft @ JScript @ } \ getUTCS econds$

Method See Also

Language Reference

Date Object Methods
getSeconds Method
setSeconds Method
setUTCSeconds Method

${\tt Microsoft @ JScript @ } \ getUTCS econds$

Method Applies To

Language Reference

Microsoft® JScript® setDate Method

Language Reference

Applies To

Microsoft® JScript® setHours Method

Language Reference

Applies To

Microsoft® JScript® setMonth Method

Language Reference

Applies To

Microsoft® JScript® setSeconds

Method

Language Reference

Applies To

Microsoft® JScript® setTime Method

Language Reference

Applies To

Microsoft® JScript® setYear Method

Language Reference

See Also

Date Object Methods
getFullYear Method
getUTCFullYear Method
getYear Method
setFullYear Method
setUTCFullYear Method

Microsoft® JScript® setYear Method

Language Reference

Applies To

Microsoft® JScript® setUTCFullYear

Method See Also

Language Reference

Date Object Methods
getFullYear Method
getUTCFullYear Method
setFullYear Method

Microsoft® JScript® setUTCFullYear

Method Applies To

Language Reference

Microsoft® JScript® setUTCHours

Method See Also

Language Reference

Date Object Methods
getHours Method
getUTCHours Method
setHours Method

Microsoft® JScript® setUTCHours

Method Applies To

Language Reference

Microsoft® JScript® setUTCMinutes

Method See Also

Language Reference

Date Object Methods
getMinutes Method
getUTCMinutes Method
setMinutes Method

Microsoft® JScript® setUTCMinutes

Method Applies To

Language Reference

Microsoft® JScript® setUTCMonth

Method See Also

Language Reference

Date Object Methods
getMonth Method
getUTCMonth Method
setMonth Method

Microsoft® JScript® setUTCMonth

Method Applies To

Language Reference

Microsoft® JScript® setUTCSeconds

Method See Also

Language Reference

Date Object Methods
getSeconds Method
getUTCSeconds Method
setSeconds Method

Microsoft® JScript® setUTCSeconds

Method Applies To

Language Reference

${\tt Microsoft @ JScript @ } to Locale String$

Method

Language Reference

Applies To

See Also

Date Object Methods

${\tt Microsoft @ JScript @ } parse \ Method$

Language Reference

Applies To

Language Reference

See Also

Date Object Methods setTime Method

Applies To

Microsoft® JScript® function

Statement See Also

Language Reference

new Operator

${\tt Microsoft @ JScript @ } parseInt\ Method$

Language Reference

Applies To

Global Object

${\tt Microsoft @ JScript @ } parseFloat$

Method Applies To

Language Reference

Global Object

Language Reference

See Also

%= Operator

Operator Behavior

Operator Precedence

Operator Summary

Microsoft® JScript® * Operator

See Also

*= Operator

Operator Behavior

Operator Precedence

Operator Summary

Microsoft® JScript® / Operator

See Also

<u>/= Operator</u>

Operator Behavior

Operator Precedence

Operator Summary

See Also

Object Object

${\tt Microsoft @ JScript @} \ \boldsymbol{Global \ Object}$

Methods

escape Method
eval Method
isFinite Method
isNaN Method
parseFloat Method
parseInt Method

unescape Method

Properties

Infinity Property
NaN Property

${\tt Microsoft @ JScript @ } {\bf Split \ Method}$

Language Reference

See Also

concat Method

RegExp Object

Regular Expression Syntax

String Object Methods

${\tt Microsoft \& JScript \& } split \ Method$

Language Reference

Applies To

String Object

${\tt Microsoft @ JScript @ } \ getVarDate$

Method See Also

Language Reference

getDate Method parse Method

${\tt Microsoft @ JScript @ } \ getVarDate$

Method Applies To

Language Reference

See Also

exec Method
RegExp Object
replace Method
search Method
String Object Methods
test Method

${\tt Microsoft @ JScript @ } replace \ Method$

Language Reference

See Also

exec Method

match Method

RegExp Object

search Method

String Object Methods

test Method

${\tt Microsoft \& JScript \& } \ replace \ Method$

Language Reference

Applies To

String Object

Microsoft® JScript®

OpenAsTextStream Method See Also

Scripting Run-Time Reference

Copy Method
CreateTextFile Method
Delete Method
Move Method
OpenTextFile Method

Microsoft® JScript®

OpenAsTextStream Method Applies To

Scripting Run-Time Reference

File Object

Microsoft® JScript® String Object

Methods

Members of String.prototype

anchor Method

big Method

blink Method

bold Method

charAt Method

charCodeAt Method

concat Method

fixed Method

fontcolor Method

fontsize Method

fromCharCode Method

indexOf Method

italics Method

lastIndexOf Method

link Method

match Method

replace Method

search Method

slice Method

small Method

split Method

strike Method

sub Method

substr Method

substring Method

sup Method

toLowerCase Method toUpperCase Method

toString Method valueOf Method

Nonmembers of String.prototype

The **String** object has no methods that are not part of the prototype.

Microsoft® JScript® Operator Summary

Computational

Addition (+)
Decrement (--)
Division (/)
Increment (++)
Modulus (%)
Multiplication (*)
Subtraction (-)
Unary negation (-)

Logical

Comma (,)
Conditional (trinary) (?:)
Equality (==)
Greater than (>)
Greater than or equal to (>=)
Identity (===)
Inequality (!=)
Less than (<)
Less than or equal to (<=)
Logical AND (&&)
Logical NOT (!)
Logical OR (||)
Nonidentity (!==)

Bitwise

```
Bitwise AND (&)
Bitwise Left Shift (<<)
Bitwise NOT (~)
Bitwise OR (|)
Bitwise Right Shift (>>)
Bitwise XOR (^)
Unsigned Right Shift (>>>)
```

Assignment

```
<u>Assignment (=)</u>
<u>Compound Assignment Operators</u>
```

Miscellaneous

delete
instanceof
new
typeof
void

Microsoft® JScript® Regular Expression Object Methods

Language Reference

compile Method exec Method test Method

Microsoft® JScript® Regular Expression Object Properties

Language Reference

lastIndex Property source Property



See Also

| Operator

Operator Behavior

Operator Precedence

See Also

^ Operator

Operator Behavior

Operator Precedence

Microsoft® JScript® /= Operator

See Also

/ Operator

Operator Behavior

Operator Precedence

Microsoft® JScript® <<= Operator

See Also

<< Operator

>> Operator

>>> Operator

Operator Behavior

Operator Precedence

Microsoft® JScript® %= Operator

See Also

% Operator

Operator Behavior

Operator Precedence

Microsoft® JScript® *= Operator

See Also

* Operator

Operator Behavior

Operator Precedence

Microsoft® JScript® >>= Operator

See Also

<< Operator

>> Operator

>>> Operator

Operator Behavior

Operator Precedence

Microsoft® JScript® -= Operator

See Also

- Operator

Operator Behavior

Operator Precedence

Microsoft® JScript® >>>= Operator

See Also

>>> Operator

<< Operator

>> Operator

Operator Behavior

Operator Precedence

Microsoft® JScript® NaN Property

See Also Applies To

Description

Returns the special value **NaN** indicating that an expression is not a number.

Syntax

NaN

Remarks

The **NaN** property (not a number) is a member of the **Global** object, and is made available when the scripting engine is initialized.

${\tt Microsoft @ JScript @} \ Remove All$

Method

See Also Applies To

Description

The **RemoveAll** method removes all key, item pairs from a **Dictionary** object.

Syntax

```
object.RemoveAll()
```

The *object* is always the name of a **Dictionary** object.

Remarks

The following code illustrates use of the **RemoveAll** method:

```
var a, d, i;  // Create some variables.
d = new ActiveXObject("Scripting.Dictionary");
d.Add ("a", "Athens");  // Add some keys and items.
d.Add ("b", "Belgrade");
d.Add ("c", "Cairo");
...
d.RemoveAll();  // Clear the dictionary.
```

See Also Applies To

Description

Sets or returns an *item* for a specified *key* in a **Dictionary** object. For collections, returns an *item* based on the specified *key*. Read/write.

Syntax

object.Item(key)[= newitem]

The **Item** property has the following parts:

Part	Description
llahi <i>oc</i> t l	Required. Always the name of a collection or Dictionary object.
II <i>KO</i> W I	Required. <i>Key</i> associated with the <i>item</i> being retrieved or added.
newitem	Optional. Used for Dictionary object only; no application for collections. If provided, <i>newitem</i> is the new value associated with the specified <i>key</i> .

Remarks

If *key* is not found when changing an *item*, a new *key* is created with the specified *newitem*. If *key* is not found when attempting to return an existing item, a new *key* is created and its corresponding item is left empty.

The following example illustrates the use of the **Item** property.

```
function DicTest(keyword)
{
  var a, d;
  d = new ActiveXObject("Scripting d.Add("a", "Athens");
  d.Add("b", "Belgrade");
  d.Add("c", "Cairo");
  a = d.Item(keyword);
  return(a);
}
```

${\tt Microsoft @ JScript @ } \pmb{Date Object}$

Methods

Members of Date.prototype

getDate Method getDay Method getFullYear Method getHours Method getMilliseconds Method getMinutes Method getMonth Method getSeconds Method getTime Method getTimezoneOffset Method getUTCDate Method getUTCDay Method getUTCFullYear Method getUTCHours Method getUTCMilliseconds Method getUTCMinutes Method getUTCMonth Method getUTCSeconds Method getVarDate Method getYear Method setDate Method setFullYear Method setHours Method setMilliseconds Method setMinutes Method

setMonth Method

setSeconds Method

setTime Method

setUTCDate Method

setUTCFullYear Method

setUTCHours Method

setUTCMilliseconds Method

setUTCMinutes Method

setUTCMonth Method

setUTCSeconds Method

setYear Method

toGMTString Method

toLocaleString Method

toUTCString Method

toString Method valueOf Method

Nonmembers of Date.prototype

parse Method UTC Method

Microsoft® JScript® getFullYear Method

See Also

Applies To

Description

Returns the year value in the **Date** object using <u>local time</u>.

Syntax

```
objDate.getFullYear()
```

Remarks

To get the year using <u>Universal Coordinated Time (UTC)</u>, use the **getUTCFullYear** method.

The **getFullYear** method returns the year as an absolute number. For example, the year 1976 is returned as 1976. This avoids the classic year 2000 problem where dates beginning with January 1, 2000 are confused with those beginning with January 1, 1900.

The following example illustrates the use of the **GetFullYear** method:

```
function DateDemo()
{
  var d, s = "Today's UTC date is: ";
  d = new Date();
```

```
s += (d.getMonth() + 1) + "/";
s += d.getDate() + "/";
s += d.getFullYear();
return(s);
}
```

Microsoft® JScript® setFullYear

Method

See Also Applies To

Description

Sets the year value in the **Date** object using <u>local time</u>.

Syntax

objDate.setFullYear(numYear[, numMonth[, numDate]])

The **setFullYear** method syntax has these parts:

Part	Description
numYear	Required. A numeric value equal to the year.
numMonth	Optional. A numeric value equal to the month. Must be supplied if <i>numDate</i> is supplied.
	Optional. A numeric value equal to the date.

Remarks

All **set** methods taking optional arguments use the value returned from corresponding **get** methods, if you do not specify an optional argument. For example, if the *numMonth* argument is optional, but not specified, JScript uses the value returned from the **getMonth** method.

In addition, if the value of an argument is greater than its range or is a negative number, other stored values are modified accordingly.

To set the year using <u>Universal Coordinated Time (UTC)</u>, use the **setUTCFullYear** method.

The range of years supported in the date object is approximately 285,616 years from either side of 1970.

The following example illustrates the use of the **setFullYear** method:

```
function SetFullYearDemo(newyear)
{
  var d, s;
  d = new Date();
  d.setFullYear(newyear);
  s = "Current setting is ";
  s += d.toLocaleString();
  return(s);
}
```

Microsoft® JScript® **setMinutes**

Method

See Also

Applies To

Description

Sets the minutes value in the **Date** object using <u>local time</u>.

Syntax

objDate.setMinutes(numMinutes[, numSeconds[, numMilli]])

The **setMinutes** method syntax has these parts:

Part	Description
numMinutes	Required. A numeric value equal to the minutes value.
	Optional. A numeric value equal to the seconds value. Must be supplied if the <i>numMilli</i> argument is used.
llniim Wiiii – I	Optional. A numeric value equal to the milliseconds value.

Remarks

All **set** methods taking optional arguments use the value returned from corresponding **get** methods, if you do not specify an optional argument. For example, if the *numMonth* argument is

optional, but not specified, JScript uses the value returned from the **getMonth** method.

To set the minutes value using <u>Universal Coordinated Time (UTC)</u>, use the **setUTCMinutes** method.

If the value of an argument is greater than its range or is a negative number, other stored values are modified accordingly. For example, if the stored date is "Jan 5, 1996 00:00:00" and **setMinutes(90)** is called, the date is changed to "Jan 5, 1996 01:30:00." Negative numbers have a similar behavior.

The following example illustrates the use of the **setMinutes** method:

```
function SetMinutesDemo(nmin, ns
{
  var d, s;
  var sep = ":";
  d = new Date();
  d.setMinutes(nmin, nsec);
  s = "Current setting is " + d.toLoc return(s);
}
```

Microsoft® JScript® lastIndex Property (Regular Expression)

See Also

Applies To

Description

Specifies the index at which to start the next match.

Syntax

*rgexp.***lastIndex** [= *index*]

The **lastIndex** property syntax has these parts:

Part	Description	
rgexp	Required. A Regular Expression object. Can be a variable name or a literal.	
	The index from which to begin the next search.	

Remarks

The **lastIndex** property is modified by the **exec** method, and the **match**, **replace**, and **split** methods of the **String** object.

The following rules apply to values of **lastIndex**:

• If **lastIndex** is greater than the length of the string, the **test** and **exec** methods fail, and **lastIndex** is set to zero.

- If **lastIndex** is equal to the length of the string, the regular expression matches if the pattern matches the empty string. Otherwise, the match fails and **lastIndex** is reset to zero.
- Otherwise, **lastIndex** is set to the next position following the most recent match.

${\tt Microsoft @ JScript @} \ \, \pmb{NaN \ Property}$

Language Reference

See Also

isNaN Method

Microsoft® JScript® NaN Property

Language Reference

Applies To

Global Object

Microsoft® JScript® RemoveAll

Method See Also

Scripting Run-Time Reference

Add Method (Dictionary)

Exists Method

Items Method

Keys Method

Remove Method

Microsoft® JScript® RemoveAll

Scripting Run-Time Reference

Method

Applies To

Dictionary Object

${\tt Microsoft \& JScript \& } \ \ \pmb{Item Property}$

Scripting Run-Time Reference

Applies To

Dictionary Object

Drives Collection

Files Collection

Folders Collection