Operators

ON THIS PAGE

OS8

Operations in the innermost level of parentheses are performed first. Evaluation then proceeds according to the precedence in the following table. Operations on the same precedence are left-associative, except for exponentiation, which is right-associative.

Description	Symbol(s)
Function Calls or Parentheses	()
Array Operators	. , []
Exponentiation	^
Negation	–, +
Multiplication, Division, Modulus	*, /, MOD
Addition, Subtraction	+, -
Comparison	<, >, = , <>, <=, >=
Logical Negation	NOT
Logical Conjunction	AND
Logical OR	OR

String Operators: The following operators work with strings: `<, >, =, <>, <=, >=, +`

Function References: The = and <> operators work on variables that contain function references and function literals.

Logical and Bitwise Operators

The `AND`, `OR`, and `NOT` operators are used for logical (Boolean) comparisons if the arguments for these operators are Boolean:

```
a = 20
b = 20
c = 20
if a = c and not(b > 40) then print "success"
```

On the other hand, if the arguments for these operators are numeric, they will perform bitwise operations:

```
x = 1 and 2   ' x is zero
y = true and false  ' y is false
```

When the `AND` or `OR` operator is used for a logical operation, only the necessary amount of the expression is executed. For example, the first statement below will print "True", while the second statement will cause a runtime error (because "invalid" is not a valid operand for `OR`):

```
print true or invalid
print false or invalid
```

Dot Operator

The "." Dot Operator can be used on any BrightScript object. It also has special meaning when used on an *roAssociativeArray* object, as well as *roXMLElement* and *roXMLList* objects. When used on a BrightScript object, it refers to an interface or method associated with that object. In the following example, `IfInt` refers to the interface and `SetInt()` refers to a method that is part of that interface:

```
i = CreateObject("roInt")
i.ifInt.SetInt(5)
i.SetInt(5)
```

Every object method is part of an interface. However, specifying the interface with the "." Dot Operator is optional. If the interface is omitted, as in the third line of the above example, each interface that is part of the object will be searched for the specified member. If there is a naming conflict (i.e. a method with the same name appears in two interfaces), then the interface should be specified.

ASSOCIATIVE ARRAYS

When the "." Dot Operator is used on an Associative Array, it is the same as calling the `Lookup()` or `AddReplace()` methods, which are member functions of the *roAssociativeArray* object:

```
aa = {}
aa.newkey = "the value"
print aa.newkey
```

Note that the parameters of the "." Dot Operator are set at compile time; unlike the `Lookup()` and `AddReplace()` methods, they are not dynamic.

The "." Dot Operator is always case insensitive: For example, the statement `aa.NewKey=55` will create the entry "newkey" in the associative array. To generate case-sensitive keys, instantiate an *roAssociativeArray* object and use the `SetModeCaseSensitive()` method.

Array and Function-Call Operators

The [ ] operator is used to access an array (i.e. any BrightScript object that has an *ifArray* interface, such as *roArray* and *roList* objects). It can also be used to access an associative array. The [ ] operator takes expressions that are evaluated at runtime, while the "." Dot Operator takes identifiers at compile time.

The ( ) operator can be used to call a function. When used on a function literal (or variable containing a function reference), that function will be called.

The following code snippet demonstrates the use of both array and function-call operators.

```
aa = CreateObject("roAssociativeArray")
aa["newkey"] = "the value"
print aa["newkey"]

array = CreateObject("roArray", 10, true)
array[2] = "two"
print array[2]

fivevar = five
print fivevar()

array[1] = fivevar
print array[1]()   ' print 5

function five() As Integer
   return 5
end function
```

ARRAY DIMENSIONS

Arrays in BrightScript are one dimensional. Multi-dimensional arrays are implemented as arrays of arrays. The [ ] operator will automatically map multi-dimensionality. For example, the following two fetching expressions are the same:

```
dim array[5,5,5]
item = array[1][2][3]
item = array[1,2,3]
```

If a multi-dimensional array grows beyond its hint size, the new entries are not automatically set to *roArray*.

Equals Operator

The = operator is used for both assignment and comparison:

```
a = 5
If a = 5 then print "a is 5"
```

Unlike the C language, BrightScript does not support use of the = assignment operator inside an expression. This is meant to eliminate a common class of bugs caused by confusion between assignment and comparison.

When assignment occurs, intrinsic types are copied, while BrightScript objects are reference counted.