Operators

Overview

Operators in DuckyScript instruct the payload to perform a given mathematical, relational or logical operation.

Math Operators

Math operators may be used to change the value of a variable

Operator	Meaning
=	Assignment
+	Add
-	Subtract
*	Multiply
/	Divide
%	Modulus
^	Exponent

Examples

Consider how the variable `$FOO` changes with each math operation.

```
REM Assign $FOO to 42
VAR $FOO = 42

REM The variable is now 42.
REM Let's add it by 1.
$FOO = ( $FOO + 1 )

REM The variable is now 43: the sum of 42 and 1.
```

```
REM Let's subtract it by 1.
$FOO = ( $FOO - 1 )

REM The variable is now 42 (again): the difference of 42 and 1.
REM Let's multiply it by 2.
$FOO = ( $FOO * 2 )

REM The variable is now 84: the product of 42 and 2.
REM Let's divide it by 2.
$FOO = ( $FOO / 2 )

REM The variable is now 42 (again): the quotient of 82 and 2.
REM Let's modulus it by 4.
$FOO = ( $FOO % 4 )

REM The variable is now 2: the signed remainder of 42 and 4.
REM Let's raise it to the power of 6.
$FOO = ( $FOO ^ 6 )

REM Our variable is now 64: the exponent of 2 and 6.
```

Comparison Operators

Comparison operators (or relational operators) will compare two values to evaluate a single boolean value.

Operator	Meaning
==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

Examples

Consider how the different comparison operators evaluate down to a single boolean value for the following variables:

```
VAR $FOO = 42
VAR $BAR = 1337
```

- The comparison `( $FOO == $BAR )` evaluates to the boolean `FALSE`.
- The comparison `( $FOO != $BAR )` evaluates to the boolean `TRUE`.
- The comparison `( $FOO > $BAR )` evaluates to the boolean `FALSE`.
- The comparison `( $FOO < $BAR )` evaluates to the boolean `TRUE`.
- The comparison `( $FOO >= $BAR )` evaluates to the boolean `FALSE`.
- The comparison `( $FOO <= $BAR )` evaluates to the boolean `TRUE`.

Order of Operations

The order of operations (order precedence) are a set of rules that define which procedures are performed first in order to evaluate an expression, similar to that of mathematics.

In DuckyScript, parentheses ( ) are required to define the precedence conventions.

```
VAR $FOO = ( 4 * 10 ) + 2

REM The expression ( 4 * 10 ) evalues to 40.
REM The expression 40 + 2 evalues to 42.
```

If multiple pairs of parentheses are required, the parentheses can be nested.

```
VAR $FOO = 42
VAR $BAR = (( 100 * 13 ) + ( $FOO - 5 ))

REM The expression 42 - 5 evalues to 37
REM The expression ( 100 * 13 ) evalues to 1300
REM The expression 1300 + 37 evalues to 1337
```

Logical Operators

Logical operators are important as they allow us to make decisions based on certain conditions. For example, when combining the result of more than one condition, the logical AND or OR logical operators will make the final determination.

These logical operators may be used to connect two or more expressions.

Operator	Description
&&	Logical AND. If both the operands are non-zero, the condition is `TRUE`.
||	Logical OR. If any of the two operands is non-zero, the condition is `TRUE`.

Examples

Considering the values of the two variables:

```
VAR $FOO = 42
VAR $BAR = 1337
```

The expression `( $FOO < $BAR )` evaluates to `TRUE`.
The expression
`( $FOO > $BAR )` evaluates to `FALSE`.

Combining these expressions, we can evaluate:

`( $FOO < $BAR ) && ( $BAR > $FOO )` Evalues down to `TRUE && TRUE`Because 42 is less than 1337 is TRUE **AND** 1337 is greater than 42 is TRUE.Both operands are non-zero (`TRUE`), therefore the final condition is `TRUE`.

`( $FOO < $BAR ) || ( $BAR == $FOO )` Evaluates as `TRUE || FALSE`Because 42 is less than 1337 is TRUE **OR** 1337 is equal to 42 is FALSE.Any of the operands are non-zero (`TRUE`), therefore the final condition is `TRUE`.

Augmented Assignment Operators

When assigning a value to a variable, the variable itself may be referenced.

Example

```
VAR $FOO = 1337
VAR $FOO = ( $FOO + 1 )
REM $FOO will now equal 1338
```

Result

The variable `$FOO` is initiated as `1337`.

`$FOO` is incremented by `1` (itself plus 1).

`$FOO` will then equal `1338`.

Bitwise Operators

Bitwise operators are operators which operate on the uint16 values at the binary level.

Operator	Description
&	Bitwise AND. If the corresponding bits of the two operands is 1, will result in 1. Otherwise if either bit of an operand is 0, the result of the corresponding bit is evaluated as 0.
|	Bitwise OR. If at least one corresponding bit of the two operands is 1, will result in 1.
>>	Right Shift. Accepts two numbers. Right shifts the bits of the first operand. The second operand determines the number of places to shift.
<<	Left Shift. Accepts two numbers. Left shifts the bits of the first operand. The second operand decides the number of places to shift.

Example

```
ATTACKMODE HID STORAGE VID_05AC PID_021E
VAR $FOO = $_CURRENT_VID
REM Because VID and PID parameters are little endian,
$FOO = (($FOO >> 8) & 0x00FF) | (($FOO << 8) & 0xFF00))
REM $FOO will now equal 0xAC05
```

Result

The value of `$_CURRENT_VID` is saved into the variable `$FOO` as `AC05`.

Using bitwise operators its endianness is swapped to `05AC`.