Eezeescript Language Reference

Script Composition

```
INCLUDE {string literal}
```

Replaces the statement with the contents of the script file with the given resource name (filename by default). The included script name will be interpreted according to the **ScriptLoader** implementation bound to the **ScriptManager**. By default, all scripts are loaded from disk.

Example:

```
INCLUDE “scripts/Common.ezs”
```

Script Embellishments

Comments

```
// {comments}
```

Allows the introduction of comments and annotations within the script. Comment lines are ignored by the compiler. Comments must be specified in their own lines and cannot be placed at the end of other statements.

Example:

```
// if orb and crystal quests completed, allow wizard quest
IF g_OrbQuestComplete THEN
    IF g_CrystalQuestComplete THEN
        SET g_WizardQuestAvailable TO TRUE
    ENDIF
ENDIF
```

Blank Lines

Blank lines are ignored by the compiler and essentially provide the script writer a means to space out the code for reasons of clarity and organisation.

Variable Assignment

Local Assignment

`SET` *`{identifier}`* `TO` *`{identifier_or_literal}`*

Assigns the given integer, float, boolean or string literal or variable to the variable with the given identifier. The scope of the variable is local to the script's execution context, unless it already exists in the global scope.

Examples:

```
SET name TO “Joe”

SET found TO TRUE

SET i TO j
```

Global Assignment

`SETGLOBAL` *`{identifier}`* `TO` *`{identifier_or_literal}`*

Assigns the given integer, float, boolean or string literal or variable to the variable with the given identifier. The variable is 'globally' available to all execution contexts for the scripts defined within a **ScriptManager** instance.

Examples:

```
SETGLOBAL pi TO 3.1415

SETGLOBAL max_items TO 10
```

Arithmetic

Addition

`ADD` *`{identifier_or_literal}`* `TO` *`{identifier}`*

Adds the value of the given variable or literal to the given variable. If one of the values involved is a string, string concatenation is used.

Examples:

```
ADD 1 TO count

ADD y TO x

ADD “hello world!” TO message
```

Subtraction

`SUBTRACT` *`{identifier_or_literal}`* `FROM` *`{identifier}`*

Subtracts the value of the given variable or numeric literal to variable with the given identifier.

Examples:

```
SUBTRACT 1 FROM count
```

```
SUBTRACT 5.0 FROM health
```

Multiplication

MULTIPLY *{itentifier}* BY *{identifier_or_literal}*

Multiplies the variable given by the identifier by the given variable or numeric literal.

Examples

```
MULTIPLY y BY x
```

```
MULTIPLY score BY 1.15
```

Division

DIVIDE *{itentifier}* BY *{identifier_or_literal}*

Divides the variable given by the identifier by the given variable or numeric literal.

Examples:

```
DIVIDE points BY team_size
```

```
DIVIDE total BY 4
```

Branching Logic

Unary Branching

```
IF {condition} THEN
    {statements}
ENDIF
```

Executes the enclosed statements if the given condition is true. The condition may consist of a single boolean variable or literal, a string comparison or numeric comparisons. Comparisons are in the form {identifier_or_literal} {operator} {identifier_or_literal} where {operator} can be '=', '!=', '>' or '<'

Examples:

```
IF orb_found THEN
   SET next_quest TO "Find the Mermaid's Shoe"
```

```
ENDIF

IF points > level_1 THEN
    SET skill_level TO 2
ENDIF
```

Binary Branching

```
IF {condition} THEN
    {statements}
ELSE
    {statements}
ENDIF
```

A variant of the IF ... THEN ... ENDIF statement block with an ELSE clause to run statements when the given condition is false.

Example:

```
IF x > y THEN
    SET max TO x
ELSE
    SET max TO y
ENDIF
```

Iteration Logic

```
WHILE {condition}
    {statements}
ENDWHILE
```

Executes the contained statements repeatedly as long as the given condition is satisfied.

Example:

```
SET product TO 1
SET count TO 0
WHILE count < power
    MULTIPLY product BY base
    ADD 1 TO count
```

```
ENDWHILE
```

Modules

Module Invocation

CALL *{identifier}*

Executes the statement block given by the block's identifier. Control is returned to the next statement after the call at the end of the block's execution.

Example:

```
CALL CalculateTotals
```

Module Definition

```
BLOCK {identifier}
    {statements}
ENDBLOCK
```

Defines a block of code that may be executed one or more times by issuing a CALL statement.

Example:

```
BLOCK CalculateTotals
    SET total TO item1
    ADD item2 TO total
ENDBLOCK
```

Host Application Interaction

Script Interrupts

YIELD

Breaks execution of the script and returns control to the host. This command may also be used to break long script runs from within the script instead of, or in tandem with, external intervention from the host.

External Calls

{command} {parameter1} ... *{parameterN}*

Executes an external command using the given variable or literal parameters. The command must be registered with the **ScriptManager** instance and its parameters must match the registered prototype, that is, it must have the correct number of parameters, each of the correct type. The command is delegated to the host only if the running **ScriptContext** is assigned to a **ScriptHandler**.

Example:

```
NPC_MoveTo 300 400

NPC_SAY “Please, help me recover the Dragon Orb!” 100
```