General Functions

General functions include those functions that are not specific to a particular interaction but can be applied in almost any script.

ArgumentList

Declaration

```
null ArgumentList(type var1[,... ])
```

Syntax

```
ArgumentList(<var1>[,... ])
```

Required Parameters

- **`var1:`** A local variable, to be initialized from the argument list of the calling instance

Optional Parameters

- **`var2,... varN:`** Additional variables to be initialized from the argument list of the calling instance

Description

This function initializes a set of local variables from its argument list.

The construction of the local variables depends on which of these cases applies:

- **Case 1: Transformation Mappings**
  When the function call is made in the mapping of a target field. (A call to the function `SetInstances()`must have been made previously.) The local variables are constructed from the corresponding global variables in the instance given by the function `SetInstances()`.
- **Case 2: Running a Script**
  When the function call is made in a script. The local variables are constructed from the corresponding arguments in the list provided by the calling `RunScript()` statement. These variables can also be addressed by index, as _1, _2...

A null value is returned by this function and can be ignored. As an alternative, see the function `GetInstance()`.

Examples

Case 1: Transformation Mappings

```
// Assuming a parent mapping contains these statements:
...
s = "SELECT key_name, key_value, key_type FROM key_values";
r = DBLookupAll("<TAG>endpoint:database/My Database</TAG>", s);
SetInstances("DETAILS", r);
...

// In the DETAILS target node, a field could
// have as a mapping:
<trans>
ArgumentList(key, value, type);
key + " = " + value + " (of type " + type + ")";
</trans>
```


Search in This Topic

Related Topics

- Cloud Studio
- Functions
- Scripts

Last updated: Jul 23, 2019

Case 2: Running a Script

```
// This code fragment calls a script
// "CalculateDisplayString":
...
RunScript("<TAG>script:CalculateDisplayString</TAG>", "John", 35);
// The result will be the string
// "John is 35 years old."
...

// The script "CalculateDisplayString", using
// names:
<trans>
ArgumentList(name, age);
name + " is " + age + " years old.";
</trans>

// Same script "CalculateDisplayString",
// using indices:
<trans>
// ArgumentList: name, age
_1 + " is " + _2 + " years old.";
</trans>
```

[Back to Top]

AutoNumber

Declaration

```
int AutoNumber()
```

Syntax

```
AutoNumber()
```

Description

Returns the number of an instance within a particular hierarchy.

WARNING: This method has been deprecated and may be removed in a future version of Jitterbit. Use either the `TargetInstanceCount()` or `SourceInstanceCount()` functions instead. The `TargetInstanceCount()` function is equivalent to this function.

Examples

Assume a target architecture has two top-level records: PO1 and PO2:

- PO1 is a parent of three child records: PO1_record1, PO1_record2, and PO1_record3.
- PO2 is a parent of two child records: PO2_record1 and PO2_record2.

When the `AutoNumber()` function is called:

- `AutoNumber()` called at the parent level returns 1 at PO1 and returns 2 at PO2.
- `AutoNumber()` at the child level of PO1 returns 1 at PO1_record1, returns 2 at PO1_record2, and returns 3 at PO1_record3, since PO1 has 3 child records.

[Back to Top]

CancelOperation

Declaration

```
void CancelOperation(string operationInstanceGUID)
```

Syntax

```
CancelOperation(<operationInstanceGUID>)
```

Required Parameters

- **`operationInstanceGUID:`** The string reference path to the operation in the current project that is to be canceled

Description

Cancels a particular operation specified by the operation's reference path. For more information, see the instructions on inserting operations under the Operations section in Jitterbit Script.

As shown in the example below, call the GetOperationQueue() function to retrieve instances of running operations. The operation instance GUID is at index 4 of the sub-arrays returned by the `GetOperationQueue()` function. See the GetOperationQueue() function for details.

Examples

```
// Cancel all instances of a particular
// operation
opt = "<TAG>operation:My Operation</TAG>";
queue = GetOperationQueue(opt);
n = Length(queue);
i = 0;
message = "Cancelling operation instance: ";
While(i < n, op_inst = queue[i][4];
  WriteToOperationLog(message + op_inst);
  CancelOperation(op_inst);
  i++;
);
```

[Back to Top]

CancelOperationChain

Declaration

```
void CancelOperationChain(string message)
```

Syntax

```
CancelOperationChain(<message>)
```

Required Parameters

- **`message`:** If a non-empty string, it will be logged as a warning message in the operation log.

Description

If the current operation has either success or failure operations, calling this method will cause those operations to be canceled. Any operations linked by a condition will also be canceled. However, any scripts in the current operation will be completed.

This can be useful if an operation is running in a loop and the condition to stop looping has been reached.

Examples

```
CancelOperationChain("The success operation does not need to run.");
```

[Back to Top]

Eval

Declaration

```
string Eval(type expToEvaluate, type defaultResult)
```

Syntax

```
Eval(<expToEvaluate>, <defaultResult>)
```

Required Parameters

- **`expToEvaluate`:** An expression to be evaluated; if valid, its result will be returned
- **`defaultResult`:** Default result to be evaluated and returned if `expToEvaluate` is not valid

Description

Evaluates the first argument; if valid, its result is returned as a string. Otherwise, the default value is evaluated and its results are returned as a string.

This can be used as a "try-catch" statement as the second argument will be evaluated if and only if the first one fails.

Examples

```
// Returns a value of "100": the string
// representation of 4 multiplied by 25:
entry = Eval(4*25,"Bad Entry");

// Returns "Bad Entry", as strings cannot be
// multiplied:
book = "";
entry = Eval(book*36.4, "Bad Entry");

// Execute a SQL statement and terminate an operation if it fails:
results = Eval(
  DBLookup("<TAG>endpoint:database/My Database</TAG>",
    "SELECT col FROM table"),
  RaiseError("Failed to execute SQL statement: "
    + GetLastError())
);
```

[Back to Top]

Get

Declaration

```
type Get(string name)

type Get(string name[, int index1, int index2,... int indexN])

type Get(array name[, int index1, int index2,... int indexN])
```

Syntax

```
Get(<name>[, <index1>, <index2>,... <indexN>])
```

Required Parameters

- **`name:`** The name of a global variable, either a scalar or an array, or an array

Optional Parameters

- **`index1,... indexN:`** Indices specifying the desired element in the array or a sub-array

Description

Returns the value of a global variable with a given name. If passed an array or the name of a global variable that is an array, returns an element of the array. See also the complementary `Set()` function.

If the first argument is either an array or the name of a global variable that is an array, the function retrieves a specific element by its index (or indices for a multi-dimensional array such as a record-set) using the remaining arguments.

Arrays are zero-indexed; the first element is at index 0 and the last element (of the array `$array`) is at index `[Length($array)-1]`.

Attempting to retrieve an element beyond the end of the array will result in a return value of null.

Examples

```
// Returns the value of the global variable "Count"
Get("Count");

// Returns the third element of an array (0-based)
Get($arr, 2);

// For arrays, this is the same as previous,
// as "arr" is equivalent to $arr in the case of arrays
Get("arr", 2);

// Returns the n-th element of the m-th array in $arr
Get($arr, m-1, n-1);
```

[Back to Top]

GetChunkDataElement

Declaration

```
type GetChunkDataElement(string name)
```

Syntax

```
GetChunkDataElement(<name>)
```

Required Parameters

- **name:** The name of the chunk variable

Description

Returns the value of the chunk variable with a given name. A chunk variable is evaluated as each chunk of data is processed. An alternative method is to use the SCOPE_CHUNK syntax of the `Set()` function. See also the `SetChunkDataElement()` and `Set()` functions.

Examples

```
// If used in a transformation mapping, this sets
// the value of the chunk variable "CustomerFileName" to
// the results of a calculation using the value of the "Customer" field
// at the time of the chunking to create a filename for that chunk:

SetChunkDataElement("CustomerFilename", "customer_" + CustomerID + ".csv");

// This global variable would be available as a variable in the
// filenames field of the connection parameters of a target as:

[CustomerFilename]

// It would also be available in scripts in the same chunk as:

GetChunkDataElement("CustomerFilename");

// With each chunk created, a unique filename for that customer ID
// will be created, such as (depending on the values of CustomerID):
customer_1009.csv
customer_2019.csv
customer_5498.csv


// Returns the value of a chunk variable
result = GetChunkDataElement("Count");
```

[Back to Top]

GetHostByIP

Declaration

```
string GetHostByIP(string ipAddress)
```

Syntax

```
GetHostByIP(<ipAddress>)
```

Required Parameters

- **ipAddress:** A string with an IP address

Description

Resolves an IP address to a host name.

Examples

```
GetHostByIP("127.0.0.1");
```

[Back to Top]

GetInputString

Declaration

```
string GetInputString(type arg)
```

Syntax

```
GetInputString(<arg>)
```

Required Parameters

- **`arg:`** A global variable

Description

Returns the unformatted input as a string given a source global variable.

This is useful if the standard Jitterbit representation of a data type (such as a date or double) is not suitable and the "raw" input is required. If this method is called on an object that is not a source global variable, an empty string is returned.

Examples

```
// The input is too large for a Jitterbit double
// return the raw input instead
$SessionId = GetInputString
(root$transaction$body$GetMachineList$req$SessionID$)
```

[Back to Top]

GetLastOperationRunStartTime

Declaration

```
date GetLastOperationRunStartTime(string operationId)
```

Syntax

```
GetLastOperationRunStartTime(<operationId>)
```

Required Parameters

- **`operationId:`** A string reference path to an operation in the current project

Description

Returns the last date and time the given operation was run. The return value is a date (which includes the date and time). To be used with a single Private Agent only.

The operation used in this function call must be defined as an operation in the current project. For more information, see the instructions on inserting operations under the Operations section in Jitterbit Script.

The returned date is in UTC (without a specific time zone). Use the `ConvertTimeZone()` function to convert to a local time, as seen in the example below.

WARNING: This function is to be used only with a single Private Agent as it is not accurate when using Cloud Agents or multiple Private Agents.

Examples

```
op = "<TAG>operation:MyOperation</TAG>";
lastOpRun = GetLastOperationRunStartTime(op);
// Converting to a local time zone
$lorInMyTimeZone = ConvertTimeZone(lastOpRun,
    "UTC", "CST");
```

[Back to Top]

GetName

Declaration

```
string GetName(type arg)
```

Syntax

```
GetName(<arg>)
```

Required Parameters

- **`arg:`** A variable or global variable

Description

Returns the name of a variable or a global variable.

Certain functions return a named global variable array; if defined, this function retrieves the name of the value.

Examples

```
x = {a="var1", b="var2"};
GetName(x[0]);
// Returns the string "a"
GetName(x)[0];
// Also returns the string "a"
```

```
// The source is a simple text and [] represents the source element
values = GetSourceInstanceArray([]);
// Returns the first field name of the source element
GetName(values[0]);
```

[Back to Top]

GetOperationQueue

Declaration

```
array GetOperationQueue([string operationTag])
```

Syntax

```
GetOperationQueue([<operationTag>])
```

Optional Parameters

- **`operationTag:`** A string reference path to an operation in the current project; otherwise, all operations in the current project are used

Description

Returns the contents of the operation queue as an array. Only operations that the current user has read access for will be returned. To be used with a single Private Agent only.

The result is returned as an array of arrays, with these elements in each sub-array:

- `0`: Operation GUID (string)
- `1`: The `IsExecuting` flag (boolean)
- `2`: Timestamp (date) for when the operation was added to the queue
- `3`: Seconds in current status (integer)
- `4`: Operation instance GUID (string)
- `5`: Operation name (string)

The operation tag argument is optional. If the operation tag argument is present, only queue entries for that particular operation will be returned. For more information, see the instructions on inserting operations under the Operations section in Jitterbit Script.

WARNING: This function is to be used only with a single Private Agent as it is not accurate when using Cloud Agents or multiple Private Agents.

Examples

```
// Write the queue for a particular operation to
// the operation log:
op = "<TAG>operation:MyOperation</TAG>"
queue = GetOperationQueue(op);
n = Length(queue);
i = 0;
// Loop over the queue entries
While(i < n,
  WriteToOperationLog("Queue Entry: "
    + "GUID=" + queue[i][0]
    + "; IsExecuting=" + queue[i][1]
    + "; Added at: " + queue[i][2]);
  i++;
);
```

[Back to Top]

GetServerName

Declaration

```
string GetServerName()
```

Syntax

```
GetServerName()
```

Description

Returns the name of the machine that the agent is running on.

Examples

```
GetServerName();
// Returns the server name
```

[Back to Top]

GUID

Declaration

```
string GUID()
```

Syntax

```
GUID()
```

Description

Returns a GUID string (a globally unique identifier, also known as a universally unique identifier or UUID).

The format of the GUID is `xxxxxxxx-xxxx-Mxxx-Nxxx-xxxxxxxxxxxx`, where M is the version (4) and N is the variant (8).

Examples

```
GUID();
// Returns a string such as "c056f89d-1f45-458e-8b25-9ecf2ed10842"
```

[Back to Top]

IfEmpty

Declaration

```
type IfEmpty(type arg, type default)
```

Syntax

```
IfEmpty(<arg>, <default>)
```

Required Parameters

- **`arg:`** An argument to evaluate to see if it is null or an empty string
- **`default:`** Default value to return if `arg` is null or an empty string

Description

Returns the default value if the first argument is null or if the string representation of the argument is an empty string. Otherwise, it returns the first argument. This is a shortcut for an `If()` function statement:

```
If(IsNull(arg)||Length(arg)==0, default, arg)
```

See also the `IsNull()` function.

Examples

```
// If the variable "myDate" is null or empty,
// returns the current date, otherwise returns "myDate"
result = IfEmpty(myDate, Now());
```

[Back to Top]

IfNull

Declaration

```
type IfNull(type arg, type default)
```

Syntax

```
IfNull(<arg>, <default>)
```

Required Parameters

- **`arg:`** An argument to evaluate to see if it is null
- **`default:`** Default value to return if `arg` is null

Description

Returns the default value if the first argument is null, else returns the first argument.

This is a shortcut for an `If()` function statement:

```
If(IsNull(arg), default, arg)
```

See also the `IsNull()` and `IfEmpty()` functions.

Examples

```
// If the variable "myDate" is null,
// returns the current date, otherwise returns "myDate"
result = IfNull(myDate, Now());
```

[Back to Top]

InitCounter

Declaration

```
long InitCounter(type counter[, long initialValue])
```

Syntax

```
InitCounter(<counter>, <initialValue>)
```

Required Parameters

- **`counter:`** The name of a variable or a reference to a global variable to be used as a counter

Optional Parameters

- **`initialValue:`** The initial value to set the counter to; default is 0

Description

Initializes a counter, optionally passing an initial value. To be used with a Single Agent only.

If no initial value is set, the initial value is set to 0. The first argument is either the name of a variable or a reference to a variable (see the examples). This method needs to be called in single-threaded contexts only. Calling this method in a multi-threaded context will result in an error. See also Using Variables with Chunking under Operation Options.

WARNING: This function must be used only with a single Agent as it results in an error in a multiple Agent context.

Examples

```
// Initialize counter to 0 using the name of a global variable
InitCounter("counter");

// Initialize counter to 100 using a reference to a global variable
InitCounter($counter, 100);
```

[Back to Top]

InList

Declaration

```
int InList(type x[, type arg1, ... type argN])
```

Syntax

```
InList(<x>[, <arg1>, ... <argN>])
```

Required Parameters

- **`x:`** An element to be compared for a match

Optional Parameters

- **`arg1...argN:`** A series of arguments that x is to be compared against

Description

Checks for `x` in the list of arguments (`arg1` through `argN`). If a match (by value) is found, this function will return an integer representing the position of the match in the list, with the first position in the list being represented by the integer 1.

If the list contains more than one instance of x, this function returns the position of the first match (the match with the lowest position index). 0 is returned if the list does not contain a matching value or if only a single argument is supplied.

Examples

```
InList("x","a","b","c","x");
// Returns 4
InList("a","a","b","c","a");
// Returns 1

InList("x","a","b","c");
// Returns 0

InList("x");
// Returns 0
```

[Back to Top]

IsInteger

Declaration

```
bool IsInteger(type x)
```

Syntax

```
IsInteger(<x>)
```

Required Parameters

- **x:** An element to be evaluated

Description

Returns true if the argument is of type integer or long or can be converted to an integer or long without any loss of information.

Examples

```
$s="1";
IsInteger($s);
// Returns true
$s="1a";
IsInteger($s);
// Returns false
$s=12.12;
IsInteger($s);
// Returns false
$s=12.00;
IsInteger($s);
// Returns true
```

[Back to Top]

IsNull

Declaration

```
bool IsNull(type x)
```

Syntax

```
IsNull(<x>)
```

Required Parameters

- **x:** An element to be evaluated

Description

Returns true if the argument is null. Applies to database fields, variables, and functions that can return nulls.

See also the `IfNull()` and `IfEmpty()` functions for shortcuts that can be used instead of this function.

Examples

```
// If the "POHeader.Vendor_Code" is null,
// it returns a string "VC", otherwise it returns the code
If(IsNull(POHeader.Vendor_Code), POHeader.Vendor_Code, "VC")
```

[Back to Top]

IsValid

Declaration

```
bool IsValid(type x)
```

Syntax

```
IsValid(<x>)
```

Required Parameters

- **x:** An element to be evaluated

Description

Returns true if the evaluation of the argument results without an error.

Examples

```
IsValid(Date("abc"))
// Returns false, since the string "abc"
// cannot be successfully converted to a date

IsValid(3/0)
// Returns false, since division by 0
// is not permitted

IsValid(0/3)
// Returns true, since 0/3 is a legal expression
// evaluating to 0
```

[Back to Top]

Length

Declaration

```
int Length(type x)
```

Syntax

```
Length(<x>)
```

Required Parameters

- **x:** An element to be evaluated

Description

Returns the length of the input argument.

The behavior of this method depends on the argument type:

- **string:** the length of the string is returned
- **array:** the number of elements in the array is returned
- **binary data:** the number of bytes is returned
- For all other types, an attempt is made to convert the argument to a string, and the length of the resulting string is returned.
- If the argument cannot be converted to a string, or the argument is null or of an unknown type, 0 is returned.

Examples

```
// String length:
Length("Mississippi"); // returns 11

// Array length:
// Count the number of email address nodes
$nodes = SelectNodesFromXMLAny("cust:EmailAddress", Customer$Any#.,
"cust=urn:xmlns:25hoursaday-com:customer");
Length($nodes);

// Binary arguments:
Length(HexToBinary("b2082fee"));
// Returns 4, because the input is a 4-byte binary value

// Numeric arguments:
Length(1234567); // Returns 7
Length(123.45678); // Returns 9

// Miscellaneous:
Length(true); // Returns 1
Length(Now()); // Returns 19 since the default date format is "yyyy-MM-DD
hh:mm:ss"
Length(Null()); // Returns 0
```

[Back to Top]

Null

Declaration

```
null Null()
```

Syntax

```
Null()
```

Description

Returns null.

Examples

This function can be used to insert a null value into specific columns of a database.

[Back to Top]

Random

Declaration

```
int Random(int min, int max)
```

Syntax

```
Random(<min>, <max>)
```

Required Parameters

- **`min:`** Integer value of the minimum random number
- **`max:`** Integer value of the maximum random number

Description

Generates a random integer number between and including the given minimum and maximum values. See also the `RandomString()` function.

Examples

```
// Creates a random number from 0 to 9999999 (inclusive)
Random(0, 9999999);


// Creates a random number from 1 to 10
Random(1, 10);
// Returns a random 7-character string
// using the characters "0123456789"
RandomString(7, "0123456789");

// Returns a random 5-digit hexadecimal string
RandomString(5, "0123456789ABCDEF");

// Returns a random 7-digit integer string
// with no leading zeroes
RandomString(1, "123456789") +
  RandomString(6, "0123456789");
```

[Back to Top]

RandomString

Declaration

```
string RandomString(int len[, string chars])
```

Syntax

```
RandomString(<len>[, <chars>])
```

Required Parameters

- **`len:`** Length of the resulting random string

Optional Parameters

- **`chars:`** String containing characters that will be used in the resulting random string

Description

Generates a random string of the given length. By default, the function uses alphanumeric characters; the set that includes a-z, A-Z, and 0-9. See also the `Random()` function.

Examples

```
// Creates a random 5-digit hexadecimal string
RandomString(5, "0123456789ABCDEF");

// Creates a random 7-digit integer string
// with no leading zeroes
RandomString(1, "123456789") + RandomString(6, "0123456789");
```

[Back to Top]

ReadArrayString

Declaration

```
array ReadArrayString(string arrayString[, string type])
```

Syntax

```
ReadArrayString(<arrayString>[, <type>])
```

Required Parameters

- **`arrayString:`** A string representation of an array

Optional Parameters

- **`type:`** A string describing the type that the array string represents, such `"string"`, `"int"`, `"double"`, `"bool"`

Description

Reads a string that represents a single or multi-dimensional array.

The array is represented by enclosing array elements with a pair of curly brackets ('{' and '}'). Each array element can be an array or a scalar element separated by comma (','). The elements in an array must be either all scalars or all arrays.

The scalar value can be represented by a CSV string. Double quotes to enclose the string are optional, unless the string contains special characters such as ",{}\n (double quotes, comma, brace brackets, tabs, line breaks, or carriage returns). Inside the double-quoted string, each double quote must be escaped by two double quotes. The optional second argument is to specify the data type of the scalar value. The type is assumed to be string if it is not explicitly specified.

Examples

```
// One-dimensional array with four string values
ReadArrayString("{John,Steve,Dave,Eric}");

// One-dimensional array with three boolean values
ReadArrayString("{1,0,1}", "bool");

// Two-dimensional array
// The first array element is an array with three string values
// The second array element is an array with two string values
// The second element of the second array contains a trailing line break
ReadArrayString('{{abc,"a,b","a""b"},{"de","d
"}}');
```

[Back to Top]

RecordCount

Declaration

```
int RecordCount()
```

Syntax

```
RecordCount()
```

Description

Returns the instance number of the target loop that is currently being generated.

If it is called in a condition, it returns the instance number of the last instance that was generated. The first time this method is called in a loop it returns 0 (zero) if called in a condition; otherwise, it returns 1 (one). The counter is reset to 0 each time a new loop is started.

NOTE: This method has been deprecated and may be removed in a future version.

Use `SourceInstanceCount()` or `TargetInstanceCount()` instead. `TargetInstanceCount()` is equivalent to this method.

Examples

`RecordCount()` returns a value of 5 while generating the fifth line in a target loop node.

[Back to Top]

ReRunOperation

Declaration

```
bool ReRunOperation([bool runSynchronously])
```

Syntax

```
ReRunOperation([<runSynchronously>])
```

Optional Parameters

- **`runSynchronously:`** Flag to indicate if the operation should be run synchronously (the default) or asynchronously

Description

Re-runs the current operation.

The behavior of this method with respect to return value and global variables is identical to the `RunOperation()` function. See that function for a description of how re-running the operation synchronously or asynchronously affects global global variables.

> **WARNING:** Since this is a recursive call, it is essential that there is a stop condition, most likely including the `CancelOperation()` function. Otherwise you will end up in an infinite loop of operation calls.

Examples

```
ReRunOperation();
// Re-runs the current operation synchronously
ReRunOperation(false);
// Re-runs the current operation asynchronously
```

[Back to Top]

RunOperation

Declaration

```
bool RunOperation(string operationId[, bool runSynchronously])
```

Syntax

```
RunOperation(<operationId>[, <runSynchronously>])
```

Required Parameters

- **`operationId:`** A string reference path to an operation in the current project

Optional Parameters

- **`runSynchronously:`** Flag to indicate if the operation should be run synchronously (the default) or asynchronously

Description

Runs an operation synchronously or asynchronously, with synchronous being the default.

The operation used in this function call must be defined as an operation in the current project. For more information, see the instructions on inserting operations under the Operations section in Jitterbit Script.

Running Synchronously

If **`run_synchronously=true`** the called operation and any success/failure operations will run inside the current operation and the current operation will wait for the whole operation chain to finish. All global variables are inherited by the called operation and any changes to the global variables will be reflected in the current operation. This is the default behavior if the second argument is not supplied. Returns `false` if the called operation resulted in a failure.

If **`run_synchronously=false`** this method puts the called operation on the Jitterbit processing queue to be processed once any operations that come before it have been processed. All global variables are inherited by the called operation but changes to those variables will not be reflected in the current operation. The current operation will continue to run independently of the called operation and there is no guarantee as to which operation will finish first. Returns `false` if the called operation could not be added to the queue. With the asynchronous mode, these global variables are passed to the called operation by

value rather than by reference, which insures that any changes to the variables are not reflected in any other operation.

If the function returns `false` to indicate a failure or if the called operation could not be queued, call `GetLastError()` to retrieve the error message.

Examples

```
// Runs the "MyOp"
RunOperation("<TAG>operation:MyOp</TAG>");
```

[Back to Top]

RunPlugin

Declaration

```
bool RunPlugin(string pluginId)
```

Syntax

```
RunPlugin(<pluginId>)
```

Required Parameters

- **`pluginId:`** A string reference path to a plugin in the current project

Description

Runs a specified plugin and then continues execution of the current script.

In the Cloud Studio UI, only those plugins that can be run inside a script are displayed; plugins that run on activities are hidden. For more information, see the instructions on inserting plugins under the Plugins section in Jitterbit Script.

Returns `true` if the plugin completes without errors. Returns `false` if the plugin could not be run or the plugin implementation itself returned an error. Call `GetLastError()` to retrieve the error message.

Examples

```
// Runs the Jitterbit HMACSHA1Generator plugin
RunPlugin("<TAG>plugin:http://www.jitterbit.com/plugins/pipeline/user
/HMACSHA1Generator</TAG>");
```

[Back to Top]

RunScript

Declaration

```
string RunScript(string scriptId[, type var1, type var2, ..., type varN])
```

Syntax

```
RunScript(<scriptId>[, <var1>, <var2>, ..., <varN>])
```

Required Parameters

- **`scriptId:`** A string reference path to a script in the current project

Optional Parameters

- **`var1...varN:`** Additional variables, to be passed to the script being called

Description

Runs the specified script and then continues execution of the current script. This method returns, on success, the return value of the called script as a string.

The script used in this function call must be defined as either a Jitterbit Script or JavaScript in the current project. For more information, see the instructions on inserting scripts under Jitterbit Script or JavaScript.

A list of values can be passed to a `RunScript` function as input variables. The script will create local variables using these values with default names such as `_1, _2 ...`.

If comprehensive names are preferred, the `ArgumentList()` function can be used to map a list of local variable names to the list of `_1, _2 ...`. See the `ArgumentList()` function for examples.

WARNING: The return type is a `string`. All other types are converted to their string equivalent. Arrays are returned as a `string`; if they contain scalar values, they can be converted to an array using the `ReadArrayString()` function. (A multidimensional array can also be converted by `ReadArrayString()`.)

WARNING: If the called script is a JavaScript script, it will not be passed any arguments. Any arguments included in the call to the `RunScript()` function will not be declared or available in the JavaScript script. The only method for passing information into a JavaScript script is to use global variables; those are variables prefaced with a $ symbol. These values can be made available inside the JavaScript script using the `Jitterbit.GetVar()` function.

Examples

```
// Runs the script "CalculateStuff"
RunScript("<TAG>script:CalculateStuff</TAG>");

RunScript("<TAG>script:CalculateStuff</TAG>",
   "abc", 1);

// Sends the script "CalculateStuff" the
// string "abc" and the number 1
// Inside "CalculateStuff", these will be
// available as _1 and _2
```

[Back to Top]

Set

Declaration

```
type Set(string name, type value)

type Set(string name, type value, int index1[, int index2, ..., int
indexN])

type Set(array name, type value, int index1[, int index2, ..., int indexN])
```

Syntax

```
Set(<name>, <value>[, <index1>, <index2>, ..., <indexN>])
```

Required Parameters

- **`name:`** The name of a global variable, either a scalar or an array
- **`value:`** A value to be assigned to the global variable

Optional Parameters

- **`index1...indexN:`** Index or indices used to describe the position of an element if setting an element in an array

Description

Sets the value of a global variable with a given name to a value, and returns the value. See also the complementary `Get()` function.

First Form: Scalars

In the first form, a string name of a global variable is set using the name and value supplied.

(Though a *local* variable can be passed as a reference, it is not advised as results can be inconsistent. Local variables are not intended to be set through this mechanism.)

See the examples below.

Second and Third Forms: Arrays

In the second and third forms, additional arguments supply the indices for setting an element in an array.

If the first argument is an array (or the name of a global variable that is an array), you can set the value of an array element by specifying its index (or indices for multi-dimensional arrays) as additional arguments.

To append data to an array, pass either a negative index value or the size of the array. The size can be determined using the `Length()` function as `Length($array)`.

Arrays are zero-indexed; the first element is at index 0 and the last element (of the array `$array`) is at index `[Length($array)-1]`. Arrays can be created with either the `Array()` or `ReadArrayString()` functions.

Attempting to set an element beyond the end of the array will result in additional null-value elements being added to the array as required to pad the array to the correct size.

SCOPE_CHUNK Prefix Syntax

Setting a variable name that is prefixed with the phrase `SCOPE_CHUNK` will create a global variable that is evaluated as each chunk of data is processed. This can be used in the creation of global variables whose value is unique to a particular chunk, and can then identify that chunk when a filename or record is created at a target. See also the `GetChunkDataElement()` and `SetChunkDataElement()` functions as an alternative method that allows the use of other variable names.

Examples

```
// Scalars:
// All of these forms are equivalent:
// they increase the global variable "count" by 1
result1 = Set("count", Get("count")+1);
$count++;
$count = $count + 1;

// Arrays:
// Appending a value to the array "arr"
// These are equivalent
Set($arr, "value", -1);
Set($arr, "value", Length($arr));

// Set the n:th entry in an array "arr"
// to the string "value"
Set($arr, "value", n-1);

// Set the n:th entry of the m:th array
// of "record_set"
Set($record_set, "value", m-1, n-1);

// SCOPE_CHUNK Prefix:
```

```
// Example from a mapping using the SCOPE_CHUNK syntax to
// create a global variable that is unique in value to a
// particular chunk.
// It uses the field "CustomerID" to identify the chunk:

Set("SCOPE_CHUNK_CustomerID",
    "customer_"+CustomerID+".csv");

// This variable will be available in the filenames field of
// the connection parameters of a target as:

[SCOPE_CHUNK_CustomerID]

// With each chunk created, a unique filename for that
// customer ID will be created, such as (depending on the
// values of Customer ID):
customer_1009.csv
customer_2019.csv
customer_5498.csv
```

[Back to Top]

SetChunkDataElement

Declaration

```
type SetChunkDataElement(string name, type value)
```

Syntax

```
SetChunkDataElement(<name>, <value>)
```

Required Parameters

- **`name:`** The name of the chunk variable
- **`value:`** The value to set the chunk variable to

Description

Sets the value of a specified chunk variable, and returns the value. A chunk variable is evaluated as each chunk of data is processed. An alternative method is to use the `SCOPE_CHUNK` syntax of the `Set()` functi on.

See also the `GetChunkDataElement()` and `Set()` functions.

Examples

```
// If used in a transformation mapping, this sets
// the value of the chunk variable "CustomerFileName"
// to the results of a calculation using the value of
// the "Customer" field at the time of the chunking
// to create a filename for that chunk:

SetChunkDataElement("CustomerFilename",
    "customer_"+CustomerID+".csv");

// This global variable would be available as a
// variable in the filenames field of the connection
// parameters of a target as:

[CustomerFilename]

// It would also be available in scripts in the same
// chunk as:
```

```
GetChunkDataElement("CustomerFilename");

// With each chunk created, a unique filename for that
// customer ID will be created, such as (depending on
// the values of Customer ID):
customer_1009.csv
customer_2019.csv
customer_5498.csv
```

[Back to Top]

Sleep

Declaration

```
void Sleep(int seconds)
```

Syntax

```
Sleep(<seconds>)
```

Required Parameters

- **seconds:** The integer number of seconds to suspend the current operation

Description

Causes execution to be suspended for a specified number of seconds.

Examples

```
// Sleeps the current operation for 1 minute
Sleep(60);
```

[Back to Top]

SourceInstanceCount

Declaration

```
int SourceInstanceCount()
```

Syntax

```
SourceInstanceCount()
```

Description

Returns the instance count of the most recent generator.

The value is independent of whether the target instance has been generated or not; the same value is returned if called in a condition script or in a mapping script.

When the first source instance is used as the generator, 1 is returned, then 2, and so forth.

See also the `TargetInstanceCount()` function.

Examples

```
// Returns the instance count of the most recent generator
currentSourceInstance = SourceInstanceCount();
```

[Back to Top]

TargetInstanceCount

Declaration

```
int TargetInstanceCount()
```

Syntax

```
TargetInstanceCount()
```

Description

Returns the instance count of a generating target loop node.

When called in a condition it returns the number of target instances that have been generated so far for the current loop node. The number returned by this method will be one less if it is called in a condition since, in a condition, it is not known if the current target instance will be generated or not.

When the first target instance is generated, 1 is returned, then 2, and so forth. If called in a condition, the sequence will instead be 0, 1, and so forth.

See also the `SourceInstanceCount()` function.

Examples

```
// Returns the instance count of the most recent target generator
currentTargetInstance = TargetInstanceCount();
```

[Back to Top]

WaitForOperation

Declaration

```
void WaitForOperation(string operationId[, int timeOutSec, int
pollIntervalSec])
```

Syntax

```
WaitForOperation(<operationId>[, <timeOutSec>, <pollIntervalSec>])
```

Required Parameters

- **`operationID:`** A string reference path to an operation in the current project

Optional Parameters

- **`timeOutSec:`** A local variable
- **`pollIntervalSec:`** A local variable

Description

Stops execution of a script or mapping until all instances of the specified operation currently in the operation queue have finished processing. This method is useful if you want to add many instances of an operation to the queue for parallel processing and then wait for all of them to finish.

The operation used in this function call must be defined as an operation in the current project. For more information, see the instructions on inserting operations under the Operations section in Jitterbit Script.

Note:

- For each operation (identified by its `operationID`) that is to be waited on, a call must be made to this method.
- Operation instances that are added (by calls to the `RunOperation()` function) after this call is made are not waited for.
- The current user needs to have read access for the operation being waited on.

The second (optional) argument is the timeout in seconds. The default timeout is 1 hour (3600 seconds) and if all the operations have not finished within this time, an error will be thrown. If you expect your operations to run for a longer time during normal conditions you must increase the timeout. You can handle this error by using the `Eval()` function.

The third (optional) argument is the poll interval in seconds. The poll interval is the time between operation queue checks. The default poll interval is 10 seconds. The default will not be a significant performance hit but if your operations are expected to run for a very long time, you may want to increase the poll interval.

Examples

```
// Add ten operation instances to the queue
// and wait for all of them to finish
i = 0;
while(i < 10,
  RunOperation("<TAG>operation:MyOperation</TAG>", false);
  i++;
);

WaitForOperation("<TAG>operation:MyOperation</TAG>");
```

[Back to Top]