# Dictionary and Array Functions

## Introduction

Dictionary and array functions provide storage structures for information in scripts. See the *Related Functions* in the side panel for additional functions that can be used with arrays.

### Difference Between Arrays and Dictionaries

Though very similar in many ways as a method to store information, arrays and dictionaries differ in these important characteristics:

- Arrays
  - Elements are ordered
  - Elements are retrieved by position
- Dictionaries
  - Elements are stored without regard to order, and have no intrinsic order
  - Elements are retrieved by a key

The syntax for setting and retrieving an element is similar (both can use the square-bracket notation) but the arrays have no keys associated with elements, and must store all items sequentially, with no gaps. Dictionaries can be sparse, with keys populated only as required.

Which one to use depends on the nature of the information. Dictionaries lend themselves to information that varies in how it is retrieved. Arrays are often better for ordered sets.

```
// Example of an array
$org.lookup.currency = Array();
$org.lookup.currency[1]="USD";
$org.lookup.currency[2]="EUR";

// Example of a dictionary
$common.dictionary.message = Dict();
AddToDict($common.dictionary.message, 'OP_START', 'Start');
AddToDict($common.dictionary.message, 'OP_END', 'End');
AddToDict($common.dictionary.message, 'API_PARMS', 'API Parameters');
```

### Finding Values in Dictionaries and Arrays

For dictionaries, the `HasKey()` function lets you test if a key is present. For an array, you can use the `FindValue()` function in a similar fashion by passing two instances of the array to the function:

```
arr = {1, 2, 3};
value = 1;
t = (FindValue(value, arr, arr) == value);
// t will be 1 (true)

value = 4;
t = (FindValue(value, arr, arr) == value);
// t will be 0 (false)
```

## AddToDict

### Declaration

```
bool AddToDict(dictionary dict, string key, type arg)
```

### Syntax

```
AddToDict(<dict>, <key>, <arg>)
```

#### Required Parameters




**Search in This Topic**

**Related Functions**

**Related Topics**

- Cloud Studio
- Functions
- Scripts


Last updated: May 15, 2019

- **`dict:`** An existing dictionary
- **`key:`** Key to be used in the dictionary
- **`arg:`** Value to be placed at that key

### Description

Adds a value to a dictionary for a specific key.

The key must be a string or have a string representation, with **null keys not allowed**. Any *value* is allowed, even null values.

Returns `true` if the value was added and the new key added to the dictionary or `false` if the key already existed and the value at that key was instead updated. If the first argument is not defined or is not a dictionary, it will be initialized to an empty dictionary before the value is added.

See also the `Dict()` function.

### Examples

```
// Initialize a dictionary prior to use
d = Dict();


// Adds a value to a dictionary "d" if a key doesn't already exist
If(!HasKey(d, "key"), AddToDict(d, "key", "value"));

// To directly assign a key-value, use this syntax:
d["key"] = "value";
```

**WARNING:** Use of a null key will cause an error and stop the execution of an operation.

[Back to Top]

## Array

### Declaration

```
array Array()
```

### Syntax

```
Array()
```

### Description

Creates an empty array. Though arrays don't need to be initialized prior to use, this method can be used to be explicit or to reset an already-existing array. Arrays are zero-based, and values are retrieved using indexes.

### Examples

```
// Create an empty array and set values
arr = Array();
arr[0] = "value1";
arr[1] = "value2";

// Create an array and set the first value directly:
arr2[0] = "value";

// Arrays can be created with { } syntax:
arr3 = {};
```

[Back to Top]

## Collection

### Declaration

```
array Collection()
```

### Syntax

```
Collection()
```

### Description

An alias for `Array()`. See the function `Array()`.

[Back to Top]

## CollectValues

### Declaration

```
array CollectValues(dictionary dict, array names)
```

### Syntax

```
CollectValues(<dict>, <names>)
```

#### Required Parameters

- **`dict:`** An existing dictionary
- **`names:`** An array of keys to be looked up in the dictionary

### Description

Returns an array containing the values corresponding to the names in the argument array, returned in the same order as the keys in the array.

### Examples

```
// Retrieve a map from a source instance
map = GetSourceInstanceMap([Values$Value.]);
names = Array();
names[0] = "Attr1";
names[1] = "Attr2";
// Array containing the values of
// the attributes named in the names array
values = CollectValues(map, names);
```

[Back to Top]

## Dict

### Declaration

```
dictionary Dict()
```

### Syntax

```
Dict()
```

### Description

Creates an empty dictionary. A dictionary is a collection of name-value pairs where the value is retrieved based on a string key value. Any value is allowed, even null values. The key must be a string or have a string representation. Null keys are not allowed. See also the `AddToDict()` function.

**WARNING:** Dictionaries **must be initialized** prior to being used.

**WARNING:** Use of a null key will cause an error and stop the execution of an operation.

### Examples

```
// Creating a dictionary "myDictionary"
myDictionary = Dict();

// Assigning a value ("text") to the key "myKey"
myDictionary["myKey"] = "text";
// Using the AddToDict function instead
AddToDict(myDictionary, "myKey", "text");
```

[Back to Top]

## GetKeys

### Declaration

```
array GetKeys(dictionary dict)
```

### Syntax

```
GetKeys(<dict>)
```

### Required Parameters

- **`dict:`** An existing dictionary

## Description

Returns an array of the keys in a dictionary. The argument must be an existing dictionary.

## Examples

```
// Retrieving the keys from the dictionary "myDictionary"
// using GetKeys and then looping through the array
// to retrieve all the values in the dictionary
// and writing them to the Operation Log
keys = GetKeys(myDictionary);
i=0;
While(i < Length(keys),
 WriteToOperationLog(myDictionary[keys[i]]);
 i++;
);
```

[Back to Top]

# GetSourceAttrNames

## Declaration

```
array GetSourceAttrNames(node n)
```

## Syntax

```
GetSourceAttrNames(<n>)
```

### Required Parameters

- **`n:`** A node in a transformation

## Description

Returns an array containing the names of the attributes of a node, in the order that the attributes appear in the node.

Compare with the function `GetSourceInstanceMap()` which returns a map of the keys (the attributes) and values for a node.

## Examples

```
// Appends all the attributes together without having
// to explicitly reference their source data elements
map = GetSourceInstanceMap([Values$Value.]);
names = GetSourceAttrNames([Values$Value.]);
i = 0;
n = Length(names);
str = "";

While(i < n,
  str = str + map[names[i]];
  i++;
);
```

[Back to Top]

# GetSourceElementNames

## Declaration

```
array GetSourceElementNames(node n)
```

## Syntax

```
GetSourceElementNames(<n>)
```

### Required Parameters

- **n:** A node in a transformation

## Description

Returns an array containing the names of the simple sub-elements for a node in the order that the attributes appear in the node.

Compare with the function `GetSourceInstanceElementMap()` which returns a map of a node.

## Examples

```
// Appends all the attributes together without having
// to explicitly reference their source data elements
map = GetSourceInstanceElementMap([Values$Value.]);
names = GetSourceElementNames([Values$Value.]);

// The values as an array of values from the sub-elements
values = CollectValues(map, names);
```

[Back to Top]

# GetSourceInstanceArray

## Declaration

```
array GetSourceInstanceArray(node n)
```

## Syntax

```
GetSourceInstanceArray(<n>)
```

### Required Parameters

- **n:** A node in a transformation

## Description

Returns an array containing the attribute's value from an element node. The value in the array is labeled with the attribute's name, and can be retrieved either by its index or by its name as in a dictionary data element.

As an alternative to this function, see `GetSourceInstanceMap()`.

## Examples

```
arr = GetSourceInstanceArray([Values$Value.]);
// To retrieve the value of the "Attr1" attribute:
// If "Attr1" is the first attribute name, you can use:
v = arr[0];
// Or, you can use the name "Attr1":
v = arr["Attr1"];
```

[Back to Top]

## GetSourceInstanceElementArray

### Declaration

```
array GetSourceInstanceElementArray(node n)
```

### Syntax

```
GetSourceInstanceElementArray(<n>)
```

#### Required Parameters

- **n:** A node in a transformation

### Description

Returns an array containing the sub-element's value from an element node. The value in the array is labeled with the name of the sub-element, and can be retrieved either by its index or by its name as in the dictionary data element.

As an alternative to this function, see `GetSourceInstanceElementMap()`.

### Examples

```
arr = GetSourceInstanceElementMap([Values$Value.]);
// To retrieve the value of the "e1" sub-element:
// If "el" is the first sub-element, you can use:
v = arr[0];
// Or, you can use the name "e1":
v = arr["e1"];
```

[Back to Top]

## GetSourceInstanceElementMap

### Declaration

```
dictionary GetSourceInstanceMap(node n)
```

### Syntax

```
GetSourceInstanceMap(<n>)
```

#### Required Parameters

- **n:** A node in a transformation

### Description

Returns a dictionary (a map) containing the attribute name and its value from an element node.

As an alternative to this function, see `GetSourceInstanceArray()`.

### Examples

```
map = GetSourceInstanceMap([Values$Value.]);
v = map["Attr1"];
// Returns the value of the "Attr1" attribute
```

[Back to Top]

## GetSourceInstanceMap

### Declaration

```
dictionary GetSourceInstanceMap(node n)
```

### Syntax

```
GetSourceInstanceMap(<n>)
```

#### Required Parameters

- **`n:`** A node in a transformation

### Description

Returns a dictionary (map) containing the attribute name and its value from an element node.

As an alternative to this function, see `GetSourceInstanceArray()`.

### Examples

```
map = GetSourceInstanceMap([Values$Value.]);
map["Attr1"];
// Returns the value of the "Attr1" attribute
```

[Back to Top]

## HasKey

### Declaration

```
bool HasKey(dictionary dict, string key)
```

### Syntax

```
HasKey(<dict>, <key>)
```

#### Required Parameters

- **`dict:`** An existing dictionary

- **key:** A key to check in the dictionary

### Description

Checks whether a dictionary contains a specified key. Returns `false` if the first argument is not a dictionary or if the key was not found. As an equivalent function that works for arrays, see the examples of the `FindValue()` function.

### Examples

```
// Update the value in a dictionary "myDictionary"
// only if the key already exists
If(HasKey(myDictionary, "key"),
    myDictionary["key"] = "value");
```

[Back to Top]

## Map

### Declaration

```
dictionary Map()
```

### Syntax

```
Map()
```

### Description

An alias for `Dict()`. See the function `Dict()`.

[Back to Top]

## MapCache

### Declaration

```
string MapCache(dictionary dict, string key, string value)
```

### Syntax

```
MapCache(<dict>, <key>, <value>)
```

#### Required Parameters

- **dict:** An existing dictionary
- **key:** A key to check in the dictionary
- **value:** A value to be used (and stored) if the key is not in the dictionary

### Description

This function caches a key/value pair to a dictionary. If the key already exists in the dictionary, the corresponding value will be returned; otherwise, the third argument will be evaluated, and that value would be stored in the dictionary for the key.

### Examples

```
// Creating a dictionary "myDictionary"
// and caching values in it,
// if they don't already exist
myDictionary = Dict();
i=0;
a=MapCache(myDictionary, "key1", ++i);
b=MapCache(myDictionary, "key2", ++i);
c=MapCache(myDictionary, "key1", ++i);
result = i;
// The result will be
// result=2, a=1, b=2, and c=1
// as c's value will be retrieved
// rather than incremented
```

[Back to Top]

# ReduceDimension

## Declaration

```
array ReduceDimension(array arrayMultiD)
```

## Syntax

```
ReduceDimension(<arrayMultiD>)
```

### Required Parameters

- **`arrayMultiD:`** A multi-dimensional array

## Description

Given a multi-dimensional array with *n* dimensions, the function returns an array with *n-1* dimensions. The lowest dimension of the input array will disappear, and their members will be collected to the next level dimension.

## Examples

```
prices = ReduceDimension(orders$company.order#.detail#.price);
// If orders$company.order#.detail#.price
// is a 2-dimensional source data element array,
// the function will return a one-dimensional array

a = {{"0_0", "0_1"}, {"1_0", "1_1"}};
result = ReduceDimension(a);
// result will be {0_0, 0_1, 1_0, 1_1}
```

[Back to Top]

# RemoveKey

## Declaration

```
bool RemoveKey(dictionary dict, string key)
```

## Syntax

```
RemoveKey(<dict>, <key>)
```

### Required Parameters

- **`dict:`** An existing dictionary
- **`key:`** A key to check in the dictionary

## Description

Removes a key-value pair with a specific key from a dictionary. The key must be a string or have a string representation, and null values are not allowed. Returns `true` if the key-value pair was removed and `false` if the key didn't exist.

## Examples

```
// Removes a key-value pair if it
// exists in dictionary "myDictionary"
If(HasKey(myDictionary, "key"),
    RemoveKey(myDictionary, "key"));
```

[Back to Top]

# SortArray

## Declaration

```
void SortArray(array arrayToSort[, bool isAscending])

void SortArray(array arrayToSort, int index[, bool isAscending])
```

## Syntax

```
SortArray(<arrayToSort>[, <isAscending>])

SortArray(<arrayToSort>, <index>[, <isAscending>])
```

### Required Parameters

- **`arrayToSort:`** An array to be sorted
- **`index:`** For multi-dimensional arrays, index of array to be sorted on (default 0)

### Optional Parameters

- **`isAscending:`** If the array is to be sorted in ascending order (the default)

## Description

Sorts an array by reference. The return value is undefined and should be ignored.

In the first form, the second parameter (optional) specifies the sort order.

In the second form, for multi-dimentional arrays, the function sorts the array according to a zero-based index specified in the second parameter (required). In the second form, the third argument (optional) specifies the sort order.

The default sort order is ascending in both forms.

After the function returns, the array will be sorted in-place. If specified, it will be sorted according to the index.

Multiple sorting of the same array is possible by applying the `SortArray()` function repeatedly.

## Examples

```
// Sorting a one-dimensional array
arr1 = {"Carol", "Bob", "Alice"};
SortArray(arr1);
// arr1 is now {Alice, Bob, Carol}

// Sorting a two-dimensional array
arr2 = {{"a", 20, 1}, {"bc", 7, 12}, {"x", 20, 13}, {"d", 5, 4}};

// Sort arr2, order by third column, descending
SortArray(arr2, 2, false);
// arr2 is now {{x, 20, 13}, {bc, 7, 12}, {d, 5, 4}, {a, 20, 1}}

// Sort arr2, order by second column, ascending
SortArray(arr2, 1);
// arr2 is now {{d, 5, 4}, {bc, 7, 12}, {x, 20, 13}, {a, 20, 1}
```

[Back to Top]