Instance Functions

These functions are intended for use in transformation mappings (or scripts called during mappings), as they use the instances of data elements (sources and targets) found in the mappings.

For details on the syntax used to describe the entity paths of data elements and for advanced usage of the instance functions, see Transformation Basics.

Known Issue

There is a known issue with using the instance functions in Cloud Studio. The hash symbol (#), used to indicate an instance, is being inserted improperly into the data element path. As a workaround, you can adjust the position of the hash manually, but upon reopening a script using this function, the symbol will need to be manually repositioned again.

The intended behavior is that the hash symbol (#), indicating an instance, is automatically inserted into the appropriate location within a data element path when a mapping that uses one of these functions is saved. Such as:

```
SumString(_Root$customer.contact#.Email, ",", true);
```

Advanced Usage

The instance functions can, in general, be nested inside each other. As they are nested, the results will move up the hierarchy, and encompass more results. These functions can return either a single value or an array of values, depending on the context in which they are used.

For example: a `Sum()` function that has a `Count()` function inside will add up the results of each invocation of the `Count()` function, and produce a total:

```
Sum(Count(_Root$customer.sales#.items#.ID));
```


Related Topics

- Cloud Studio
- Functions
- Scripts
- Transformations

Last updated: Dec 20, 2019

Count

Declaration

```
int Count(type de)

int Count(array arr)
```

Syntax

```
Count(<de>)

Count(<arr>)
```

Required Parameters

- **`de:`** (First form) An entity path to instances of a data element in a source or a target
- **`arr:`** (Second form) An array; all the elements of the array must be of the same type

Description

Counts all instances of a data element at a particular hierarchical level in a source or target, where that data element contains a valid value (and is not null).

The function returns either an integer or an array of instances, depending on the context in which it is called.

WARNING: There is a known issue with using this function in Cloud Studio. The hash symbol (#), used to indicate an instance, is being inserted improperly into the data element path. As a workaround, you can adjust the position of the hash manually, but upon reopening a script using this function, the symbol will need to be manually repositioned again.

Examples

Assume a database contains a field "Quantity" in a table "Items" that is a child of "POHeader" (a purchase order), and that there are many items within a *POHeader*. Then, this statement returns the number of item rows for a particular *POHeader* that have values in the *Quantity* column that are not null:

```
Count(POHeader.Items#.Quantity);
```

In this case, assume a datafile with multiple instances in it, with *customers* that have *sales* that have *items*; and each *item* has an *ID* field. This statement will count how many different items are in each sale and use the `Sum()` function to add together all of the items returned for each sale; this will be the total number of different items purchased:

```
Sum(Count(_Root$customer.sales#.items#.ID));
```

[Back to Top]

CountSourceRecords

Declaration

```
int CountSourceRecords()
```

Syntax

```
CountSourceRecords()
```

Description

Returns the number of source instances for a target node, when the target node is referring to a parent of a mapping field.

If the target node is not a loop node, the function returns 1. See also the `SourceInstanceCount()` function.

NOTE: The streaming mode of a *Flat-to-Flat* transformation would not be affected by the use of this function, while the streaming mode would be turned off for an *XML-to-Flat* transformation.

WARNING: There is a known issue with using this function in Cloud Studio. The hash symbol (#), used to indicate an instance, is being inserted improperly into the data element path. As a workaround, you can adjust the position of the hash manually, but upon reopening a script using this function, the symbol will need to be manually repositioned again.

Examples

Assume a source with *customer* records that have instances of *sales* with instances of *items* with a field *Type*:

```
// This statement shows the instance count of a record compared
// to the total number of source records
"Record " + SourceInstanceCount() + " of " + CountSourceRecords();
```

[Back to Top]

Exist

Declaration

```
bool Exist(type v, type de)

bool Exist(type v, array arr)
```

Syntax

```
Exist(<v>, <de>)

Exist(<v>, <arr>)
```

Required Parameters

- **`v:`** A value to be found
- **`de:`** (First form) An entity path to instances of a data element in a source or a target
- **`arr:`** (Second form) An array; all the elements of the array must be of the same type and be the same type as v

Description

Checks for the existence of a value (`v`) in instances of a data element (`de`) or an array (`arr`) and returns true (or false) depending if it is found.

The function returns either a boolean or an array of instances, depending on the context in which it is called.

WARNING: There is a known issue with using this function in Cloud Studio. The hash symbol (#), used to indicate an instance, is being inserted improperly into the data element path. As a workaround, you can adjust the position of the hash manually, but upon reopening a script using this function, the symbol will need to be manually repositioned again.

Examples

Assume a source with *customer* records that have instances of *sales* with instances of *items* with a field *Type*:

```
// Returns if true if the value "subscription" is
// found in any instances of a field "customer.sales.items.Type"
// at the level of "sales":
Exist("subscription",_Root$customer.sales.items#.Type);

// To test this at the next highest level of the hierarchy,
// at the level of the customer,
// enclose this in a nested "Exist", testing for "true":
Exist(true, Exist("subscription",_Root$customer.sales#.items#.Type));

// The last statement answers, at the customer level, if a customer
// has any items in any sales with a Type field equal to "subscription"
```

[Back to Top]

FindByPos

Declaration

```
type FindByPos(int pos, type de)

type FindByPos(int pos, array arr)
```

Syntax

```
FindByPos(<pos>, <de>)

FindByPos(<pos>, <arr>)
```

Required Parameters

- **`pos:`** The index (from which occurrence; 1-based) to retrieve the value
- **`de:`** (First form) An entity path to instances of a data element in a source or a target; or
  `arr:` (Second form) An array; all the elements of the array must be of the same type

Description

Returns the value of a data element from an instance that occurs multiple times. It can also be used to return an element of an array, in a 1-based fashion.

If a negative number is specified for the occurrence or array, counting will begin from the last row or element. Note that the index is 1-based.

WARNING: There is a known issue with using this function in Cloud Studio. The hash symbol (#), used to indicate an instance, is being inserted improperly into the data element path. As a workaround, you can adjust the position of the hash manually, but upon reopening a script using this function, the symbol will need to be manually repositioned again.

Examples

```
// Assume a database has a child-parent relationship
// where for each parent the child occurs multiple times

// To retrieve the second child, use:
FindByPos(2, ParentTab.ChildTab#.Value$);

// To retrieve the last child, use:
FindByPos(-1, ParentTab.ChildTab#.Value$);
```

[Back to Top]

FindValue

Declaration

```
type FindValue(type0 v, type1 de1, type2 de2)
```

Syntax

```
FindValue(<v>, <de1>, <de2>)
```

Required Parameters

- **`v:`** A value to be searched for
- **`de1:`** An entity path to instances of a data element in a source or a target, to be used as a match
- **`de2:`** An entity path to instances of a data element in a source or a target, to be returned if a match is found

Description

Searches multiple instances of a data element (`de1`) looking for the value specified in `v`. If the function finds the value, it returns the value in the field specified in the third parameter (`de2`) for that found instance. If the value is not found, the function returns null. See also the `HasKey()` function.

> **WARNING:** There is a known issue with using this function in Cloud Studio. The hash symbol (#), used to indicate an instance, is being inserted improperly into the data element path. As a workaround, you can adjust the position of the hash manually, but upon reopening a script using this function, the symbol will need to be manually repositioned again.

Examples

This statement will search the instances of `B` under `A` and check the contents of `field1`. It will select the first instance of `B` it finds where `field1` contains "`ID`", and then return the value of `field2` from that same instance:

```
FindValue("ID", A.B#.field1, A.B#.field2);
```

These statements show how to implement a test of an array for inclusion of a value. It searches an array for a value, and returns true if found and false if not. It is the array equivalent to the dictionary `HasKey()` function. Note that two instances of the same array are passed 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)
```

[Back to Top]

GetInstance

Declaration

```
type GetInstance()
```

Syntax

```
GetInstance()
```

Description

This function returns the instance data element which was defined by calling a `SetInstances()` function during the generation of the parent. As an alternative to this function, see the `ArgumentList()` function.

> **WARNING:** There is a known issue with using this function in Cloud Studio. The hash symbol (#), used to indicate an instance, is being inserted improperly into the data element path. As a workaround, you can adjust the position of the hash manually, but upon reopening a script using this function, the symbol will need to be manually repositioned again.

Examples

Assume one of the parent mappings of a transformation contains these statements:

```
...
r1=DBLookupAll("<TAG>endpoint:database/My Database</TAG>",
    "SELECT key_name, key_value, key_type FROM key_values");
SetInstances("DETAILS", r1);
r2={"MS","HP"};
SetInstances("COMPANIES", r2);
...
```

In the "DETAILS" target node, we may create a mapping condition using:

```
<trans>
GetInstance()["key_value"] != "";
// Same as GetInstance()[0] != ""
</trans>
```

or, in one of the attributes, the mapping may contain:

```
<trans>
x=GetInstance();
x["key_name"] + "=" + x["key_value"];
// Same as x[0] + "=" + x[1]
</trans>
```

In one of the attributes of the "COMPANIES" target node, the mapping may contain:

```
<trans>
GetInstance();
// This will return
// "MS" for the first instance
// "HP" for the second instance
</trans>
```

[Back to Top]

Max

Declaration

```
type Max(type de)

type Max(array arr)
```

Syntax

```
Max(<de>)

Max(<arr>)
```

Required Parameters

- **de:** (First form) An entity path to instances of a data element in a source or a target
- **arr:** (Second form) An array; all the elements of the array must be of the same type

Description

Returns the maximum value of instances of a data element at a particular level in the hierarchy of a data structure. It will check all instances at that level and return the largest. It can also be used to return the maximum value of an array.

WARNING: There is a known issue with using this function in Cloud Studio. The hash symbol (#), used to indicate an instance, is being inserted improperly into the data element path. As a workaround, you can adjust the position of the hash manually, but upon reopening a script using this function, the symbol will need to be manually repositioned again.

Examples

Assume a database contains a field `"Quantity"` in a table `"Items"` that is a child of `"POHeader"` (a purchase order), and that there are many items within a *POHeader*. Then, this statement returns the maximum value of *Quantity* for any item for a particular *POHeader*:

```
Max(POHeader.Items#.Quantity);
```

[Back to Top]

Min

Declaration

```
type Min(type de)

type Min(array arr)
```

Syntax

```
Min(<de>)

Min(<arr>)
```

Required Parameters

- **`de:`** (First form) An entity path to instances of a data element in a source or a target
- **`arr:`** (Second form) An array; all the elements of the array must be of the same type

Description

Returns the minimum value of instances of a data element at a particular level in the hierarchy of a data structure. It will check all instances at that level and return the smallest. It can also be used to return the minimum value of an array.

WARNING: There is a known issue with using this function in Cloud Studio. The hash symbol (#), used to indicate an instance, is being inserted improperly into the data element path. As a workaround, you can adjust the position of the hash manually, but upon reopening a script using this function, the symbol will need to be manually repositioned again.

Examples

Assume a database contains a field `"Quantity"` in a table `"Items"` that is a child of `"POHeader"` (a purchase order), and that there are many items within a *POHeader*. Then, this statement returns the minimum value of *Quantity* for any item for a particular *POHeader*:

```
Min(POHeader.Items#.Quantity);
```

[Back to Top]

ResolveOneOf

Declaration

```
type ResolveOneOf(type de)

type ResolveOneOf(array arr)
```

Syntax

```
ResolveOneOf(<de>)

ResolveOneOf(<arr>)
```

Required Parameters

- **de:** (First form) An entity path to instances of a data element in a source or a target
- **arr:** (Second form) An array; all the elements of the array must be of the same type

Description

Returns the first non-null value from instances of a data element. This function is generally used for retrieving the value of a "one-of" source data element. It can also be used with arrays, and will return the first non-null element.

WARNING: There is a known issue with using this function in Cloud Studio. The hash symbol (#), used to indicate an instance, is being inserted improperly into the data element path. As a workaround, you can adjust the position of the hash manually, but upon reopening a script using this function, the symbol will need to be manually repositioned again.

[Back to Top]

SetInstances

Declaration

```
null SetInstances(string nodeName, array de)
```

Syntax

```
SetInstances(<nodeName>, <de>)
```

Required Parameters

- **nodeName:** A target's name
- **de:** An entity path to instances of a data element in the target

Description

Defines the source instances for a target loop node. Normally, a loop target instance is generated from a loop source instance. Sometimes the data may come from other sources. This function is intended for cases where the data is in multiple sets and each set generates a single target element.

The instance is a data element which could be a simple value, or an array of data elements. When creating the target, each instance will be used to generate a target instance.

To see how to use an instance data element, see the `GetInstance()` and `ArgumentList()` functions.

This function should be called in the mappings of the parent node of the intended target. If no leaf node is available in the parent, you can create a condition node that calls this function. The condition should end with **true** so that it always is accepted.

The function should not be called more than once with the same target node, as the last call overrides previous ones. To avoid being overridden, you may create multiple-mapping-folders.

A null data element is returned from this function and should be ignored.

WARNING: There is a known issue with using this function in Cloud Studio. The hash symbol (#), used to indicate an instance, is being inserted improperly into the data element path. As a workaround, you can adjust the position of the hash manually, but upon reopening a script using this function, the symbol will need to be manually repositioned again.

Examples

Assume that:

- there is a common parent for the "DETAILS" and "COMPANIES" target nodes;
- both are loop nodes; and
- a multiple-mapping-folder has been created for the "DETAILS" target node.

```
...
r1 = DBLookupAll("<TAG>endpoint:database/My Database</TAG>",
  "SELECT key_name, key_value FROM key_values");
SetInstances("DETAILS", r1);
SetInstances("DETAILS#1", r1);
// DETAILS#1 is the name of the
// 1st multiple-mapping-folder for DETAILS

r2 = {"MS", "HP", "Apple"};
SetInstances("COMPANIES", r2);

// Note: Renaming the display name of a
// multiple-mapping-folder doesn't change
// the folder's actual name, which can be
// found by control-clicking the node and using
// "Copy node name to clipboard"
...
```

[Back to Top]

SortInstances

Declaration

```
null SortInstances(string nodeName, array sourceDataElements1[, bool
sortOrder, ..., array sourceDataElementsN, bool sortOrderN])
```

Syntax

```
SortInstances(<nodeName>, <sourceDataElements1>[, <sortOrder>, ...,
<sourceDataElementsN>, <sortOrderN>])
```

Required Parameters

- **`nodeName:`** Name of the target loop elements to sort
- **`sourceDataElements:`** An entity path to instances of a data element in a source or a target

Optional Parameters

- **`sourceDataElementsN:`** An entity path to instances of a data element in a source or a target
- **`sortOrder:`** An optional sort order, default true for ascending. The argument is not optional if multiple **`sourceDataElements`** arguments are provided.

Description

Sorts the generation of target loop data elements based on one or more data elements in the source or target.

All the sorting instances must have the same number of instances as the number of target instances.

The sorting order is assumed to be ascending, and an optional scalar argument can be put next to each sorting data element to override the default sorting order. If the `sortOrder` is false, the sorting order is descending.

The target loop data elements will be sorted first by the instances of the first source data elements, and then sorted by the instances of the second data elements, and so on.

This function must be called in the mappings of the parent node. If there is no field to map in the parent node, either a script can be called with this function or a condition added for that purpose.

A null value is returned from this function and should be ignored.

> **WARNING:** There is a known issue with using this function in Cloud Studio. The hash symbol (#), used to indicate an instance, is being inserted improperly into the data element path. As a workaround, you can adjust the position of the hash manually, but upon reopening a script using this function, the symbol will need to be manually repositioned again.

Examples

```
// The target node "detail" will be ordered
// by "price" from high to low and,
// if the prices are the same for two items,
// the node will be ordered by "quantity" from low to high
SortInstances("detail", Invoice$Item#.price, false, Invoice$Item#.
quantity);
```

This next example could be used in a condition in a mapping to sort all sales for each customer by date. It would be placed at the level of the **`customer`** node. Note the inclusion of the statement **`true`** at the end of the code block so that the condition is always accepted:

```
<trans>
SortInstances("SalesOrders", _Root$customer.sales#.SalesDate);
true
</trans>
```

[Back to Top]

Sum

Declaration

```
type Sum(type de)

type Sum(array arr)
```

Syntax

```
Sum(<de>)

Sum(<arr>)
```

Required Parameters

- **`de:`** (First form) An entity path to instances of a data element in a source or a target
- **`arr:`** (Second form) An array; all the elements of the array must be of the same type

Description

Takes the value of each instance of a data element at a particular hierarchical level and returns the sum. The data type of both `de` and `arr` must be one of integer, long, float, double, or string. The data types of all instances or all elements must be the same.

If the array is empty, 0 (zero) is returned. Though null values will be ignored in arrays with another data type, an array with just nulls will return an error.

> **WARNING:** There is a known issue with using this function in Cloud Studio. The hash symbol (#), used to indicate an instance, is being inserted improperly into the data element path. As a workaround, you can adjust the position of the hash manually, but upon reopening a script using this function, the symbol will need to be manually repositioned again.

Examples

Assume a database containing a field "Quantity" in a table "Items" that is a child of POHeader (there are many items within one POHeader).

```
// Returns the sum of the field "Quantity" for
// all items for a particular "POHeader"
Sum(POHeader.Items#.Quantity);
```

[Back to Top]

SumCSV

Declaration

```
string SumCSV(type de)

string SumCSV(array arr)
```

Syntax

```
SumCSV(<de>)

SumCSV(<arr>)
```

Required Parameters

- **`de:`** (First form) An entity path to instances of a data element in a source or a target
- **`arr:`** (Second form) An array; all the elements of the array must be of the same type

Description

Concatenates each instance of a field of a data element or each element of an array, with a comma delimiter between each instance or element.

If the field or array element contains special characters such as line feeds or commas, the field or array element is enclosed by double quotes. No delimiter is added after the last instance or element is concatenated.

See also the `SumString()` function for a similar function but with additional options.

> **WARNING:** There is a known issue with using this function in Cloud Studio. The hash symbol (#), used to indicate an instance, is being inserted improperly into the data element path. As a workaround, you can adjust the position of the hash manually, but upon reopening a script using this function, the symbol will need to be manually repositioned again.

Examples

```
// Concatenates all instances of a field of email addresses
// with a comma between each address
SumCSV(_Root$customer.contact#.Email);
```

[Back to Top]

SumString

Declaration

```
string SumString(type de[, string delimiter, bool omitLast])

string SumString(array arr[, string delimiter, bool omitLast])
```

Syntax

```
SumString(<de>[, <delimiter>, <omitLast>])

SumString(<arr>[, <delimiter>, <omitLast>])
```

Required Parameters

- **`de:`** (First form) An entity path to instances of a data element in a source or a target
- **`arr:`** (Second form) An array; all the elements of the array must be of the same type

Optional Parameters

- **`delimiter:`** A string to delimit the items; default value is a semicolon
- **`omitLast:`** A flag indicating if to include the delimiter after the last item; default value is false

Description

Concatenates each instance of the specified data elements or each element of an array, with a delimiter automatically appended to the end of each concatenated string.

If the parameter `omitlast` is true, the delimiter after the last string is omitted.

See also the `SumCSV()` function.

WARNING: There is a known issue with using this function in Cloud Studio. The hash symbol (#), used to indicate an instance, is being inserted improperly into the data element path. As a workaround, you can adjust the position of the hash manually, but upon reopening a script using this function, the symbol will need to be manually repositioned again.

Examples

```
// Concatenates all instances of a field of email addresses
// with a comma between each address,
// but does not place a delimiter after the last address
SumString(_Root$customer.contact#.Email, ",", true);
```

[Back to Top]