Macros

Macros

Macros are used to return data from the system in various formats, to test for conditions, map data from GUI or bulk loader input to various elements in the system (in conjunction with configuration templates) and to access data in workflow and wizard steps.

Various macro functions are available. These serve as boolean operators or can be used to carry out various numeric functions, string manipulation functions, list functions, time functions, and hierarchy related functions.

Macros can be created for re-use, named and stored as an instance of the Macro data model. When re-used, the reference prefix syntax is of the format: macro.<macroname>.

Named macros and macro functions can be nested within other macros.

Macro Syntax Brackets

Macros can have any of the following markup:

- {{ and }} indicate macros that resolve to single values.

  The value can also return an object. A direction parameter is also available for hierarchy searching. This is indicated by ||.

- {# and #} indicate macros that resolve to lists of values.
- {% and %} indicate macros that resolve to dictionaries
- ((and)) indicate macros that test for a condition are enclosed in round brackets ((and)) - these macros evaluate to True or False).

  The comparison operators that are available for these macros are: ==, !=, <, >, <=, >=.

  The OR operator in a test is |, for example:

  ```
  (( device.cuc.PagerDevice.Undeletable|ObjectId:input.ObjectId|
  direction:local == False ))
  ```

- ((test)) <if value> <else value> - IF ELSE type conditional macro.
- ((test)) <value> ((test)) <value> <value> - IF ELSEIF ELSE-type macros combine tests and result values if the test resolves to True or False. The logic is IF (test) THEN <value> ELSE IF (test) THEN <value> ELSE <value>.

Example:

```
((self.a == self.b)) <foo-{{CallManager.host}}>
((self.b == self.c)) <foo-{{CallManager.username}}>
<foo-{{CallManager.version}}>
```

This macro tests for the equality of values in a calling model (referenced by 'self') and returns an evaluation for the condition that is True. The evaluation refers in dot notation to attributes of a Data Model called 'CallManager' and concatenates the result with a string 'foo-'.

- 'SELECT FROM WHERE'-type macros returning single-, dictionary- and list type values and can take parameters. the format is:
  - {{ SELECT FROM | WHERE }}
  - {% SELECT FROM | WHERE %}
  - {# SELECT FROM | WHERE #}

Dot notation

A dot-notation is used to refer to a macro function, model attribute, a defined variable, step reference, or non-attribute value in the model schema.

- **fn.macrofunction_name** - identifies a macro function.
- **self.attribute** - used to refer to the current value of an attribute in the model where the macro applies.

  Here, *attribute* should be an attribute name of the calling model in which the macro is referenced (usually a configuration template). In other words, the macro should be associated with a configuration template of a resource that is referenced.
- **previous.attribute** - used to show the previously saved value of an attribute as opposed to the existing value in the case where a model is updated.
- **input.attribute** - the values input via any of the GUI, or bulk load, for each variable or context..
- **pwf.variablename** - identifies a variable value defined as a provisioning workflow set step variable.
- **cft.attribute** - identifies the values input in a Configuration Template via the current value of the *context_var* of a *foreach* loop in the Configuration Template.
- **modelname.attribute** - this notation defaults to the Data Model type.
- **modeltype.modelname.attribute** - is used for other non-data model types.
- **modeltype.modelname.attribute.NUMBER** - is used to refer to attribute NUMBER-1 where there is more than one attribute, that is, the first attribute reference is *modeltype.model.name.attribute.0*.
- **macro.name** - used to refer to a defined macro by name.
- **workflow.stepSTEPNUMBER.pkid** - used to override the Context Hierarchy by specifying the context using the pkid of stepSTEPNUMBER, where STEPNUMBER can be 1,2 and so on.
- Non-attribute notation allows the following for a model:
  - __pkid

- __bkey
- __hierarchy
- __hierarchy_friendly_path
- __hierarchy_friendly_parent_path

Some examples of this syntax:

- {# data.Countries__pkid #}
- {# data.CallManager.__bkey #}
- {{data.CallManager.__bkey | host: 172.29.248.150 }}
- {{data.CallManager.__hierarchy | host:172.29.248.150 }}
- {{data.CallManager.__hierarchy_friendly_path |host:172.29.248.150 }}
- {{data.CallManager.__hierarchy_friendly_parent_path | host:172.29.248.150 }}

Meta attribute properties can also be used in a macro filter.

- To indicate sequence instance with SEQ - the value is a loop sequence number starting from 1 or a wizard step number:
  - It is obtained from a **Foreach List Macro** in a Provisioning Workflow or a **Foreach Elements** loop in a Configuration Template.

    This value can be used to refer to an attribute of a model. An array item in a Configuration Template has a **Foreach Elements** loop with a variable *phoneX*:

    ```
    {# self.Phone.{{fn.subtract input.phoneX.SEQ, 1}}.lines.line #}
    ```

    that refers to an attribute (line in a list of phones), starting with the first one:

    self.Phone.0.lines.line
  - This value can be used to refer to a Wizard step (stepSTEPNUMBER) in its Configuration Template. A **Foreach Elements** loop with a variable *step* that holds a list of STEPNUMBER obtained from the wizard:

    ```
    {# input.define_wizard_steps #}
    ```

    so that stepSTEPNUMBER can be incremented with:

    step{ {cft.step.SEQ} }

SELECT FROM WHERE Macro Syntax

The types of return values are indicated by the syntax:

- {{SELECT FROM | WHERE}} for single values or objects
- {%SELECT FROM | WHERE%} for dictionaries
- {#SELECT FROM | WHERE#} for lists

The SELECT FROM part is a model reference and uses dot notation.

For lists and dictionaries, the SELECT FROM notation can contain a wild card asterisk * to return all matching values.

Examples returning all User attributes:

```
{# User.* |username: js54321, last_name: 'van Dever' #}
{% User.* |username: js54321 %}
```

The WHERE part is one or more comma-separated name:value pairs of attribute values of the model reference. If the value itself has a comma, it should be wrapped in quotes.

For example:

```
{{ data.Countries.* | country_name: 'India, Republic of' }}
```

The value can also contain:

- a reference to a defined macro used to return a value, as in:

  ```
  {# data.DRTPBXMeta.* | PBX:macro.getHost #}
  ```

- a boolean value using the corresponding macro function, as in:

  ```
  {{ data.DATA1.name | code: fn.true }}
  ```

The WHERE part also supports:

- Asterisk “*” for filtering.

  Examples:

  ```
  {{ device.cucm.Phone.lines.line | name: "SEPD13B004F0719",
  lines.line.*.dirn.pattern:"1006",
  lines.line.*.dirn.routePartitionName:
  "AllowEmerCalls-NewSite4" }}
  ```

  If there is a “*” in the WHERE clause:

  - the results list is reduced with that specific clause
  - the specific WHERE clause is a list

  Note that only one “*” supported and if the WHERE clause has an invalid fieldname, it will be ignored and still return the data.

- regular expression type syntax:

  ```
  field:/regex/
  ```
  Example:

  ```
  {# data.Countries.country_name |
  country_name:/ia$/ #}
  ```

  returns names that end in “ia”:

  ```
  [ "Australia", "Saudi Arabia" ]
  ```
  Similarly, to exclude a list of countries matching “ia” in the name:

  ```
  {# data.Countries.country_name |
  country_name:/[^ia]$/ #}
  ```

Macros cannot be nested in the regex. This will **not** work:

```
{# data.Countries.* | iso_country_code: /[^({{input.ISO}})]/ #}
```

But this will work:

Macro ISO_REGEX:

```
/[^({{input.ISO}})]/
```

Macro:

```
{# data.Countries.* |
iso_country_code:macro.ISO_REGEX #}
```

- attributes not in the data of the model schema
  - __pkid
  - __bkey
  - __hierarchy
  - __hierarchy_friendly_path
  - __hierarchy_friendly_parent_path

For example, given the following:

Macro USA_pkid:

```
{{ data.Countries.__pkid | iso_country_code:USA }}
```

Then Macro:

```
{{ data.Countries.country_name |
__pkid:macro.USA_pkid }}
```

will return

```
{"United States of America"}
```

Note that when using __hierarchy_friendly_path in a WHERE clause, the option clause direction will be ignored, for example:

```
{# data.Countries. | __hierarchy_friendly_path: sys.TestMacros #}*
```

will only return Countries at this hierarchy node.

The SELECT-FROM-WHERE macros can also take additional parameters that restrict results:

- | direction:[up|down|local|parent|below|above]
- | device:[pkid of device]
- | ndl:[pkid of ndl]

The direction option can be added to return values relative to the current hierarchy position. The default direction is down:

- *direction:up* - Upwards. Include current hierarchy.
- *direction:down* - Downwards. Include current hierarchy.
- *direction:local:* - On this level only. Include current hierarchy.

- *direction:parent* - Parent only. Exclude current hierarchy, in other words, search the parent as local.
- *direction:below* - Downwards. Exclude current hierarchy.
- *direction:above* - Upwards. Exclude current hierarchy.

In a 'SELECT-FROM-WHERE'-type macro, a single bar indicates the direction, for example:

```
{{data.Countries|iso_country_name:AUS|direction:up}}
```

If used in other macro types, a double bar is used for parameters, for example:

```
{# data.SiteDefaultsDoc.defaultcucphonesystem || direction:local #}
```

When traversal is *up* or *above*, results will be ordered starting with ones at the lowest hierarchies. Otherwise, results will be ordered starting with the ones at the highest hierarchies. Results at the same hierarchy will be in arbitrary order.

Added to the direction option is an optional limit specifier. When used, the results returned by a list comprehension will be limited to the specified count, for example:

```
{# data.test_user.name || direction:above,limit:2 #}
```

This will return the first two names of *data/test_user* instances at the closest ancestors.

By default, for the following filter specifiers values apply to lists if they are not present:

- *skip* (default: 0) - skip over a number of values before listing
- *limit* (default: 2000) - number of values to return in the list

So, if the first list macro was:

```
{# data.test_user.name || skip:0,limit:2000 #}
```

then the second batch of results can be obtained by:

```
{# data.test_user.name || skip:2000,limit:2000 #}
```

In addition, a *title* filter can be applied if the SELECT-FROM query is for a string to only return values matching its value, with a regular expression, for example:

{# data.Countries.country_name || title:.*of$ #}

returns:

```
["India, Republic of","Oman, Sultanate of","Qatar, State of","Singapore,
Republic of","Sweden, Kingdom of"]
```

Device option:

A device or ndl (Network Device List) pkid can be specified to restrict a query, for example, assume a named macro MY_CUCM_PKID_150:

```
{{ data.CallManager.__pkid | host:172.29.248.150,port:8443 }}
```

then we can select from the specified device as follows:

```
{{ device.cucm.HuntList.__pkid | name: "DR-Test1" |
device:macro.MY_CUCM_PKID_150 }}
```

```
{{ device.cucm.HuntList.__hierarchy | name: "DR-Test1" | device:
macro.MY_CUCM_PKID_150 }}
```

```
{{ device.cucm.HuntList.__hierarchy_friendly_path | name: “DR-Test1” |
device: macro.MY_CUCM_PKID_150 }}
{{ device.cucm.HuntList.__hierarchy_friendly_parent_path | name: “DR-Test1”
| device: macro.MY_CUCM_PKID_150 }}
```

In a GUI Rule, [and] indicate references to values in the current usage context of the GUI Rule if the macro is added as a Value to the GUI Rule.

A GUI rule Action can also have an API call as its Source. The references are current context hierarchy pkid's or to field attribute names in the WHERE section of SELECT FROM WHERE-type macros - enclosed in square brackets [ ].

For example:

```
/api/tool/Macro/?method=evaluate&hierarchy=[hierarchy]
&input={{Countries.iso_country_code |
country_name:[countries.name]}}
```

The syntax in a GUI Rule for a Wizard also uses [], in the format [stepData.STEPNAME.ATTRIBUTE], for example:

```
[stepData.SubscriberType.role]
```

Macro Nesting

To nest macro calls, create a named macro. Nesting inside macros is not supported.

For example, the following incorrect macro:

```
(( fn.list_in Kit, {# fn.split {{ fn.one device.ldap.user.streetAddress
|sAMAccountName: bjones }} #} == True)).
```

should be split up into named macros:

- macro: LDAP_USER

  ```
  {{fn.one device.ldap.user.streetAddress | sAMAccountName: bjones }}
  ```

- macro: LDAP_USER_ADDRESS_LIST

  ```
  {# fn.split macro.LDAP_USER #}
  ```

So the correct macro usage should be:

```
(( fn.list_in Kit, macro.LDAP_USER_ADDRESS_LIST == True))
```

Macro Syntax to Filter by Meta Properties

Macro results can also be filtered by the *meta* data of a resource.

A typical resource instance has associated *meta* data, for example:

```
meta: {
title: "Australia - AUS"
cached: true
tags: [0]
schema_version: "0.1.5"
summary: "true"
references: {...}-
```

```
actions: [...12]-
model_type: "data/Countries"
path: [2]
summary_attrs: [...3]-
business_key: {...}-
tagged_versions: [0]
}
```

The following macro fields are supported to filter by these properties:

- __meta.business_key
- __meta.model_type
- __meta.schema_version
- __meta.system_resource
- __meta.tags
- __meta.title_format
- __meta.uri
- __meta.version_tag

The macro fields can be combined with model attribute names in a comma separated, for example:

```
{# data.Countries.country_name,__meta.schema_version |
country_name:Australia #}
```

Output:

```
[
{
"country_name": "Australia",
"__meta": {
"schema_version": "0.1.5"
}
}
]
```

As a further example: if the *system_resource* is set in the *meta* section of the resource, then the following macro can be used:

```
{# data.ConfigurationTemplate.name | __meta.system_resource: true #}
```

The output is all the Configuration Template names where it is a system resource:

```
[
"AddFeature_Attributes_View",
"AddFeature_DM",
"AddFeature_DOMM",
"AddFeature_DOMM_DM",
"AddFeature_PKG",
"AddFeature_PWF_Add","AddFeature_PWF_Add_DM",
"AddFeature_PWF_Del",
"AddFeature_PWF_Del_DM",
"AddFeature_PWF_Mod",
"AddFeature_PWF_Mod_DM",
"AddFeature_SelectModels_View",
"AddWizard_GuiRules",
"AddWizard_Wizard"
]
```

For devices, the following are examples of macros that are available for the device NDL:

```
__device_meta.ndl.name
```

```
__device_meta.ndl.data/CallManager.pkid
```

Numeric Functions

- *fn.zeropad* - Left pad a given number with zeros up to a given pad number.
- *fn.minval* - For integers, return the minimum value of a provided list.
- *fn.maxval* - For integers, return the maximum value of a provided list.
- *fn.add* - Add two integers.
- *fn.subtract* - Subtract two integers.
- *fn.multiply* - Multiply two integers.
- *fn.divide* - Divide two integers.

Examples:

Example	Output
{{fn.zeropad 123,6}}	`000123`
{{fn.minval 2,3,130,1,30}}	`1`
{{fn.maxval 2,3,130,1,30}}	`130`
{{fn.add 2, 3}}	`5`
{{fn.subtract 2, 3}}	`-1`
{{fn.multiply 2, 3}}	`6`
{{fn.divide 20, 10}}	`2`

String Functions

- fn.index - Return the i'th item of an iterable, such as a list or string.
- fn.mask - Return a mask of (length + modifier) instances of char.
- fn.length - Return the length of a string.
- fn.split - Split a string by delimiter, returning a list.
- fn.join - Join a string by delimiter. If no delimiter is provided, the list is returned as a single string.
- fn.title - Return a string in title case.
- fn.upper - Return an uppercase version of the input string.
- fn.lower - Return a lower case version of the input string.
- fn.contains - Return true or false if string is contained in another.

- fn.sub_string - Return the substring of a string from a start to an end position.
- fn.containsIgnoreCase - Return true or false if string is contained in upper- or lower case.
- fn.containsStartsWith - Return true or false if source string is the start of target string.
- fn.containsStartOf - Return true or false if start of source string is target string.
- fn.isexactly - Return true or false if source string is exactly the same as target string.
- fn.replace - Replace target substring for source substring in source string.

Examples:

Example	Output
{{fn.index 'foo bar baz', 5}}	'b'
{{fn.mask X, 2, 3}}	XXXXXX
{{fn.length This is a valid string}}	22
{# fn.split foo/bar/baz,/ #}	['foo', 'bar', 'baz']
{{ fn.join 1234,: }} {{ fn.join 1234 }}	1:2:3:4 1234
{{fn.title 'foo bar baz'}}	'Foo Bar Baz'
{{fn.upper somevalue}}	SOMEVALUE
{{fn.lower SOMEVALUE}}	somevalue
{{fn.contains needle, haystack}} {{fn.contains hay, haystack}} ((fn.contains Kit, 1234 Kit Creek == True))	false true true
{{fn.sub_string haystack 0, 2}} {{fn.sub_string haystack 7, 7}}	hay k
{{fn.containsIgnoreCaseaaa, bbbaAaccc }}	true
{{ fn.containsStartsWith aaa, aaaffccffdd }}	true
{{ fn.containsStartOf ffnnccgg,ffnn }}	true
{{ fn.isexactly source1,source1 }}	true
{{ fn.replace dddddAAAc,AAA,FFF }}	dddddFFFc

List Functions

- fn.list_index: Return a specified item from a list. Zero is the first item.
- fn.list_index_item: Return the position of item in list.
- fn.list_count: Return the number of items in a list. Note that if the list is known and empty, the count is 0, but if the list is not known, then the count is 1, because the returned message string count is 1.
- fn.list_count_item: Return the number of times item is in a list.
- fn.list_contain: Return true or false if item is in a list or not.
- fn.list_append: Returns a list with item appended.
- fn.list_pop: Return the last item of the list.
- fn.list_insert: Return a list with item inserted at position.
- fn.list_insert_no_dup: Return a list with item inserted at position and all duplicates ignored.
- fn.list_remove: Return a list with all instances of item or list of items removed.
- fn.list_remove_dup: Return a list with all instances of item or list of items removed, including duplicates.
- fn.list_reverse: Return a list that is the reverse of a given list.
- fn.list_extend: Return a list that is an extension of list 1 with list 2.
- fn.list_extend_no_dup: Return the concatenation of list1 and list2, ignoring duplicates.
- fn.sequence: Return a sequence of numbers running from the first value to the second value, optionally padded with zeroes to be the length of a third value.
- fn.list_sort: Return a sorted list; by ascending (A) or descending (D) order.
- fn.one: Return a single result from a list. This is used to convert a single element list to a string.

  If fn.one is called with a string, it returns the string unchanged. The string can be a macro that might get the value of another attribute from a context, such as input.some_variable.
- fn.as_list: Return a string result as a list. If fn.as_list is called with a string, it returns a list. Again, abc could be a macro that resolves to the value of an attribute in its context.
- fn.list_empty: Returns an empty list
- fn.list_set_intersect: Given two lists, return the intersection as a list.
- fn.list_set_union: Given two lists, return the union as a list.
- fn.list_set_left: Given two lists, return a list of items in the left list only.
- fn.list_set_right: Given two lists, return a list of items in the right list only.

Example	Output
`{{fn.list_index 2, data.Countries country_name}}`	“Canada” if this is the third item.

Example	Output
`MACRO1={# fn.sequence 40, 43 #}` `{{ fn.list_index_item 42, macro.MACRO1 }}`	2

`{{fn.list_count data.Countries}}`
`{{fn.list_count input.does_not_exist}}`
`{{fn.list_count non_existant_namespace. does_not_exist}}`
	25 if the list has 25 items.
	0
	1

`MACRO1={# data.Countries.`
`international_access_prefix #}`
`{{fn.list_index_item 00,macro.MACRO1 }}`
	19

`{{fn.list_contain 'AUS',`
`data.Countries.iso_country_code}}`
`{{fn.list_contain 'AUZ',`
`data.Countries.iso_country_code}}`
	true
	false

`MACRO1={# fn.sequence 40, 43 #}` `{{fn.list_append 999, macro.MACRO1 }}`	[‘40’, ‘41’, ‘42’, ‘43’, ‘999’]
`MACRO1={# fn.sequence 40, 43 #}` `{{fn.list_pop macro.MACRO1}}`	“43”
`MACRO1={# fn.sequence 40, 43 #}` `{{fn.list_insert 1, 999, macro.MACRO1 }}`	[‘40’, ‘999’, ‘41’, ‘42’, ‘43’]

`MACRO1={# fn.sequence 40, 43 #}`
`MACRO7={# fn.sequence 39, 41 #}`
`{{fn.list_insert_no_dup macro.MACRO1, macro.MACRO7}}`
	[‘40’,‘41’,
	‘42’,‘43’,‘39’]

`MAC1={# fn.sequence 40,43 #}`
`MAC7={# fn.sequence 40,41 #}`
`MAC3={{fn.list_insert 1,43, macro.MAC1 }}`
`{{fn.list_remove 43, macro.MAC3 }}`
`{{fn.list_remove macro.MAC7, macro.MAC1 }}`
	[‘40’,‘41’,‘42’]
	[‘42’,‘43’]

Example	Output
Given the following list is the result of the regex for iso_country_code: {# data.Countries. iso_country_code | iso_country_code:/AU/ #} ["AUS","SAU","AUS","AUS"] {# fn.list_remove_dup data.Countries. iso_country_code | iso_country_code:/AU/ #}	['AUS','SAU']
MACRO1={# fn.sequence 40, 43 #} {{fn.list_reverse macro.MACRO1}}	[‘43’,‘42’,‘41’,‘40’]

MACRO1={# fn.sequence 40, 43 #}
MACRO9={# fn.sequence 50, 52 #}
{{fn.list_extend macro.MACRO1, macro.MACRO9 }}
	[‘40’, ‘41’, ‘42’, ‘43’, ‘50’, ‘51’, ‘52’]

MACRO1={# fn.sequence 40, 43 #}
MACRO8={# fn.sequence 42, 45 #}
{{fn.list_extend_no_dup macro.MACRO1, macro.MACRO8 }}
	[‘40’,‘41’,‘42’,‘43’,‘44’, ‘45’]

{# fn.sequence 40, 43 #} {# fn.sequence 110, 100, 4 #}	[‘40’,‘41’,‘42’,‘43’] [‘0110’,‘0109’,‘0108’,‘0107’, ‘0106’,‘0105’,‘0104’,‘0103’, ‘0102’,‘0101’,‘0100’]
MACRO1={# fn.sequence 40, 43 #} {{fn.list_sort macro.MACRO1, D}} {{fn.list_sort macro.MACRO1, Descending}} MACRO5={# fn.sequence 110, 108, 4 #} {{fn.list_sort macro.MACRO5, A}}	[‘43’,‘42’,‘41’,‘40’] [‘43’,‘42’,‘41’,‘40’] [‘0110’,‘0109’,‘0108’] [‘0108’,‘0109’,‘0110’]

Example	Output
{{fn.one data.Countries. iso_country_code | emergency_access_ prefix:'112'}} {{fn.one abc}}	A single result, e.g. 'FRA' 'abc'
{{fn.as_list data.Countries. country_name|country_name : 'Bahrain'}} {{fn.as_list abc}}	['Bahrain'] ['abc']
{{fn.list_empty}}	[]

MACRO1={# fn.sequence 40, 43 #}
MACRO2={# fn.sequence 41, 42 #}
{# fn.list_set_intersect macro.MACRO1, macro.MACRO2 #}
	['41','42']

MACRO1={# fn.sequence 40, 43 #}
MACRO2={# fn.sequence 41, 45 #}
{# fn.list_set_union macro.MACRO1, macro.MACRO2 #}
	['40','41','42','43','44','45']

MACRO1={# fn.sequence 40, 43 #}
MACRO2={# fn.sequence 41, 45 #}
{# fn.list_set_left macro.MACRO1, macro.MACRO2 #}
	['40']

MACRO1={# fn.sequence 40, 43 #}
MACRO2={# fn.sequence 41, 45 #}
{# fn.list_set_right macro.MACRO1, macro.MACRO2 #}
	['44','45']

Rule Filter Functions

The "filter by rule" function returns a list of resource instance data for a given model type. The schema of the model type in question must define a rules object of the form:

```
{
  'rules': {
```

```
        'hierarchy_types': [<list of data/HierarchyNodeType business keys>]
    }
}
```

The model `data/Role` is one example of such a model type. Filtering is applied using the current hierarchy context or else based on an explicit hierarchy type name.

Macro format:

```
{{ fn.filter_by_rule <rule name>,
                     <model type>,
                     <direction>,
                     <attribute path to return>,
                     <hierarchy type name> }}
```

Argument descriptions:

<rule name>: The name of the rule being used to filter results. Supported values: `hierarchy_types`

<model type>: The model type of the instances to be filtered.

<direction>: The search direction of the results. Possible values are:

- `all` - search at current hierarchy, ancestors, and descendants.
- `up` - search at current hierarchy and ancestors.
- `down` - search at current hierarchy and descendants.
- `local` - search at current hierarchy only.

<attribute path to return>: [optional] The path dot (.) delimited path to a single attribute to return

- If not specified the returned result will contain a list of objects.
- If specified the returned result will contain a list the given attribute.
- Must be “null” if this field is not required, while <hierarchy type name> is supplied.

<hierarchy type name>: [optional] The name of the hierarchy type to filter by. This will be looked up from the current hierarchy going up.

Examples:

```
{{ fn.filter_by_rule hierarchy_types,data/Role,up,name,Customer }}
```

Returns all the names of the roles that are permitted at “Customer” hierarchy type. Lookup is done from the current hierarchy upwards.

```
{{ fn.filter_by_rule hierarchy_types,data/Role,up,null,Customer }}
```

Returns full instance data of the roles that are permitted at “Customer” hierarchy type. Lookup is done from the current hierarchy upwards.

```
{{ fn.filter_by_rule hierarchy_types,data/Role,up,name }}
```

Returns all the names of the roles that are permitted at the hierarchy type of the current hierarchy context. Lookup is done from the current hierarchy upwards.

```
{{ fn.filter_by_rule hierarchy_types,data/Role,up }}
```

Returns full instance data of the roles that are permitted at the hierarchy type of the current hierarchy context. Lookup is done from the current hierarchy upwards.

Macro Function

- *fn.evaluate* : Evaluate the string using the macro interpreter. The string can be a macro and can also contain macro names to be evaluated.

The function is so that we can save data as a macro and then evaluate it when we read it again.

Example	Output

Where self.x is :
`{# data.Countries.iso_country_code |`
`country_name:'South Africa' #}`
Then:
`{# fn.evaluate self.x #}`
Where MACRO1 is:
`{{ data.Countries.emergency_access_prefix|`
`iso_country_code:FRA }}`
Where MACRO2 is :
`{# data.Countries.iso_country_code |`
`emergency_access_prefix:macro.MACRO1 #}`
Then:
`{# fn.evaluate macro.MACRO2 #}`
	['ZAF']
	['DNK','HKG','FRA','DEU',
	'IND','ITA','NLD','NZL', 'SAU,'ESP','SWE','ZAF', 'CHE','TUR']

CUCM Functions

`fn.cucm_get_line_details` - Specify the line pattern and site name and use the Macro Evaluator function to view the line parameters for the specified line.

To return the result for a single line parameter, append the required parameter to the end of the macro

Example	Output
{{fn.cucm_get_line_details 4025, VS-Corp-NewYork}}	{ "is_line_shared": true, "remote_destination_profiles": "RDP_vdevenr"], "device_profiles": ["UDP_vdevenr"], "phones": ["SEP002155D547F7", "SEP111122223333"] }
{{fn.cucm_get_line_details 4025, VS-Corp-NewYork, is_line_shared}}	true
{{fn.cucm_get_line_details 4025, VS-Corp-NewYork, phones}}	SEP002155D547F7 SEP111122223333

Subscriber Functions

- *fn.process_subscriber_line_data*: Used in workflows - a single parameter called input is the workflow input context, containing line data. The function returns line patterns and partitions found in any of Phone, DeviceProfile, RemoteDestinationProfile.
- *fn.process_subscriber_wizard_data*: Used in workflow CFT - a single parameter called input is the workflow input context, containing wizard data. The function returns Configuration Template data use for adding a relation/Subscriber instance.

Examples:

Example	Output
{{ fn.process_subscriber_line_data input }}	[{ "pattern": "10003", "routePartitionName": "Site-23m-Customer 1 Site A" },{ "pattern": "10005", "routePartitionName": "Site-23m-Customer 1 Site A" }]

Example	Output
`{{ fn.process_subscriber_wizard_data input }}`	template data for a Configuration Template

Example Configuration Template data snippet:

```
"data": {
"description": "Template for provisioning a Subscriber from
wizard using custom macro function",
"name": "AddSubscriberWizard_CFT",
"target_model_type": "relation/Subscriber",
"template": "{{ fn.process_subscriber_wizard_data input }}"
}
```

Zero, Unset and Boolean Functions

Function	Description	Example	Output
fn.zero	Return a zero value.	{{fn.zero}}	0
fn.unset	Return an empty string.	{{fn.unset}}	""
fn.true	Return a boolean True.	{{fn.true}}	true
fn.false	Return a boolean False.	{{fn.false}}	false

Time Functions

- *fn.now*: Return the date and time at this moment. An optional format parameter is available.

Example	Output
{{fn.now}} {{fn.now "%Y%m%d"}} macro.DAY="%A %m/%d/%Y" {{fn.now macro.DAY}}	`2013-04-18 10:50:52.105130` 20140327 "Thursday 03/27/2014"

Supported date and time formats

%a	abbreviated weekday name according to the current locale
%A	full weekday name according to the current locale
%b	abbreviated month name according to the current locale

%B	full month name according to the current locale
%c	preferred date and time representation for the current locale
%C	century number (the year divided by 100 and truncated to an integer, range 00-99)
%d	day of the month as a decimal number (range 01 to 31)
%D	same as m/d/y
%e	day of the month as a decimal number, a single digit is preceded by a space (range ‘ 1’ to ‘31’)
%g	like G, but without the century
%G	The 4-digit year corresponding to the ISO week number
%h	same as b
%H	hour as a decimal number using a 24-hour clock (range 00 to 23)
%I	hour as a decimal number using a 12-hour clock (range 01 to 12)
%j	day of the year as a decimal number (range 001 to 366)
%m	month as a decimal number (range 01 to 12)
%M	minute as a decimal number
%n	newline character
%p	either ‘AM’ or ‘PM’ according to the given time value, or the corresponding strings for the current locale
%P	like p, but lower case
%r	time in a.m. and p.m. notation equal to I:M:S p
%R	time in 24 hour notation equal to H:M
%S	second as a decimal number
%t	tab character
%T	current time, equal to H:M:S
%u	weekday as a decimal number [1,7], with 1 representing Monday
%U	week number of the current year as a decimal number, starting with the first Sunday as the first day of the first week

%V	The ISO 8601:1988 week number of the current year as a decimal number, range 01 to 53, where week 1 is the first week that has at least 4 days in the current year, and with Monday as the first day of the week.
%w	day of the week as a decimal, Sunday being 0
%W	week number of the current year as a decimal number, starting with the first Monday as the first day of the first week
%x	preferred date representation for the current locale without the time
%X	preferred time representation for the current locale without the date
%y	year as a decimal number without a century (range 00 to 99)
%Y	year as a decimal number including the century
%z	numerical time zone representation
%Z	time zone name or abbreviation
%%	a literal '%' character

Hierarchy Functions

- `fn.hierarchy` - Return the UUID of the current node.
- `fn.hierarchy_parent` - Return the UUID of the parent.
- `fn.hierarchy_path` - Return the current node hierarchy as a list of UUIDs.
- `fn.hierarchy_parent _path` - Return the current node parent hierarchy as a list of UUIDs.
- `fn.hierarchy_friendly _path` - Return the current node hierarchy as a dot-separated hierarchy string.
- `fn.hierarchy_friendly _parent_path` - Return the current node parent hierarchy as a dot-separated hierarchy string.

Example	Output
`{{fn.hierarchy}}`	`'52162d552afa433946245bcb'`
`{[fn.hierarchy_parent}}`	`'52162d522afa433941245ba0'`

Example	Output
{{fn.hierarchy_path}}	['1c0efeg2c0deab10da595101', '52162d4c2afa433940245ba3', '52162d4e2afa43393b245ba2', '52162d522afa433941245ba0', '52162d552afa433946245bcb']
{[fn.hierarchy_parent _path}}	['1c0efeg2c0deab10da595101', '52162d4c2afa433940245ba3', '52162d4e2afa43393b245ba2', '52162d522afa433941245ba0']
{{fn.hierarchy_friendly _path}}	'sys.GenCorp.SuperCom. ABCGroup.Branch1'
{[fn.hierarchy_friendly _parent_path}}	'sys.GenCorp.SuperCom. ABCGroup'

Request Functions

Function	Description	Example
fn.request_user_name	Return the logged-in username.	{{fn.request_user_name}}
fn.request_user_role	Return the logged-in user role.	{{fn.request_user_role}}
fn.request_user_email	Return the logged-in user email address	{{fn.request_user_email}}
fn.request_user_pkid	Return the logged-in user pkid.	{{fn.request_user_pkid}}

Internal Number Inventory Functions

The `fn.lines` macro functions use the value of a CUSTOMER_INI_ENABLED macro at the relevant hierarchy:

- If `((True))`, then apply the function to the Internal Number Inventory at the hierarchy: `data.InternalNumberInventory.internal_number`.
- If `((False))`, then apply the function to `device.cucm.Line.pattern`, optionally with a specified `routePartitionName`.

All macros will check the CUSTOMER_INI_ENABLED macro first. These macros exist at the required hierarchy level and have a value of `((True))` or `((False))`.

- CUSTOMER_INI_ENABLED macro is `((False))`:

Give the following patterns and route partition on the Unified Communications Manager:

Number	Partition
1000	
2000	Site-REL103-Customer
3000	
4000	Site-REL103-Customer
5000	Site-REL103-Customer
6000	

- `fn.lines` - Return a list of lines. With no parameter, list all the lines on the associated Unified Communications Manager. If the optional parameter is a route partition, list the lines in the partition. If the optional parameter is custom, show an empty list.

Example	Output

`{{fn.lines}}`	`[{u'value': u'1000', u'title': u'1000'},`
	`{u'value': u'2000', u'title': u'2000'},`
	`{u'value': u'3000', u'title': u'3000'},`
	`{u'value': u'4000', u'title': u'4000'},`
	`{u'value': u'5000', u'title': u'5000'},`
	`{u'value': u'6000', u'title': u'6000'}]`

`{{fn.lines Site-REL103-Customer}}`	`[{u'value': u'2000', u'title': u'2000'},`
	`{u'value': u'4000', u'title': u'4000'},`
	`{u'value': u'5000', u'title': u'5000'}]`

`{{fn.lines custom}}`	`[]`

- CUSTOMER_INI_ENABLED macro is `((True))`:

Given the following properties of an example Internal Number Inventory:

Number	In Use	Available
1000	N	Y
2000	N	Y
3000	N	N
4000	Y	Y
5000	Y	Y
6000	Y	N

- `fn.lines` - Return a list of available lines on the Internal Number Inventory, with used lines indicated as (used).
- `fn.lines_available_only` - Return a list of available lines from the Internal Number Inventory, with <TITLE> in brackets.
- `fn.lines_used_only` - Return a list of used lines from the Internal Return a Number Inventory, with <TITLE> in brackets after each.
- `fn.lines _unavailable_only` - Return a list of unavailable lines from the Internal Number Inventory, with <TITLE> in brackets after each.
- `fn.lines_unused_only` - Return a list of unused lines from the Internal Number Inventory, with <TITLE> in brackets after each.
- `fn.lines_available_used` - Return a list of available and used lines from the Internal Number Inventory, with <TITLE> in brackets after each.
- `fn.lines_available_unused` - Return a list of available and unused lines from the Internal Number Inventory, with <TITLE> in brackets after each.
- `fn.lines_unavailable_used` - Return a list of unavailable, used lines from the Internal Number Inventory, with <TITLE> in brackets after each.
- `fn.lines_unavailable_unused` - Return a list of unavailable and unused lines from the Number Inventory, with <TITLE> in brackets after each.

Example	Output

{{fn.lines}}	`[{u'value': u'1000', u'title': u'1000'},`
	`{u'value': u'2000', u'title': u'2000'},`
	`{u'value': u'4000', u'title': u'4000 (used)'},`
	`{u'value': u'5000', u'title': u'5000 (used)'}]`

Example	Output
{{fn.lines_available_only avail}}	[{u'value': u'1000', u'title': u'1000 (avail)'}, {u'value': u'2000', u'title': u'2000 (avail)'}, {u'value': u'4000', u'title': u'4000 (avail)'}, {u'value': u'5000', u'title': u'5000 (avail)'}]
{{fn.lines_used_only inuse}}	[{u'value': u'4000', u'title': u'4000 (inuse)'}, {u'value': u'5000', u'title': u'5000 (inuse)'}]
{{fn.lines_unavailable_only}}	[{u'value': u'3000', u'title': u'3000'}, {u'value': u'6000', u'title': u'6000 (used)'}]
{{fn.lines_unused_only}}	[{u'value': u'1000', u'title': u'1000'}, {u'value': u'2000', u'title': u'2000'}, {u'value': u'3000', u'title': u'3000'}]
{{fn.lines_available_used}}	[{u'value': u'4000', u'title': u'4000 (used)'}, {u'value': u'5000', u'title': u'5000 (used)'}]
{{fn.lines_available_unused}}	[{u'value': u'1000', u'title': u'1000'}, {u'value': u'2000', u'title': u'2000'}]
{{fn.lines_unavailable_used}}	[{u'value': u'6000', u'title': u'6000'}]
{{fn.lines_unavailable_unused}}	[{u'value': u'3000', u'title': u'3000'}]

Localization Functions

- `fn.localize` - Return a value that is localized, in other words it will be translated if a translation exists. It is used with a macro call that returns a value from a data store.
- `fn.list_installed_languages` - List the installed languages on the system. This includes languages in the administrator GUI and selfservice GUI.

- `fn.list_installed_languages_admin` - List the installed languages in the administrator GUI.
- `fn.list_installed_languages_selfservice` - List the installed languages in the selfservice GUI.
- `fn.list_installed_languages_by_role` - List the installed languages based on the User Role Interface value.

  The User Role Interface value is a parameter of this function:

  - admin - installed languages in the admin GUI
  - self-service - selfservice GUI
  - none - union of admin and selfservice languages

Example	Output
{{fn.localize data.LocalizedModelStore. localized_value | where_clause }}	A localized value is returned.
{# fn.list_installed_languages #}	[{'value':'en-us','title':'English'}, {'value':'de-de','title':'German'}]
{# fn.list_installed_languages _admin #}	[{'value':'en-us','title':'English'}]
{# fn.list_installed_languages _selfservice #}	[{'value':'en-us','title':'English'}, {'value':'de-de','title':'German'}]
{# fn.list_installed_languages__by_role none #}	[{'value':'en-us','title':'English'}, {'value':'de-de','title':'German'}]

Log Functions

- `fn.log` - Given a log level and message, display it in the log. Log levels can be: debug, critical, warn, error or info.
- `fn.txn_log` - Given a message, display it in the transaction log on the GUI.

  The macro is typically added to a workflow “Set” step for debugging purposes.

Example	Output

`{{fn.log info, This is an informational message.}}`
`{{fn.log debug, Debug message.}}`
	In `app.log`:
	`INFO This is an informational message.`
	`DEBUG Debug message.`

Example	Output
A workflow (1PWF) step Set variable “bar” is a message with the value of “name”. `{{fn.txn_log pwf.name}}`	Step 1 - Set workflow context (1PWF): `{ "set_list": [ { "set_var_name": "name", "set_var_value": "foo" }, { "set_var_name": "bar", "set_var_value": "{{fn.txn_log pwf.name}}" } ], "bar": "foo", "name": "foo" }`

Object Keys Functions

Example object:

```
{
"input": {
"object": {
"boolean_1": true,
"boolean_2": false,
"boolean_3": true,
"string_1": "1",
"string_1_dup": "1",
"string_2": "2",
"integer_1": 1,
"integer_1_dup": 1,
"integer_2": 2
}
}
```

- `fn.object_keys` - Given an object and additional optional parameter, return the list of keys that match the parameter value, or all the keys if no parameter value is given.

Examples:

Example	Output
{{ fn.object_keys input.object,true }}	["boolean_1","boolean_3"]
{{fn.object_keys input.object,"1"}}	["string_1","string_1_dup"]
{{fn.object_keys input.object,1}}	["integer_1","integer_1_dup"]
{{fn.object_keys input.object}}	["boolean_1","boolean_2","boolean_3", "string_1","string_1_dup","string_2", "integer_1","integer_1_dup","integer_2"]

Macro and Macro Function Nesting

Macros and macro functions can be used as arguments of macros and macro functions. Consider the examples below:

1. Define a macro Masklen as {{fn.length This is a valid string}}.
2. Define a macro as {{fn.mask X macro.Masklen 0}}.
3. The result is evaluated as 'XXX...' to the length of 'This is a valid string'.

Conversion Functions

- `fn.pkid_to_bkey` - Given a pkid, return the business key.
- `fn.bkey_to_pkid` - Given a business key, return the pkid. Provide the data type as an argument.
- `fn.as_int` - Given a string, return an integer
- `fn.as_string` - Given an integer, return a string.
- `fn.as_bool` - Given strings "True", "TRUE", "T", "t", "1", return True. Given strings "False", "FALSE", "F", "f", "0", return False.
- `fn.as_list` - Given input, return it as a list. List input is returned as is.

Examples:

Example	Output
`macro: USA_pkid = {{data.Countries.__pkid|iso_country_code:USA}}` `{{fn.pkid_to_bkey macro.USA_pkid}}`	`"[u'United States of America',` `u'USA', u'']"`
`macro: a_country_bkey = {{data.Countries.__bkey|__pkid:macro.USA_pkid}}` `{{fn.bkey_to_pkid macro.a_country_bkey, data/Countries}}`	`"52d3eba8893d57373f842acb"`
`{{fn.as_int "1"}}`	`1`
`{{fn.as_string 12345}}`	`"12345"`
`{{fn.as_bool "T"}}` `{{fn.as_bool "0"}}`	`true` `false`

Example	Output
{#fn.as_list foo bar#}	['foo bar']
{#fn.as_list 'foo,bar'#}	['foo,bar']
{#fn.as_list ['foo','bar']#}	['foo','bar']

Conditional Logic Macro Function

Conditional logic in macros is supported by the `fn.conditional_logic` function that takes two parameters:

- the name of an instance of the `data/ConditionalLogic` data model.
- a value that serves as input to the `data/ConditionalLogic` data model.

The namespaces that can be used as `left_expression` values in the data model can depend on the reference to the data model in a Provisioning Workflow. The namespaces - that can also be referenced in full - include:

- `{{self}}`
- `{{previous}}`
- `{{input}}`
- `{{cft}}`
- `{{pwf}}`

Consider the following example `data/ConditionalLogic` data model called "Is_SLC_Allowed":

```
"conditions": [
    {
        "unary_operator": "NOT",
        "right_expression": "{{ logic.DATA }}",
        "conditional_operator": "AND",
        "condition": "contains",
        "left_expression": "{{ pwf.SLCS }}"
    },
    {
        "unary_operator": "NOT",
        "right_expression": "{{ input.CURRENT_SLC }}",
        "conditional_operator": "AND",
        "condition": "containsStartOf",
        "left_expression": "{{ logic.DATA }}"
    },
    {
        "right_expression": "{{ input.CURRENT_SLC }}",
        "condition": "containsStartsWith",
        "unary_operator": "NOT",
        "left_expression": "{{ logic.DATA }}"
    }
]
```

Also suppose the Provisioning Workflow for this example has a list variable `SLCS` and receives an `input.CURRENT_SLC` value.

Furthermore, during the call of the `fn.conditional_logic` function in the Provisioning Workflow, it receives a *scalar* value as an argument, for example:

```
{{ fn.conditional_logic Is_SLC_Allowed, 128 }
```

- The scalar value reference `{{ logic.DATA }}` can be omitted from either `left_expression` or `right_expression`. Its reference is then assumed.
- The input value is referenced as `{{ input.CURRENT_SLC }}`
- The list that is the Provisioning Workflow variable, is `{{ pwf.SLCS }}`

As another example, consider a `data/ConditionalLogic` model called "TestData" with three conditions:

```
"conditions": [
                {
                   "conditional_operator": "OR",
                   "left_expression": "{{input.DATA}}",
                   "condition": "contains",
                   "right_expression": "AAA"
                },
                {
                   "conditional_operator": "AND",
                   "left_expression": "{{input.DATA}}",
                   "condition": "contains",
                   "right_expression": "BBB"
                },
                {
                   "left_expression": "{{input.DATA}}",
                   "condition": "contains",
                   "right_expression": "CCC",
                   "unary_operator": "NOT"
                }
```

The following function checks if a received input value "AAAaaaBBBaaaCCc" fulfills the condition: contains "AAA" OR "BBB" AND NOT "CCC", as in the macro test using a scalar value:

```
{{ fn.conditional_logic TestData, AAaaaBBBaaaCCc }}
```

The condition resolves to true.

Finally in the following example, the conditional function is used as a condition in a Provisioning Workflow. The Data Model instance of data/ConditionalLogic called "Does Newland Exist" tests a single string matching condition:

```
"data": {
    "conditions": [
        {
           "right_expression": "Newland",
           "condition": "isexactly",
           "left_expression": "{{pwf.EXIST}}"
        }
    ],
    "name": "Does Newland Exist"
}
```

The Provisioning Workflow step to apply a Configuration Template if the condition is false. So the step is carried out only if there is not already a `country_name` called "Newland".

```
"workflow": [
  {
    "templates": [
      {
        "conditions": [
          {
            "condition": "(( fn.conditional_logic \"Does Newland Exist\" == False ))"
          }
        ],
        "template": "CFT1"
      }
    ],
    "entity": "data/Countries",
    "set_list": [
      {
        "set_var_name": "EXIST",
        "set_var_value": "{{data.Countries.country_name|country_name:Newland}}"
```

```
      }
    ],
    "method": "add",
    "entity_type": "model"
  }
```

HTTP GET Function

- `fn.request_get` - Return in JSON format the response of an HTTP request. The HTTP request must start with `http://localhost`.

Example request:

```
{{ fn.request_get http://localhost/api/data/ Countries/properties }}
```

The output can be assigned to a variable so that properties can be referenced.

Example with output snippet:

```
http://localhost/api/data/Countries/properties
{
    "meta": {
       "query": "/api/data/Countries/properties/?hierarchy=[hierarchy]&format=json"
    },
   "choices": [
      [
           "cli_on_prefix",
           "cli_on_prefix"
      ],
      [
           "country_name",
           "country_name"
      ],
      [
...
```

Example of instance output with GET request for an instance with `[pkid]`:

```
http://localhost/api/data/Countries/54e1de60edec65160652e402
...
  ],
     "business_key": {
         "hierarchy": true,
         "unique": [
             "country_name",
             "iso_country_code"
         ]
     },
     "tagged_versions": []
},
"data": {
    "iso_country_code": "MEX",
     "pstn_access_prefix": "9",
     "pkid": "54e1de60edec65160652e403",
     "default_user_locale": "English United States",
     "network_locale": "United States",
     "standard_access_prefix": "0",
     "international_access_prefix": "00",
     "country_name": "Mexico",
     "international_dial_code": "52",
     "emergency_access_prefix": "066",
     "national_trunk_prefix": "01"
    }
```

Macro Examples Simple

Simple macros must always resolve to one value only.

```
{{data.Countries.iso_country_code | country_name:'South Africa'}}
'ZAF'
```

Call to a non-existent macro.

```
{{macro.DoesNotExist}}
{u'code': 6003,
u'http_code': 400,
u'message': u'Macro lookup of macro.DoesNotExist failed at hierarchy sys',
u'transaction': None}
```

First Partition member name in CSS PSTN-CSS-Cape-Town.

```
{{ device.cucm.Css.members.member.0.routePartitionName | name: 'PSTN-CSS-Cape-Town' }}
'PHONES-PT-Cape-Town'
```

First Partition member UUID in CSS PSTN-CSS-Cape-Town.

```
{{ device.cucm.Css.members.member.0.uuid | name: 'PSTN-CSS-Cape-Town' }}
'{7AF255DC-3A05-A1B4-9E5E-95CD48C3C95F}'
```

Macro Examples List Macro

Syntax for List macros is between {##}. The results are in a list format: comma-separated results and between [ ] All fields in Countries model.

```
{# data.Countries.* #}
[{u'cli_on_prefix': u'',
u'country_name': u'Australia',
u'data_type_': u'data/Countries',
u'default_user_locale': u'English United States',
u'emergency_access_prefix': u'000',
u'international_access_prefix': u'011',
u'international_dial_code': u'61',
u'iso_country_code': u'AUS',
u'national_trunk_prefix': u'0',
u'network_locale': u'United States',
u'premium_access_prefix': u'8',
u'pstn_access_prefix': u'9',
u'service_access_prefix': u'13'},
{u'cli_on_prefix': u'',
u'country_name': u'Bahrain',
u'data_type_': u'data/Countries',
u'default_user_locale': u'English United States',
u'emergency_access_prefix': u'999',
u'international_access_prefix': u'00',
u'international_dial_code': u'973',
u'iso_country_code': u'BHR',
u'national_trunk_prefix': u'',
u'network_locale': u'United States',
u'premium_access_prefix': u'',
u'pstn_access_prefix': u'9',
u'service_access_prefix': u''},
.........]
```

Selected fields in Countries model.

```
{# data.Countries.country_name, iso_country_code #}
[{u'country_name': u'Australia',
u'iso_country_code': u'AUS'},
{u'country_name': u'Bahrain',
u'iso_country_code': u'BHR'},
{u'country_name': u'Canada',
u'iso_country_code': u'CAN'},
{u'country_name': u'Denmark',
u'iso_country_code': u'DNK'},
```

```
...
...
{u'country_name': u'United States of America',
u'iso_country_code': u'USA'}
]
```

Specifying one field in the list will return only a list of values and not a key-value pair list.

```
{# data.Countries.country_name #}
[u'Australia',
u'Bahrain',
u'Canada',
...
...
u'United States of America']
```

Device types: a list of all line patterns in the null partition.

```
{# device.cucm.Line.pattern,routePartitionName | routePartitionName:'NullPartition'#}
[{u'pattern': u'55554444',
u'routePartitionName': u'NullPartition'},
{u'pattern': u'8100240105',
u'routePartitionName': u'NullPartition'},
{u'pattern': u'5544332211',
u'routePartitionName': u'NullPartition'},
{u'pattern': u'55667722',
u'routePartitionName': u'NullPartition'},
{u'pattern': u'8765653',
u'routePartitionName': u'NullPartition'},
{u'pattern': u'66776767',
u'routePartitionName': u'NullPartition'},
{u'pattern': u'3009',
u'routePartitionName': u'NullPartition'},
{u'pattern': u'656574747',
u'routePartitionName': u'NullPartition'},
...
... ]
```

Nested structures.

```
{# device.cucm.Css.* | name: 'PSTN-CSS-Cape-Town'#}
[{u'clause': u'PHONES-PT-Cape-Town:PSTN-PT-Cape-Town:
Pickup-PT-Cape-Town:CallPark-PT-Cape-Town',
u'hierarchy': u'5171010ecc2e19483c11291b',
u'members': {u'member': [{u'index': 1,
u'routePartitionName': u'PHONES-PT-Cape-Town',
u'uuid': u'{7AF255DC-3A05-A1B4-9E5E-95CD48C3C95F}'},
{u'index': 2,
u'routePartitionName': u'PSTN-PT-Cape-Town',
u'uuid': u'{5FA76732-0074-108A-3A91-23D7C6CAC2E1}'},
{u'index': 3,
u'routePartitionName': u'Pickup-PT-Cape-Town',
u'uuid': u'{F789964F-C95D-4095-F6C7-48E587CBFAD8}'},
{u'index': 4,
u'routePartitionName': u'CallPark-PT-Cape-Town',
u'uuid': u'{B4817113-0F32-6E7F-67B2-20645CFC4509}'}]},
u'name': u'PSTN-CSS-Cape-Town',
u'partitionUsage': u'General',
u'uuid': u'{E678A23E-866A-7CE8-AD0F-8AF138E10A18}'}]
{# device.cucm.Css.name,members | name: 'PSTN-CSS-Cape-Town'#}
[{u'members': {u'member': [{u'index': 1,
u'routePartitionName': u'PHONES-PT-Cape-Town',
u'uuid': u'{7AF255DC-3A05-A1B4-9E5E-95CD48C3C95F}'},
{u'index': 2,
u'routePartitionName': u'PSTN-PT-Cape-Town',
u'uuid': u'{5FA76732-0074-108A-3A91-23D7C6CAC2E1}'},
{u'index': 3,
u'routePartitionName': u'Pickup-PT-Cape-Town',
u'uuid': u'{F789964F-C95D-4095-F6C7-48E587CBFAD8}'},
{u'index': 4,
u'routePartitionName': u'CallPark-PT-Cape-Town',
u'uuid': u'{B4817113-0F32-6E7F-67B2-20645CFC4509}'}]},
u'name': u'PSTN-CSS-Cape-Town'}]
{# device.cucm.Css.name,members.member | name: 'PSTN-CSS-Cape-Town'#}
[{u'members.member': [{u'index': 1,
u'routePartitionName': u'PHONES-PT-Cape-Town',
```

```
u'uuid': u'{7AF255DC-3A05-A1B4-9E5E-95CD48C3C95F}'},
{u'index': 2,
u'routePartitionName': u'PSTN-PT-Cape-Town',
u'uuid': u'{5FA76732-0074-108A-3A91-23D7C6CAC2E1}'},
{u'index': 3,
u'routePartitionName': u'Pickup-PT-Cape-Town',
u'uuid': u'{F789964F-C95D-4095-F6C7-48E587CBFAD8}'},
{u'index': 4,
u'routePartitionName': u'CallPark-PT-Cape-Town',
u'uuid': u'{B4817113-0F32-6E7F-67B2-20645CFC4509}'}],
u'name': u'PSTN-CSS-Cape-Town'}]
{# device.cucm.Css.name,members.member.2 | name: 'PSTN-CSS-Cape-Town'#}
[{u'members.member.2': {u'index': 3,
u'routePartitionName': u'Pickup-PT-Cape-Town',
u'uuid': u'{F789964F-C95D-4095-F6C7-48E587CBFAD8}'},
u'name': u'PSTN-CSS-Cape-Town'}]
{# device.cucm.Css.name,members.member.2.routePartitionName | name:
'PSTN-CSS-Cape-Town'#}
[{u'members.member.2.routePartitionName': u'Pickup-PT-Cape-Town',
u'name': u'PSTN-CSS-Cape-Town'}]
```

Create a Macro In-line

To create a macro in-line, enter the macro directly in:

1. Default value in a Data Model.
2. Default value in a Configuration Template.
3. Condition of an Operation in a Provisioning Workflow.

Create a Value Substitution Macro

To write a value substitution macro:

1. Determine where to enter the macro: in-line or for re-use.
2. Determine the reference of the value to resolve:
   - Data Model reference syntax is “data/datamodel.attribute” or “datamodel.attribute”. The first instance of the datamodel.attribute will be the target.
   - Syntax for a reference to an attribute in a Domain Model is “self.attribute”
3. Enter any static text that should combine with the evaluated macro - if required. Static text is entered outside the “{{” and “}}”.

Substitution Macro Examples

```
{{CallManager.host}}
http://{{CallManager.host}}
http://{{CallManager.host}}/{{CallManager.username}}:
```

Create an Evaluation Macro

To write an evaluation macro:

1. Identify tests and result values:
   - A simple test resolves to True or False.
   - An *If-Then-Else* test resolves to a value.
2. For a simple test, identify the values and operator to resolve to True or False.
3. For an *If-Then-Else test*, identify *If-*, *Else-* and default conditions and values.

Evaluation Macro Examples

((DATA1.d1 == y))

Explanation: this macro evaluates to true or false if DATA1.d1 is equal to y.

((EVALUATE1.val == y)) <{{CallManager.host}}-Enabled> ((EVALUATE1.val == n))<{{CallManager.host}}-Disabled> <{{CallManager.host}}-Not set>

Explanation: this macro evaluates data model called “EVALUATE1” with attribute “val” - to value of data model called “CallManager” and with attribute “host”:

- Appended with “-Enabled” if EVALUATE1.val is y
- Appended with “-Disabled” if EVALUATE1.val is n
- Otherwise appended with “-Not set”