Virtual Product API

Release 15.5.0.12

CONTACT Software

Sep 04, 2018

CHAPTER 1

BOM Positions

1.1 cs.vp.bom.bomqueries

Collections of methods for efficiently querying a product structure.

`cs.vp.bom.bomqueries.`**`complete_flat_bom`**`(`**args*, ***kwds*`)`

Return a dictionary containg the flat boms of **all** the assemblies in the product structure of the roots.

The keys have the form:

(assembly_nr, assembly_idx)

The values are lists of Records, containing the flat bom.

Parameters

- **`arguments`** (*`positional`*) – instances of cs.vp.items.Item of which the flat boms have to be computed
- **`additional_condition`** – an sql condition which can be used to filter the result

`cs.vp.bom.bomqueries.`**`flat_bom`**`(`**roots*, ***kwargs*`)`

Return a RecordSet of all the bom positions present in the product structure of one of the roots. Computes the result efficiently making only one database query.

Parameters

- **`arguments`** (*`positional`*) – instances of cs.vp.items.Item of which the flat boms have to be computed
- **`additional_condition`** – an sql condition which can be used to filter the result
- **`searched_item`** (*`cs.vp.items.Item`*) – If given, only the bom positions in the usages structure of searched_item are returned
- **`variant_filter`** – if given the filter is applied and only the bom position in the filtered structure are navigated
- **`bomfilter`** – if given the filter is forwarded to bomfilter_func and applied and only the bom position in the filtered structure are navigated
- **`bomfilter_func`** – if given the function is called to apply the bomfilter
- **`part_attributes`** – Attributes from the relation teile_stamm, which have to be joined in the result.

Returns a record set containing all the bom position in the product structure of one of the given roots

`cs.vp.bom.bomqueries.`**`flat_bom_dict`**`(`**roots*, ***kwargs*`)`

Same as flat_bom but returns a dictionary. The keys are of the form (teilenummer, t_index) and the values are the children of the given item in the product structure.

`cs.vp.bom.bomqueries.`**`quantities`**`(`**roots*, ***kwargs*`)`

Computes for every part in the product structures of the roots the aggregated quantity.

Uses a recursive query if variant_filter is not given, otherwise steps through the structures.

Parameters

- **`roots`** – instances of cs.vp.items.Item of which the bom quantities have to be computed
- **`variant_filter`** – if given the filter is applied and only the bom position in the filtered structure are navigated
- **`bomfilter`** – if given the filter is forwarded to bomfilter_func and applied and only the bom position in the filtered structure are navigated
- **`bomfilter_func`** – if given the function is called to apply the bomfilter

Returns a record set containing all the bom position in the product structure of one of the given roots

`cs.vp.bom.bomqueries.`**`get_components`**`(`**args*, ***kwds*`)`

Given an item or a bom position return a list of its bom positions. Some part attributes can be accessed efficiently from the result of this method using the method `get_item_attr`.

Parameters

- **`item_or_component`** – an item or a bom position. It can be an instance of `cdb.objects.Object` or of `cdb.sqlapi.Record`
- **`searched_item`** – If specified only the bom positions in the usages structure of searched_item are returned. Must be an instance of `cdb.objects.Object`
- **`make_object`** – if true instances of `cdb.objects.Object` are returned, otherwise instances of `cdb.sqlapi.Record` are returned. Setting it to `False` will give better performance.

`cs.vp.bom.bomqueries.`**`get_item_attr`**`(`*comp*, *attr*`)`

Retrieves a part attribute from the bom position.

If the bom position has been constructed using get_components, it retrieves the attributes item_object_id, is_mbom and cdb_depends_on efficiently.

1.2 cs.vp.bom.diffutil.differences

Compute the differences between two product structures.

`cs.vp.bom.diffutil.differences.`**`get_differences`**`(`*lbom*, *rbom*, *product_object_id=None*, *variant_filter=None*, *bomfilter=None*`)`

Computes the differences between an engineering BOM and a manufacturing BOM.

Important: This method only works when the attribute `mbom_mapping_tag` of the bom positions is set correctly.

Parameters

- **`lbom`** (an instance of `cs.vp.items.Item`) – the engineering BOM
- **`rbom`** (an instance of `cs.vp.items.Item`) – the manufacturing BOM
- **`variant_filter`** – variant filter object, used to filter the BOMs
- **`bomfilter`** – bomfilter dict, used to filter the BOMs

Returns

an iterable which provides dictionary-like objects with the following keys:

- teilenummer
- t_index
- lbom_quantity
- rbom_quantity
- item_object_id

1.3 cs.vp.bom.mapping

Compute a mapping between two product structures.

`cs.vp.bom.mapping.`**`compute_mapping`**`(`**args*, ***kwds*`)`

Compute a mapping between the assembly components of litem and ritem by comparing the device tag. The mapping is returned as a dictionary where keys have the format

(baugruppe, b_index, teilenummer, t_index, position, variante, auswahlmenge)

and values are lists of tuples with the same format.

Returns (mapping, lunmapped, runmapped)

1.4 cs.vp.bom.usages

Efficiently compute the usages of some items inside a product structure.

`cs.vp.bom.usages.`**`get_all_usages`**`(`**roots*`)`

compute usages for all the parts under the bom.

Parameters **`roots`** – The boms under which the usages are searched for. Can be instances of `cs.vp.items.Items`, `cs.vp.bom.AssemblyComponent` or `cdb.sqlapi.Record`.

Returns a map (teilenummer, t_index) -> [(teilenummer, t_index)]

`cs.vp.bom.usages.`**`get_usages`**`(`*items*, **boms*`)`

Compute a list of those items, which use the given items in their product structure and are contained in the product structure of some given boms. Return their object ids

Parameters

- **`items`** (a list of instances of `cs.vp.items.Item`) – a list the items, of which the usages are searched for
- **`boms`** (instances of `cs.vp.items.Item`) – a list of boms, under which the usages are searched for

Returns a list of strings

cs.vp.classification.sml

Some subject characteristics tools

The following example shows how an property check can be done:

```python
class MyItem(cs.vp.items.Item):

    def on_modify_post_mask(self, ctx):
        errmsgs = sml.checkPropertyFormat(ctx.dialog, self.sachgruppe)
        if errmsgs:
            raise ue.Exception(1024, "\n".join(errmsgs))

    def on_modify_dialogitem_change(self, ctx):
        errmsgs = sml.checkPropertyFormat(ctx.dialog, ctx.object.sachgruppe,
                                          ctx.changed_item)
        if errmsgs:
            raise ue.Exception(1024, "\n".join(errmsgs))
```

`cs.vp.classification.sml.`**`checkPropertyFormat`**(*item*, *generic_group=''*, *property=''*)

Checks, whether the subject characteristic property values of `item` are correct. `item` is the classified part that contains the property values to be checked. If a value can't be accessed it will not be checked. `generic_group` is the subject characteristic that contains the definition of the properties. If it is empty it will be retrieved from `item`. If the parameter `property` is empty all numeric properties will be checked - if it contains an database attribute, only this attribute will be checked.

The function returns a list of internationalized error messages or an empty list, if all checks has been successful.

`cs.vp.classification.sml.`**`getFQSMLAttrIdentifier`**(*pset_id*, *prop_id*)

Retrieves an identifier that identifies the property within a `cdb.mom.CDBObjectHandle`. The function returns `None` if there is no property with the identification `prop_id` within the class list of characteristics `pset_id`. Note that `prop_id` is the value of `cdbsml_pset_prop.prop_mk`. You can use this identifier if you set a value within a user exit, e.g. if you call `cdb._ctx.cdbserver.Context.set`.

`cs.vp.classification.sml.`**`getSMLAttrIdentifier`**(*pset_id*, *prop_id*)

Retrieve an identifier that can be used to access the property `prop_id` of the subject characteristic `pset_id` within context adaptor objects like `ctx.dialog`. The function returns `None` if there is no property with the identification `prop_id` within the class list of characteristics `pset_id`. Note that `prop_id` is the value of `cdbsml_pset_prop.prop_mk`.

`cs.vp.classification.sml.`**`AddDescriptiveText`**(*part*, *attribute*)

Apply *`BuildDescriptiveText`* (page 4) to `part` and update part's `attribute` with the result. `part` should be an updatable of type `cdb.sqlapi.Record`. The function retrieves the template to generate the description from the part's property set (`part.sachgruppe`).

`cs.vp.classification.sml.`**`BuildDescriptiveText`**(*template*, *part*, *properties*)

Process a `template` string. Replace all occurences of `[modifier(propname)!expr|format]` with the data of a property named `propname` in one of the dictionaries `properties` or `part`.

List of Figures

List of Tables

Index