Virtual Product

Release 15.5.0.12

CONTACT Software

Sep 04, 2018

Contents

CHAPTER 1

Introduction

CONTACT Virtual Product provides a comprehensive model of the growing virtual product as support for work in progress. Its carefully harmonized components include products and product portfolios, variability models and product variants, components and their logistical variants in the form of articles or materials, product structures and BOMs, as well as configuration management for the control of product changes.

CHAPTER 2

BOM management

2.1 General details

A part (object of the `part` class) is simultaneously the BOM header. BOM items (objects of the `bom_item` class) are assigned to it.

When determining the BOM, the system usually uses one of the relationships from the *Relationship directory* that has a name beginning with `CDB::Relationship::STL`. For these relationships there is the special feature that, for performance reasons, no SQL select statement is transmitted if the `baugruppenart` attribute of the `part` class is empty. Instead, the system assumes in a case like this that there is no BOM for the part.

The system even automatically assigns the attribute when creating the first BOM item for a part with the value `Baugruppe`.

Note: Particularly for data transfer from another system, it must be noted that the `teile_stamm.baugruppe` database attribute for items with BOM receives the value `Baugruppe` - otherwise the BOM might not be visible.

2.2 BOM export from CAD systems

The content of this chapter covers configuring the server-side BOM export as used in conjunction with Workspace Manager. Refer to the selection titled “BOM export from CAD systems” in the *CONTACT Virtual Product* application manual.

2.2.1 Overview

The following configuration options are available:

- You can select from among various methods for exporting BOMs.
- You can configure which data from the generated BOM is displayed in the table preview.
- You can select attributes combined together based on their BOM items (default: `teilenummer` and `t_index`).
- You can select attributes whose value is updated automatically when repeatedly exporting (default: `menge`).
- You can adjust the increment used when issuing new item numbers (`STL Schrittweite` CAD configuration switch).

If a method for importing CAD BOM information (from a BOM editor) has been selected, you can also configure a mapping of CAD BOM information attributes to attributes.

Furthermore, BOM items for specific parts can be suppressed using object rules.

All of the settings are managed in groups with the exception of the increment. Each configuration group is assigned to a CAD system. In addition, the validity range for a configuration group can be limited to different users by using roles. The default settings apply to all CAD systems that do not have a special configuration available and to all users holding the "Engineering" common role. The priority of the configuration group is used to make decisions if ambiguities emerge. Ambiguities can emerge if there are several configuration groups for the same CAD system and an unambiguous selection cannot be made using user roles either. A *higher* value for "priority" means that a configuration group takes priority. The priority is assigned automatically but can be changed manually.

The configuration of the BOM export can be accessed in the navigation tree via *Administration/Configuration* → *Administration* → *Integrations* → *CAD Configuration* → *BOM Export*.

A restart is not required after changes have been made to the configuration.

2.2.2 Adjusting the preview

This section uses examples to cover how to configure the display of the preview, and to do so such that the new configuration is valid for all CAD systems.

Start this process by copying the default configuration and specifying a unique name for the copied configuration (e.g. *Adjusted Display*). Note that the value for the priority of the new configuration is assigned automatically.

Deleting the copy lets you return to the default configuration later on without issue.

The new configuration can now be modified by double-clicking on it. Select the *Table Columns* tab to adjust the display of the preview. Here you can modify, create or delete the columns. Choose Create.

This opens a mask for creating table columns. Here you have the option of selecting the *Attribute* to be displayed from a list. This provides both attributes from BOM items (at the highest level) as well as attributes for the associated part (accessible via the "Reference_1: Item" entry).

For instance, you can select the `mengeneinheit` attribute using the `Item` relationship.

For specific applications, it may be necessary to enter the attribute name manually. This is the case in instances when using what are known as "Localized Fields". "Localized Fields" are covered in the "Multilingual attributes" section of the PowerScript manual. The `Item.designation` attribute provides an example of access to a "Localized Field".

Another option of handling multilingual attributes is the use of the "CADDOK_ISOLANG". environment variable. If an `name_$(CADDOK_ISOLANG)` attribute is configured, for example, its value is read from the `name_de` or `name_en` (or `name_it`, `name_fr`, etc.) database attribute depending on the configured language.

A table column also requires a *Label* for display in the table header. General system labels are used for this. Select the `quant_unit` label, for example.

The *Position* of a table column determines its horizontal position in relation to other columns. The value automatically increases, but it can be adjusted to insert a new column between existing columns. For example, you could enter `35` to position the new column directly after the existing "Quantity" column.

2.2.3 Activating the automatic export of subassemblies

The *Generate from Component Structure* method is the default method. This method involves only generating a BOM for the part in the document where the operation was called.

In order to generate BOMs for the parts of referenced documents automatically, the *Generate from Component Structure (Recursively) BOM Generation Method* must be selected in the data sheet for a configuration group. It is selected from the list of options.

2.2.4 Configuring the attribute mappings for a specific CAD system

This section describes the creation of a configuration specifically for a selected CAD system. This assumes that the data from the system already contains BOM information and this information has also been transferred from

Workspace Manager in the form of file information (*.appinfo* files). This requires configuring the appropriate export method and demonstrating how values can be mapped to attributes from the BOM information.

Start by copying the `Default` configuration as a starting basis for the CAD-specific configuration to be created. Give the new configuration a unique name and select the *CAD system* from the list. The *Priority* is defined automatically *after* copying so that the new configuration replaces existing configurations for the same CAD system.

Set the *BOM Generation Method* to *Readout from Appinfo* in order to take over the BOM information from the files.

Finish copying and then double-click to open the new configuration. Switch to the *Appinfo Mappings* tab. Each entry in this table maps exactly *one* value to an attribute. Generate a new entry using the pop-up menu. On the mask, start by selecting the target attribute (*CDB attribute*). Note that each attribute can be used only once. The name of the value for the BOM information (*Appinfo name*) has to be entered manually. Consult the BOM display in Workspace Manager in this regard and take the linking-specific export settings into consideration.

Searches usually look for the value under the *Attributes* of a BOM item. Checking the *Property* check box causes searches to look under *Properties* instead.

Note: It is important to ensure that the value range of appinfo values and system attributes is compatible. If the values have to be converted, this can be implemented as part of reimplementing an export method.

Note: No attribute mapping with the target attribute `position` may be created. The attribute `position` will always be filled automatically from the appinfo attribute `posno`. If `posno` is lacking, new position numbers will be generated automatically.

2.2.5 Evaluating document references in BOM information

BOM items transmitted by Workspace Manager in the form of file information can contain references to the documents of used parts. These references are not taken into account when using the *Readout from Appinfo* export method for performance reasons, even if information on the used parts is missing (empty `teilenummer`).

Selecting the *Readout from Appinfo, with Document References* export method disables this restriction. Document references are resolved in the same way as in the *Generating from Component Structure* method in this case.

2.2.6 Adapting to CAD systems with semi-finished products (use of key attributes)

BOM items are combined in the default configuration if they refer to the same installed part, in other words, if `teilenummer` and `t_index` are identical. This results in the `menge` attribute being set to the total of the combined entries.

In some situations, it is desirable that imported BOM items are not combined even though they reference the same part. This could be the case when it comes to semi-finished products. This is why the BOM export can be configured so that items that differ in relation to certain attributes stay as standalone BOM items.

This requires that the attributes in question have been mapped to system attributes beforehand (see *Configuring the attribute mappings for a specific CAD system* (page 3)).

If this is the case, the corresponding attributes can be added on the *Key attribute* tab.

Note: Selecting attributes that are updated during repeated exports provides another configuration option. This is usually desired for key attributes. This means you should also add new key attributes under the *Synchronization attributes* tab.

Note: The attribute `position` may be used neither as a key attribute nor as an update attribute.

2.2.7 Excluding components using object rules

There are three object rules used to identify parts that are not used to generate a BOM item. These rules do not contain any predicates by default and, as a result, do not match any parts. Specifically, this includes the following rules:

- `BOM: Ignore Component Item`: This rule is applied to parts ("Items"). Each BOM item is checked to see if the rule applies to the installed part. A BOM entry is not generated if it applies. It is important to ensure that the `cs.vp.items.Item` *Type* is selected when defining predicates.

  This rule is taken into consideration for all export methods provided.

- `BOM: Ignore Component Document`: This rule is applied to documents. When handling CAD references, the system checks if the rule applies to the referenced document. The `cs.documents.Document` *Type* is selected for defining predicates.

  This rule is evaluated by the *Generate from Component Structure (Recursively)* and *Readout from Appinfo, with Document References* export methods.

- *BOM: Recursive Ignore Component*: This rule is applies to documents. It is used solely by the *Generate from Component Structure (Recursively)* export method. A separate BOM is not generated for referenced documents where the rule applies. The `cs.documents.Document` *Type* is selected for defining predicates.

Refer to *CONTACT Elements Platform: Administration and Configuration* for defining object rules.

2.2.8 Ignore variants using object rules

For some CAD systems, it can be desirable to ignore certain variants when exporting the BOM. In particular, no warning or fault message should appear if a part has not been assigned to a variant of this type.

There are two object rules for this purpose:

- `BOM: Ignore Variant`: CAD variants applicable to this rule are always ignored when determining the installed components. Instead, the main assembly (part of the CAD document) is used.
- `BOM: Ignore Singleton Variant`: CAD variants applicable to this rule are ignored if there is only a single variant for the model involved. In the default configuration, this rule is configured so that types of variants for the SolidWorks CAD system are filtered out.

The `cs.vp.cad.CADVariant` *Type* is selected for defining predicates. For defining the terms, the attributes of the CAD document themselves can also be added along with the attributes of the variant.

These rules are evaluated by the *Generate from Component Structure (Recursively)* and *Readout from Appinfo, with Document References* export methods.

2.2.9 Issuing item numbers

New BOM items normally contain item numbers determined using the highest item number included to date and the `STL Schrittweite` CAD configuration switch.

If a component has only been replaced by a higher index value when reconciling the BOM, however, a special case comes into effect: The item is reused if the part number is recognized as being unique. This situation is shown in the form of two items in the preview for technical reasons: A deleted entry with the old index value and a new entry with the new index value for the component.

Note: The existence of several replaced items for the part number has no effect in determining the order of the resulting items.

2.2.10 Using a project-specific method

Selecting the *Document Specific Method* export method for a CAD system enables the actual selection of the module to be delegated to the document.

In order to allow the selection of the method to be made on the document mask, the corresponding field first has to be reconfigured to be visible. This requires searching under *Administration/Configuration* → *Configuration* → *User Interface* → *Mask Configuration* → *Advanced* → *Mask Items* for elements with the `bom_method` value for the `Attribut` field. Uncheck the *Hidden* check box for the two mask items found this way. The client has to be restarted for this change to take effect.

An export method can now be selected for the affected documents (using the appropriate generation application (*BOM Generation Method* field).

If a document does not have a separate export method even though the document-specific method for the CAD system has been selected, then the system uses priorities and user roles to search for another configuration. If necessary, the default configuration is used.

2.2.11 Configuration of a custom method for BOM export

The implementation of export methods as part of customizing is covered in the English programming manual of the "Workspace" package. If an implementation such as this is present in the form of a class, it is very simple to integrate the new method into the system.

Use the create button for the *BOM Generation Method* field in the data sheet for a BOM configuration. On the mask, assign a unique, descriptive name for the new method and specify the full name for the new Python class (*Full qual. Python class name*). It is important that the Python class can be found by the objects framework dynamically. The export method can now be used in configurations.

If the class is not found, error messages are logged and a default method is used instead. The client has to be restarted for the changes to take effect if changes are made to the implementation's code.

2.2.12 Taking variants into account

Variants can be taken into consideration for resolving references when using the *Generate from Component Structure (Recursively)* and *Readout from Appinfo, with Document References* export methods.

This requires explicitly enabling the option for transmitting variant information in *Workspace Manager*. See *Workspaces* Administration manual, Chapter "Configuration".

Users must also assign parts to variants. This is described in detail in the *CONTACT Virtual Product* application manual (section *Using variants*).

There are not currently any other configuration option for this functionality.

2.2.13 Replace manually created BOM Items

Usually only positions with an identical CAD source will be considered during a comparison of a Bill of Materials. If manual positions have to be replaced automatically, this can be configured in the data Sheet of a Bill of Materials configuration. You can use the field *CAD source to be replaced* to do that.

If you enter *manual* here, all those manual positions will be replaced by automatically generated positions, which coincide with their key attributes. The values for *CAD source* and *Quantity* and further configured update attributes will be adjusted.

If the field *CAD source to be replaced* is empty, positions with varying CAD source, furthermore, won't be considered.

CHAPTER 3

Class List of Characteristics and Characteristics Groups

This chapter outlines how to configure and manage particular performance properties of the system in relation to class list of characteristics and characteristics groups.

3.1 Introduction

The CLC module contains the functions for administration and use of characteristics groups, characteristics and class lists of characteristics based on DIN 4000, 4001 and 4002. Only aspects of the administration are documented in the following. How to create, maintain and use class lists of characteristics is described in the *CONTACT Virtual Product* application manual (classification of parts, issuing characteristics, search based on characteristics, etc.).

Configuration access can be reached in the menu tree by selecting *Products* → *CLC*.

3.2 Characteristics Groups / Characteristics Group Hierarchy

The data that enables navigation to the class lists of characteristics from the menu tree or from the CLC structure catalog is generated from the entries in the characteristics group hierarchy. The `smfm` properties can be used to set the identifier for characteristics groups in the application area.

Important: After modifying the `smfm` property, the characteristics group hierarchy must be generated again in order for the modifications to take effect in the entire application area.

3.3 Characteristics

The general configuration of characteristics is described in the *CONTACT Virtual Product* application manual.

Subsequent modifications to characteristics definitions and, in particular, to the data type and the number of places before and after the decimal point should be made with caution. *Modification of the data type of characteristics* (page 9) indicates the options under the ORACLE database system.

Table 3.1: Modification of the data type of characteristics

From	To	Without data	With data
Int	Int	x	x (no effect on SQL type, DIN entry only)
Int	Float	x	x
Int	Text	x	–
Float	Float	x	x (no effect on SQL type, DIN entry only)
Float	Int	x	–
Float	Text	x	–
Text	Text	x	x (extension only, no reduction)
Text	Int	x	–

3.4 Generating a class list of characteristics

After creating or modifying a class list of characteristics and its characteristics, it must be "generated" initially or again in order for these modifications to become visible to the user.

The generation process should be initiated manually by selecting the *Rebuild Class Lists of Characteristics* pop-up menu item for the respective class list of characteristics (see *CONTACT Virtual Product* application manual). The generation methods can be found in the corresponding cdb.objects classes.

When generating the class list of characteristics, a Data Dictionary class (see *CONTACT Elements Platform: Administration and Configuration*, Section *Facets*) is generated or adapted based on the characteristics of the class list of characteristics and its properties. For this class, the standard methodology is then used to generate the schema, interface elements and operations. This methodology has been extended with the following aspects:

Label of the Class The general label for the characteristics group class can be set using the error message with the message label `cdbsml_label_desc`. Attributes of the `cdbsml_propset` relation can be used within the designation. The configuration `pset_id + " (" + name_de + ")"` could lead, for example, during runtime to the output of *10-0280-0020-0020 (wire nails)*.

Label of the Attribute The attribute designation in the data dictionary is derived from the characteristics identifier. Therefore SQL key words, such as `create`, `delete`, `update`, `table`, `lower`, etc., cannot be used. The identifier is adapted if necessary, for example, if it begins with numbers or underscores or contains capital letters.

Generating a Standard Mask If a class list of characteristics contains characteristics, the fields from the mask are first copied, before the attributes of the class list of characteristics are added to the mask. This template mask has to be defined for the `public` group and contains by default the fields in which the CLC Images are displayed. The `sml_base_mask` is used as the template for the *Information* and *Modify* operations. If the `sml_base_mask_n` exists, a Create mask is generated based on this template mask. If it does not exist, the Change mask is also used for creating. If special search masks are to be generated, use `sml_base_mask_s` as the template.

Generating Mask Fields The `teilenummer` and `t_index` fields are not added to the mask.

The field label is based on the `pmld` property.

Generating a Representation Table The `teilenummer` and `t_index` fields are not added to the table.

The field label is based on the `ptld` property.

Like other definitions, generated automatically for masks, mask elements, tables and table elements can be subsequently adapted with regard to the layout. Keep in mind that these adaptations may be overwritten for another generation using the `srmt=true` property or have to be maintained elsewhere for `srmt=false` CLC modifications (new, changed or deleted characteristics).

3.5 Deleting a class list of characteristics

The deletion of a class list of characteristics can be initiated by selecting the *Delete* menu item for the respective class list of characteristics. Confirm the confirmation prompt with *yes*. The corresponding entry is deleted from the bar. The referenced characteristics, views and hierarchy entries are deleted, which means only the relationships are deleted, however not the objects or elements themselves.

If the associated class has already been generated, classified parts can still be used. If this is not wanted, you have to delete this class and clear the `sachgruppe` field in the corresponding parts.

3.6 Integration of GUI elements

If a part is classified, the associated CLC mask tab is always displayed if a mask composition is configured for the corresponding operation in which a tab with the tab name `$Facet:sachgruppe` is configured. If such a tab exists in the screen array for creation of a new part, this tab is replaced by the screen configuration of the CLC class for the creation operation for the facet.

In tables, the table of the CLC class is shown where a field with the attribute `$Facet:sachgruppe` and the column type `Placeholder` is configured.

3.7 Properties

The following system-wide presettings (properties) are available which can be configured for specific persons and/or groups:

[string] smfm =‘<$NAME>‘ Defines how a characteristics group is to be named in the application area. Permissible values are:

$ID Classification flag.

$LONG The long designation of the characteristics group

$NAME Name of the class list of characteristics.

Combinations such as `$NAME` (`$ID`) are possible.

[string] smls =‘<>‘ Defines the sorting sequence of the characteristics group in the characteristics group hierarchy. If the value is `klassifikation`, sorting is based on the classification (`$ID`). Otherwise, they are sorted by the characteristics group name.

[boolean] srmt = <true> Permissible values are:

true Masks and tables which contain the characteristics are created automatically when generating class lists of characteristics. Any manual modifications made to the masks and tables are overwritten!

false No masks or tables are created when generating class lists of characteristics.

`true` applies to the initial generation, regardless of the value.

[string] pmld = <> Defines the format string for the generation of the mask labels for the individual characteristics. The characteristic label is generated according to the following standard pattern if the format string entry is empty, not present or faulty.

Characteristic Label (Characteristic Identification) [Characteristic Unit]

Note: If the standard generation results in a label with a character length that exceeds the permissible value for the associated database attribute, the characteristic label section is truncated accordingly. If a characteristic label of which the length of the characters exceeds the permissible value of the corresponding database attribute is generated on the basis of a valid explicit format string entry, this label is cut off accordingly.

The format string may contain variable, constant and/or optional elements. To combine the individual elements to form the complete label, all individual entries must be connected with a plus sign.

Format specifications

- Names for the corresponding attributes of the respective characteristic and its usage (referencing) are allowed as variable elements. Furthermore, additional known environment variables such as `CADDOK_ISOLANG` are permissible. They are entered in the form of `$(CADDOK_ISOLANG)`.
- Constant elements are entered in double quotation marks.
- Elements - in square brackets - can be indicated as optional. Elements indicated as optional are only accepted for the complete label if the evaluation of each of the individual variable elements results in a value which is not empty.

Example: `pmld` property

Specification of the format string for the generation of a class list of characteristics label

```
name_$(CADDOK_ISOLANG)+[" ("+prop_mk+")"]+[" \["+prop_unit+"\]"]
```

Note: Opening or closing square brackets or plus signs must be masked with a backslash when entered in a constant label element.

`[string] ptld ='<>'` Defines the format string for the generation of the table labels for the individual characteristics. The characteristic label is generated in the form of a three-line column header according to the following standard pattern if the format string entry is empty, not present or faulty.

```
Feature name
Feature characteristic
Feature unit
```

Note: If a characteristic label is generated of which the length of the characters exceeds the permissible value of the corresponding database attribute, the label is cut off accordingly.

The format string may contain variable, constant and/or optional elements. To combine the individual elements to form the complete label, all individual entries must be connected with a plus sign.

Format specifications

- Names for the corresponding attributes of the respective characteristic and its usage (referencing) are allowed as variable elements. Furthermore, additional known environment variables such as `CADDOK_ISOLANG` are permissible. They are entered in the form of `$(CADDOK_ISOLANG)`.
- Constant elements are entered in double quotation marks. Furthermore, the `'\n'` character can be used as a separator for the multiline arrangement of individual elements.
- Elements - in square brackets - can be indicated as optional. Elements indicated as optional are only accepted for the complete label if the evaluation of each of the individual variable elements results in a value which is not empty.

Example: `ptld` property

Specification of the format string for the generation of a class list of characteristics label

```
name_$(CADDOK_ISOLANG) + "\n" + prop_mk + "\n" + prop_unit
```

Note: Opening or closing square brackets or plus signs must be masked with a backslash when entered in a constant label element.

3.8 Characteristic Overall Search

The system provides the option of performing a characteristic overall search. To activate the characteristic overall search, the characteristic identifiers and the corresponding values must be integrated in the search mask. When configuring the corresponding mask fields, the value `cdb::argument::sml.` has to be put in front in the *Attribute* field. Fields configured in this way are displayed only for searches and are automatically suppressed in all other contexts. Such fields are also automatically suppressed if a search is made anyway within the context of a characteristics group.

For the IDs, the value `cdb::argument::sml.propid_nr` is configured in the *Attribute* field; `nr` can accept values from `01` to `99`, thus `cdb::argument::sml.propid_01`, `cdb::argument::sml.propid_02`, etc., `cdb::argument::sml.propid_99`. For the assigned values, the value `cdb::argument::sml.propval_<nr>` is specified in the *Attribute* field.

If values are entered in the fields for the characteristic overall search, all characteristics groups which contain all specified characteristics are edited to the search. The selected characteristics are automatically displayed at the start of the assigned view table. It is not possible to sort the result list according to multiple criteria.

Characteristics groups should never contain the same characteristic several times. If this is nevertheless the case, the entered search criterion must apply to all values of the characteristic. Only one value is displayed in the view table.

CHAPTER 4

Variant Editor

4.1 Open in CAD Systems

The Variant Editor offers the possibility to open a variant in a CAD System. You can extend this functionality to support further authoring systems. For this you must write a plugin and register it within the Variant Editor. You can define the plugin in any Python module.

In order to register a plugin, you must import the method *register_plugin* from the module `cs.vp.variants.apps.generatorui`. You can do it as follows:

```
from cs.vp.variants.apps.generatorui import register_plugin
```

The method `register_plugin` expects a Python Dictionary as parameter, which must contain the following key/value pairs:

- `icon`: Name of the icon for the button
- `label`: Name of the label for the tooltip
- `json`: a JSON method
- `json_name`: the public name of the JSON method

These parameters **must** provide values in order to assign a new CAD system to the dropdown menu successfully.

Additionally you can provide the following optional parameters:

- `position`: Determines the position in the dropdown menu.
- `open_new_window`: Determines if the operation should open a new browser window. The default value is `True`.

The CAD systems will be displayed in the dropdown menu in descending order according to their position number.

If the parameter `position` is not provided, the default value 0 will be assigned. That means that all the registered plugins whose attribute `position` does not contain an explicit value will be added at the end of the dropdown menu. All the plugins whose attribute `position` has been registered with an explicit value > 0 will be displayed in the dropdown menu at a higher position.

An example for the call of the method `register_plugin` could be:

```
register_plugin({"icon": "cdbvp_show_in_catia",
                 "label": "cdbvp_show_in_catia",
                 "json_name": "show_in_catia",
                 "json": show_in_catia,
                 "position": 10})
```

The ID of a configured icon plus the ID of a configured label are provided. Additionally a JSON method and ist name are provided. Finally the optional parameter `position` is set to determine the order of the entries in the dropdown menu.

The JSON method should return a python dictionary which contains the key `url` on the one hand and a value which can be processed by the client on the other hand. This can be for instance the URL of an operation. You can find an example of such a JSON method in `cs.vp.variants.apps.generatorui.show_in_catia`.

4.2 Embedded mode

The variant editor has an *embedded mode*, which allows you to embed it as a Panel in the client, or in an external application. If the embedded mode is active the title bar will not be shown and the *product details* area is per default collapsed.

To activate the *embedded mode* call the variant editor with the URL param `embedded=1`.

4.3 Columns lock

The variant editor gives you the ability to lock the first four columns, which contains descriptive information on variants, while scrolling horizontally. You can set the default behavior using the property *vecf*. Set it to *true* if the lock should be active by default. You can find properties under *Administration/Configuration -> Configuration -> Properties*.

Each user can activate or deactivate this lock, using a button in the toolbar.

CHAPTER 5

Variant matrix

5.1 Adapting the toolbars

A toolbar with various operations for the selected variant is available in the top right section of the variant matrix.

The `VariantMatrixVariant` operation context allows you to define which operations should be available in this toolbar. The operations must be visible in the menu and must be applicable to a `SingleObject` of type `cdbvp_variant`.

CHAPTER 6

Filtering Maximum BOMs

Developping interfaces can be accomplished by filtering Maximum BOMS on the basis of python expressions.

In this case you create fewer objects as if you created regular selection predicates.

To use python expressions as a filter, you must instantiate one object of the class `cdbvp_bom_string_predicate` per BOM Item to be filtered. This is possible using the REST API. The expression will be saved in the attribute `expression`.

The expressions can contain python operators and constants. Variables will be evaluated as follows:

1. On the basis of the variable's name the corresponding characteristic will be searched via the attribute *ERP Code*.
2. The attribute `ERP Code` of the property value in the variant replaces the variable.

CHAPTER 7

mBOM Manager

7.1 Registration of operations

The mBOM Manager already provides a set of standard operations. These standard operations can be extended by custom operations via a registration mechanism. Therefore you have to implement an operation in form of a Python dictionary and register this operation into the mBOM Manager. The implementation of an operation can be done in any module.

For registration of operations the method *register_operation* of module `cs.vp.bom.diffutil.pages` is needed. Importing has to be done as follows:

```
from cs.vp.bom.diffutil.pages import register_operation
```

The method `register_operation` expects a Python dictionary as parameter which has to provide following key-value-pairs:

- `name`: Name of the configured operation
- `is_ebom_op`: Flag for assignment of operation to eBOM or mBOM
- `is_item_op`: Flag for assignment of operation to a part or a bom item

The parameters **must have** assigned values to successfully register a new operation to particular dropdown menus of the mBOM Manager.

In addition there are three optional parameters:

- `needsReload`: Flag for reloading after operation call, default value True
- `unselect`: Flag for deselection of tree node after operation call, default value True
- `multi_select_op`: Determines if operation is callable on multiple parts, default value False

The operations will be displayed in order of their registration.

An exemplified call of `register_operation` function would be as follows:

```
register_operation({"name": "CDB_Workflow",
                    "is_ebom_op": False,
                    "is_item_op": True,
                    "needsReload": True,
                    "unselect": False,
                    "multi_select_op": False})
```

The name of a configured operation is passed. Additionally a flag is provided which determines that the operation is only available for mBOM nodes and another flag determines that the operation can only be called on parts. Besides two optional parameters are provided to determine that the eLink application has to be reloaded after call of operation and to determine that the part should not be deselected after call of operation. Furthermore the last parameter determines that the operation is only callable on a single part.

7.2 Controlling display of difference table

With the help of the object rule `mBOM Manager: Ignore differences` you are able to configure which differences shall **not** be displayed in the differences table. The object rule checks on part level. All mBOM parts which do not have a reference to an eBOM are filtered out by this object rule by default.

7.3 Synchronization of engineering and manufacturing BOMs

The synchronization of engineering and manufacturing BOMs requires that the attribute `mbom_mapping_tag` of the class `bom_item` is set correctly.

The attribute contains a mapping between the bom positions of the engineering bom and those of the manufacturing bom. Correspondants positions must have the same value for the attribute `mbom_mapping_tag`.

If the manufacturing is created using the standard operations the attribute is automatically set. If you create the manufacturing using an interface or a third party system, you should ensure that the attribute is set correctly.

List of Tables

C

E

I

J

L

M

N

O

P

R

S

U

V