BOM Generation

Release 15.2.1.7

CONTACT Software

Aug 13, 2018

Contents

CHAPTER 1

Introduction

CONTACT BOM Generation enables the automatic, rule-based generation and synchronization of engineering BOMs with 3D CAD model structures.

1.1 Powerscript API

PowerScript is the development environment of the system, which enables you to develop add ons, user exits, stand alone scripts or complex applications based on or integrated with the *CONTACT Elements Platform*. This document only describes the APIs that are important for the `bomcreator` package. A powerscript manual is part of the *CONTACT Elements Platform* documenation.

CHAPTER 2

Customization of BOM algorithms

2.1 Implementing a new BOM reader

The cs.vp.bomcreator module offers a general mechanism to automatically derive a BOM (bill of materials) from CAD documents (and other business objects) with appropriate information. Examples for concrete implementations of this mechanism are contained in the *CONTACT Workspaces* package. These implementations work on CAD documents that were imported with the Workspace Manager.

In this section, you will learn how to extend this functionality with your own algorithms. This gives you the flexibility to use arbitrary files, file structures or system objects as the source for your automatically generated bills of materials, if you are willing to invest in the implementation effort.

To read about this functionality from a user perspective, see section "Stuecklisten-Ausleitung aus CAD-Systemen" in chapter "Stuecklisten und Produktstrukturen" of the *CONTACT Virtual Product* user manual. To learn about the configuration options, read chapter "Stuecklistenverwaltung", section "Stuecklistenausleitung aus CAD-Systemen" in the *CONTACT Virtual Product* administration manual.

2.1.1 Example: Extracting a BOM from a CSV file

BOM readers are implemented as classes that directly or indirectly are derived from the abstract base class *cs.vp.bomcreator.bomreader.BOMReader* (page 9). This class defines a very flexible interface. Most custom implementation don't need this flexibility and should derive from *cs.vp.bomcreator.bomreader.SimpleBOMReader* (page 8) instead.

As a simple example, the following listing shows the implementation of a BOM reader that reads its information from a file with lines of comma-separated values. The implementation assumes that it is invoked on a document with a text file as the first primary file.

```
from cdb.objects import ByID
from cs.vp.bomcreator.bomreader import SimpleBOMReader
import csv


class CSVBOMReader(SimpleBOMReader):
    """
    A BOM-Reader that interprets a CSV-file as a bill of materials.
    """

    def fill_BOM(self, bom_context):
        """
        :Parameters:
         - bom_context: bomreader.BOMContext
        """
        doc = ByID(bom_context.object_id)
        if doc.PrimaryFiles:
            anchor_file = doc.PrimaryFiles[0]
```

```
            content = anchor_file.get_content()     # String
            lines = content.splitlines()            # list of Strings

            # now parse the lines and create a BOM item for each
            # assumption: every line has the format "part number,part index"
            for part_no, part_index in csv.reader(lines):
                self.add_position(teilenummer=part_no,
                                  t_index=part_index)
```

As you see, this class only has one method: `fill_BOM()`. It takes a *cs.vp.bomcreator.bomreader.BOMContext* (page 9) as its sole argument. From this `bom_context` object the method navigates to the first primary file of the document and extracts its contents. The contents string is split into lines which are parsed as pairs of part numbers and part indices. Using this part information, the method fills the internal BOM object by repeated calls to `add_position()`.

Before it can be used, the class has to be made available to the system. See section "*Integrating a new BOM reader* (page 6)".

This implementation has some weaknesses which will be discussed in "*Improvements to the example* (page 3)". But first we describe the features that even this simple implementation gets for free.

2.1.2 Advantages of using a `SimpleBOMReader`

BOMs generated by custom BOM readers

- will automatically be merged with existing BOMs in the database,
- are displayed in the interactive preview, and
- will be stored in the database after user confirmation.

When creating a BOM with a class derived from `SimpleBOMReader`, it will also automatically take into account any configuration settings:

- BOM items will be merged using the correct key attributes.
- Positions will be displayed according to the configured display columns.
- When storing positions in the database, only the configured attributes will be updated.
- New BOM items without a value for the `position` attribute will be assigned a position value according to the configured step width.

2.1.3 Improvements to the example

The implementation of the `CSVBOMReader` in the example above has some problems. This section shows how to avoid them.

- Reading the complete file contents with the `get_contents()` method can be problematic with large files. You should check the `cdbf_size` attribute before, or use a sandbox instead.
- The implementation does not handle the case where a line does not match the expected format.

  But more importantly, it also does not check whether the item that a position references, actually exists. BOM reader implementations should always verify the existence of items. Otherwise the whole BOM import will fail and the user will see an error message only after they tried to save the BOM.

  Both problems can be fixed by rewriting the `for`-loop like this:

```
for line_tuple in csv.reader(lines):
    if len(line_tuple) != 2:
        self.add_error("Unexpected line: '%s'" % str(line_tuple))
    else:
        (part_no, part_index) = line_tuple
```

```
        item = Item.ByKeys(part_no, part_index)
        if item is None:
            self.add_error("Item does not exist: '%s-%s'"
                           % (part_no, part_index))
        else:
            self.add_position(teilenummer=part_no,
                              t_index=part_index)
```

This implementation needs an additional import:

```
from cs.vp.items import Item
```

The messages passed to `add_error()` will be shown to the user, who can then decide whether they want to save the BOM nevertheless.

- The implementation does not respect object rules which may be configured to filter some BOM items. This can be fixed by using the method `ignore_component_item()` and change the last statement to:

```
if not self.ignore_component_item(item):
    self.add_position(teilenummer=part_no,
                      t_index=part_index)
```

2.1.4 Merging attribute values of merged positions

When multiple positions are merged on the basis of their key attributes, the `menge` attribute is automatically aggregated. Other attributes are not merged, however. In this case, the value of the position that was added first is always used.

If you need to change this behaviour, you can check the return value of the `add_position()` method. Whenever the call to `add_position()` results in a merge with an existing position, the existing position is returned as an instance of `cs.vp.bomcreator.bom.BOMEntry`. Otherwise `None` is returned. The `attrs` field of `cs.vp.bomcreator.bom.BOMEntry` is a dictionary and can be manipulated in custom code. In this example, we manipulate the `auswahlmenge` attribute (which is NOT identical to `menge`):

```
existingEntry = self.add_position(teilenummer=part_no, auswahlmenge=2.0)
if existingEntry is not None:
    # update auswahlmenge manually
    existingEntry.attrs['auswahlmenge'] += 2.0
```

2.1.5 Alternatives to `SimpleBOMReader`

Customizations that need to create and display more than one BOM in one step, have to derive from the more general interface *cs.vp.bomcreator.bomreader.BOMReader* (page 9) instead.

It is an an important design decision whether to use the methods of the *cs.vp.bomcreator.factory.Factory* (page 12) class to create BOMs and BOM items or whether to create these objects manually. If you use the factory methods, the resulting BOMs will automatically respect the current configuration settings. If you want to work independently from these settings, you should avoid the factory methods.

The following example illustrates a BOM reader that creates two BOMs using the `Factory` class. It assumes that there are two pre-existing items `self.part` and `self.assembly2`:

```
class MultipleBOMReader(BOMReader):
"""
An example of a BOMReader that create more than one BOM.
"""
    def __init__(self, bom_context, factory, custom_state):
        # MISSING: prepare test data self.part and self.assembly2
```

```
        # create the main bom for the main assembly (taken from the CAD document)
        self._bom = factory.create_BOM()
        # ... and add a position
        self._bom.create_and_add_entry(teilenummer=self.part.teilenummer, t_
↪index=self.part.t_index, menge=1.0)

        # create another BOM for our test assembly, also containing a position
        bom2 = factory.create_BOM_for_assembly(self.assembly2)
        bom2.create_and_add_entry(teilenummer=self.part.teilenummer, t_index=self.
↪part.t_index, menge=1.0)

        # return both BOMs as a list
        self.boms = [self._bom, bom2]
```

2.2 Customizing an existing BOM reader

Often it is not necessary to implement a new BOM method from scratch when handling customization requests. If the requirements cannot be met with configuration changes and object rules alone, it may suffice to extend an existing class by overriding a specific method.

All existing implementations support an explicit customization point: the method `adjust_and_filter()`. We will show how to use it in the next section.

2.2.1 Example: Converting attributes from Workspace Manager file information

For this example, we assume that we are importing the BOM from a CAD system, using the Workspace Manager. We also assume that the package *CONTACT Workspaces* is installed.

The problem that is example tries to solve is: the `menge` attribute does not have the required format. It uses the comma as the decimal separator instead of following the English convention. To convert the value, we can use a customization that like this:

```
from cs.vp.bomcreator.cadbominforeader import CADBomInfoReader

class CustomCADBomInfoReader(CADBomInfoReader):
    def adjust_and_filter(self, attributes):
        menge = attributes.get('menge')
        try:
            if isinstance(menge, basestring):
                menge2 = string.replace(menge, ',', '.')
                attributes['menge'] = float(menge2)
            return True
        except ValueError:
            self.add_error("Could not convert 'menge' value: '%s'" % str(menge))
            return False
```

`adjust_and_filter()` is a method that is supported by all BOM reader implementations. It works on the level of single BOM items and allows to change the value of attributes, to add new attribute values, and to discard a position completely.

As its input, the method takes a dictionary of attribute values. This dictionary may contain a value for any of the attributes of the system's `bom_item` class. However, if the data comes from an external source, you cannot make any assumptions about which attributes are present or about the format of their values.

The above example checks whether there is a `menge` attribute and whether it is a string. If this is the case, it replaces all commas with periods, tries to convert the resulting string to a float and stores the float in the dictionary. If no error occurs, the method returns `True` which indicates that the BOM position should not be discarded. If an

exception occurs because the `menge` string still cannot be converted to float, the method returns `False` and the BOM item will be filtered out.

> **Note:** This specific implementation can be problematic if input data contains commas as thousands separators. In this case you need a more sophisticated algorithm.

2.3 Integrating a new BOM reader

A new class like `CSVBomReader` should be part of a new or existing CONTACT Elements package. In this way, custom BOM implementations can easily be distributed and updated.

The fully qualified Python class name must be configured as described in section "Konfiguration einer eigenen Methode zur Stuecklistenausleitung" in the *CONTACT Virtual Product* administration manual.

2.4 Using callbacks

Sometimes it is necessary to maintain additional database relations whenever BOM items are written or deleted. The easiest way to do this is to override one or more of these methods in your `BOMReader` implementation:

- `post_write_component()`
- `post_write_bom()`
- `post_write_boms()`

However, you should be aware that this only works with BOMs which were created with one of the methods in the *cs.vp.bomcreator.factory.Factory* (page 12) class. For details, see the documentation of *cs.vp.bomcreator.bomreader.BOMReader* (page 9).

> **Warning:** You cannot use instance variables in these methods. If you need persistent values, use `set_user_attribute()` of class *cs.vp.bomcreator.bom.GeneratedBOM* (page 14) instead.

2.5 Shared position numbers

Position numbers are typically automatically created for every new BOM entry, using the configured step width. If your goal is to have multiple BOM entries which receive the same generated position number, you can use the class *cs.vp.bomcreator.bom.SharedPosition* (page 16). For example:

```
shared_pos = SharedPosition()
bom.create_and_add_entry(teilenummer=part_nr_1, position=shared_pos)
bom.create_and_add_entry(teilenummer=part_nr_2, position=shared_pos)
```

The two entries will receive their common position number when the BOM is synchronized with the database.

The class `SharedPosition` can also be used with the method meth:`add_position` of BOM readers.

2.6 Customizing the order of entries for generating position numbers in CADBomInfoReader

When position numbers are automatically assigned to new BOM entries, the order of entries as added to the BOM is taken into account, e.g. earlier entries get a lower position number.

But sometimes the required behavior is different. Implementations derived from the class `CADBomInfoReader` can override the method `sort_key()` to influence the order. For example, to sort by the attribute "netto_breite" before assigning position numbers, add this implementation to your BOM reader:

```
def sort_key(self, entry):
    return entry.attrs['netto_breite']
```

By default, `CADBomInfoReader` uses the order of the XML elements in the appinfo file.

2.7 Configuring an operation for other classes than CAD documents

This package comes with a predefined interactive operation for creating BOMs from CAD documents. This operation is called `cdbwsm_createbom` and belongs to the data dictionary class `model`. The operation opens a preview of the BOM and allows the user to save it to the database.

It is possible to define such an operation for other data dictionary classes. Classes must fulfill these requirements:

- They *must* have an attribute `cdb_object_id` with the usual semantics.
- They *must* have a 1:1 PowerScript relation named `Item` which returns the associated item. This item will receive the BOM when it is saved by the user.
- They *may* have an attribute `erzeug_system` which describes the associated application. The application determines the applied BOM configuration and the value of the `cadsource` attribute of the generated components.

The operation must be defined as a `SingleObject` operation of type `PowerScript`. To implement the operation, connect to it and start the preview application like this:

```
from cdb import sig

@sig.connect(OwnClass, "operation_name", "now")
def create_bom_from_own_class(self, ctx):
    ctx.url("powerscript/"
            "cs.vp.bomcreator.app/"
            "bomcreator_confirm?cdb_object_id=%s" % self.cdb_object_id)
```

Or, if your class does *not* have an attribute `erzeug_system`:

```
from cdb import sig

@sig.connect(OwnClass, "operation_name", "now")
def create_bom_from_own_class(self, ctx):

    cadsource = "ProE"  # replace with your own application!

    ctx.url("powerscript/"
            "cs.vp.bomcreator.app/"
            "bomcreator_confirm?cdb_object_id=%s&cadsource=%s" %
            (self.cdb_object_id, cadsource))
```

The built-in `BOMReader` implementations all only work on CAD documents. So you also need to implement your own `BOMReader` if your new operation should do something useful. The `cdb_object_id` of the business object will be available as part of the *`cs.vp.bomcreator.bomreader.BOMContext`* (page 9) that is passed to your `BOMReader`.

2.8 Reference

2.8.1 cs.vp.bomcreator.bomreader

This module contains the classes and interfaces that are needed for the most common customization scenarios.

SimpleBOMReader

class cs.vp.bomcreator.bomreader.**SimpleBOMReader**(*bom_context*, *factory*, *_custom_state*)

Bases: *cs.vp.bomcreator.bomreader.BOMReader* (page 9)

Base class for BOM reader implementations that

- create only a single BOM
- should automatically respect the current configuration (for display, merging and updating)

Derived classes only have to implement *fill_BOM()* (page 8).

They don't need to define their own `__init__` method.

Instances of derived classes are always created by the framework, never by user code. Instances are only used *once*. If a new BOM creation process is started, a fresh instance is created.

add_error(*error_msg*)

Adds an error message that will be logged and shown to the user. Calling this method does not prevent a successful BOM import.

Parameters

- `error_msg`: String

add_position(***attrs*)

Adds a position to the BOM of this reader.

Depending on the configuration and the data, this may not actually add a new position. It may instead just increase the value of the `menge` attribute of an existing position. In this case, the existing position is returned.

This method respects the result of the inherited method `adjust_and_filter()`, so the position may not actually be added if a custom implementation filters it.

Parameters

- any of the attributes of the CIM DATBASE class `bom_item` may be used as a keyword argument
- `teilenummer`: part number; **required**
- `t_index`: part version; optional; if not given, the latest version of the part is used
- `position`: determines the item order of a BOM; if not given, an increasing value is automatically assigned
- `menge`: amount; if not given, it will be set to 1.0
- ...

add_warning(*msg*)

Adds a warning message that will be logged and shown to the user.

Parameters

- `msg`: String

fill_BOM(*bom_context*)

This method must be overridden by derived classes.

The method has the task to fill the associated BOM by calling *add_position()* (page 8). It may call *add_error()* (page 8)/*add_warning()* (page 8) to show hints to the user.

It may also raise *BOMFatalError* (page 9) in case of errors that prevent any creation of a meaningful BOM.

Parameters

- bom_context: *BOMContext* (page 9)

BOMContext

class cs.vp.bomcreator.bomreader.**BOMContext**(*object_id*, *teilenummer*, *t_index*, *cadsource*, *global_user_hints*, *variant_id=None*)

Information about the objects involved in a BOM import.

This class bundles the input to a BOM creation method. It is a simple data container.

Attributes

- **object_id: String** ID of the business object that the operation was applied to. For now, this is always a document, but implementations of BOMReader should not rely on this assumption.
- **teilenummer: String** part number of the assembly that will receive the generated BOM
- **t_index: String** part version of the assembly that will receive the generated BOM
- **cadsource: String** application of the business object
- **global_user_hints: cs.vp.bomcreator.bom.UserHintList** list of global errors and hints
- **'variant_id': String or None** optional variant id of the variant whose BOM should be created In this case, (teilenummer, t_index) identifies the assembly of the variant.

BOMFatalError

class cs.vp.bomcreator.bomreader.**BOMFatalError**

Should be raised from *BOMReader* (page 9) implementations when no BOM could be created at all. As a result, the "Save" button will not be visible in the user interface, and the framework will not attempt to use the result of the reader.

BOMReader

class cs.vp.bomcreator.bomreader.**BOMReader**

Abstract Base Class for all BOM reader implementations.

This is the most general interface. For most custom implementations, *SimpleBOMReader* (page 8) will be the better choice as the base class.

Derived classes must have a __init__ method with the following signature:

```
def __init__(self,bom_context,factory,custom_state)
```

- **bom_context: *BOMContext* (page 9)** input data
- **factory: *cs.vp.bomcreator.factory.Factory* (page 12)** factory to create BOMs and BOM entries in accordance witht the current configuration

- **custom_state: dictionary** may contain implementation-specific state that is needed across multiple, recursive invocations of a reader

__init__ may raise a *BOMFatalError* (page 9).

If it does not raise an error, instances must have an attribute boms which contains a list of *cs.vp.bomcreator.bom.GeneratedBOM* (page 14) (typically a singleton list, but multiple BOMs are supported, too)

Instances of BOMReader are only used *once*. If a new BOM creation process is started, a fresh instance will be used.

Implementations that want to support recursive invocation also need to supply a list of IDs of encountered documents in the attribute referenced_docs and optionally a list of the variant ids of the those documents in 'referenced_variant_ids'. see *cs.vp.bomcreator.bomreader.make_recursive_reader_type()* (page 12).

Derived classes may have a static method is_applicable_to():

```
@staticmethod
def is_applicable_to(object_id)
```

which must return False if the reader type cannot be used on the given object. object_id is typically the cdb_object_id of a document, but may also be the id of other business object types in future implementations.

adjust_and_filter(*attributes*)

This method is a customization point to create specialized implementations that derive from existing BOM readers.

attributes is a dictionary of bom_item attribute values. No guarantees are given about which attribute values are contained or about their format.

The attributes may be modified to adjust the input data.

Returns a Boolean indicating whether the entry with these attributes should be kept (True) or discarded (False).

It is the responsibility of concrete implementations to **call** this method and respect the result.

It is the responsibility of customizations of these implementations to **override** this method if they want to filter or adjust BOM entries.

ignore_component_document(*document*, *user_hints=None*)

Checks whether the given document matches the object rule BOM: Ignore Component Document.

ignore_component_item(*item*, *user_hints=None*)

Checks whether the given item matches the object rule BOM: Ignore Component Item.

ignore_variant(*cad_variant*, *user_hints=None*)

Checks whether the given CADVariant matches the object rule BOM: Ignore Variant or whether it is the only variant and matches the rule BOM: Ignore Singleton Variant.

log(*msg*)

Write a log message, prepended by the class name.

post_write_bom(*bom*, *user_hints*)

This method is called after all components of a BOM have been written to the database but before the transaction is committed and warnings and errors are displayed. By default, it does nothing.

Derived classes may verify constraints and possibly cancel the process by raising an exception derived from RuntimeError.

WARNING: You cannot use instance variables in this method. If you need persistent values, use set_user_attribute() of *cs.vp.bomcreator.bom.GeneratedBOM* (page 14) instead.

Parameters

- bom: *cs.vp.bomcreator.bom.GeneratedBOM* (page 14)
- user_hints: *cs.vp.bomcreator.bom.UserHintList* (page 15)

post_write_boms (*boms*, *user_hints*)
This method is called after all BOMs of a document have been written to the database.

If multiple BOMReader classes have been used to create the BOMs, this method will only be called for the BOMReader class of the first BOM.

bom contains only the successfully written BOMs.

WARNING: You cannot use instance variables in this method. If you need persistent values, use set_user_attribute() of *cs.vp.bomcreator.bom.GeneratedBOM* (page 14) instead.

Parameters

- boms: list of *cs.vp.bomcreator.bom.GeneratedBOM* (page 14)
- user_hints: *cs.vp.bomcreator.bom.UserHintList* (page 15)

post_write_component (*bom*, *bom_entry*, *user_hints*)
This method is called immediately after a component of a BOM has been written to the database. By default, it does nothing.

Derived classes may inspect the attributes and status of the component (bom_entry.attrs, bom_entry.status), add warnings and errors to the user_hints or even cancel the process of writing the BOM by raising an exception derived from RuntimeError.

WARNING: You cannot use instance variables in this method. If you need persistent values, use set_user_attribute() of class *cs.vp.bomcreator.bom.GeneratedBOM* (page 14) instead.

Parameters

- bom: *cs.vp.bomcreator.bom.GeneratedBOM* (page 14)
- bom_entry: cs.vp.bomcreator.bom.BOMEntry
- user_hints: *cs.vp.bomcreator.bom.UserHintList* (page 15)

pre_show_component (*bom*, *bom_entry*, *user_hints*)
This method is called immediately before the preview is generated. At this point, automatically generated position numbers have become available. By default, it does nothing.

Derived classes may inspect and change the attributes (bom_entry.attrs).

If values of an existing component are adjusted, the method is responsible for adjusting the status of the component! You should use update_attribute() of class :class:'cs.vp.bomcreator.bom.BOMEntry'.

WARNING: You cannot use instance variables in this method. If you need persistent values, use set_user_attribute() of class *cs.vp.bomcreator.bom.GeneratedBOM* (page 14) instead.

Parameters

- bom: *cs.vp.bomcreator.bom.GeneratedBOM* (page 14)
- bom_entry: cs.vp.bomcreator.bom.BOMEntry
- user_hints: *cs.vp.bomcreator.bom.UserHintList* (page 15)

use_kernel_operations()
Whether to use kernel operations for saving changes in BOMs.

If True, the BOMs are saved similarly to running the operation interactively. Notably, user exits (customization code) will be executed.

If False, the BOMs are saved using a lower-level mechanism which does not call user-exits but is much faster.

Returns boolean

make_recursive_reader_type

cs.vp.bomcreator.bomreader.**make_recursive_reader_type**()

Given a class that implements the BOMReader interface, creates a class that also implements this interface, but additionally creates BOMs for the referenced docs.

In other words: Takes a non-recursive bom reader C and creates a recursive one, based on C.

Ignores a document if it has encountered the document before (breaking cycles).

Parameters

- **reader_class: class** a class derived from cdb.bom.bomreader.BOMReader instances of this class have to return a list of IDs of referenced documents in the attribute referenced_docs
- **result_order: ResultOrder** how to sort the tree of BOMs; either ResultOrder.DepthFirst or ResultOrder.BreadthFirst
- **max_depth: int** maximum recursion depth (0=no limit)

2.8.2 cs.vp.bomcreator.factory

Factory

class cs.vp.bomcreator.factory.**Factory**(*bom_context*)

Creates

- instances of *cs.vp.bomcreator.bomreader.BOMReader* (page 9) (varying in their concrete subtype)
- instances of BOMEntry (varying in the attributes they contain), and
- instances of GeneratedBOM (only varying in the headers for display)

based on the currently valid configuration.

The valid configuration depends on the CAD source and on the roles of the current user.

Instances of this class are used only *once*. They are not recycled for new BOM creation processes.

The __init__ method finds out the correct configuration to use and loads and evaluates it.

Parameters

- **bom_context: *cs.vp.bomcreator.bomreader.BOMContext* (page 9)** input data

Custom implementations of BOM readers never create a factory manually. It is passed to them via their __init__ method instead.

create_BOM()

Creates a *cs.vp.bomcreator.bom.GeneratedBOM* (page 14) that is adjusted according to the configuration and which represents a BOM of the item associated with this factory.

create_BOMEntry(***attrs*)

Creates a BOMEntry according to the configuration.

Parameters

- **attrs: dictionary containing values of bom_item attributes;** must contain at least teilenummer

create_BOMReader(*custom_state=None*, *create_boms=True*)

Creates an object implementing the *cs.vp.bomcreator.bomreader.BOMReader* (page 9) interface. The concrete type of the returned object depends on configuration settings.

Parameters

- custom_state: dictionary (internal state of implementations)

create_BOM_for_assembly(*item*)

Creates a *cs.vp.bomcreator.bom.GeneratedBOM* (page 14) for the given item. The generated BOM will respect the configuration settings of this Factory but is not attached to the original item.

Parameters

- item: a cs.vp.items.Item

create_assembly_and_BOM(***kwargs*)

Creates a new cs.vp.items.Item and a *cs.vp.bomcreator.bom.GeneratedBOM* (page 14) attached to this item.

WARNING: This must only be used if a matching assembly does not yet exist, i.e. a call to this method should always be guarded by condition for the existing assembly.

The arguments are preset attributes for the new item. The item is created using a CDB operation.

get_attribute_mapping()

Retrieves the configured mapping from source attributes to CDB attributes. The meaning of "source attributes" is implementation-specific. This configuration part is only useful for some BOMReader implementations.

The result is a list of 3-tuples: (cdb_name, source_name:string, is_property)

- cdb_name: String; the name of an attribute of class AssemblyComponent
- source_name: String; the name of a source attribute. The details depend on the specific implementation.
- is_property: int; 1 if the source is considered a property; 0 if it is an attribute

sync_BOMEntries(*existing*, *new*)

Synchronizes an existing cs.vp.bomcreator.bom.BOMEntry with a new one. "new" will be overwritten with the result.

This method is typically not called directly by client code.

2.8.3 cs.vp.bomcreator.bom

Classes representing BOMs and their entries.

BOM

class cs.vp.bomcreator.bom.**BOM**

A dictionary of BOMEntry that keeps information about the order of entries.

The order information is needed because it represents the sort value of occurrences which in turn is needed to give meaningful position values to new items and to display items in the right order.

add_entry(*new_entry*)

Adds an entry to this BOM. If an equivalent entry already exists, updates menge and the occurrences. In this case, the existing entry is returned.

Parameters

- new_entry: cs.vp.bomcreator.bom.BOMEntry

entries()
Entries in the order they were added.

entries_not_in(*other*)
Returns the list of BOMEntrys that are in self but not in other. The result is ordered as the entries were added to self.

from_json(*js*)
Deserialize into an existing instance.

index_less_equal(*other*)
compares two boms but doesn't consider the index of the entries

index_less_keys()

intersecting_entries(*other*)
Returns the list of BOMEntrys that are both in self and other. The result is unordered.

Parameters

- other: *cs.vp.bomcreator.bom.BOM* (page 13)

GeneratedBOM

class cs.vp.bomcreator.bom.**GeneratedBOM**(*bom_context*, *factory*, *headers*, *new_positions_step*)
Bases: *cs.vp.bomcreator.bom.BOM* (page 13)

A bill of materials that is generated from a business object (typically a CAD document). It is associated with an assembly (given by bom_context).

After *synchronize()* (page 15), it represents a merged bill of materials, containing information about entries to be created/deleted/changed.

It also contains information to display the contents and status of all BOM positions.

BOMs are normally created using method create_BOM() of class *cs.vp.bomcreator.factory.Factory* (page 12).

Its methods are normally called by the server code, not directly by customization code.

Parameters

- bom_context: *cs.vp.bomcreator.bomreader.BOMContext* (page 9)
- factory: *cs.vp.bomcreator.factory.Factory* (page 12)
- **headers: list of strings that serve as column headers of** the HTML table
- new_position_step: int

create_and_add_entry(***attrs*)
Creates a BOM entry for this BOM, taking into account the current configuration. If an entry already exists for the given key attributes, the existing entry is returned.

Parameters

- **attrs: dictionary containing values of bom_item attributes;** must contain at least teilenummer

get_assembly()
Returns the cs.vp.items.Item that this BOM belongs to.

get_user_attribute(*key*, *default=None*)
Retrieve a value previously set with *set_user_attribute* (page 15).

set_post_write_bom(*callback*)
Registers a callback that is called when a BOM was written. See *cs.vp.bomcreator.bomreader.BOMReader* (page 9) for details.

set_post_write_component(*callback*)
Registers a callback that is called whenever a BOM component was written. See *cs.vp.bomcreator.bomreader.BOMReader* (page 9) for details.

set_pre_show_component(*callback*)
Registers a callback that is called for every component immediatley before the preview is generated. See *cs.vp.bomcreator.bomreader.BOMReader* (page 9) for details.

set_user_attribute(*key*, *val*)
Stores arbitrary key/value pairs in this BOM. Both keys and values should be simple, JSON-serializable values. This can be useful for custom BOMReader implementations.

set_visual_indent(*indent*)
Change the visual indent of the BOM in the preview. This can be useful if this BOM is displayed as part of a a hierarchy of BOMs.

Parameters

- `indent`: int

synchronize(*json_data=True*)
Creates a merged version of this BOM and the BOM of the associated assembly already existing in the database. The result is represented in this object, ready to be displayed and (after user confirmation) to be stored in the database.

returns bom_changed, bom_content_changed bom_changed is true if any record is changed included index changes

on the same position

bom_content_changed is true if a content changed or other componets are added or deleted (without regarding the t_index)

write(*user_hints*)
Writes a generated BOM into the database (for the associated assembly). *synchronize()* (page 15) must have been called before.

Parameters

- `user_hints`: *cs.vp.bomcreator.bom.UserHintList* (page 15)

UserHintList

class `cs.vp.bomcreator.bom.`**UserHintList**
List of error messages and hints that have occurred during a BOM creation. Automatically logs all added messages. Implementations that derive from `SimpleBOMReader` don't have to use this class directly.

Implements a list-like interface.

append(*msg*)
Adds a string and logs it as a message.

append_error(*msg*)
Adds a string and logs it as an error.

append_success(*msg*)
Adds a success message and logs it as a message.

extend(*other*)
Adds all messages of another UserHintList but does not log the messages again.

SharedPosition

class cs.vp.bomcreator.bom.**SharedPosition**

This class is useful if multiple BOM entries should receive the same generated position number.

It can be used as the value of the 'position' attribute of an entry. If an instance of SharedPosition is used for multiple entries, they will receive the same position number when the BOM is synchronized.

CHAPTER 3

Creating a BOM automatically

3.1 Introduction

The cs.vp.bomcreator module is designed for customizable, interactive creation of bills of materials (BOM) and includes a preview function. However, in some situations customers need BOMs to be created and updated automatically without user intervention whenever new data enters the system. It is possible to implement such behaviour with some manual customization.

3.2 Requirements

To implement automatic BOM creation we first need a suitable integration point where all needed data is available. The exact user exit or signal to use depends on the type of input data.

For data imported with the Workspace Manager, the right choice is the signal `wsmcommit`, emitted by the Workspace Manager communication component. This signal is only available in **Workspace Manager 3.1** and newer.

3.3 Basic implementation

The cs.vp.bomcreator.bomreader Python module contains a function `create_and_save_bom` which is suitable for this use case. It considers the current configuration and creates a BOM for the article associated with the given CAD document. Note: Depending on the configuration, it may also create BOMs for referenced documents and their articles.

The function is called like this, if you are using **Workspace Manager 3.1.1-4**:

```
from cdb import sig
from cs.documents import Document
from cs.vp.bomcreator.bom import UserHintList
from cs.vp.bomcreator.bomreader import create_and_save_bom

@sig.connect(Document, "wsmcommit", "post")
def create_bom_after_commit(doc):
    try:
        hasArticle = bool(doc.Item)
        isDrawing = doc.z_categ2 == "CAD-Zeichnung"  # customize!
        if hasArticle and not isDrawing:
            errors_and_hints = UserHintList()
            success = create_and_save_bom(doc, errors_and_hints)
    except:
        pass  # better: log exception
```

In this example, we connect the code to the Workspace Manager-specific signal `wsm_commit` to make sure that the BOM is created and updated whenever the data of a document is committed.

`create_and_save_bom` logs most errors and warnings automatically. However, errors will not be visible to the user!

> **Warning:** BOMs must not be created from drawing documents. The mechanism how to exclude drawings is dependend on the concrete installation and must be adjusted (comparing "z_categ2" to "CAD-Zeichnung" will not work in the general case).

If you are using **Workspace Manager 3.1.5** or newer, the signature is slightly changed:

```
...
@sig.connect(Document, "wsmcommit", "post")
def create_bom_after_commit(doc, ctx):
...
```

If you are using **Workspace Manager 3.3** or newer, you can improve this example by returning any errors and warnings to the Workspace Manager where they will be displayed after the commit operation.

The Workspace Manager expects error messages in a format that is different from the format that is used by the BOM creation. So some glue code is necessary:

```
import traceback
from cdb import sig
from cs.documents import Document
from cs.vp.bomcreator.bom import UserHintList
from cs.vp.bomcreator.bomreader import create_and_save_bom
from cs.wsm.result import Result, Error, Info


@sig.connect(Document, "wsmcommit", "post")
def create_bom_after_commit(doc, ctx):
    res = Result()
    try:

        hasArticle = bool(doc.Item)
        isDrawing = doc.z_categ2 == "CAD-Zeichnung"  # customize!
        if hasArticle and not isDrawing:
            errors_and_hints = UserHintList()
            create_and_save_bom(doc, errors_and_hints)
            res += convert_to_result(errors_and_hints)

    except Exception:
        res += Error("Unexpected exception while creating BOM for document %s-%s:\n
↪%s"
                     % (doc.z_nummer, doc.z_index, traceback.format_exc()))
    return res


def convert_to_result(userHintList):
    res = Result()
    for msg, alertType in userHintList:
        msg = u"%s" % msg  # convert to unicode if bytestring
        if alertType == 'alert-error':
            res += Error(msg)
        elif alertType == 'alert-warning':
            res += Info(msg)
    return res
```

V

A

B

C

E

F

G

I

L

P

S

U

W