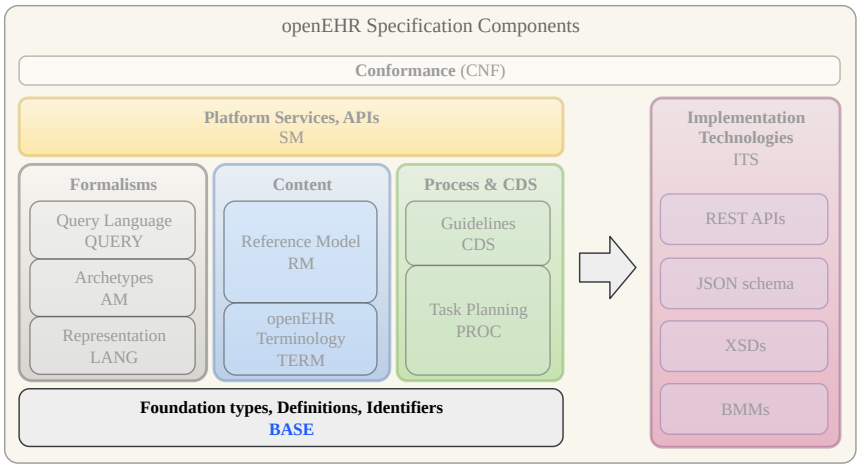
Table of Contents


openEHR

BMM Persistence Model and Syntax

Issuer: openEHR Specification Program (https://www.openehr.org/programs/specification)	
Release: BASE Release-1.0.4	**Status**: STABLE
Revision: [latest_issue]	**Date**: [latest_issue_date]
Keywords: reflection, meta-model, UML	



Amendment Record

Issue	Details	Raiser	Completed
2.3.2	Separate from main BMM specification; Support generic types as class ancestors; add new type `P_BMM_BASE_TYPE` as parent of `P_BMM_SIMPLE_TYPE`, `P_BMM_GENERIC_TYPE` and `P_BMM_OPEN_TYPE`; Remove archetype-related schema meta-data.	T Beale	27 Apr 2018
2.2.2	Improve and update introductory text in the Overview section.	E Sundvall, T Beale	03 Nov 2017
2.2.1	Rename `P_BMM_SIMPLE_TYPE_OPEN` to `P_BMM_OPEN_TYPE`; Rename `P_BMM_GENERIC_PARAMETER.create_bmm_generic_parameter_definition` to `create_bmm_generic_parameter`; Remove inheritance from `P_BMM_PACKAGE_CONTAINER` to `P_BMM_MODEL_ELEMENT`; Make `P_BMM_MODEL_ELEMENT` abstract; Correct `P_BMM_ENUMERATION.items_names` cardinality to multiple; Constrain `P_BMM_GENERIC_PARAMETER.name` to one character and upper case.	C Nanjo, T Beale	02 Mar 2017
2.2.0	Remove class `P_BMM_CLASSIFIER`. Add missed inheritance from `P_BMM_PROPERTY` to `P_BMM_MODEL_ELEMENT`.	T Beale	20 Jun 2016
	Add missing inheritance from `P_BMM_SCHEMA` to `BMM_SCHEMA_CORE`. Correct naming of `bmm_xxx_definition`, `create_bmm_xxx_definition` in `P_BMM_XXX` classes to `bmm_xxx`, `create_bmm_xxx` etc.	T Beale	18 Apr 2016
2.1.0	Initial writing based on ADL Workbench implementation.	T Beale	08 Feb 2016

Acknowledgements

Primary Author

- Thomas Beale, Ars Semantica; openEHR Foundation Management Board.

Contributors

This specification has benefited from formal and informal input from the openEHR and wider health informatics community. The openEHR Foundation would like to recognise the following people for their contributions.

- Patrick Langford, NeuronSong LLC, Utah, USA
- Claude Nanjo MA African Studies., M Public Health, Cognitive Medical Systems Inc., California, USA
- Erik Sundvall PhD, Linkoping University, Sweden

Trademarks

1. Preface

1.1. Purpose

This document describes a persistence model for the Basic Meta-Model (BMM) known as `P_BMM`, that may be used as a basis for serialisation of BMM models. It may be considered as an approximate replacement for the UML XMI for data-only models. It is human-readable and writable, and supports generic types (open and closed), container types, and multiple inheritance.

1.2. Status

This specification is in the STABLE state. The development version of this document can be found at {openehr_base_bmm_persistence}[www.openehr.org/releases/BASE/latest/bmm_persistence.html].

Known omissions or questions are indicated in the text with a 'to be determined' paragraph, as follows:

TBD: (example To Be Determined paragraph)

1.3. Feedback

Feedback may be provided on the openEHR languages specifications forum (https://discourse.openehr.org/c/specifications/bmm-el).

Issues may be raised on the specifications Problem Report tracker (https://specifications.openehr.org/components/BASE/open_issues).

To see changes made due to previously reported issues, see the BASE component Change Request tracker (https://specifications.openehr.org/components/BASE/history).

1.4. Conformance

Conformance of a data or software artifact to an openEHR specification is determined by a formal test of that artifact against the relevant openEHR Implementation Technology Specification(s) (ITSs) (https://specifications.openehr.org/releases/ITS/latest/index), such as an IDL interface or an XML-schema. Since ITSs are formal derivations from underlying models, ITS conformance indicates model conformance.

1.5. Tooling

The openEHR ADL Workbench (AWB) (https://www.openehr.org/downloads/ADLworkbench) fully implements this specification, and provide a convenient way of illustrating BMM semantics. The screenshots used in this specification are all from the ADL Workbench. The tool is written in the Eiffel language, and is available as open source on Github (https://github.com/openEHR/adl-tools). The BMM libraries can be found in the EOMF Github repository (https://github.com/wolandscat/EOMF/tree/master/library/bmm).

A modelling tool known as Archie implements BMM in Java, and can be found in the openEHR Github area (https://github.com/openEHR/archie).

2. Overview

2.1. Conceptual Approach

This specification defines a model that may be used as the basis for a serial format for the Basic Meta-Model (BMM). The formalism described here is an adaptation of pure object serialisation approaches used in mainstream programming languages. Instead of directly materialising a BMM instance graph from serial form, the initial materialised form is a graph of `P_BMM_XXX` classes, with a subsequent in-memory model-to-model transform step required to produce a materialised BMM model.

The `P_BMM_*` classes perform two functions. Firstly, they are a modified and simplified version of the `BMM_*` classes that enable for example symbolic referencing via class names, syntactical type names to be used etc, rather than the full explosion of fine-grained objects that would result from a direct serialisation of `BMM_*` classes. This enables an object model represented internally (to a tool, say) in `BMM` form, converted to `P_BMM` form, and then serialised to a `.bmm` file, to be easily readable and editable by human users.

The second is that `.bmm` files function as schemas that support schema inclusion and therefore re-use, in a similar manner to the XML schema languages. Thus, a single logical BMM model can be expressed as a *number* of `.bmm` schema files which are actually `P_BMM_*` object serialisations of parts of the BMM model. A schema reading component has to resolve the schema inclusions and ultimately `BMM_*` object instantiations to obtain the in-memory form of the model.

We thus talk of the `P_BMM_*` classes as a 'model of a BMM *schema*' and the `BMM_*` classes as a 'model of a BMM *model*', where the latter is understood as the fully computable in-memory object structure with all name references resolved to object references.

The `P_BMM` format is not the only serial format possible for BMM, and alternatives, e.g. a more syntactic style reminiscent of OMG IDL, are possible.

The normal use of P_BMM schemas is as follows:

- create one or more `.bmm` schema files, using the `P_BMM_` form of the model. This is easy to understand by using the example schema and/or copying other examples;
- locate these files in a suitable place for use with a tool such as the openEHR ADL Workbench (https://www.openehr.org/downloads/ADLworkbench) and LinkEHR (http://linkehr.com).

2.2. Concrete Format

BMM models are normally expressed as schema text files that support inclusion and re-use. The default file format has historically been openEHR ODIN syntax (https://specifications.openehr.org/releases/LANG/latest/odin.html), and BMM tools to date support this format. However any common format that can express typed object models may be used, including JSON (with type markers), YAML, and XML. The examples shown in this specification are primarily in ODIN, but a tool implementing BMM may choose to serialise in and out of another preferred format.

3. Persistence Package

3.1. Overview

The `persistence` package, shown below, defines a simplified form of the main BMM model suitable for persisting and human authoring. The openEHR BMM schemas (https://github.com/openEHR/reference-models/tree/master/models/openEHR/Release-1.0.2/BMM) are authored in the `P_BMM` form of the BMM, using the openEHR ODIN syntax (https://specifications.openehr.org/releases/LANG/latest/odin.html).

Figure 1. base.bmm.persistence Package

The general approach taken in this model is that attributes name *`bmm_xxx`* and of type `BMM_XXX` are derived from the persisted attributes of the `P_BMM_XXX` classes of this model. In other words, they are in-memory only references to reconstructed instances of `BMM_XXX` types.

The general approach taken in this model is that attributes name *`bmm_xxx`* and of type `BMM_XXX` are derived from the persisted attributes of the `P_BMM_XXX` classes of this model. In other words, they are in-memory only references to reconstructed instances of `BMM_XXX` types.

3.2. Class Definitions

3.2.1. P_BMM_MODEL_ELEMENT Class

Class	*P_BMM_MODEL_ELEMENT (abstract)*	
Description	Persistent form of `BMM_MODEL_ELEMENT`.	
Attributes	**Signature**	**Meaning**
0..1	**documentation**: `String`	Optional documentation of this element.

3.2.2. P_BMM_PACKAGE_CONTAINER Class

Class	P_BMM_PACKAGE_CONTAINER	
Description	Persisted form of a model component that contains packages.	
Attributes	**Signature**	**Meaning**
1..1	**packages**: `Hash<String, P_BMM_PACKAGE>`	Package structure as a hierarchy of packages each potentially containing names of classes in that package in the original model.

3.2.3. P_BMM_SCHEMA Class

Class	P_BMM_SCHEMA

Description	Persisted form of `BMM_SCHEMA`.	
Inherit	`P_BMM_PACKAGE_CONTAINER`, `BMM_SCHEMA`	
Attributes	**Signature**	**Meaning**
0..1 (redefined)	**primitive_types**: `List<P_BMM_CLASS>`	Primitive type definitions. Persisted attribute.
0..1 (redefined)	**class_definitions**: `List<P_BMM_CLASS>`	Class definitions. Persisted attribute.
0..1 (redefined)	**includes**: `Hash<String,BMM_INCLUDE_SPEC>`	Inclusion list, in the form of a hash of individual include specifications, each of which at least specifies the id of another schema, and may specify a namespace via which types from the included schemas are known in this schema. Persisted attribute.
Functions	**Signature**	**Meaning**
0..1 (effected)	**validate_created** *Pre_state*: `state = State_created` *Post_state*: `passed implies state = State_validated_created`	Implementation of `validate_created()`
0..1 (effected)	**load_finalise** *Pre_state*: `state = State_validated_created` *Post_state*: `state = State_includes_processed or state = State_includes_pending`	Implementation of `load_finalise()`
0..1 (effected)	**merge** (other: `P_BMM_SCHEMA[1]`) *Pre_state*: `state = State_includes_pending` *Pre_other_valid*: `includes_to_process.has (included_schema.schema_id)`	Implementation of `merge()`
0..1 (effected)	**validate**	Implementation of `validate()`

0..1 (effected)	**create_bmm_model** *Pre_state*: `state = P_BMM_PACKAGE_STATE.State_includes_processed`	Implementation of *`create_bmm_model()`*
1..1	**canonical_packages** (): `P_BMM_PACKAGE`	Package structure in which all top-level qualified package names like `xx.yy.zz` have been expanded out to a hierarchy of `BMM_PACKAGE` objects.

3.2.4. P_BMM_PACKAGE Class

Class	**P_BMM_PACKAGE**	
Description	Persisted form of a package as a tree structure whose nodes can contain more packages and/or classes.	
Inherit	`P_BMM_PACKAGE_CONTAINER`, `P_BMM_MODEL_ELEMENT`	
Attributes	**Signature**	**Meaning**
1..1	**name**: `String`	Name of the package from schema; this name may be qualified if it is a top-level package within the schema, or unqualified. Persistent attribute.
0..1	**classes**: `List<String>`	List of classes in this package. Persistent attribute.
0..1	**bmm_package_definition**: `BMM_PACKAGE`	`BMM_PACKAGE` created by *`create_bmm_package_definition`*.
Functions	**Signature**	**Meaning**
0..1	**merge** (other: `P_BMM_PACKAGE[1]`)	Merge packages and classes from other (from an included `P_BMM_SCHEMA`) into this package.
0..1	**create_bmm_package_definition**	Generate a `BMM_PACKAGE_DEFINITION` object and write it to *`bmm_package_definition`*.

3.2.5. P_BMM_TYPE Class

Class	***P_BMM_TYPE (abstract)***	
Description	Persistent form of `BMM_TYPE`.	
Attributes	**Signature**	**Meaning**
0..1	**bmm_type**: `BMM_TYPE`	Result of *`create_bmm_type()`* call.

Functions	Signature	Meaning
0..1 (abstract)	**create_bmm_type** (a_schema: `BMM_MODEL[1]` , a_class_def: `BMM_CLASS[1]`)	Create appropriate `BMM_XXX` object; effected in descendants.
1..1 (abstract)	**as_type_string** (): `String`	Formal name of the type for display.

3.2.6. P_BMM_CLASS Class

Class	P_BMM_CLASS	
Description	Definition of persistent form of `BMM_CLASS` for serialisation to ODIN, JSON, XML etc.	
Inherit	`P_BMM_MODEL_ELEMENT`	
Attributes	**Signature**	**Meaning**
1..1	**name**: `String`	Name of the class. Persisted attribute.
0..1	**ancestors**: `List<String>`	List of immediate inheritance parents. Persisted attribute.
0..1	**properties**: `Hash<String,P_BMM_PROPERTY>`	List of attributes defined in this class. Persistent attribute.
0..1	**is_abstract**: `Boolean`	True if this is an abstract type. Persisted attribute.
0..1	**is_override**: `Boolean`	True if this class definition overrides one found in an included schema.
0..1	**generic_parameter_defs**: `Hash<String,P_BMM_GENERIC_PARAMETER>`	List of generic parameter definitions. Persisted attribute.
1..1	**source_schema_id**: `String`	Reference to original source schema defining this class. Set during `BMM_SCHEMA` materialise. Useful for GUI tools to enable user to edit the schema file containing a given class (i.e. taking into account that a class may be in any of the schemas in a schema inclusion hierarchy).
0..1	**bmm_class**: `BMM_CLASS`	`BMM_CLASS` object built by *`create_bmm_class_definition`* and *`populate_bmm_class_definition`* .

1..1	**uid**: `Integer`	Unique id generated for later comparison during merging, in order to detect if two classes are the same. Assigned in post-load processing.
Functions	**Signature**	**Meaning**
1..1	**is_generic** (): `Boolean` *Post*: `Result := generic_parameter_defs /= Void`	True if this class is a generic class.
0..1	**create_bmm_class**	Create *`bmm_class_definition`*.
0..1	**populate_bmm_class** (a_bmm_schema: `BMM_MODEL[1]`)	Add remaining model elements from Current to *`bmm_class_definition`*.

3.2.7. P_BMM_GENERIC_PARAMETER Class

Class	**P_BMM_GENERIC_PARAMETER**	
Description	Persistent form of `BMM_GENERIC_PARAMETER`.	
Inherit	`P_BMM_MODEL_ELEMENT`	
Attributes	**Signature**	**Meaning**
1..1	**name**: `String`	Name of the parameter, e.g. 'T' etc. Persisted attribute. Name is limited to 1 character, upper case.
0..1	**conforms_to_type**: `String`	Optional conformance constraint - the name of a type to which a concrete substitution of this generic parameter must conform. Persisted attribute.
0..1	**bmm_generic_parameter**: `BMM_PARAMETER_TYPE`	`BMM_GENERIC_PARAMETER` created by *`create_bmm_generic_parameter`*.
Functions	**Signature**	**Meaning**
0..1	**create_bmm_generic_parameter** (a_bmm_schema: `BMM_MODEL[1]`)	Create *`bmm_generic_parameter`*.
Invariants	*Inv_generic_name*: `name.count = 1 and name.is_upper`	

3.2.8. P_BMM_PROPERTY Class

Class	***P_BMM_PROPERTY (abstract)***

Description	Persistent form of `BMM_PROPERTY`.	
Inherit	`P_BMM_MODEL_ELEMENT`	
Attributes	**Signature**	**Meaning**
1..1	**name**: `String`	Name of this property within its class. Persisted attribute.
0..1	**is_mandatory**: `Boolean`	True if this property is mandatory in its class. Persisted attribute.
0..1	**is_computed**: `Boolean`	True if this property is computed rather than stored in objects of this class. Persisted Attribute.
0..1	**is_im_infrastructure**: `Boolean`	True if this property is info model 'infrastructure' rather than 'data'. Persisted attribute.
0..1	**is_im_runtime**: `Boolean`	True if this property is info model 'runtime' settable property. Persisted attribute.
0..1	**type_def**: `P_BMM_TYPE`	Type definition of this property, if not a simple String type reference. Persisted attribute.
0..1	**bmm_property**: `BMM_PROPERTY<BMM_TYPE>`	BMM_PROPERTY created by create_bmm_property_definition.
Functions	**Signature**	**Meaning**
0..1	**create_bmm_property** (a_bmm_schema: `BMM_MODEL[1]`, a_class_def: `BMM_CLASS[1]`)	Create *`bmm_property_definition`* from `P_BMM_XX` parts.

3.2.9. P_BMM_BASE_TYPE Class

Class	***P_BMM_BASE_TYPE (abstract)***
Description	Persistent form of `BMM_PROPER_TYPE`.
Inherit	`P_BMM_TYPE`

3.2.10. P_BMM_SIMPLE_TYPE Class

Class	**P_BMM_SIMPLE_TYPE**
Description	Persistent form of `BMM_SIMPLE_TYPE`.
Inherit	`P_BMM_BASE_TYPE`

Attributes	Signature	Meaning
1..1	**type**: `String`	Name of type - must be a simple class name.
0..1 (redefined)	**bmm_type**: `BMM_SIMPLE_TYPE`	Result of *`create_bmm_type()`* call.

3.2.11. P_BMM_OPEN_TYPE Class

Class	P_BMM_OPEN_TYPE	
Description	Persistent form of `BMM_PARAMETER_TYPE`.	
Inherit	`P_BMM_BASE_TYPE`	
Attributes	**Signature**	**Meaning**
1..1	**type**: `String`	Simple type parameter as a single letter like 'T', 'G' etc.
0..1 (redefined)	**bmm_type**: `@@`	Result of *`create_bmm_type()`* call.

3.2.12. P_BMM_GENERIC_TYPE Class

Class	P_BMM_GENERIC_TYPE	
Description	Persistent form of `BMM_GENERIC_TYPE`.	
Inherit	`P_BMM_BASE_TYPE`	
Attributes	**Signature**	**Meaning**
1..1	**root_type**: `String`	Root type of this generic type, e.g. `Interval` in `Interval<Integer>`.
1..1	**generic_parameter_defs**: `List<P_BMM_TYPE>`	Generic parameters of the root_type in this type specifier if non-simple types. The order must match the order of the owning class's formal generic parameter declarations. Persistent attribute.
0..1	**generic_parameters**: `List<String>`	Generic parameters of the *`root_type`* in this type specifier, if simple types. The order must match the order of the owning class's formal generic parameter declarations. Persistent attribute.
0..1 (redefined)	**bmm_type**: `BMM_GENERIC_TYPE`	Result of *`create_bmm_type()`* call.

Functions	Signature	Meaning
0..1	**generic_parameter_refs** (): `List<P_BMM_TYPE>`	Generic parameters of the root_type in this type specifier. The order must match the order of the owning class's formal generic parameter declarations

3.2.13. P_BMM_CONTAINER_TYPE Class

Class	P_BMM_CONTAINER_TYPE	
Description	Persistent form of `BMM_CONTAINER_TYPE`.	
Inherit	`P_BMM_TYPE`	
Attributes	Signature	Meaning
1..1	**container_type**: `String`	The type of the container. This converts to the *`root_type`* in `BMM_GENERIC_TYPE`. Persisted attribute.
0..1	**type_def**: `P_BMM_BASE_TYPE`	Type definition of *`type`*, if not a simple String type reference. Persisted attribute.
0..1	**type**: `String`	The target type; this converts to the first parameter in *`generic_parameters`* in `BMM_GENERIC_TYPE`. Persisted attribute.
0..1 (redefined)	**bmm_type**: `BMM_CONTAINER_TYPE`	Result of *`create_bmm_type()`* call.
Functions	Signature	Meaning
1..1	**type_ref** (): `P_BMM_BASE_TYPE`	The target type; this converts to the first parameter in *`generic_parameters`* in `BMM_GENERIC_TYPE`. Persisted attribute.

3.2.14. P_BMM_SINGLE_PROPERTY Class

Class	P_BMM_SINGLE_PROPERTY	
Description	Persistent form of `BMM_SINGLE_PROPERTY`.	
Inherit	`P_BMM_PROPERTY`	
Attributes	Signature	Meaning

0..1	**type**: `String`	If the type is a simple type, then this attribute will hold the type name. If the type is a container or generic, then type_ref will hold the type definition. The resulting type is generated in type_def.
0..1	**type_ref**: `P_BMM_SIMPLE_TYPE`	Type definition of this property computed from *`type`* for later use in *`bmm_property`*.
0..1 (redefined)	**bmm_property**: `BMM_PROPERTY<BMM_SIMPLE_TYPE>`	`BMM_PROPERTY` created by *`create_bmm_property_definition`*.
Functions	**Signature**	**Meaning**
1..1	**type_def** (): `P_BMM_SIMPLE_TYPE`	Generate *`type_ref`* from *`type`* and save.

3.2.15. P_BMM_SINGLE_PROPERTY_OPEN Class

Class	**P_BMM_SINGLE_PROPERTY_OPEN**	
Description	Persistent form of a `BMM_SINGLE_PROPERTY_OPEN`.	
Inherit	`P_BMM_PROPERTY`	
Attributes	**Signature**	**Meaning**
0..1	**type_ref**: `P_BMM_OPEN_TYPE`	Type definition of this property computed from *`type`* for later use in *`bmm_property`*.
0..1	**type**: `String`	Type definition of this property, if a simple String type reference. Really we should use *`type_def`* to be regular in the schema, but that makes the schema more wordy and less clear. So we use this persisted String value, and compute the *`type_def`* on the fly. Persisted attribute.
0..1 (redefined)	**bmm_property**: `BMM_PROPERTY<BMM_OPEN_TYPE>`	`BMM_PROPERTY` created by *`create_bmm_property_definition`*.
Functions	**Signature**	**Meaning**
1..1	**type_def** (): `P_BMM_OPEN_TYPE`	Generate *`type_ref`* from *`type`* and save.

3.2.16. P_BMM_GENERIC_PROPERTY Class

Class	**P_BMM_GENERIC_PROPERTY**
Description	Persistent form of `BMM_GENERIC_PROPERTY`.
Inherit	`P_BMM_PROPERTY`

Attributes	Signature	Meaning
0..1 (redefined)	**type_def**: `P_BMM_GENERIC_TYPE`	Type definition of this property, if not a simple String type reference. Persistent attribute.
0..1 (redefined)	**bmm_property**: `BMM_PROPERTY<BMM_GENERIC_TYPE>`	`BMM_PROPERTY` created by *`create_bmm_property_definition`*.

3.2.17. P_BMM_CONTAINER_PROPERTY Class

Class	P_BMM_CONTAINER_PROPERTY	
Description	Persistent form of `BMM_CONTAINER_PROPERTY`.	
Inherit	`P_BMM_PROPERTY`	
Attributes	**Signature**	**Meaning**
0..1	**cardinality**: `Interval<Integer>`	Cardinality of this property in its class. Persistent attribute.
0..1 (redefined)	**type_def**: `P_BMM_CONTAINER_TYPE`	Type definition of this property, if not a simple String type reference. Persistent attribute.
0..1 (redefined)	**bmm_property**: `BMM_CONTAINER_PROPERTY`	`BMM_PROPERTY` created by *`create_bmm_property`*.
Functions	**Signature**	**Meaning**
0..1 (redefined)	**create_bmm_property** (a_bmm_schema: `BMM_MODEL[1]`, a_class_def: `BMM_CLASS[1]`)	Create *`bmm_property_definition`*.

3.2.18. P_BMM_ENUMERATION Class

Class	P_BMM_ENUMERATION<T>	
Description	Persistent form of `BMM_ENUMERATION` attributes.	
Inherit	`P_BMM_CLASS`	
Attributes	**Signature**	**Meaning**
0..1	**item_names**: `List<String>`	
0..1	**item_values**: `List<Any>`	

0..1 (redefined)	**bmm_class**: `BMM_ENUMERATION`	`BMM_CLASS` object build by *`create_bmm_class_definition`* and *`populate_bmm_class_definition`*.

3.2.19. P_BMM_ENUMERATION_STRING Class

Class	**P_BMM_ENUMERATION_STRING**	
Description	Persistent form of `BMM_ENUMERATION_STRING`.	
Inherit	`P_BMM_ENUMERATION`	
Attributes	**Signature**	**Meaning**
0..1 (redefined)	**bmm_class**: `BMM_ENUMERATION_STRING`	`BMM_CLASS` object build by *`create_bmm_class_definition`* and *`populate_bmm_class_definition`*.

3.2.20. P_BMM_ENUMERATION_INTEGER Class

Class	**P_BMM_ENUMERATION_INTEGER**	
Description	Persistent form of an instance of `BMM_ENUMERATION_INTEGER`.	
Inherit	`P_BMM_ENUMERATION`	
Attributes	**Signature**	**Meaning**
0..1 (redefined)	**bmm_class**: `BMM_ENUMERATION_INTEGER`	`BMM_CLASS` object build by *`create_bmm_class_definition`* and *`populate_bmm_class_definition`*.

4. BMM Persistence Syntax

4.1. Overview

A BMM schema is normally written in the ODIN syntax (https://specifications.openehr.org/releases/LANG/latest/odin.html), although any other serialisation supporting typed object models may be used, including JSON (with type markers), YAML, XML etc. The ODIN form is described here. The structures are direct ODIN serialisations of the `P_BMM_XXX` classes in the `persistence` package.

4.2. Header Items

The following shows the header items of a BMM schema, which correspond to the 'persistent' attributes of the class `P_BMM_SCHEMA`.

```ODIN
-----------------------------------------------------
-- P_BMM version on which these schemas are based.
-----------------------------------------------------
bmm_version = <"2.3">

-----------------------------------------------------
-- schema identification
-- (schema_id computed as <rm_publisher>_<schema_name>_<rm_release>)
-----------------------------------------------------
rm_publisher = <"openehr">
schema_name = <"adltest">
rm_release = <"1.0.2">
model_name = <"TEST_PKG">

-----------------------------------------------------
-- schema documentation
-----------------------------------------------------
schema_revision = <"1.0.36">
schema_lifecycle_state = <"stable">
schema_description = <"openEHR schema to support test archetypes">
```

4.3. Inclusions

```ODIN
includes = <
    ["1"] = <
        id = <"openehr_basic_types_1.0.2">
    >
>
```

4.4. Package Definition

The packages definition should be self-explanatory: just name the classes and packages in a recursive fashion.

NOTE only top-level package ids can be paths (i.e. contain the '.' character)

NOTE only classes defined in the same schema can be referenced in the package section in that schema.

NOTE make sure that the ODIN 'keys' are the same as the 'name' attributes in each block.

```ODIN
packages = <
    ["org.openehr.test_pkg"] = <
        name = <"org.openehr.test_pkg">
        classes = <"WHOLE", "SOME_TYPE", "BOOK", "CHAPTER", "ENTRY", "CAR", "CAR_BODY">
    >
>
```

4.5. Class Definitions

4.5.1. Classes for Primitive Types

Definitions for primitive types in a BMM schema are just normal class definitions within a `primitive_types` block. Types that are included here are usually types corresonding to primitives in target programming languages, XML schema or other downstream technologies. These types can be be detected as primitive types by tools, but otherwise are processed in the same way as types defined in the main `class_definitions` group.

> **NOTE** unlike UML, all container types such as `List<T>`, `Hash<V,K>` etc are exlicit in a BMM schema, and consequently, such types are normally defined (including with generic parameters) in a BMM schema.

```ODIN
primitive_types = <
    ["Any"] = <
        name = <"Any">
        is_abstract = <True>
    >
    ["Ordered"] = <
        name = <"Ordered">
        is_abstract = <True>
        ancestors = <"Any">
    >
>
```

4.5.2. Non-primitive Classes

The main group of class definitions in a schema occurs within the *`class_definitions`* block. Each definition is a keyed ODIN object block correspnding to a serialised `P_BMM_CLASS` object, where the key is the class name. Since `name` is a BMM meta-model attribute, the class definition always contains its ODIN key.

The possible class-level meta-properties:

- *`name`* - class name - any capitalisation can be used, usually one of CamelCase or so-called UPPER_SNAKE_CASE;
- *`ancestors`* - states classes from which this class inherits, as an ODIN String list;
- *`is_abstract`* - indicates that the class cannot be instantiated directly;
- *`properties`* - ODIN block containing definitions consisting of `P_BMM_PROPERTY` descendants, keyed by property name.

4.5.3. Simple Classes

Simple classes are those whose type is the same as the class, as opposed to generic classes and enumerated types (see below).

```ODIN
class_definitions = <
    ["ITEM"] = <
        name = <"ITEM">
        ancestors = <"Any">
        is_abstract = <True>
        properties = <
            -- properties here
        >
    >
    -- more classes here
>
```

4.5.3.1. Class properties

Class properties from the original model are expressed using ODIN object blocks keyed by property name. Since there are multiple possible descendants of `P_BMM_PROPERTY`, ODIN type markers must be used to indicate which subtypes is used in each case.

The possible BMM meta-properties of all property types are as follows:

- *`name`* - `String` name of the property in its owning class in the model - use camelCase or snake_case as appropriate;
- *`is_mandatory`* - `Boolean` flag indicating whether the property is mandatory within its class.

The following shows the class `ELEMENT` with two properties *`null_flavour`* and *`value`* of BMM meta-type `P_BMM_SINGLE_PROPERTY`, i.e. corresponding to a single-valued attribute from the original model.

```ODIN
    ["ELEMENT"] = <
        name = <"ELEMENT">
        ancestors = <"ITEM">
        properties = <
            ["null_flavour"] = (P_BMM_SINGLE_PROPERTY) <
                name = <"null_flavour">
                type = <"DV_CODED_TEXT">
                is_mandatory = <True>
            >
            ["value"] = (P_BMM_SINGLE_PROPERTY) <
                name = <"value">
                type = <"DATA_VALUE">
            >
        >
    >
```

4.5.3.2. Container Properties

The following is a `P_BMM_CONTAINER_PROPERTY` definition for the model property `items: List<ITEM>` in the `ELEMENT` class. The type is expressed in the *`type_def`* part which indicates the type of the container, which must be defined elsewhere in the schema, typically in the primitive types. The optional *`cardinality`* meta-property indicates cardinality of the container, and is expressed as a ODIN range. The default is `|0..*|`.

```ODIN
    ["ELEMENT"] = <
        name = <"ELEMENT">
        ancestors = <"ITEM">
        properties = <
            ["items"] = (P_BMM_CONTAINER_PROPERTY) <
                name = <"items">
                type_def = <
                    container_type = <"List">
                    type = <"ITEM">
                >
                cardinality = <|>=1|>
                is_mandatory = <True>
            >
        >
    >
```

4.5.4. Generic Classes

Generic classes are those that have one or more substitutable generic type parameters. Such classes are therefore *type generators*, since actual types are formed by substitution of concrete types (typically simple classes) for the formal type parameters. The following example shows a generic class `Interval` with `generic_parameter_defs` of `T` which is constrained to conform to the type `Ordered`. This structure defineds the type `Interval<T→Ordered>`, with the same meaning as UML and programming languages supporting generic (templated) types.

Generic classes will normally contain one or more properties whose formal type is the generic type parameter, i.e. the `T` in this example, as is the case below where the model properties *`lower`* and *`upper`* are both declared to be of type `T`. This declaration necessitates the use of the BMM meta-type `P_BMM_SINGLE_PROPERTY_OPEN`.

```ODIN
    ["Interval"] = <
        name = <"Interval">
        ancestors = <"Any">
        generic_parameter_defs = <
            ["T"] = <
                name = <"T">
                conforms_to_type = <"Ordered">
            >
        >
        properties = <
            ["lower"] = (P_BMM_SINGLE_PROPERTY_OPEN) <
                name = <"lower">
                type = <"T">
            >
            ["upper"] = (P_BMM_SINGLE_PROPERTY_OPEN) <
                name = <"upper">
                type = <"T">
            >
        >
    >
```

Given the presence of generic classes in a BMM schema, derived generic types can be used as the type of properties in other classes, for which the BMM meta-type `P_BMM_GENERIC_PROPERTY` is used. The folowing example shows first a generic class `DV_INTERVAL` defined in a similar way to `Interval` above, and then a class `SOME_TYPE` containing the property *`qty_interval_attr`* whose model type is `DV_INTERVAL<DV_QUANTITY>`. In the latter type declaration, the `DV_INTERVAL` is the *`root_type`* and `DV_INTERVAL` the *`generic_parameter`*.

```odin
    ["DV_INTERVAL"] = <
        name = <"DV_INTERVAL">
        ancestors = <"Interval", "DATA_VALUE">
        generic_parameter_defs = <
            ["T"] = <
                name = <"T">
                conforms_to_type = <"DV_ORDERED">
            >
        >
    >

    ["SOME_TYPE"] = <
        name = <"SOME_TYPE">
        ancestors = <"Any", ...>
        properties = <
            ["qty_interval_attr"] = (P_BMM_GENERIC_PROPERTY) <
                name = <"qty_interval_attr">
                type_def = <
                    root_type = <"DV_INTERVAL">
                    generic_parameters = <"DV_QUANTITY">
                >
            >
        >
    >
```

Type declarations can also be nested types, for example and container followed by a generic type. In the following the *careProvider* attribute is declared to be of model type `List<ResourceReference<Party>>`. Any level of type nesting is allowed.

```odin
    ["Patient"] = <
        name = <"Patient">
        ancestors = <"Any">
        properties = <
            ["careProvider"] = (P_BMM_CONTAINER_PROPERTY) <
                name = <"careProvider">
                type_def = <
                    container_type = <"List">
                    type_def = (P_BMM_GENERIC_TYPE) <
                        root_type = <"ResourceReference">
                        generic_parameters = <"Party">
                    >
                >
                cardinality = <|>=0|>
            >
        >
    >
```

The following property definition is based on the class `REFERENCE_RANGE`, and in this case, has a generic parameter type that is another generic type: `DV_INTERVAL<DV_QUANTITY>`. To express this, we use `generic_parameter_defs` instead of just *`generic_parameters`* to specify a type structure, rather than just a string type name. Note that `generic_parameter_defs` is a logical list in general, since there can always be more than one generic parameter, i.e. 'T', 'U' etc, even though it is most commonly just one. Accordingly, the usual ODIN keyed hash structure is used with each member being keyed by a generic parameter name, below `["T"]`.

```odin
    ["REFERENCE_RANGE"] = <
        name = <"REFERENCE_RANGE">
        ancestors = <"Any">
        generic_parameter_defs = <
            ["T"] = <
                name = <"T">
                conforms_to_type = <"DV_ORDERED">
            >
        >
        properties = <
            ["range"] = (P_BMM_SINGLE_PROPERTY) <
                name = <"range">
                type = <"DV_INTERVAL">
                is_mandatory = <True>
            >
        >
    >

    ["RANGE_OF_INTERVAL_OF_QUANTITY"] = <
        name = <"RANGE_OF_INTERVAL_OF_QUANTITY">
        ancestors = <"Any", ...>
        properties = <
            ["range"] = (P_BMM_GENERIC_PROPERTY) <
                name = <"range">
                type_def = <
                    root_type = <"REFERENCE_RANGE">
                    generic_parameter_defs = <
                        ["T"] = (P_BMM_GENERIC_TYPE) <
                            root_type = <"DV_INTERVAL">
                            generic_parameters = <"DV_QUANTITY">
                        >
                    >
                >
            >
        >
```

The following example just does the same as the one above, but shows an (unrealistic) but legal case of multiple, mixed, nested generic parameters corresponding to the model property definition `range: REFERENCE_RANGE<DV_INTERVAL<DV_QUANTITY>, Integer, List<DV_QUANTITY>, List<DV_INTERVAL<DV_QUANTITY>>>`. The rules for expressing types is clearly illustrated:

- use 'type' for simple string type refs; use *`type_def`* for structure types;
- within `P_BMM_GENERIC_TYPE`, use *`generic_parameters`* for a list of string types;
- use *`generic_parameter_defs`* for a list of complex type references.

```
ODIN
    ["CRAZY_TYPE"] = <
        name = <"CRAZY_TYPE">
        ancestors = <"Any">
        properties = <
            ["range"] = (P_BMM_GENERIC_PROPERTY) <
                name = <"range">
                type_def = <
                    root_type = <"REFERENCE_RANGE">
                    generic_parameter_defs = <
                        ["T"] = (P_BMM_GENERIC_TYPE) <
                            root_type = <"DV_INTERVAL">
                            generic_parameters = <"DV_QUANTITY">
                        >
                        ["U"] = (P_BMM_SIMPLE_TYPE) <
                            type = <"Integer">
                        >
                        ["V"] = (P_BMM_CONTAINER_TYPE) <
                            type = <"DV_QUANTITY">
                            container_type = <"List">
                        >
                        ["W"] = (P_BMM_CONTAINER_TYPE) <
                            type_def = (P_BMM_GENERIC_TYPE) <
                                root_type = <"DV_INTERVAL">
                                generic_parameters = <"DV_QUANTITY">
                            >
                            container_type = <"List">
                        >
                    >
                >
            >
        >
```

4.5.5. Enumerated Types

In a BMM schema, enumerated types are treated as constrained forms of standard types with open ranges, currently only `Integer` and `String`. They are accordingly represented using class definitions of the meta-types `P_BMM_ENUMERATION_INTEGER` and `P_BMM_ENUMERATION_STRING` respectively. In either case, just names (*`items_names`* meta-property) or both names and values (*`item_values`* meta-property) can be specified.

The following example shows two variants of am enumerated type based on the `Integer` primitive type.

```
ODIN
    ["PROPORTION_KIND"] = (P_BMM_ENUMERATION_INTEGER) <
        name = <"PROPORTION_KIND">
        ancestors = <"Integer">
        item_names = <"pk_ratio", "pk_unitary", "pk_percent", "pk_fraction", "pk_integer_fraction">
    >

    ["PROPORTION_KIND_2"] = (P_BMM_ENUMERATION_INTEGER) <
        name = <"PROPORTION_KIND_2">
        ancestors = <"Integer">
        item_names = <"pk_ratio", "pk_unitary", "pk_percent", "pk_fraction", "pk_integer_fraction">
        item_values = <0, 1001, 1002, 1003>
    >
```

The following example shows two similar examples based on `String`.

```
["MAGNITUDE_STATUS"] = (P_BMM_ENUMERATION_STRING) <
    name = <"MAGNITUDE_STATUS">
    ancestors = <"String", ...>
    item_names = <"le", "ge", "eq", "approx_eq">
    item_values = <"<=", ">=", "=", "~">
>

["NAME_PART"] = (P_BMM_ENUMERATION_STRING) <
    name = <"NAME_PART">
    ancestors = <"String", ...>
    item_names = <"FIRST", "MIDDLE", "LAST">
>
```
ODIN

4.6. Inheritance

In the case of inheritance from simple classes, the `ancestors` list of Strings may be used to simply name the types (which are the same as class names), as seen in the above examples. In the case of generic inheritance, the ancestors are generic types, which may be open, partially closed or fully closed. The following example shows a class model containing generic inheritance in UML (using the closest approximation available in UML), followed by the equivalent P_BMM schema. In the latter, a structured `ancestor_defs` section is used instead of the `ancestors` String list, in the same way as for the `P_BMM_GENERIC_PROPERTY` examples above.

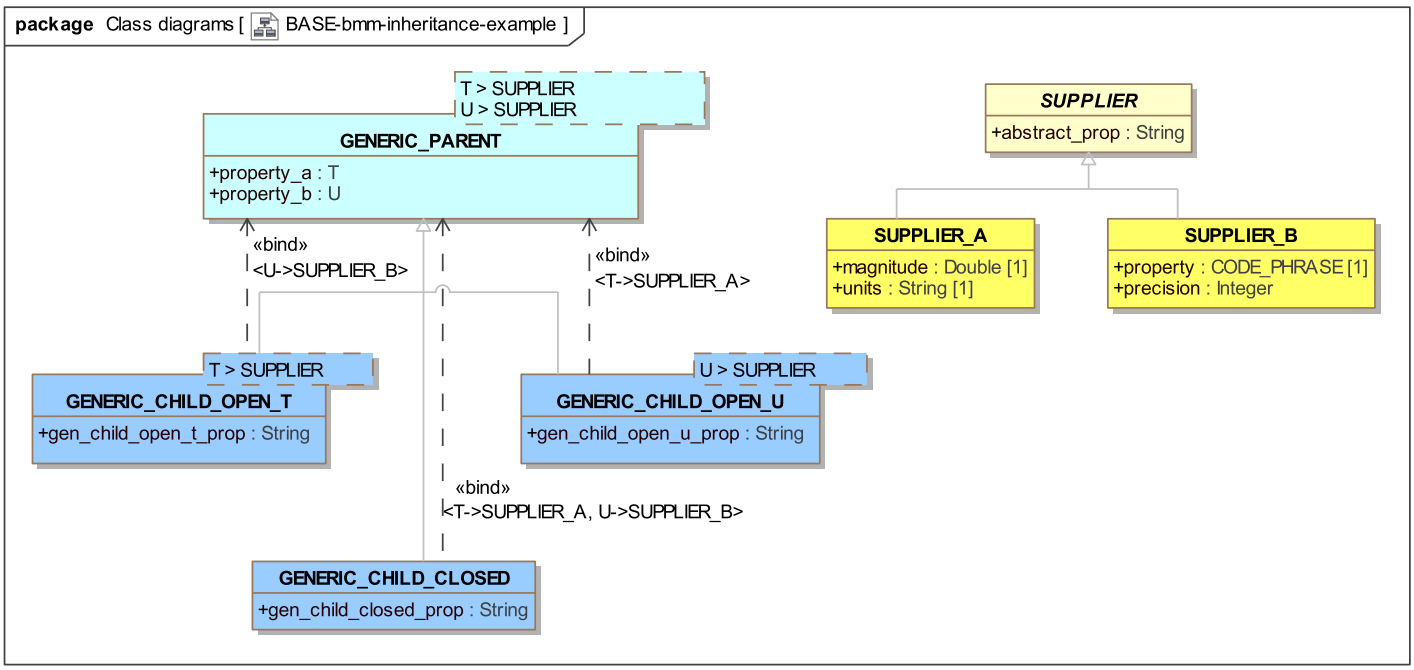


Figure 2. Generic inheritance eaxmple model

ODIN

```
    ["GENERIC_PARENT"] = <
        name = <"GENERIC_PARENT">
        generic_parameter_defs = <
            ["T"] = <
                name = <"T">
                conforms_to_type = <"SUPPLIER">
            >
            ["U"] = <
                name = <"U">
                conforms_to_type = <"SUPPLIER">
            >
        >
        properties = <
            ["property_a"] = (P_BMM_SINGLE_PROPERTY_OPEN) <
                name = <"property_a">
                type = <"T">
            >
            ["property_b"] = (P_BMM_SINGLE_PROPERTY_OPEN) <
                name = <"property_b">
                type = <"U">
            >
        >
    >

    ["SUPPLIER"] = <
        name = <"SUPPLIER">
        is_abstract = <True>
        properties = <
            ["abstract_prop"] = (P_BMM_SINGLE_PROPERTY) <
                name = <"abstract_prop">
                type = <"String">
            >
        >
    >

    ["SUPPLIER_A"] = <
        name = <"SUPPLIER_A">
        ancestors = <"SUPPLIER">
        properties = <
            ["magnitude"] = (P_BMM_SINGLE_PROPERTY) <
                name = <"magnitude">
                type = <"Double">
                is_mandatory = <True>
            >
            ["units"] = (P_BMM_SINGLE_PROPERTY) <
                name = <"units">
                type = <"String">
                is_mandatory = <True>
            >
        >
    >

    ["SUPPLIER_B"] = <
        name = <"SUPPLIER_B">
        ancestors = <"SUPPLIER">
        properties = <
            ["property"] = (P_BMM_SINGLE_PROPERTY) <
                name = <"property">
                type = <"CODE_PHRASE">
                is_mandatory = <True>
            >
            ["precision"] = (P_BMM_SINGLE_PROPERTY) <
                name = <"precision">
                type = <"Integer">
            >
        >
    >

    ["GENERIC_CHILD_OPEN_T"] = <
        name = <"GENERIC_CHILD_OPEN_T">
```

```
        ancestor_defs = <
            ["GENERIC_PARENT<T,SUPPLIER_B>"] = (P_BMM_GENERIC_TYPE) <
                root_type = <"GENERIC_PARENT">
                generic_parameters = <"T", "SUPPLIER_B">
            >
        >
        generic_parameter_defs = <
            ["T"] = <
                name = <"T">
                conforms_to_type = <"SUPPLIER">
            >
        >
        properties = <
            ["gen_child_open_t_prop"] = (P_BMM_SINGLE_PROPERTY) <
                name = <"gen_child_open_t_prop">
                type = <"String">
            >
        >
    >

    ["GENERIC_CHILD_OPEN_U"] = <
        name = <"GENERIC_CHILD_OPEN_U">
        ancestor_defs = <
            ["GENERIC_PARENT<SUPPLIER_A,U>"] = (P_BMM_GENERIC_TYPE) <
                root_type = <"GENERIC_PARENT">
                generic_parameters = <"SUPPLIER_A", "U">
            >
        >
        generic_parameter_defs = <
            ["U"] = <
                name = <"U">
                conforms_to_type = <"SUPPLIER">
            >
        >
        properties = <
            ["gen_child_open_u_prop"] = (P_BMM_SINGLE_PROPERTY) <
                name = <"gen_child_open_u_prop">
                type = <"String">
            >
        >
    >

    ["GENERIC_CHILD_CLOSED"] = <
        name = <"GENERIC_CHILD_CLOSED">
        ancestor_defs = <
            ["GENERIC_PARENT<SUPPLIER_A,SUPPLIER_B>"] = (P_BMM_GENERIC_TYPE) <
                root_type = <"GENERIC_PARENT">
                generic_parameters = <"SUPPLIER_A", "SUPPLIER_B">
            >
        >
        properties = <
            ["gen_child_closed_prop"] = (P_BMM_SINGLE_PROPERTY) <
                name = <"gen_child_closed_prop">
                type = <"String">
            >
        >
    >

>
```

Last updated 2021-07-22 14:07:24 +0100