Document Title	Generic Structure Template
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	202

Document Status	published
Part of AUTOSAR Standard	Foundation
Part of Standard Release	R21-11

Document Change History

Date	Release	Changed by	Description
2021-11-25	R21-11	AUTOSAR Release Management	• Improve Collection • Restructure UML Tags • Update Life Cycle States
2020-11-30	R20-11	AUTOSAR Release Management	• Extend Splitable • Migration of document to standard FO
2019-11-28	R19-11	AUTOSAR Release Management	• Refine Splitable • Extent AttributeValueVariationPoint • Introduce TracableTable • Changed Document Status from Final to published
2018-10-31	4.4.0	AUTOSAR Release Management	• Update Splitable • Include ARMQL • Refine atp.Status
2017-12-08	4.3.1	AUTOSAR Release Management	• Introduction of FileInfoCmmment • Ordered collections • Naming conventions in variant handling patterns
2016-11-30	4.3.0	AUTOSAR Release Management	• Extend AttributeValuePattern for enumeration • Editorial changes • Control the production of specification documents • Added section on Special Data Group Definitions

2015-07-31	4.2.2	AUTOSAR Release Management	• Update View Approach • Combinations of status values • Update Inline Text Model Element
2014-10-31	4.2.1	AUTOSAR Release Management	• Propagation of LifeCycleState • Editorial changes
2014-03-31	4.1.3	AUTOSAR Release Management	• Update of blueprint topics • Extension of variant handling topics • Editorial changes
2013-10-31	4.1.2	AUTOSAR Release Management	• Editorial changes • Extension of formula language
2013-03-15	4.1.1	AUTOSAR Administration	• Editorial changes including tagged specification items • Support of build action manifest • Support of roles and rights • Added life cycle support • Support of collections and collectable elements
2011-12-22	4.0.3	AUTOSAR Administration	• Editorial changes including tagged specification items • Improvements in UML usage (M3), especially mark obsolete elements • Improved specification of primitives, primitive definition, formula language, category • Improved variant handling and blueprint support • Improved support for instanceRef and arrays • Improved definition of package structures

2009-12-18	4.0.1	AUTOSAR Administration	• Editorial changes • Improvements in variant handling (Package content, composed predefined variants) • Align Formula language with ASAM General Expression Language • Generalized approach for anntoations • Improved aligment with ASAM - FSX • Document the admin.* uml tags. • Support global referenceing and tracing
2010-02-02	3.1.4	AUTOSAR Administration	• restructured the document • support for variant handling • support for abstract structures • documentation support • detailed primitives • general modeling information required to understand other templates
2008-08-13	3.1.1	AUTOSAR Administration	• Legal disclaimer revised • Rename document from "Template Modeling Patterns" to "Generic Structure Template"
2007-12-21	3.0.1	AUTOSAR Administration	• Updated Attributes of Identifiable • Added "Hint to the Users" • Added document identification no • Added document classification
2007-01-24	2.1.15	AUTOSAR Administration	• Legal disclaimer revised • "Advice for users" revised • "Revision Information" added
2006-05-16	2.0	AUTOSAR Administration	Second release

Disclaimer

This work (specification and/or software implementation) and the material contained in it, as released by AUTOSAR, is for the purpose of information only. AUTOSAR and the companies that have contributed to it shall not be liable for any use of the work.

The material contained in this work is protected by copyright and other types of intellectual property rights. The commercial exploitation of the material contained in this work requires a license to such intellectual property rights.

This work may be utilized or reproduced without any modification, in any form or by any means, for informational purposes only. For any other purpose, no part of the work may be utilized or reproduced, in any form or by any means, without permission in writing from the publisher.

The work has been developed for automotive applications only. It has neither been developed, nor tested for non-automotive applications.

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

Table of Contents

References

[1] Meta Model
AUTOSAR_MMOD_MetaModel

[2] Standardization Template
AUTOSAR_TPS_StandardizationTemplate

[3] AUTOSAR XML Schema Production Rules
AUTOSAR_TPS_XMLSchemaProductionRules

[4] Specification of ECU Resource Template
AUTOSAR_TPS_ECUResourceTemplate

[5] System Template
AUTOSAR_TPS_SystemTemplate

[6] Specification of Manifest
AUTOSAR_TPS_ManifestSpecification

[7] Standardized M1 Models used for the Definition of AUTOSAR
AUTOSAR_MOD_GeneralDefinitions

[8] Predefined Names in AUTOSAR
AUTOSAR_TR_PredefinedNames

[9] List of Basic Software Modules
AUTOSAR_TR_BSWModuleList

[10] Basic Software Module Description Template
AUTOSAR_TPS_BSWModuleDescriptionTemplate

[11] XML Schema 1.0
http://www.w3.org/TR/xmlschema-1

[12] ANTLR parser generator V3

[13] C++ Operator Precedence
http://www.cppreference.com/wiki/operator_precedence

[14] Collection of blueprints for AUTOSAR M1 models
AUTOSAR_MOD_GeneralBlueprints

[15] Issue Exchange Format V3.0.0
http://www.asam.net

[16] Container Catalog XML Model Specification
http://www.asam.net

[17] ASAM MCD 2MC ASAP2 Interface Specification
http://www.asam.net
ASAP2-V1.51.pdf

[18] Methodology for Classic Platform
AUTOSAR_TR_Methodology

[19] Specification of RTE Software
AUTOSAR_SWS_RTE

[20] Software Component Template
AUTOSAR_TPS_SoftwareComponentTemplate

[21] Unified Modeling Language: Superstructure, Version 2.0, OMG Available Specification, ptc/05-07-04
http://www.omg.org/cgi-bin/apps/doc?formal/05-07-04

[22] ASAM AE Functional Specification Exchange Format V1.0.0
http://www.asam.net
AE-FSX_V1.0.0.pdf

[23] OASIS open exchange table model
http://www.oasis-open.org/specs/tm9901.html

[24] Software Process Engineering Meta-Model Specification
http://www.omg.org/spec/SPEM/2.0/

1 Introduction

This document contains the specification of the AUTOSAR `Generic Structure Template`. Actually, it has been created as a supplement to the formal definition provided by the AUTOSAR meta-model [1]. In other words, this document in addition to the formal specification provides introductory description and rationale for the parts of the AUTOSAR meta-model relevant for almost all AUTOSAR templates.

Nevertheless, the core part of the specification is directly based on the content of the AUTOSAR meta-model. Therefore, this document contains a summary of the main concepts of the AUTOSAR meta-model, see chapters 1.3 and 1.4.

This document provides reference information and is not intended to be read in a sequence. Nevertheless it contains as major aspects:

1. Chapter 3 explains the top level structure which is common to all AUTOSAR templates.
2. Mechanisms used to design AUTOSAR templates:
   (a) Chapter 2 describes an essential aspects of the AUTOSAR Template UML profile which are necessary to understand the AUTOSAR template documents.
   (b) Chapter 4 describes general template classes which are collected similar to the standard library of a compiler.
   (c) Chapter 5 explains abstract classes with abstract relationships. These structures implement **particular concepts** applicable to all AUTOSAR templates. These concepts are applied by specializing these abstract classes and in particular specializing the abstract relationships.
   (d) Chapter 6 explains in general the approach to apply by model transformation (as for example used for Variant handling).
3. Some specific applications of design mechanisms in the meta-model
   (a) Chapter 7 describes the implementation of variant handling within AUTOSAR templates based on meta-modeling patterns (as described in 6).
   (b) Chapter 8 describes the implementation of splitable elements within AUTOSAR M2 models. The corresponding AUTOSAR M1 model view is explained in F.2).
   (c) Chapter 9 describes the documentation support.

1.1 Scope

The scope of this document covers information which is required to understand the AUTOSAR templates and the core mechanisms used to define these templates.

Aspects of UML modeling required to perform the template Modeling tasks are out of the scope of this document.

1.2 Document Conventions

Technical terms are typeset in mono spaced font, e.g. `PortPrototype`. As a general rule, plural forms of technical terms are created by adding "s" to the singular form, e.g. `PortPrototype`s. By this means the document resembles terminology used in the AUTOSAR XML Schema.

This document contains constraints in textual form that are distinguished from the rest of the text by a unique numerical constraint ID, a headline, and the actual constraint text starting after the ⌈ character and terminated by the ⌋ character.

The purpose of these constraints is to literally constrain the interpretation of the AUTOSAR meta-model such that it is possible to detect violations of the standardized behavior implemented in an instance of the meta-model (i.e. on M1 level).

Makers of AUTOSAR tools are encouraged to add the numerical ID of a constraint that corresponds to an M1 modeling issue as part of the diagnostic message issued by the tool.

The attributes of the classes introduced in this document are listed in form of class tables. They have the form shown in the example of the top-level element AUTOSAR:

Please note that constraints are not supposed to be enforceable at any given time in an AUTOSAR workflow. During the development of a model, constraints may legitimately be violated because an incomplete model will obviously show inconsistencies.

However, at specific points in the workflow, constraints shall be enforced as a safeguard against misconfiguration.

The points in the workflow where constraints shall be enforced, sometimes also known as the "binding time" of the constraint, are different for each model category, e.g. on the classic platform, the constraints defined for software-components are typically enforced prior to the generation of the RTE while the constraints against the definition of an Ecu extract shall be applied when the Ecu configuration for the Com stack is created.

For each document, possible binding times of constraints are defined and the binding times are typically mentioned in the constraint themselves to give a proper orientation for implementers of AUTOSAR authoring tools.

Let `AUTOSAR` be an example of a typical class table. The first rows in the table have the following meaning:

Class: The name of the class as defined in the UML model.

Package: The UML package the class is defined in. This is only listed to help locating the class in the overall meta model.

Note: The comment the modeler gave for the class (class note). Stereotypes and UML tags of the class are also denoted here.

Base Classes: If applicable, the list of direct base classes.

The headers in the table have the following meaning:

Attribute: The name of an attribute of the class. Note that AUTOSAR does not distinguish between class attributes and owned association ends.

Type: The type of an attribute of the class.

Mul.: The assigned multiplicity of the attribute, i.e. how many instances of the given data type are associated with the attribute.

Kind: Specifies, whether the attribute is aggregated in the class (`aggr` aggregation), an UML attribute in the class (`attr` primitive attribute), or just referenced by it (`ref` reference). Instance references are also indicated (`iref` instance reference) in this field.

Note: The comment the modeler gave for the class attribute (role note). Stereotypes and UML tags of the class are also denoted here.

Please note that the chapters that start with a letter instead of a numerical value represent the appendix of the document. The purpose of the appendix is to support the explanation of certain aspects of the document and does not represent binding conventions of the standard.

The verbal forms for the expression of obligation specified in [TPS_STDT_00053] shall be used to indicate requirements, see Standardization Template, chapter Support for Traceability ([2]).

The representation of requirements in AUTOSAR documents follows the table specified in [TPS_STDT_00078], see Standardization Template, chapter Support for Traceability ([2]).

1.3 Methodology for Defining Formal Templates

Figure 1.1 illustrates the overall methodology used to define formal templates using System Template as an example. A precise and concise model of the information that needs to be captured in AUTOSAR XML files is provided in [1]

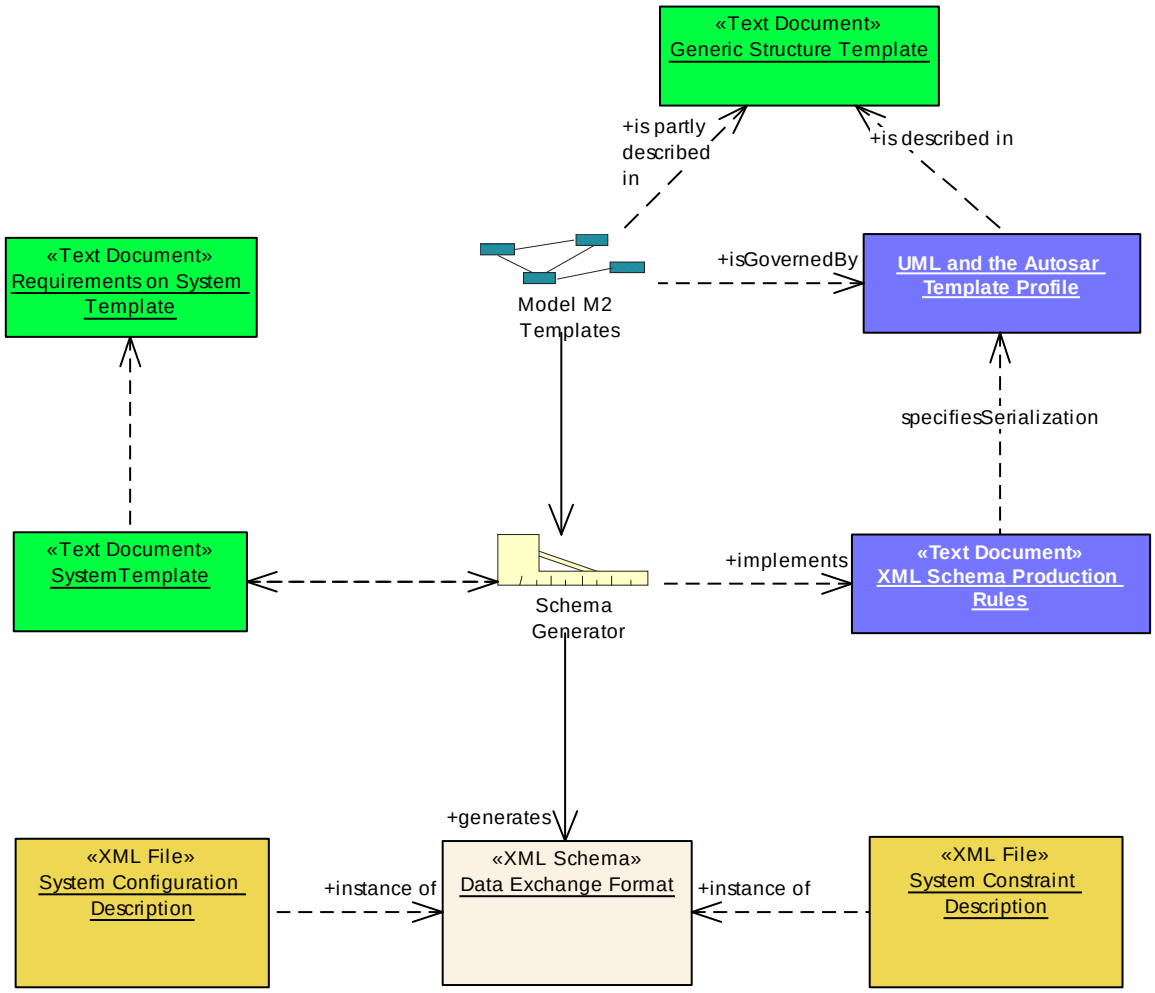


Figure 1.1: Methodology to define templates in AUTOSAR (using SystemTemplate as example

The following documents describe the various aspects of the methodology:

1. The **template document** (in this example `System Template`) describes the information that can be captured in the template, independently from the mapping of this model on XML-technology. It contains an elaborate description of the semantics (the precise meaning) of all the information that can be captured within the relevant parts of the AUTOSAR meta-model.

2. The model called **M2 Templates** in the AUTOSAR meta-model [1] contains the structure of the AUTOSAR templates modeled in UML. The model is annotated using notes which are also represented as class tables in the template documents.

3. The document called **Generic Structure Template** (this document) is represented e.g. as predefined Classes in the meta-model which are incorporated in the generated schema.

4. The **Template UML Profile and Modeling Guide** describes the basic concepts that were applied when creating content of the meta-model. This information is presented in chapter 2.

5. The document called **XML Schema Production Rules** [3] describes how XML is used and how the meta-model designed in the "Software Component Template" should be translated by the "Schema Generator" (MDS) into XML-Schema (XSD) "Data Exchange Format".

   This "formalization strategy" is supposed to be used for all data that is formally described in the meta-model. In particular this document is worth to read in order to understand the mapping of the meta-model and the XML based AUTOSAR template.

6. The **Data Exchange Format** is represented as an XML schema automatically generated out of the AUTOSAR meta-model using the approach and the patterns defined in the **XML Schema Production Rules**. This schema is typically used as input to AUTOSAR tools.

7. The **M1-level descriptions** (in figure 1.1 illustrated as "System configuration description" and "System Constraint Description") are XML files that can be validated against the XML schema and later on follow the specifications in the relevant "template document". In other words, the XML files are instances of the schema defining the XML representation of the template.

1.4 Organization of the Meta-Model

Figure 1.2 sketches the overall structure of the meta-model, which formally defines the vocabulary required to describe AUTOSAR software-components. As the diagram points out, other template specifications (e.g. `ECU Resource Template` [4] and `System Template` [5]) also use the same modeling approach in order to define an overall consistent model of AUTOSAR software description.

The dashed arrows in the diagram describe dependencies in terms of import-relationships between the packages within the meta-model. For example, the package `SWComponentTemplate` imports meta-classes defined in the packages `GenericStructure` (described in this document) and `ECUResourceTemplate` [4].

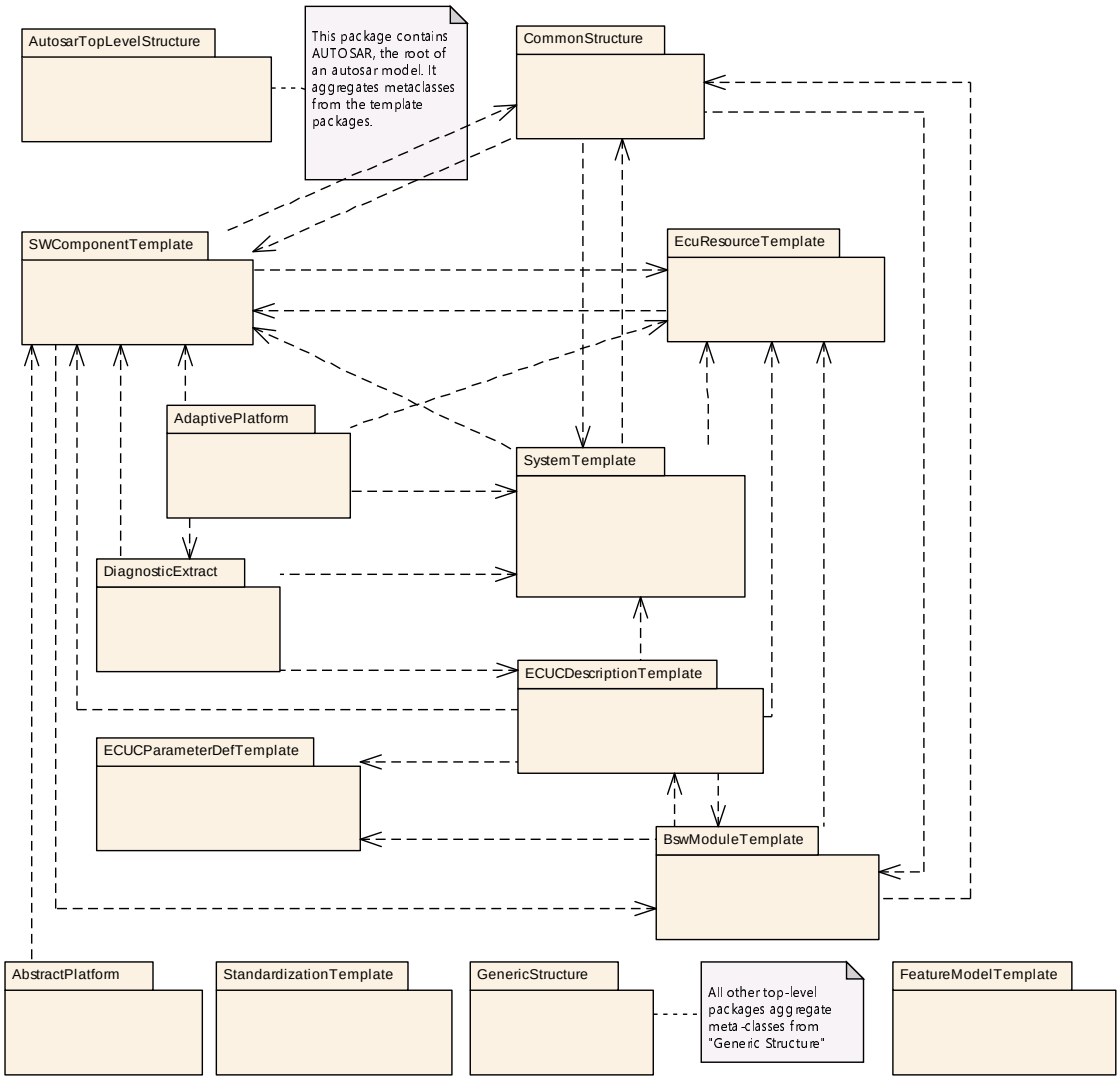


Figure 1.2: Structure of the meta-model

For clarification, please note that the package `GenericStructure` contains some fundamental infrastructure meta-classes and common patterns. As these are used by all other template specification the dependency associations are not depicted in the diagram for the sake of clarity.

2 Usage of UML in AUTOSAR Templates

The AUTOSAR meta-model is defined as an UML model. Therefore basic knowledge of UML is required to understand the AUTOSAR template documents.

2.1 UML Diagrams

The diagrams in the AUTOSAR Template documents are consistent with UML 2.0. The underlying model (the AUTOSAR metamodel) is assumed to be complete even though certain elements might not be shown in a particular diagram to simplify understanding. Nevertheless the class tables show all relevant information.

The coloring of the diagrams is usually explained in the surrounding text. But in general the meta-classes in light green color are those which are taken from ASAM/MSR.

Representation of instance refs is shown in Figure 5.9 (see [TPS_GST_00044]).

2.2 The AUTOSAR meta-model hierarchy

The complete meta-model hierarchy for AUTOSAR templates is shown in figure 2.1. Unlike the classical four-layer architecture used by OMG, five meta levels are shown. Starting at the lowest, most concrete meta level those are:

- **M0: AUTOSAR objects**
  This is the realization of an AUTOSAR system at work: For example a real ECUs executing a software image containing for instance the windshield wiper control software.
- **M1: AUTOSAR models**
  Models on this meta level are built by the AUTOSAR developers. They may define a software component called "windshield wiper" with a certain set of ports that is connected to another software component and so on. On this level all artifacts required to describe an AUTOSAR system are detailed, including re-usable types as well as specific instances of such types.

  The AUTOSAR software is loaded in to individual ECUs for individual vehicles. This loading means that the M1 Model is instantiated.

  Note that such an AUTOSAR model can be represented using various formats ranging from XML, to C even to PDF.
- **M2: AUTOSAR meta-model**
  On this meta level the vocabulary for AUTOSAR templates is defined. This vocabulary later can be used by developers of AUTOSAR based ECU systems.

  For example it is **defined on M2** that in AUTOSAR we have an entity called "software component" which among others aggregate an entity called "port". This

definition ensures that the developer of an AUTOSAR software component can describe his particular component and its ports. This description is called an AUTOSAR model and **resides on M1**.

- **M3: UML profile for AUTOSAR templates**
  The AUTOSAR templates on M2 are built according to the meta-model defined on M3. As discussed before this is UML together with a particular UML profile to better support template modeling work.

  Formally a template on M2 is still an instance of UML, but at the same time the template profile is applied, i.e. that additionally rules set out by the stereotypes in the profile need to be observed. The relevant details of the profile are specified in chapter 2.3 and chapter 2.4.

Note that an AUTOSAR model can be represented using various formats ranging from XML, to C even to PDF. The conversion between these formats is called "transformation", while the fact that an AUTOSAR model follows to the AUTOSAR meta-model is called "instantiation". An AUTOSAR model (M1) is therefore called an instance of **the** AUTOSAR meta-model (M2).

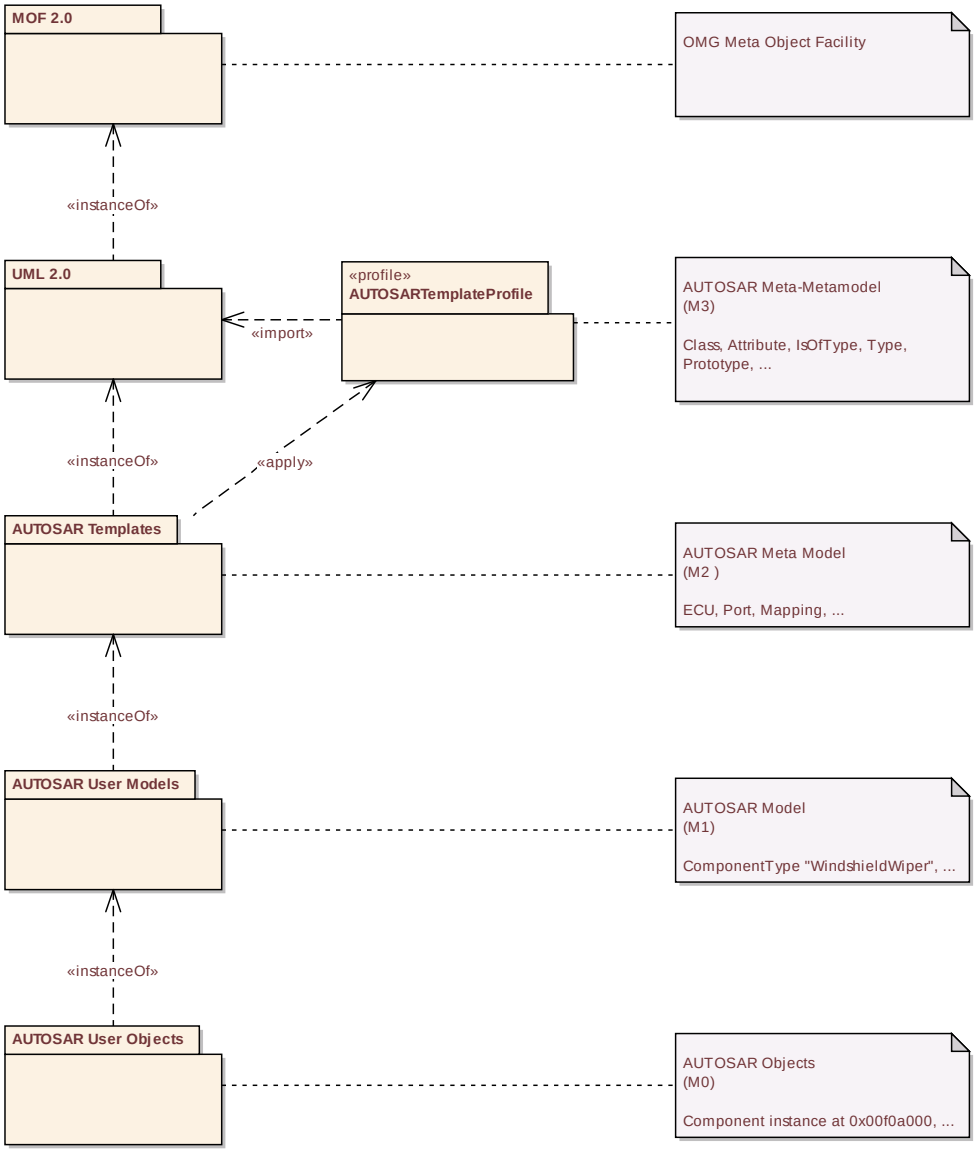


Figure 2.1: Meta Model Hierarchy

2.3 Stereotypes

The AUTOSAR Template Profile uses the following stereotypes[1]:

- **[TPS_GST_00022]** ≪`atpAbstract`≫ **applicable to relations (associations, aggregations)** ⌈This indicates that the relationship is abstract. There needs to be specialized relation in every concrete subclass redefining the abstract relation. This stereotype is there to provide a better visualization in the diagrams. The fact that the relation is abstract is modeled by defining the role as “derived” in the model. It is also indicated by a “/” in front of the role name in the diagrams.

[1] the names of these stereotypes start with `atp` which is the abbreviation for "AUTOSAR Template Profile"

Relations of ≪atpAbstract≫ exist only in the superclass and are not inherited to subclasses. They *need to be redefined* in the subclasses[2].⌋()

- **[TPS_GST_00023] ≪atpDerived≫ applicable to relations (associations, aggregations)** ⌈This indicates that the relationship exists in the subclasses by inheritance. It further indicates that in M1 models the relation is calculated (derived) from other information.

  There are two types of calculation:

  – **general**
  indicates that the value is calculated by a method which is described in the note of the abstract relation. For example the atpBase is calculated as the container of the first atpContextElement.

  – **derived union**
  indicates that it is derived as the union of all concrete relations.

  ⌋()

  For example, the aggregation from AtpClassifier to AtpFeature with role atpFeature is ≪atpDerived≫, SwComponentType has an atpFeature association **in addition** to component, port etc. This atpFeature is calculated as the union of the concrete features.

  derived union means that for a given component type, its atpFeature property holds its ports AND its contained component prototypes AND its contained connectors. This allows to define the instance reference on abstract level.

  Refer to chapter 5 for further details.

- **[TPS_GST_00024] ≪atpMixed≫ applicable to classes** ⌈

  This is applied to meta-classes only and indicates a mixed content model **without** intermixed text.⌋()

- **[TPS_GST_00025] ≪atpMixedString≫ applicable to classes** ⌈

  This is a mixed content model **with** intermixed text. This is applied to meta-classes only.⌋()

  For more details see chapter 2.3.1.

- **[TPS_GST_00026] ≪atpObject≫ applicable to classes** ⌈

  This is an implicit base class. It can only provide attributes with tagged with xml.attribute=true.⌋()

  For more details see chapter 6.3.3.

- **[TPS_GST_00027] ≪atpSplitable≫ applicable to relations** ⌈By using the stereotype ≪atpSplitable≫ the meta-model can explicitly define how in-

[2] In consequence of such a redefinition the XSD-generator ignores such abstract relations.

stances of the meta-model may be distributed over several files. By default all data is stored in one single file. If the ≪atpSplitable≫ is applied, then the associated or aggregated information may be stored in different files.⌋()

- **[TPS_GST_00427] ≪atpIdentityContributor≫ applicable to aggregations, associations and primitive attributes** ⌈This stereotype is applied to aggregations, associations, and primitive attributes which contribute to the identity of a M1 element, for example, the shortName attribute. The properties stereotyped with ≪atpIdentityContributor≫ contribute to the splitkey which controls the merging of splitable elements from partial models.⌋()

  See chapter 8 for more information about splitable elements and splitkeys.

- **[TPS_GST_00028] ≪atpVariation≫ applicable to classes and relations** ⌈This indicates variant handling. It is applied to meta-classes as well as to associations or aggregations.⌋()

  For more details see chapter 7.

- **[TPS_GST_00029] ≪atpUriDef≫ applicable to associations** ⌈This indicates that the essential information is only the full qualified name of the reference.target. This is then used as a whole as identifier for a particular purpose. The association acts as the definition of a kind of "Universal Resource Identifier" in the AUTOSAR model.

  Note that in this case only the full qualified shortName path is important, and not the shortName of the target nor the target itself. The particular semantics and therefore the subsequent processing depends on the individual use case.⌋()

  Therefore it is not always necessary to really follow the references of stereotype ≪atpUriDef≫. Tools should not warn about dangling references of this stereotype unless explicitly requested by the user respectively the particular use case.

  For example in EcucReferenceDef.destination the reference indicates that valid targets of the EcucReferenceValue shall be EcucContainerValues whose definition is derived from the target of EcucReferenceDef.destination. But this can be verified even if the target EcucReferenceDef.destination is not really available.

- **[TPS_GST_00030] ≪instanceRef≫ applicable to dependencies** ⌈This is used to provide a simplified representation of instance references within diagrams.⌋()

- **[TPS_GST_00426] ≪instanceRef≫ in combination with ≪atpUriDef≫** ⌈If the two stereotypes ≪instanceRef≫ and ≪atpUriDef≫ are applied to a dependency, then the ≪atpUriDef≫ semantic defined in [TPS_GST_00029] shall be applied to all sub-references of the instance reference.⌋()

  See chapter 5.1.3 for more details due to instance references.

- **[TPS_GST_00031]** ≪**isOfType**≫ **applicable to associations** ⌈This is used to emphasize the concrete relationship between prototypes (subclasses of AtpPrototype and types (subclasses of AtpType).⌋*()*

  This stereotype influences in generation of associations according to chapter 6.3.2[3].

  [constr_2633] Existence of reference decorated with stereotype ≪isOfType≫ ⌈If a subclass of AtpPrototype defines a reference decorated with stereotype ≪isOfType≫ to a subclass of AtpType, then this reference shall always exist.⌋*()*

2.3.1 Mixed Content (≪atpMixed≫, ≪atpMixedString≫)

If a meta-class has several attributes (which may include aggregations or references), there are cases in which the "serialized" representation (like XML) in the M1 model adds semantics to the actual order and to the number of occurrences of the attribute instances. It may be also be required, that the same attribute appears in multiple instances (in M1) which are mixed with other attribute instances. This situation cannot be expressed in UML in a simple way.

In addition, if a model requires to describe documentation-like information, it often will need to mix formal content and text. An example of such a model is a an embedded link in HTML: markup of formal information bits is mixed into regular text, as shown in the following example:

```
[...]meet <a href="/wiki/Runtime" title="Runtime">runtime</a> requirements
of automotive devices[...]
```

This example illustrates that the "mixed content" feature is is well known in the XML world, where it is called mixed content[4].

[TPS_GST_00032] Basic Features of Mixed Content ⌈The following list indicates the features of mixed content from a modeling point of view. Within a mixed content instance

- a set of formally defined model elements may appear an arbitrary number of times in arbitrary order
- but the actually present order is relevant in terms of semantics of the whole object, and
- In case of ≪atpMixedString≫ unqualified text may be mixed in between any of formally defined elements.

⌋*()*

[3] Note that this stereotype is redundant to the fact that such associations need to redefine the role atpType directly or indirectly.

[4] http://www.w3schools.com/schema/schema_complex_mixed.asp

This mechanism is supported in AUTOSAR through stereotypes

- ≪atpMixedString≫ which allows text between the data elements ([TPS_GST_00025])
- ≪atpMixed≫ which allows any mix of the properties of such a class in any order ([TPS_GST_00024])

The latter stereotype does not allow for mixed-in text, but keeps the definition of order in terms of syntax and semantics.

[TPS_GST_00033] Upper Multiplicity in Mixed content ⌈A mixed content class will aggregate or reference a number of other classes in a template model. The target multiplicity of those relations are typically 1, since the overall number of occurrences is arbitrary by definition given in [TPS_GST_00032].

If, however, multiplicity is different from 1, a required grouping is specified.⌋*()*

For example if the target multiplicity is 2, always a pair of those objects (not just a single object) shall be put into the mixed content, and so on.

Figure 2.2 illustrates how it works. The M1 Model is shown in XML. Please note

1. MixedContent can be an arbitrary mix any order of a b c d e in any order. The order is semantically important. This is the same significance as if an aggregation of upper multiplicity > 1 is annotated as {ordered} in UML[5].
2. c is of upper multiplicity > 1 therefore the wrapper for multiplicity is there
3. e is legally missing since overall number of occurrences is arbitrary by definition.

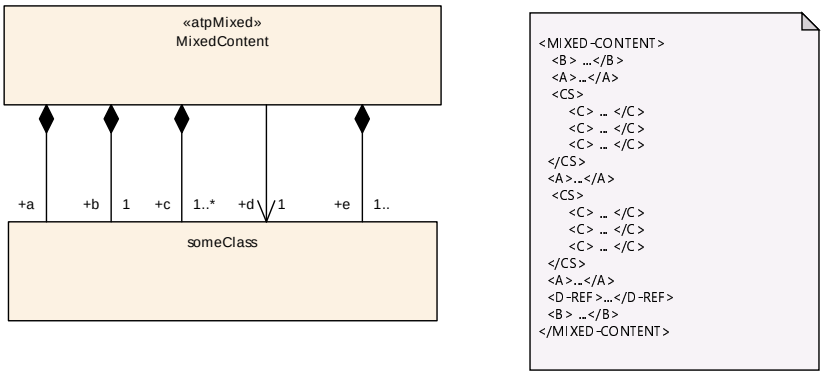


Figure 2.2: Mixed content

[TPS_GST_00045] Inherited properties in mixed content ⌈

- Inherited mixed properties are part of the mixed content and can freely be mixed with own mixed properties.
- Inherited properties from classes, which are not ≪atpMixed≫ themselves, are **not** part of the mixed content.

[5]UML it is not possible to denote this annotation for classes.

- Attributes (with `xml.attribute` set to `true`) and inherited attributes are **not** part of the mixed content.

Note further that in ≪`atpMixedString`≫ there are no inherited properties other than attributes with `xml.attribute` set to `true`.⌋*()*

2.3.2 StructuredComment Elements (≪`atpStructuredComment`≫)

AUTOSAR supports StructuredComment to provide auxiliary information with the goal to create a comment.

[TPS_GST_00381] ≪**`atpStructuredComment`**≫ ⌈Elements marked as ≪`atpStructuredComment`≫ contain information that have no semantics in the model and may be ignored on model level.⌋*()*

[TPS_GST_00382] **Interaction of ≪`atpStructuredComment`≫ and ≪`atpSplitable`≫** ⌈When merging multiple physical files according to splitable elements all elements marked as ≪`atpStructuredComment`≫ and all child elements may be ignored.⌋*()*

The listing 2.1 illustrates the use of ≪`atpStructuredComment`≫ by providing information about the generating tool and its version.

Listing 2.1: File Info Comment in ARXML file

```
<AUTOSAR xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xmlns="http://autosar.org/schema/r4.0"
     xsi:schemaLocation="http://autosar.org/schema/r4.0␣AUTOSAR_00044.xsd">
  <FILE-INFO-COMMENT>
    <SDGS>
      <SDG GID="GENERATION-INFO">
        <SD GID="TOOL-VERSION">ToolA.1.2.3</SD>
        <SD/>
      </SDG>
    </SDGS>
  </FILE-INFO-COMMENT>
  <ADMIN-DATA>
```

Class	**FileInfoComment**			
Package	M2::AUTOSARTemplates::AutosarTopLevelStructure			
Note	This class supports StructuredComment to provide auxiliary information with the goal to create a comment.			
Base	*ARObject*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
sdg	Sdg	*	aggr	This property allows to keep special data which is not represented by the standard model. It can be utilized to keep e.g. tool specific data.

Table 2.1: FileInfoComment

2.4 UML Tags

The AUTOSAR Template Profile uses the following UML tags. Note that only those tags are mentioned here which directly influence the semantic content of the meta-model.

[TPS_GST_00364] UML tags are attached to the target end of relations if suitable ⌈
Unless specified differently with the particular UML tag, UML tags are attached to the target end of a relation (association, aggregation). For Dependency and Generalization UML tags are associated with the connector itself (to overcome limitation of UML tool (EA) used in AUTOSAR).⌋*()*

- **[TPS_GST_00049]** **`atp.recommendedPackage`** ⌈
  This tag provides a recommended package name for objects of the given meta-class. Thereby it provides a value for `{kind}`. Usually it is the name of the meta-class to which the tag is attached. Note that
  - this tag is propagated to subclasses
  - this tag only applies to subclasses of `PackageableElement`

  ⌋*()* The value for `{kind}` is described in Chapter 3.1.

  Possible values are the manifests defined in [6], e.g. Machine Manifest, Execution Manifest, Service Instance Manifest.
- **[TPS_GST_00050]** **`atp.Splitkey`** ⌈
  This specifies the identifying key of ≪`atpSplitable`≫ relations. The tag specifies a comma separated list of the attributes defined in [TPS_GST_00416]. The attributes are the foundation for constructing the identity of M1 elements in partial models as defined in [TPS_GST_00047].⌋*()*

For more details refer to Chapter 8.

2.4.1 Life Cycle Tags

[TPS_GST_00297] Tags to denote life cycle information ⌈AUTOSAR denotes the status with respect to life cycle on entities in UML model by tags with the name `atp.Status*` and `map.Status`.⌋*()*

- **[TPS_GST_00051]** **`atp.Status`** ⌈
  This tag allows to specify the current state of a meta-model entity with respect to its life cycle. It is applicable to classes, aggregations, associations and attributes.

  The following values are supported:

  valid This indicates that the related entity is a valid part of the document.

draft This indicates that the related entity is introduced newly in the meta-model but still experimental. This information is published but is subject to be changed without backward compatibility management.

candidate This shall be used for a related entity that is not yet valid but shall still be recognized by the AUTOSAR BWC checker for the XML schema. Any BWC-incompatible change of the related entity shall be reported by the BWC checker.

obsolete This indicates that the related entity is obsolete and kept in the meta-model for compatibility reasons. If this tag is set, the note shall express the recommended alternative solution.

removed This indicates that the related entity is still in the meta-model for whatever reason (e.g. in context of lifeCycles). It shall not be used and should not even appear in documents. An AUTOSAR release does not contain such elements. It is intended for AUTOSAR internal development. In M1 use case, such removed elements are not included in an `.arxml` delivery but can be referenced in a `LifeCycleInfoSet` by using the ≪`atpUriDef`≫ attribute of type `Referrable`: `lcObject`, respectively `useInstead`.

The allowed combinations of status values in classes and aggregations / references are illustrated in tables 2.2 and 2.3. If the tag is not specified, the related entity is a valid part of the current meta-model. The tag should be applied to the target end of an association.⌋*()*

The tag can be applied to the association, if it is intended to be shown in notes within diagrams.

Non-validated newly introduced elements are typically created in status 'draft', independently of the assumed confidence level. They can be changed in future releases in an incompatible way with respect to AUTOSAR schema.

To mitigate the situation, meta-model elements which are introduced by a concept result in the lifecycle status 'candidate'. Model elements with the new lifecycle status become subject to the compatibility checker and this avoids a situation where such a model element is accidentally changed in a incompatible way that goes undetected.

The lifecycle status 'candidate' is only applicable for meta-model elements and not for the lifecycle status of specification items.

Note that [TPS_GST_00051] focuses on M2 entities (meta-model) while [TPS_STDT_00064] expresses the same for M1 entities (model).

Note that Listing 12.1 provides these values as AUTOSAR arxml file.

- **[TPS_GST_00413] Default value of `atp.Status`** ⌈
The default value of `atp.Status` for a meta-class is 'valid', an association inherits the value from the source, an aggregation and an attribute inherit the value from the parent.⌋*()*

- **[TPS_GST_00295]** **`atp.StatusRevisionBegin`** ⌈
  This tag indicates the AUTOSAR-Revision from which on the status denoted in `atp.Status` is viable. This corresponds to the `periodBegin` as specified in [TPS_GST_00244].⌋*()*
- **[TPS_GST_00296]** **`atp.StatusRevisionEnd`** ⌈
  This tag indicates the AUTOSAR-Revision from which on the status denoted in `atp.Status` is viable. This corresponds to the `periodEnd` as specified in [TPS_GST_00244].⌋*()*
- **[TPS_GST_00274]** **`atp.StatusComment`** ⌈This represents a short note about the current status according to [TPS_GST_00051]. It is primarily applied to status values other than "valid".⌋*()*
- **[TPS_GST_00362]** **`map.Status`** ⌈This tag allows the definition of a life cycle for upstream mappings. The values of this tagged value are linked to the values of `atp.status` [TPS_GST_00051].⌋*()*

The status values in aggregations / references and classes are illustrated in table 2.2 and table 2.3. There '1' means allowed and '0' means not allowed combinations. The tables represent direct children no inherit children.

	Status of (Aggregation/Reference/Attribute)				
Status of Class (parent/source)	draft	candidate	valid	obsolete	removed
draft	1	0	0	1	1
candidate	0	1	0	1	1
valid	1	1	1	1	1
obsolete	0	0	0	1	1
removed	0	0	0	0	1

Table 2.2: Matrix of allowed status value combinations (Aggregation/Reference/Attribute)

	Status of Class (child/target)				
Status of (Aggregation/Reference/Attribute)	draft	candidate	valid	obsolete	removed
draft	1	0	1	0	0
candidate	0	1	1	0	0
valid	0	0	1	0	0
obsolete	1	1	1	1	0
removed	1	1	1	1	1

Table 2.3: Matrix of allowed status value combinations (Class)

The scope of table 2.2 and table 2.3 is illustrated in figure 2.3.

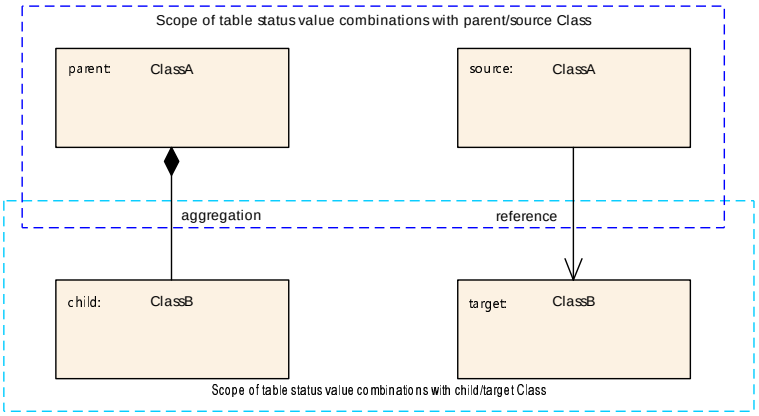


Figure 2.3: Status value combinations

An invalid case: status of (Aggregation/Reference) = "valid" and status of Class (parent/source) = "obsolete"; is illustrated in figure 2.4.

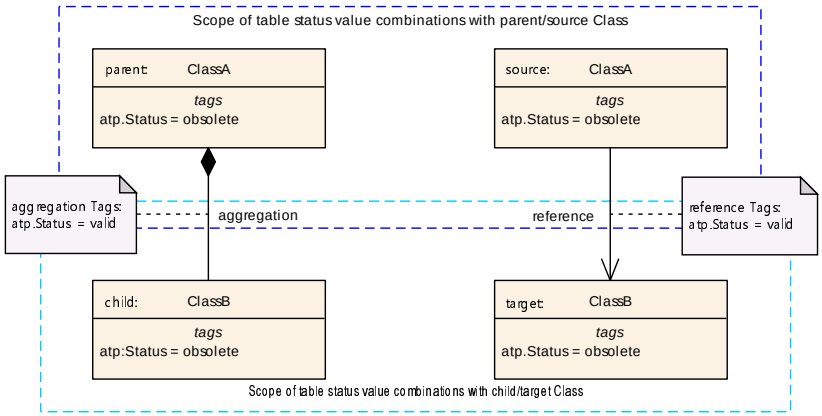


Figure 2.4: Invalid combination of status values

[TPS_GST_00370] **`atp.EnumerationLiteralIndex`** ⌈This tag allows the definition of an index for particular enumerator literals. This index allows to specify an enumerator literal as a result of a numerical function (in particular the result of a formula expression). The index is unique within one particular Enumeration meta-class.⌋*()*

Listing 2.2: Example for enum attibute value variation point

```
<!-- this is an enum attibute value variation point which should provide
    integer value note that this examples assumes that SwBaseType.byteOrder
    is variant which is not really the case. -->
<SW-BASE-TYPE>
   <SHORT-NAME>ByteOrderBasetype</SHORT-NAME>
   <LONG-NAME>
      <L-4 L="EN">byte order basetype</L-4>
   </LONG-NAME>
   <BASE-TYPE-ENCODING>NONE</BASE-TYPE-ENCODING>
   <BYTE-ORDER BASE="DefaultEnumMappingTables" ENUM-TABLE="ByteOrderEnum">
      <SYSC-REF BASE="mysystemconstants">processor</SYSC-REF> == 42 ? 0 : 1
         </BYTE-ORDER>
</SW-BASE-TYPE>
```

In listing 2.2 the ByteOrder is calucated by a formal expression. In case the result of this formal expression is 42 it leads to "0". This `numericalValue` marks an index on M2 level and is mapped to EnumerationLiteral `MOST-SIGNIFICANT-BYTE-FIRST` of the `ByteOrderEnum`. Leads the result of the calculation to "1" it ends up in the EnumerationLiteral `MOST-SIGNIFICANT-BYTE-LAST` of the `ByteOrderEnum`. See `EnumerationMappingTable` in [7]. The `numericalValue` is not used in C-Code or at runtime.

2.4.2 Document Production Control Tags

AUTOSAR specifies a means to control the appearance of platform specific content in specification documents. The semantics of these restrictions are explained further:

- on documentation level [TPS_GST_00428]
- on modelling level [TPS_GST_00431]

The means, is meta-model artifact specific and explained further in:

- meta-model classes, connectors and attributes [TPS_GST_00372]
- meta-model diagrams [TPS_GST_00429]
- meta-model descriptions [TPS_GST_00430]

[TPS_GST_00428] Standards restriction on document level ⌈The usage of a standards restriction controls the appearance of model elements in generated document artifacts within the scope of the mentioned standard. If a standards restriction is applied to a model element then this model element shall appear only in the generated artifacts of the standards identified by the values of the restriction.⌋*()*

[TPS_GST_00431] Implications of a standards restriction on modelling level ⌈Model elements whose visibility is excluded from a respective platform due to [TPS_GST_00372] are also implicitly excluded from any further semantical meaning in a model even if they are visibly present in a model.⌋*()*

Without [TPS_GST_00431] it would be necessary to also write formal AUTOSAR constraints in addition to the restrictions from [TPS_GST_00372] to further restrict any usage or association of those excluded artifacts in a model. This would increase the number of constraints significantly. The obvious example would be a model, where the restricted artifacts may be present, but when the standards restriction is taken into account, have no documentary semantics.

[TPS_GST_00371] Tags to control the production of specification documents ⌈UML-Tags with the name mmt.* control the production of specification documents.⌋ *()*

- **[TPS_GST_00372]** **`mmt.RestrictToStandards`** ⌈The usage of this tag controls the appearance of model elements such as meta-classes, connectors and attributes in generated artifacts with respect to the mentioned stan-

dard. If mmt.RestrictToStandards is applied to a model element then this model element shall appear only in the generated artifacts of the standard identified by the value of mmt.RestrictToStandards. The value of mmt.RestrictToStandards is a comma separated list with one or more of the following values "CP", "AP", "FO", "TC", "TA". White-space around the values in the list is not significant.

Example: Tags:mmt.RestrictToStandards = "CP, AP"

If no standards restriction tag is applied, or, if the list in the value part is empty, it is assumed no restriction applies and thus the artifact is generated for all standards.⌋*()*

AUTOSAR defines a single common meta-model for all platforms. This allows reuse of existing meta-classes. Some meta-classes will have attributes, aggregations or references that are mutually exclusive for the respective standards, e.g. the attributes relevant for the classic platform shall only appear in generated artifacts for the classic platform.

- **[TPS_GST_00353] mmt.templateTable** ⌈
  This tag is used to associate templates with the given variation Point. The value is a white space separated list of templates denoted by names according to [TR_PDN_00003]. In particular it is the abbrName defined by a Keyword of classification DocumentAbbreviation in set "DocumentAbbreviations" specified in [8] (e.g. SWCT).

  mmt.templateTable tag is applied to any package with objects contributing to the meta-model.

  mmt.templateTable applies transitively to all sub-packages of the package in which it is defined. Nevertheless sub-packages may override the value provided by mmt.templateTable on ancestor packages.⌋*()*

[TPS_GST_00429] Application of a standards restriction for meta-model diagrams ⌈The application of a standards restriction for meta-model diagrams is a Diagram Note, where the value is a JSON formatted key / value, where the key=mmt.RestrictToStandards and the value is a string containing a comma separated list with one or more of the following values: "CP", "AP", "FO". White-space around the values in the list is not significant.

Example: {'mmt.RestrictToStandards' : 'CP, AP'}

If no standards restriction is applied, or, if the list in the value part is empty, it is assumed no restriction applies and thus the artifact is generated for all of those standards.⌋*()*

[TPS_GST_00430] Application of a standards restriction for meta-model description fields ⌈The application of a standards restriction for meta-class or connector descriptions (role notes) which shall appear in the generated documentation, uses an embedded parser token in the description field to assign text blocks to platforms which is processed by the AUTOSAR documentation tooling suite.

Example:

```
@RESTRICT_TO_STANDARD:CP:AP!
 Classic and Adaptive text.
@RESTRICT_TO_STANDARD:FO!
 Foundation text.
@END_RESTRICT_TO_STANDARD!
```

If no standards restriction is applied, or, if the list in the value part is empty, it is assumed no restriction applies and thus the artifact is generated for all of those standards.⌋*()*

2.4.3 Variant Handling Tags

[TPS_GST_00298] Tags to denote Variant Handling Properties ⌈UML- Tags with the name vh.* relate to the variant handling.⌋*()*

- **[TPS_GST_00052]** **`vh.latestBindingTime`** ⌈
  This tag controls the binding time of variant handling.⌋*()*

  For more details see chapter 7.6 [TPS_GST_00182].

2.4.4 Schema Production Configuration Tags

[TPS_GST_00291] UML-tags for Configuration of XML schema production ⌈UML-Tags with the name `xml.*` relate to the XML serialization of AUTOSAR models. They basically do not influence the semantics of an M1 model, but ensure that the XML serialization can be controlled by the meta-model.

They also provide means to tweak the schema such that backwards compatibility of the schema can be ensured while the meta-model evolves.

For more details how to apply these tags and about the impact of these tags, refer to chapter "4 Configuration of XML schema production" in [3].⌋*()*

- **[TPS_GST_00053]** **`xml.xsd.*` etc.** ⌈
  These tags allow to define details of primitive types by using XSD restrictions. Even if it is specified by means of a technology specific definition it applies to the meta-model independently of the storage technology.⌋*()*
- **[TPS_GST_00054]** **`xml.xsd.customType`** ⌈
  This tag is applicable to a ≪`primitive`≫. It specifies the name of the `xsd:simpleType` which represents the primitive type.⌋*()*
- **[TPS_GST_00055]** **`xml.attribute`** ⌈
  determines if the UML-attribute is serialized as an XML attribute. This tag allows to control the XML serialization and is not relevant for the semantics of an M1 model.⌋*()*

- **[TPS_GST_00056]** **`xml.attributeRef`** ⌈
  determines if the UML-attribute is serialized as a reference to a global XML attribute. If set to `true` serializes the property as a reference to a global attribute. Applicable only if `xml.attribute` is set to `true`.

  The name of the referenced attribute is specified in `xml.name`. The namespace prefix of the referenced attribute is specified in `xml.nsPrefix`.⌋*()*

- **[TPS_GST_00057]** **`xml.enforceMinMultiplicity`** ⌈
  If true, enforce minimum multiplicity; otherwise, it is "0". In order to allow for transmitting partial information, the minimum multiplicity is not enforced by default in the standardized schema.⌋*()*

- **[TPS_GST_00058]** **`xml.enforceMaxMultiplicity`** ⌈
  If true, enforce maximum multiplicity; otherwise, it is "unbounded". By default `xml.enforceMaxMultiplicity` is true.⌋*()*

- **[TPS_GST_00059]** **`xml.globalElement`** ⌈
  If true, a global `xsd:element` is created for the tagged class. This `xsd:element` can be used as the root element of an instance of the schema. This tag needs to be explicitly defined in the AUTOSAR meta-model. Usually only the meta-class AUTOSAR is represented by a globally defined XML element.⌋*()*

- **[TPS_GST_00060]** **`xml.mds.type`** ⌈
  determines the data type for a primitive if this is a primitive type generated by the meta-model tool. Major example for such a generated type is given by `REFERRABLE-SUBTYPES-ENUM`. This tag shall be applied to a ≪`primitive`≫ which then acts as a proxy to the type denoted in `xml.mds.type`.⌋*()*

- **[TPS_GST_00061]** **`xml.name`** ⌈
  Provides the name of a schema fragment (element, attribute, group, etc.) that represents the role or class.

  If not explicitly defined in the AUTOSAR meta-model, then this value is calculated as explained in [3].⌋*()*

- **[TPS_GST_00062]** **`xml.nsPrefix`** ⌈
  This tag may be applied to

  attribute determines the name space prefix for properties with `xml.attributeRef` set to `true`

  package determines the name space prefix used for the schema based on this package.

  ⌋*()*

- **[TPS_GST_00063]** **`xml.nsUri`** ⌈
  determines the name space URI used for the schema based on this package. The format of the name space URI is defined in [3]. If not explicitly defined in the AUTOSAR meta-model, then this value is implicitly specified as explained in [3].⌋
  ()

- **[TPS_GST_00064]** **`xml.roleElement`, `xml.roleWrapperElement`, `xml.typeElement`, `xml.typeWrapperElement`** ⌈
These tags allow to control the XML serialization, in particular the creation of XML-Elements and are not relevant for the semantics of an M1 model. For more details see [3].⌋*()*

- **[TPS_GST_00065]** **`xml.sequenceOffset`** ⌈
Determines the sequence in which the properties are serialized in XML. If this tag is missing, the properties are serialized in alphabetical order. This sequence is relevant for easier maintenance of the XML artifacts but are not relevant for the semantics of an M1 Model.⌋*()*[6]

- **[TPS_GST_00066]** **`xml.systemIdentifier`** ⌈
Determines the System identifier which should be used in instances dedicated to this schema. If not explicitly defined in the AUTOSAR meta-model, then this value is implicitly specified as explained in [3].⌋*()*

2.4.5 Adminstrative UML Tags

[TPS_GST_00292] Adminstrative UML Tags ⌈For document administration of the meta-model, the UML tags with the name pattern `admin.*` are applied to a particular package (for example `M2::AUTOSAR Templates::ReadMe`). If this package is referenced in the configuration files of the "Meta-Model Tool" (MMT) the values are forwarded to the generated artifacts (e.g. MMOD_XMLSchema or MOD_ECUConfigurationParameters).

In addition to the UML tags the disclaimer for the generated artifacts is taken from the package note of this package.⌋*()*

The following UML tags apply:

- **[TPS_GST_00067]** **`admin.documentClassification`** ⌈
Denotes the classification of the meta-model (Standard resp. Auxiliary)⌋*()*

- **[TPS_GST_00068]** **`admin.documentIdentificationNo`** ⌈
This represents the AUTOSAR document number.⌋*()*

- **[TPS_GST_00069]** **`admin.documentOwner`** ⌈
This denotes the maintainer of the meta-model.⌋*()*

- **[TPS_GST_00070]** **`admin.documentResponsibility`** ⌈
This denotes the responsible authority of the meta-model.⌋*()*

- **[TPS_GST_00071]** **`admin.documentStatus`** ⌈
This denotes the status of the meta-model.⌋*()*

- **[TPS_GST_00072]** **`admin.documentTitle`** ⌈
This denotes the title assigned to the meta-model.⌋*()*

[6] If the sequence is relevant, this is denoted as {ordered} multiplicity in the model.

- **[TPS_GST_00073] `admin.documentVersion`** ⌈
  This denotes the official version of the meta-model. Note that the document version is not related to the `admin.partOfRelease` so a version number like 3.1.12 doesn't necessarily refer to the R3.1 branch of AUTOSAR.⌋*()*

  AUTOSAR distinguishes an external and an internal AUTOSAR release number:

  – The external AUTOSAR release number is given in the format R<yy>.<mm> (e.g. R20-11) and identifies a specific publication of an AUTOSAR release for the external community, see [TPS_GST_00074].

  – The internal AUTOSAR release number is given in the format defined by the RevisionLabelString (e.g. 4.6.0) and is used for the internal identification and reference of an AUTOSAR release. The internal AUTOSAR release number is used, e.g. in BSW modules, the XML schema, and ARXML files, see [TPS_GST_00076].

- **[TPS_GST_00074] `admin.partOfRelease`** ⌈
  This denotes the external AUTOSAR release (e.g. R20-11) in which the meta-model is published. Note that this tag is necessary as it's being used by the tooling in order to control the details of various generators such as:

  – the insertion of hardwired `xsd:simpleType` named `REF` for AUTOSAR release less than 4.0.

  – the processing of primitives according to their implementation.

  – structural differences of artifacts (e.g. classtables) among the AUTOSAR releases.

  ⌋*()* Details to the implementation of primitives are described in Chapter 6.3.1.

- **[TPS_GST_00075] `admin.releaseDate`** ⌈
  This denotes the date of the AUTOSAR release in which the meta-model is published⌋*()*

- **[TPS_GST_00076] `admin.revision`** ⌈
  denotes the particular revision of the internal AUTOSAR release in which the meta-model is published.⌋*()*

2.4.6 Upstream Mapping Specification Tags

[TPS_GST_00299] Tags to specify Upstream Mapping ⌈UML-Tags with the name `map.{template}.*` relate to the description of upstream mapping. Upstream mapping describes if and how entities (M1 or M2) in artifacts crated latter in the methodology (also called downstream) relate to entities created earlier (also called upstream).⌋ *()*

- **[TPS_GST_00301] Placeholder `{template}` in Upstream Mapping Specification Tags** ⌈This denotes the applicable upstream template, in which the map-

ping will be listed. These names follow [TR_PDN_00003]. In particular it is the `abbrName` defined by a `Keyword` of `classification` `DocumentAbbreviation` in set "DocumentAbbreviations" specified in [8] (e.g. `SWCT`).⌋*()*

- **[TPS_GST_00300]** **`map.{template}.desc`** ⌈This provides a description of the mapped entity.⌋*()*
- **[TPS_GST_00302]** **`map.{template}.m2element`** ⌈This denotes the reference to M2 entity, e.g. `SystemSignal.length`⌋*()*
- **[TPS_GST_00303]** **`map.{template}.rule`** ⌈This gives textual description how to transform the data, e.g. 1:1 mapping⌋*()*
- **[TPS_GST_00304]** **`map.{template}.type`** ⌈This represents the mapping quality. The following values apply:

  local no mapping needed since parameter local to BSW

  partial some data can be automatically mapped but not all

  full all data can be mapped automatically

  ⌋*()*

[TPS_GST_00363] **map.Id** ⌈This tag allows the unique identification of a given upstream mapping for the purpose of tracing.⌋*()*

3 AUTOSAR Top Level Structure

AUTOSAR uses a common top level structure for all AUTOSAR templates. This approach leaves maximum flexibility to design the artifacts in the AUTOSAR methodology. Figure 3.1 illustrates the AUTOSAR top level structure.

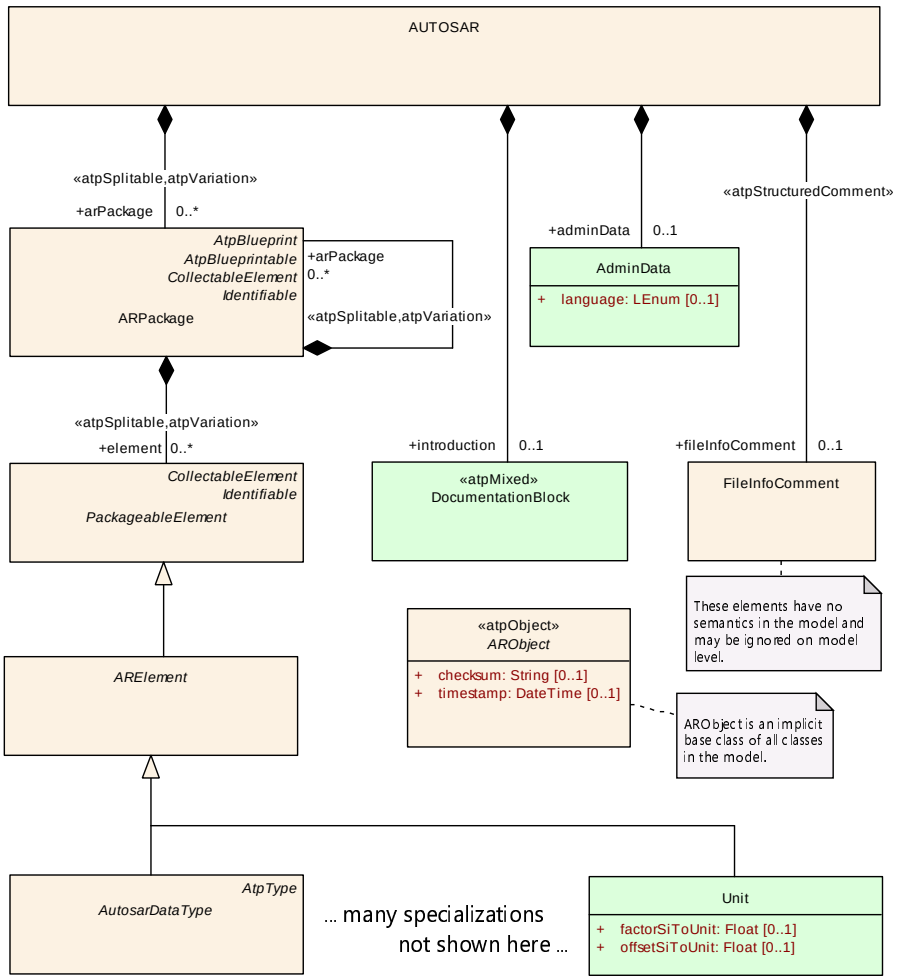


Figure 3.1: Top level structure of AUTOSAR templates

[TPS_GST_00077] Top-Level Structure of an AUTOSAR Model ⌈The meta-class `AUTOSAR` is the root of all templates. `AUTOSAR` contains multiple `ARPackage`s as `arPackage`.⌋*()*

`ARPackage` can be arbitrarily nested. These packages contain `PackageableElement`s which represent particular autonomous entities of AUTOSAR templates. The most prominent specialization of this is `ARElement` (see Chapter 4.2).

Note that all[1] AUTOSAR meta-classes inherit from `ARObject` (see Chapter 4.1).

[TPS_GST_00078] AUTOSAR top level AdminData ⌈The top level structure also contains `AdminData` which specifies two major aspects of an AUTOSAR artifact:

[1] Obviously `ARObject` dos not inherit from itself.

- change management information specified as DocRevision
- language status of the document

⌋()

For more details see Chapter 4.4).

[TPS_GST_00079] Language Status of an Artifact ⌈The language status of the artifact specifies:

1. the "master" language of the document specified in the attribute language in the top level AdminData.
2. the additional languages which are in the document. This is specified as usedLanguages which is a MultiLanguagePlainText which serves as a list of languages used in the document.

⌋()

For more details on the multi language approach see Chapter 9.6. The following example illustrates the top-level structure of an ARXML file. This file is maintained in English as well as in German. English is the master language.

Listing 3.1: Top-Level Structure of an ARXML file

```
<?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns="http://autosar.org/schema/r4.0"
      xsi:schemaLocation="http://autosar.org/schema/r4.0␣AUTOSAR_4-1-3.xsd"
         >
  <ADMIN-DATA>
    <LANGUAGE>EN</LANGUAGE>
    <USED-LANGUAGES>
      <L-10 L="EN" xml:space="default">English</L-10>
      <L-10 L="DE" xml:space="default">German</L-10>
    </USED-LANGUAGES>
  </ADMIN-DATA>
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>demo</SHORT-NAME>
      <ELEMENTS>
        <!--
          autosar elements here
        -->
      </ELEMENTS>
    </AR-PACKAGE>
  </AR-PACKAGES>
</AUTOSAR>
```

see also AUTOSAR

An AUTOSAR artifact is organized in ARPackages which contains so called PackageableElements elements. These are defined on its own nature, they exist independently from each other and are used by associations. For example a computation method is defined on its own. It is used by data definitions through a reference.

For more details about ARPackage please refer to Chapter 4.2.

3.1 Identifying M1 elements in packages

Packages are used to organize AUTOSAR M1 models. AUTOSAR GBR itself publishes M1 models as part of the released standard. In order to clearly identifying such model elements, the following rules apply:

- **[TPS_GST_00080] Package Structure for AUTOSAR delivered Models** ⌈Model elements standardized by AUTOSAR and delivered as ARXML live in a top-level Package of which the shortName is AUTOSAR.

  This means that data elements which are defined by an OEM or supplier should **not** live in a top level package named AUTOSAR.⌋*()*

  [TPS_GST_00081] Pattern for AUTOSAR delivered Models ⌈The package structure of AUTOSAR delivered models follow the pattern:

```
/AUTOSAR
    /{module}                                       -- identify the spec
       /{kind}s[_Blueprint | _Example]              -- identify the kind
                                                    --          of object
```

  ⌋*()*

  Note that AUTOSAR typically delivers Blueprints. For more details see [TPS_STDT_00067].

  An example structure is

```
/AUTOSAR
    /ComM
       /ApplicationDataTypes_Blueprint          [BLUEPRINT]
       /BswModuleEntrys_Blueprint               [BLUEPRINT]
       /CompuMethods_Blueprint                  [BLUEPRINT]
       /DataConstrs_Blueprint                   [BLUEPRINT]
       /DataTypeMappingSets_Blueprint           [BLUEPRINT]
       /Documentations                          [STANDARD]
       /ImplementationDataTypes_Blueprint       [BLUEPRINT]
       /ImplementationDataTypes                 [STANDARD]
       /ModeDeclarationGroups_Blueprint         [BLUEPRINT]
       /SwcBswMappings_Blueprint                [BLUEPRINT]
       /SwComponentTypes_Blueprint              [BLUEPRINT]
       /BswModuleDescriptions_Blueprint         [BLUEPRINT]
```

  In this example, there is a package with blueprints for implementation data types as well as for implementation data types which are finally implemented as STANDARD.

Another example is

```
/AUTOSAR
    /AISpecification
       /DataConstrs                              [STANDARD]
       /PhysicalDimensions                       [STANDARD]
       /Units                                    [STANDARD]
       /ApplicationDataTypes_Blueprint           [BLUEPRINT]
       /CompuMethods_Blueprint                   [BLUEPRINT]
       /PortInterfaces_Blueprint                 [BLUEPRINT]
       /PortPrototypeBlueprints_Blueprint        [BLUEPRINT]
       /ApplicationDataTypes_Example             [EXAMPLE]
       /BlueprintMappingSets_Example             [EXAMPLE]
       /CompuMethods_Example                     [EXAMPLE]
       /PortInterfaces_Example                   [EXAMPLE]
       /SwComponentTypes_Example                 [EXAMPLE]
```

This example shows a use case which provides STANDARD, BLUEPRINT and EXAMPLE.

[TPS_GST_00082] Package Structure for ECUC parameter definitions ⌈Note that that for compatibility reasons, the ECUC package structure is kept in AUTOSAR 4.0 as

```
/AUTOSAR
      /EcucDefs
```

⌋()

- **[TPS_GST_00083] Pattern for AUTOSAR defined Model Elements** ⌈Model elements for which AUTOSAR specification already defines a standardized name (such as platform types) shall live in a package path according to the following pattern:

  `/AUTOSAR_{module}[_{postfix}]/{kind}s`⌋()

In these given patterns, the following placeholders apply:

- **[TPS_GST_00017] `{module}` denotes a Module Designator** ⌈The module designator is one of
  - the module, library etc. (as API service prefix according to [9] (e.g. `CanIf`, `Ifx`, `Compiler`)
  - the `abbrName` defined by a `Keyword` of `classification` `ModuleDesignator` in set "VirtualModules" specified in [8] (e.g. `AISpecification`).

  ⌋()

- **[TPS_GST_00084] `{postfix}` denotes the particular implementation** ⌈If and only if multiple implementations of the BSW Module appear in the same system the a postfix is added to the package structure.⌋*()*
- **[TPS_GST_00085] `{kind}` denotes the kind of element** ⌈The value is the name of a subclass of `ARElement` with an appended plural-'s'. The particular package names are specified for each `ARElement` using the UML tag `atp.recommendedPackage` (see [TPS_GST_00049]) and shown as such in the class tables (e.g. `BswModuleDescriptions` derived from `BswModuleDescription`.⌋*()*

[TPS_GST_00086] Category of `ARPackage` ⌈The value of attribute `category` of an `ARPackage` can be taken as an indication about the nature of the `ARPackage`'s content. Certain values of attribute `category` are standardized by AUTOSAR: `STANDARD`, `BLUEPRINT`, `EXAMPLE`, `ICS`. For the definition of custom values of attribute `category` [TPS_GST_00016] applies.⌋*()*

- **[TPS_GST_00087] `BLUEPRINT`** ⌈
  Elements in such a package act as a kind of "blueprint" for real objects.

  This applies in particular to objects such as `PortInterface` which are not modeled particularly to be a "blueprint" but still are specializations of `AtpBlueprint` respectively `AtpBlueprintable`.⌋*()*

  For example, an authoring tool provides the such predefined `PortInterface` as a kind of toolbox from which the definitions can be copied to a real project.

  Model elements in such packages may be only defined partially. Therefore particular semantic constraints may apply. See [TPS_STDT_00002] in [2] for more details.

  [constr_2501] Blueprint of blueprints are not supported ⌈Note that objects modeled particularly as a "blueprint" (e.g. `PortPrototypeBlueprint`) also live in a package of category `BLUEPRINT`. Strictly speaking this means that they can be "blueprints" of "blueprints". This indirection is not intended and not supported.⌋*()*
- **[TPS_GST_00088] `STANDARD`** ⌈
  Elements which are standardized by the submitter of the related top level package and can be used as is for processing (e.g. ECU Parameter definitions).⌋*()*

  Note that this also allows to represent stakeholder specific standard elements since `STANDARD` is not limited to AUTOSAR internal application.
- **[TPS_GST_00196] `ICS`** ⌈
  Elements which form an Implementation Conformance Statement.⌋*()*

  [constr_4055] ICS may not contain blueprints ⌈Since an Implementation Conformance Statement always describes a set of one or more fully configured software modules, a package with category `ICS` it is not allowed to contain subpackages at any level which have the category `BLUEPRINT`.⌋*()*

[constr_2573] ICS shall not reference examples ⌈ICS is like a productive Model and therefore shall not reference to an EXAMPLE. Such a reference would be useless since the target needs to be ignored in the ICS.⌋*()*

For more details on the content of an Implementation Conformance Statement refer to [10].

- **[TPS_GST_00089] EXAMPLE** ⌈
  Elements in EXAMPLE package illustrate how to apply for example Elements defined in STANDARD or BLUEPRINT packages. Elements in EXAMPLE packages shall be ignored by generators etc.⌋*()*

[TPS_GST_00090] Non Standardized Category of ARPackage ⌈Model elements which do not fall in to one of these categories should live in a package with a category mutually agreed between the stakeholders. It is also possible to have no category at all in this case.⌋*()*

[constr_2515] Categories of packages shall not conflict ⌈If a non empty category is defined for a package, then all sub packages shall have empty category or the same category. See table 3.1. Additionally, the "Rules for references between elements in packages with specific categories" shall apply. See table 3.2.⌋*()*

	child + category (also indirect children)						
parent category	empty	BLUEPRINT	STANDARD	EXAMPLE	ICS	cus-tom1	cus-tom2
empty	ok	ok	ok	ok	ok	ok	ok
BLUEPRINT	ok	ok	conflict	conflict	conflict	conflict	conflict
STANDARD	ok	conflict	ok	conflict	conflict	conflict	conflict
EXAMPLE	ok	conflict	conflict	ok	conflict	conflict	conflict
ICS	ok	conflict	conflict	conflict	ok	conflict	conflict
custom1	ok	conflict	conflict	conflict	conflict	ok	conflict
custom2	ok	conflict	conflict	conflict	conflict	conflict	ok

Table 3.1: Rules for categories of sub packages

	target package category (if category is empty, then the parent category applies)						
category of package that contains reference source element (if category is empty, then the parent category applies)	empty	BLUEPRINT	STANDARD	EXAMPLE	ICS	cus-tom1	cus-tom2
empty	ok	ok	ok	ok	ok	ok	ok
BLUEPRINT	ok	ok	ok	conflict	ok	conflict	conflict
STANDARD	ok	conflict	ok	conflict	conflict	conflict	conflict
EXAMPLE	ok	ok	ok	ok	ok	conflict	conflict
ICS	ok	conflict	ok	conflict [2]	ok	conflict	conflict
custom1	ok	ok	ok	ok	ok	ok	ok
custom2	ok	ok	ok	ok	ok	ok	ok

Table 3.2: Rules for references between elements in packages with specific categories

It is possible to maintain a reference from a blueprint to the “actual” objects which were derived from this blueprint. The meta-class `BlueprintMappingSet` can be used for this. Particular compatibility rules may be applicable and defined in the appropriate templates. For more details refer to [2].

Class	**BlueprintMappingSet**			
Package	M2::AUTOSARTemplates::CommonStructure::StandardizationTemplate::BlueprintMapping			
Note	This represents a container of mappings between "actual" model elements and the "blueprint" that has been taken for their creation. **Tags:**atp.recommendedPackage=BlueprintMappingSets			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
blueprintMap	AtpBlueprintMapping	*	aggr	This represents a particular blueprint map in the set.

Table 3.3: BlueprintMappingSet

[2] see [constr_2573] for details

3.2 The role of ARPackage, ARElement and Identifiable et. al.

The AUTOSAR meta-model uses some abstract classes which represent various abilities with respect of model organization. A synopsis of the same is provided in Figure 3.2.

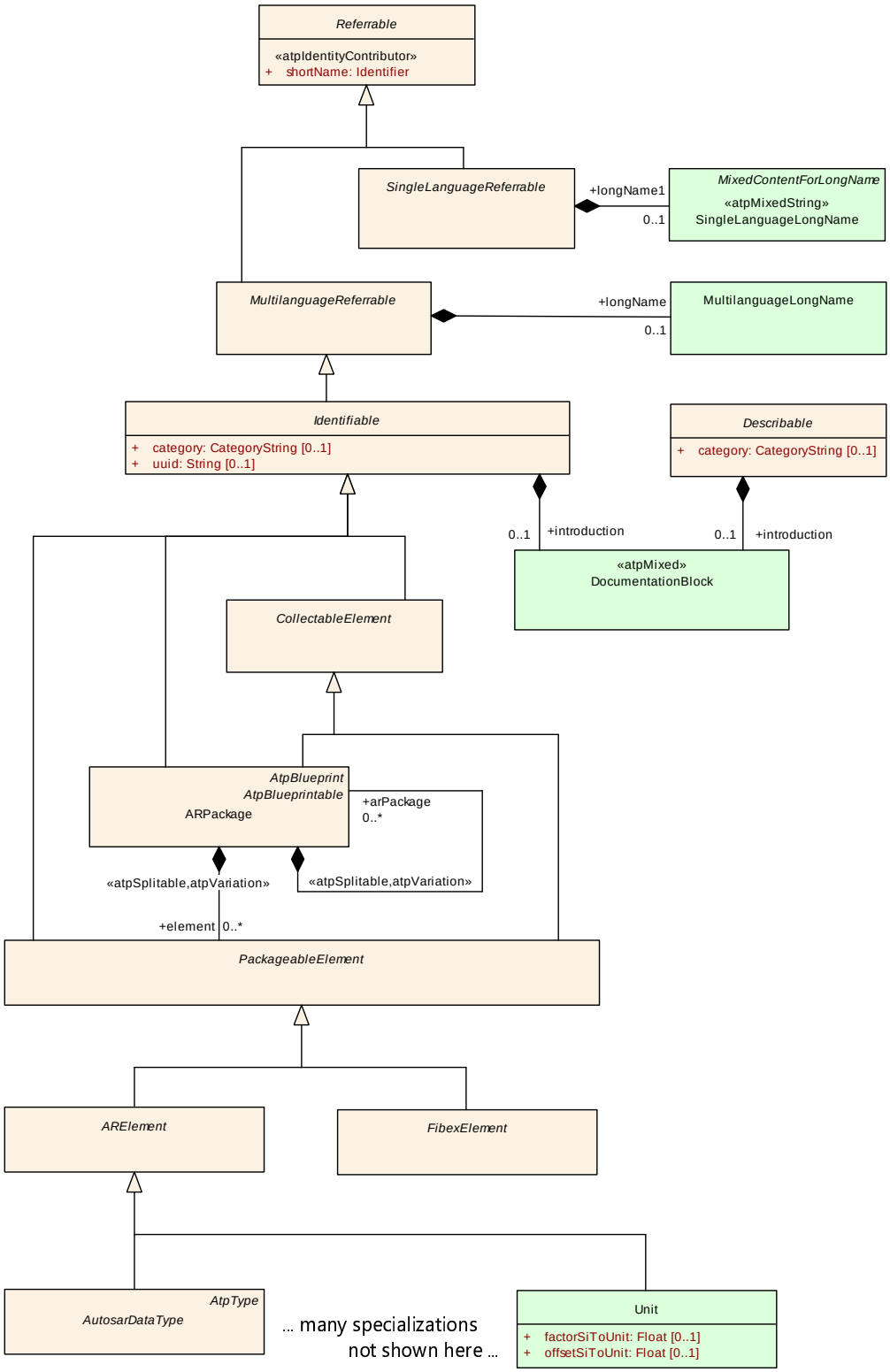


Figure 3.2: Synopsis of model organization classes

About Referrable:

- Referrable is an abstract meta-class which has a shortName acting as a target for references. Referrable is mainly used for reference targets with a small footprint such as Sdg. Referrable is specialized in SingleLanguageReferrable/MultilanguageReferrables.
- Specialization s of SingleLanguageReferrable are applicable as elements which are embedded in a text which is already part of a multilingual object (so called inline elements as described in Chapter 9.2.8).
- Specializations of MultilanguageReferrables apply to multilingual objects which should have a relatively small footprint such as DefList, Traceable.

About Describable:

- Describable is an abstract meta-class which represents the ability to provide a description but not being Referrable or Identifiable. It is mentioned here for completeness.

About Identifiable:

- Identifiable is an abstract class which inherits from MultilanguageReferrable. This is used to identify the essential objects in an AUTOSAR model. Related to Referrable it provides further means to identify the element such as desc. Obviously anything which can be identified shall also be referable. Therefore, Identifiable is a specialization of Referrable.
- Nested Identifiables establish a **hierarchical name space**.

  Note that if Referrable would contain further Referrables it would **not** be a hierarchical name space because name spaces are established by Identifiable only[3].

  See chapter 4.3.1 for more details.
- arPackage in AUTOSAR is the top-most Identifiable in an AUTOSAR model. This top-level ARPackage is the root of the name space hierarchy.
- The shortName of an Identifiable (more precisely of an Referrable) shall be (case insensitively) unique in the containing Identifiable. Examples are
  - shortName of an ARElement (e.g. ApplicationSwComponentType) shall be unique within ARPackage (which is Identifiable). But there might be other ApplicationSwComponentTypes with the same shortName in a **different** ARPackage.
  - The shortName of an PortPrototype shall be unique within an ApplicationSwComponentType (which is Identifiable). But there might be PortPrototypes of the same name in other ApplicationSwComponentTypes.

[3]Nevertheless such a nesting of Referrables does not occur in the AUTOSAR meta-model.

About CollectableElement:

- CollectableElement represents the ability to be part of a collection (in particular to be referenced by a collection). This meta-class does not introduce further attributes nor further possible aggregations.
- Even if CollectableElement is similar to ARElement it is not the same because it handles an entirely different aspect.

About ARPackage:

- ARPackage can be nested: ARPackage can contain ARPackage in the role arPackage.
- Thereby ARPackage consists of nested branches (ARPackages) and leaves (subclasses of PackageableElement, in particular subclasses of ArElement).
- ARPackage is a specialization of Identifiable. Otherwise it would not contribute to the name space hierarchy.
- PackageableElement is an abstract class which represents the ability that the objects can be defined stand-alone. These objects do not require a context. Such objects are sometimes called "first class citizens".
- PackageableElement (ARElement FibexElement) cannot contain further PackageableElements (ARElements resp. FibexElement).

About ARElement and FibexElement:

- ARElement is an abstract class which contributes to an AUTOSAR model in general.
- FibexElement is an abstract class which represents the ability that the elements contribute particularly to the description of communication and topology of a system.
- ARElement and FibexElement is Identifiable. So the derived elements also have a shortName.
- ARElement FibexElement may contain further elements derived from Identifiable.

4 General Template Classes

The nature of the general template classes given below is similar to the standard library of a compiler: a set of predefined structures and elements to be used in an AUTOSAR template model.

4.1 ARObject - Common Attributes for all Classes

[TPS_GST_00091] **ARObject** ⌈ARObject is one meta-class which is inherited by all other meta-classes.⌋ *()*

The related pattern is shown in Figure 6.9.

see also ARObject

4.2 Packages in AUTOSAR

AUTOSAR M1 models can be organized as a number of packages, represented by class ARPackage. It allows allows to put together the model elements, e.g. in form of an OEM or project specific package containing entities like a windshield wiper software component.

The self aggregation (role arPackage) shows that packages may in fact contain other packages. Besides those, a package may contain an arbitrary number of elements, represented by the abstract class ARElement. Such an AUTOSAR element is an entity for which it makes sense to be defined in its own semantic context (stand-alone). An example for such an ARElement is the definition of a reusable software component type. On the other hand a parameter of an operation does not make sense to be defined stand-alone since its semantics is defined within and therefore highly dependent on the enclosing context: the operation.

[TPS_GST_00092] The purpose of a ARPackage ⌈The purpose of ARPackage is to

- organize AUTOSAR M1 models and establish a name space for the elements in the package (as all Identifiables do)
- define the basis for relative references

⌋ *()*

For details about relative references, see chapter 6.3.2.2.

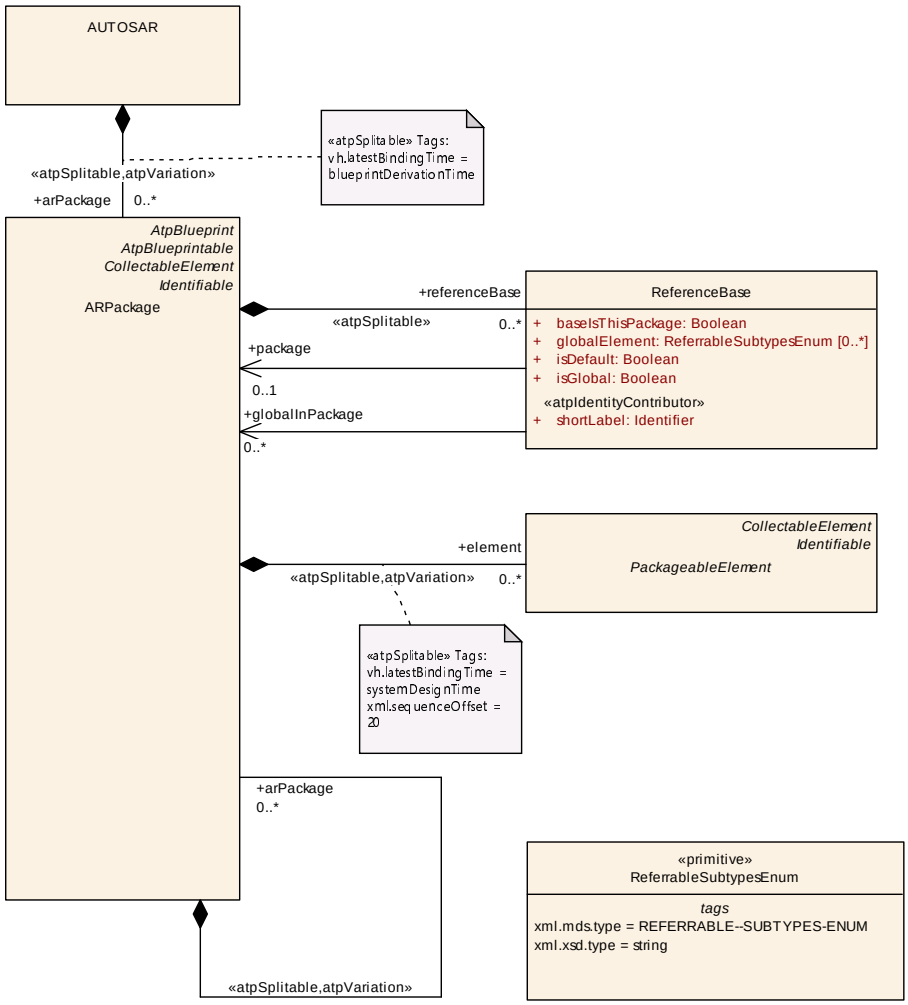


Figure 4.1: AUTOSAR package

Note that the aggregation of `AUElement` in `ARPackage` is subject to variation. The main use-case for this is to specify component alternatives with different interfaces within a product line architecture.

[constr_2537] Variation of `PackageableElement` is limited to components resp. modules ⌈Variation of `AUElement` in `ARPackage` shall be applied only to elements on a kind of component level. In particular this is `BswModuleDescription`, `Documentation`, `Implementation`, `SwComponentType`, `TimingExtension`. This constraint only applies if the `PackageableElement` is not a blueprint.⌋*()*

[constr_2509] Uniqueness of `ReferenceBase.shortLabel` in the scope of an `ARPackage` ⌈The `shortLabel` of any given `ReferenceBase` defined in the scope of an `ARPackage` shall be unique within the scope of the enclosing `ARPackage`.⌋*()*

Note that each level of the package hierarchy may have an independent set of `ReferenceBase`s.

[constr_2510] only one default `ReferenceBase` ⌈Only one `ReferenceBase` per level can be marked as default (`default`="true").⌋*()*

[constr_2574] `globalInPackage` for global elements only ⌈`ReferenceBase.globalInPackage` is allowed only if `isGlobal` is set to true.⌋*()*

[constr_2538] Global reference is limited to certain elements ⌈The ability to perform a global reference is limited to `Chapter`, `Topic1`, `Caption`, `Traceable`, `XrefTarget`, `Std`, `Xdoc`, `Xfile`⌋*()*

Class	**ARPackage**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::ARPackage			
Note	AUTOSAR package, allowing to create top level packages to structure the contained AUTOSAR Elements. ARPackages are open sets. This means that in a file based description system multiple files can be used to partially describe the contents of a package. This is an extended version of MSR's SW-SYSTEM.			
Base	*ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
arPackage	ARPackage	*	aggr	This represents a sub package within an ARPackage, thus allowing for an unlimited package hierarchy. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=arPackage.shortName, arPackage.variationPoint.shortLabel vh.latestBindingTime=blueprintDerivationTime xml.sequenceOffset=30
element	PackageableElement	*	aggr	Elements that are part of this package **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=element.shortName, element.definition, element.variationPoint.shortLabel vh.latestBindingTime=systemDesignTime xml.sequenceOffset=20
referenceBase	ReferenceBase	*	aggr	This denotes the reference bases for the package. This is the basis for all relative references within the package. The base needs to be selected according to the base attribute within the references. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=referenceBase.shortLabel xml.sequenceOffset=10

Table 4.1: ARPackage

Class	***PackageableElement*** (abstract)			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::ARPackage			
Note	This meta-class specifies the ability to be a member of an AUTOSAR package.			
Base	*ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	*ARElement*, EnumerationMappingTable, *FibexElement*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table 4.2: PackageableElement

Class	***ARElement*** (abstract)			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::ARPackage			
Note	An element that can be defined stand-alone, i.e. without being part of another element (except for packages of course).			
Base	*ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Subclasses	AclObjectSet, AclOperation, AclPermission, AclRole, AliasNameSet, ApplicationPartition, *AutosarDataType*, *BaseType*, BlueprintMappingSet, BswEntryRelationshipSet, BswModuleDescription, BswModuleEntry, BuildActionManifest, CalibrationParameterValueSet, ClientIdDefinitionSet, ClientServerInterfaceToBswModuleEntryBlueprintMapping, Collection, CompuMethod, ConsistencyNeedsBlueprintSet, ConstantSpecification, ConstantSpecificationMappingSet, CpSoftwareCluster, CpSoftwareClusterBinaryManifestDescriptor, CpSoftwareClusterMappingSet, CpSoftwareClusterResourcePool, CryptoEllipticCurveProps, CryptoServiceCertificate, CryptoServiceKey, CryptoServicePrimitive, CryptoServiceQueue, CryptoSignatureScheme, DataConstr, DataExchangePoint, DataTransformationSet, DataTypeMappingSet, *DiagnosticCommonElement*, DiagnosticConnection, DiagnosticContributionSet, DltContext, DltEcu, Documentation, E2EProfileCompatibilityProps, EcucDefinitionCollection, EcucDestinationUriDefSet, EcucModuleConfigurationValues, EcucModuleDef, EcucValueCollection, EndToEndProtectionSet, EthIpProps, EthTcpIpIcmpProps, EthTcpIpProps, EvaluatedVariantSet, FMFeature, FMFeatureMap, FMFeatureModel, FMFeatureSelectionSet, FlatMap, GeneralPurposeConnection, HwCategory, HwElement, HwType, IPSecConfigProps, IPv6ExtHeaderFilterSet, *IdsCommonElement*, IdsDesign, *Implementation*, InterpolationRoutineMappingSet, J1939ControllerApplication, KeywordSet, LifeCycleInfoSet, LifeCycleStateDefinitionGroup, LogAndTraceMessageCollectionSet, McFunction, McGroup, ModeDeclarationGroup, ModeDeclarationMappingSet, OsTaskProxy, PhysicalDimension, PhysicalDimensionMappingSet, *PortInterface*, PortInterfaceMappingSet, PortPrototypeBlueprint, PostBuildVariantCriterion, PostBuildVariantCriterionValueSet, PredefinedVariant, RapidPrototypingScenario, SdgDef, SignalServiceTranslationPropsSet, SomeipSdClientEventGroupTimingConfig, SomeipSdClientServiceInstanceConfig, SomeipSdServerEventGroupTimingConfig, SomeipSdServerServiceInstanceConfig, SwAddrMethod, SwAxisType, SwComponentMappingConstraints, *SwComponentType*, SwRecordLayout, SwSystemconst, SwSystemconstantValueSet, SwcBswMapping, System, SystemSignal, SystemSignalGroup, TDCpSoftwareClusterMappingSet, TcpOptionFilterSet, *TimingExtension*, TlsConnectionGroup, TlvDataIdDefinitionSet, TransformationPropsSet, Unit, UnitGroup, ViewMapSet			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table 4.3: ARElement

Class	**ReferenceBase**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::ARPackage			
Note	This meta-class establishes a basis for relative references. Reference bases are identified by the short Label which shall be unique in the current package.			
Base	*ARObject*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
baseIsThisPackage	Boolean	1	attr	This indicates that this base is established by the current package. In this case the association "package" can be derived as the qualified shortName of the enclosing package. If the value of baseIsThisPackage is set to true then one of the following shall be true: • target of the association "package" shall be the enclosing package. • association "package" is omitted. **Tags:**xml.sequenceOffset=28
globalElement	ReferrableSubtypes Enum	*	attr	This attribute represents a meta-class for which the global referencing is supported via this reference base. **Tags:**xml.sequenceOffset=29

△

Class	**ReferenceBase**			
globalIn Package	ARPackage	*	ref	This represents the ability to express that global elements live in various packages which do not have a common ancestor package. Packages mentioned by Reference Base.globalInPackage are used in addition to the one in ReferenceBase.package. **Tags:**xml.sequenceOffset=28
isDefault	Boolean	1	attr	This attribute denotes if the current ReferenceBase is the default. Note that there can only be one default reference base within a package. **Tags:**xml.sequenceOffset=20
isGlobal	Boolean	1	attr	This indicates that the target of the applicable reference can be resolved via the non-qualified shortName. This requires that the shortName of the target is unique within the package referenced in the reference base. The default is false. Note that the reference base also maintains a list of elements which may be referenced using a "global Reference". **Tags:**xml.sequenceOffset=25
package	ARPackage	0..1	ref	This association specifies the basis of all relative references with the base equals shortLabel. This association shall exist unless the value of baseIsThis Package is set to true. **Tags:**xml.sequenceOffset=30
shortLabel	Identifier	1	attr	This is the name of the reference base. By this name, particular references can denote the applicable base. **Stereotypes:** atpIdentityContributor **Tags:**xml.sequenceOffset=10

Table 4.4: ReferenceBase

Primitive	**ReferrableSubtypesEnum**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	This primitive is a proxy for an enum generated by the MMT. It allows to refer to any subclass of Referrable. Due to technical reasons the possible values are not shown in this class table. **Tags:** xml.mds.type=REFERRABLE–SUBTYPES-ENUM xml.xsd.type=string

Table 4.5: ReferrableSubtypesEnum

4.3 Referrable and Identifiable

The main direct abstract base classes of `Identifiable` are `Referrable` and `MultilanguageReferrable`.

[TPS_GST_00096] Main Purpose of `Referrable` ⌈The abstract base class `Referrable` represents the ability to be the target of a reference. Such references can be simply links in a documentation or association contributing to the data model of an AUTOSAR system.⌋*()*

The essential attribute of Referrable is the shortName.

- **[TPS_GST_00097] Purpose of shortName** ⌈The shortName (provided by Referrable) unambiguously identifies the object within the context given by the first ancestor Identifiable.⌋*()*

The essential attribute of MultilanguageReferrable is the longName.

- **[TPS_GST_00099] Purpose of longName** ⌈longName (provided by MultilanguageReferrable) which contains the headline of the object as **single line** of text. The content longName is targeted to human readers. Therefore the long name can be described in different languages.⌋*()*

It is important to understand the big picture of the properties of Identifiable.

[TPS_GST_00095] Main Purpose of Identifiable ⌈The abstract base class Identifiable extends the Referrable with descriptive and administrative features.⌋*()*

- **[TPS_GST_00100] Purpose of desc** ⌈desc contains a brief description of **what** the object is. This is intended to help a human being to identify the object. It is one **single paragraph** also provided in multiple languages.⌋*()*
- **[TPS_GST_00101] Purpose of adminData** ⌈adminData contains administrative information about the object such as version information, language setting etc. This information also has identifying character.⌋*()*
- **[TPS_GST_00365] Purpose of uuid** ⌈uuid may be used in a user's tool environment to uniquely identify AUTOSAR elements, for example, when merging AUTOSAR elements into a company-specific data base. The uuid has no semantic meaning for an AUTOSAR model and there is no requirement for AUTOSAR tools to manage the uuid.⌋*()*

In addition to these identifying properties, Identifiable carries further properties which are of specifying purpose. For simplification of the model, these properties are defined in Identifiable since they apply to mostly all identifiable objects representing the technical structure of an AUTOSAR system:

- **[TPS_GST_00102] Purpose of category** ⌈category contains a keyword expressing a specific use case of the Identifiable. To some extent category can be compared with stereotypes in UML. The applicable categories are specified in the constraints of the objects in question.⌋*()*

  [TPS_GST_00016] Values for category ⌈In general it is allowed to extend the categories defined in the template specifications by user-defined values. In this case the user is responsible to avoid any conflict with existing or future defined AUTOSAR categories. This can be achieved for example by using an appropriate prefix.

  Anyhow the constraints of specific elements may restrict the category to exactly the defined ones and in this case an extension is not allowed.⌋*()*

- **[TPS_GST_00103] Purpose of `introduction`** ⌈`introduction` contains introductory documentation about **how** the identified object is built or maybe used.⌋ *()*
- **[TPS_GST_00104] Purpose of `annotation`** ⌈`annotation` contains development annotations⌋ *()*

  For more details on `Annotation` see chapter 4.11).

As an example, these properties for the AUTOSAR project would be:

- shortName: AUTOSAR
- longName: AUTomotive Open Systems ARchitecture
- desc: AUTOSAR is an open and standardized automotive software architecture, jointly developed by automobile manufacturers, suppliers and tool developers.
- introduction:

  AUTOSAR

  - paves the way for innovative electronic systems that further improve performance, safety and environmental friendliness
  - is a strong global partnership that creates one common standard: "Cooperate on standards, compete on implementation"
  - ...

The base class `Identifiable` has further content related attributes, which are shown in figure 4.2:

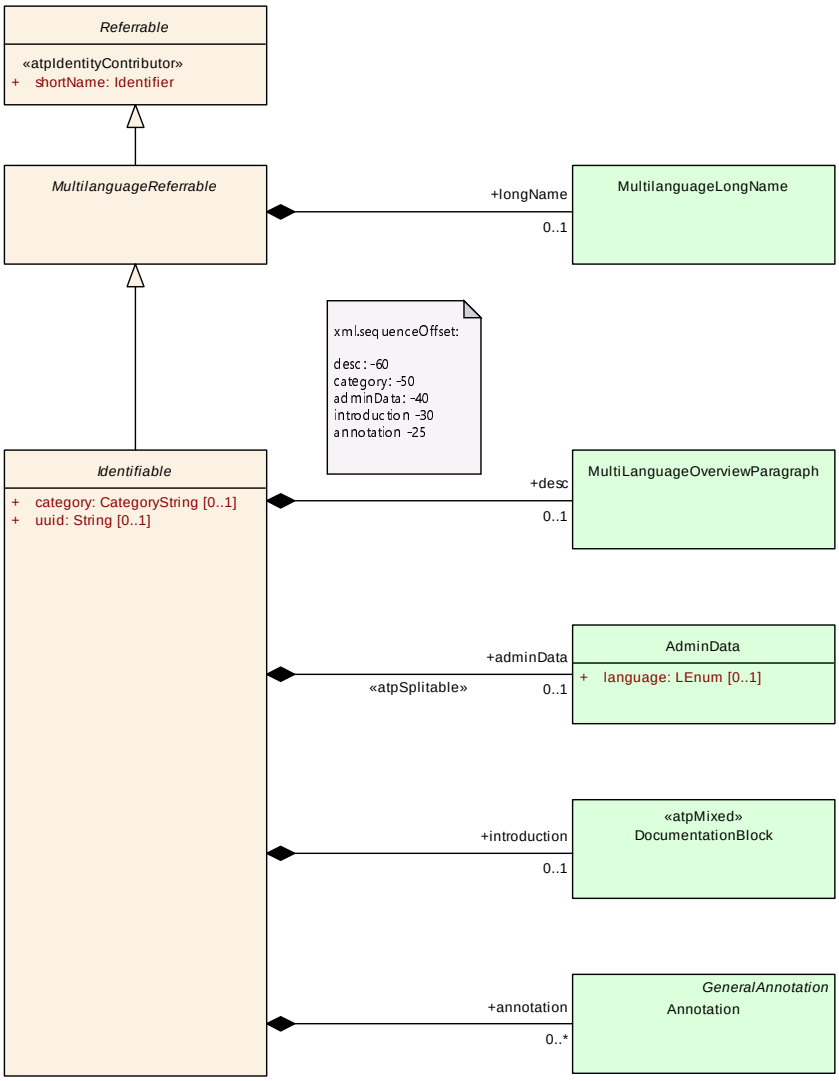


Figure 4.2: Identifiable

The base class `Referrable` and its specializations are shown in figure 4.3:

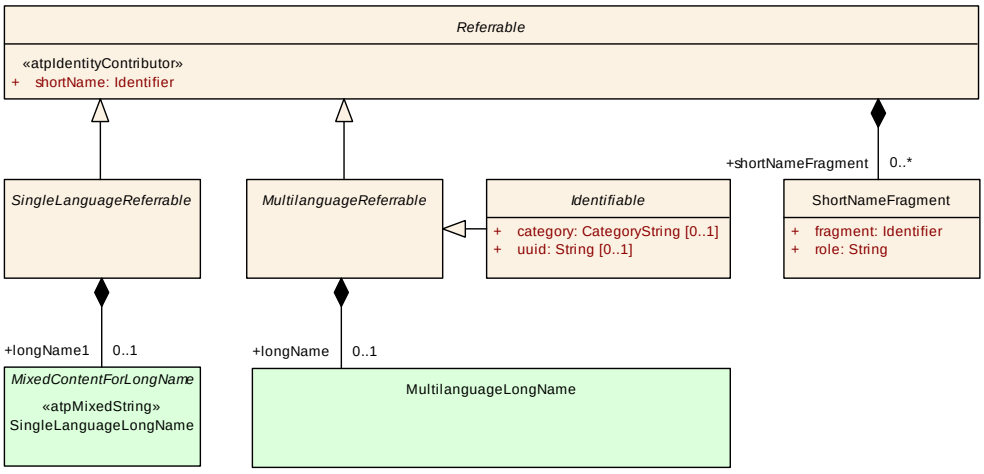


Figure 4.3: Referrable

If a shortName is assembled from different components then it is a good practice to provide the ShortNameFragments as a formal specification of the assembly instruction.

In 4.1 the composed shortName is "A__B". The first ShortNameFragment is "A" and has the role "prefix". The second ShortNameFragment is "B" and has the role "suffix". The terms "prefix" and "suffix" are just examples and no matter of standardization. They shall only illustrate this example. AUTOSAR does not standardized any value of the role attribute.

Listing 4.1: Example for shortNameFragments

```
<SHORT-NAME>A__B</SHORT-NAME>
<SHORT-NAME-FRAGMENTS>
  <SHORT-NAME-FRAGMENT>
    <ROLE>prefix</ROLE>
    <FRAGMENT>A</FRAGMENT>
  </SHORT-NAME-FRAGMENT>
  <SHORT-NAME-FRAGMENT>
    <ROLE>suffix</ROLE>
    <FRAGMENT>B</FRAGMENT>
  </SHORT-NAME-FRAGMENT>
</SHORT-NAME-FRAGMENTS>
```

This procedure allows to compose and to separate the atomic and composed shortNames without losing the original information.

For more details about DocumentationBlock please refer to chapter 9.2.

The attributes are given in the following class tables:

Class	***Identifiable*** (abstract)
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Identifiable
Note	Instances of this class can be referred to by their identifier (within the namespace borders). In addition to this, Identifiables are objects which contribute significantly to the overall structure of an AUTOSAR description. In particular, Identifiables might contain Identifiables.
Base	*ARObject*, *MultilanguageReferrable*, *Referrable*
Subclasses	ARPackage, *AbstractDoIpLogicAddressProps*, *AbstractEvent*, *AbstractImplementationDataTypeElement*, *AbstractSecurityEventFilter*, *AbstractSecurityIdsmInstanceFilter*, *AbstractServiceInstance*, AppOsTask ProxyToEcuTaskProxyMapping, ApplicationEndpoint, ApplicationError, ApplicationPartitionToEcuPartition Mapping, AsynchronousServerCallResultPoint, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *Atp Feature*, AutosarOperationArgumentInstance, AutosarVariableInstance, *BinaryManifestAddressable Object*, BinaryManifestItemDefinition, *BinaryManifestResource*, BinaryManifestResourceDefinition, Block State, BswInternalTriggeringPoint, BswModuleDependency, *BuildActionEntity*, BuildActionEnvironment, CanTpAddress, CanTpChannel, CanTpNode, Chapter, ClassContentConditional, ClientIdDefinition, ClientServerOperation, Code, *CollectableElement*, ComManagementMapping, *CommConnectorPort*, *CommunicationConnector*, *CommunicationController*, Compiler, ConsistencyNeeds, ConsumedEvent Group, CouplingPort, *CouplingPortStructuralElement*, *CpSoftwareClusterResource*, CpSoftwareCluster ResourceToApplicationPartitionMapping, CpSoftwareClusterToEcuInstanceMapping, CpSoftwareCluster ToResourceMapping, *CryptoServiceMapping*, DataPrototypeGroup, DataTransformation, Dependency OnArtifact, *DiagEventDebounceAlgorithm*, DiagnosticConnectedIndicator, DiagnosticDataElement, DiagnosticDebounceAlgorithmProps, DiagnosticFunctionInhibitSource, *DiagnosticRoutineSubfunction*, DltApplication, DltArgument, DltLogChannel, DltMessage, DoIpInterface, DoIpLogicAddress, DoIp RoutingActivation, ECUMapping, *EOCExecutableEntityRefAbstract*, EcuPartition, EcucContainerValue, *EcucDefinitionElement*, EcucDestinationUriDef, EcucEnumerationLiteralDef, EcucQuery, EcucValidation ▽

▽

△

Class	***Identifiable*** (abstract)			
	△ Condition, EndToEndProtection, EthernetWakeupSleepOnDatalineConfig, EventHandler, ExclusiveArea, *ExecutableEntity*, *ExecutionTime*, FMAttributeDef, FMFeatureMapAssertion, FMFeatureMapCondition, FMFeatureMapElement, FMFeatureRelation, FMFeatureRestriction, FMFeatureSelection, FlatInstanceDescriptor, FlexrayArTpNode, FlexrayTpConnectionControl, FlexrayTpNode, FlexrayTpPduPool, *FrameTriggering*, GeneralParameter, GlobalTimeGateway, *GlobalTimeMaster*, *GlobalTimeSlave*, *HeapUsage*, HwAttributeDef, HwAttributeLiteralDef, HwPin, HwPinGroup, IPSecRule, IPv6ExtHeaderFilterList, ISignalToIPduMapping, ISignalTriggering, *IdentCaption*, InternalTriggeringPoint, J1939SharedAddressCluster, J1939TpNode, Keyword, LifeCycleState, LinScheduleTable, LinTpNode, Linker, MacMulticastGroup, McDataInstance, MemorySection, ModeDeclaration, ModeDeclarationMapping, ModeSwitchPoint, NetworkEndpoint, *NmCluster*, NmEcu, *NmNode*, NvBlockDescriptor, *PackageableElement*, ParameterAccess, PduActivationRoutingGroup, PduToFrameMapping, PduTriggering, PerInstanceMemory, *PhysicalChannel*, PortElementToCommunicationResourceMapping, PortGroup, *PortInterfaceMapping*, PossibleErrorReaction, ResourceConsumption, RootSwCompositionPrototype, RptComponent, RptContainer, RptExecutableEntity, RptExecutableEntityEvent, RptExecutionContext, RptProfile, RptServicePoint, RteEventInCompositionSeparation, RteEventInCompositionToOsTaskProxyMapping, RteEventInSystemSeparation, RteEventInSystemToOsTaskProxyMapping, RunnableEntityGroup, *SdgAttribute*, SdgClass, SecureCommunicationAuthenticationProps, SecureCommunicationFreshnessProps, SecurityEventContextProps, *ServerCallPoint*, *ServiceNeeds*, SignalServiceTranslationElementProps, SignalServiceTranslationEventProps, SignalServiceTranslationProps, SocketAddress, SomeipTpChannel, *SpecElementReference*, *StackUsage*, StaticSocketConnection, StructuredReq, SwGenericAxisParamType, SwServiceArg, SwcServiceDependency, SwcToApplicationPartitionMapping, SwcToEcuMapping, SwcToImplMapping, SystemMapping, TDCpSoftwareClusterMapping, TDCpSoftwareClusterResourceMapping, TcpOptionFilterList, TimingCondition, *TimingConstraint*, *TimingDescription*, TimingExtensionResource, TimingModeInstance, TlsCryptoCipherSuite, TlsCryptoCipherSuiteProps, Topic1, TpAddress, TraceableTable, TraceableText, *TracedFailure*, *TransformationProps*, TransformationTechnology, Trigger, VariableAccess, VariationPointProxy, ViewMap, VlanConfig, WaitPoint			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
adminData	AdminData	0..1	aggr	This represents the administrative data for the identifiable object. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=adminData xml.sequenceOffset=-40
annotation	Annotation	*	aggr	Possibility to provide additional notes while defining a model element (e.g. the ECU Configuration Parameter Values). These are not intended as documentation but are mere design notes. **Tags:**xml.sequenceOffset=-25
category	CategoryString	0..1	attr	The category is a keyword that specializes the semantics of the Identifiable. It affects the expected existence of attributes and the applicability of constraints. **Tags:**xml.sequenceOffset=-50
desc	MultiLanguageOverviewParagraph	0..1	aggr	This represents a general but brief (one paragraph) description what the object in question is about. It is only one paragraph! Desc is intended to be collected into overview tables. This property helps a human reader to identify the object in question. More elaborate documentation, (in particular how the object is built or used) should go to "introduction". **Tags:**xml.sequenceOffset=-60
introduction	DocumentationBlock	0..1	aggr	This represents more information about how the object in question is built or is used. Therefore it is a DocumentationBlock. **Tags:**xml.sequenceOffset=-30

▽

△

Class	***Identifiable*** (abstract)			
uuid	String	0..1	attr	The purpose of this attribute is to provide a globally unique identifier for an instance of a meta-class. The values of this attribute should be globally unique strings prefixed by the type of identifier. For example, to include a DCE UUID as defined by The Open Group, the UUID would be preceded by "DCE:". The values of this attribute may be used to support merging of different AUTOSAR models. The form of the UUID (Universally Unique Identifier) is taken from a standard defined by the Open Group (was Open Software Foundation). This standard is widely used, including by Microsoft for COM (GUIDs) and by many companies for DCE, which is based on CORBA. The method for generating these 128-bit IDs is published in the standard and the effectiveness and uniqueness of the IDs is not in practice disputed. If the id namespace is omitted, DCE is assumed. An example is "DCE:2fac1234-31f8-11b4-a222-08002b34c003". The uuid attribute has no semantic meaning for an AUTOSAR model and there is no requirement for AUTOSAR tools to manage the timestamp. **Tags:**xml.attribute=true

Table 4.6: Identifiable

Primitive	**Identifier**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes			
Note	An Identifier is a string with a number of constraints on its appearance, satisfying the requirements typical programming languages define for their Identifiers. This datatype represents a string, that can be used as a c-Identifier. It shall start with a letter, may consist of letters, digits and underscores. **Tags:** xml.xsd.customType=IDENTIFIER xml.xsd.maxLength=128 xml.xsd.pattern=[a-zA-Z][a-zA-Z0-9_]* xml.xsd.type=string			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
blueprintValue	String	0..1	attr	This represents a description that documents how the value shall be defined when deriving objects from the blueprint. **Tags:** atp.Status=draft xml.attribute=true
namePattern	String	0..1	attr	This attribute represents a pattern which shall be used to define the value of the identifier if the identifier in question is part of a blueprint. For more details refer to TPS_StandardizationTemplate. **Tags:**xml.attribute=true

Table 4.7: Identifier

Class	**MultilanguageLongName**
Package	M2::MSR::Documentation::TextModel::MultilanguageData
Note	This meta-class represents the ability to specify a long name which acts in the role of a headline. It is intended for human readers. Per language it should be around max 80 characters.

▽

△

Class	**MultilanguageLongName**			
Base	*ARObject*			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
l4	LLongName	1..*	aggr	This is the long name in one particular language. **Tags:** xml.roleElement=true xml.roleWrapperElement=false xml.sequenceOffset=20 xml.typeElement=false xml.typeWrapperElement=false

Table 4.8: MultilanguageLongName

Class	<<atpMixedString>> **SingleLanguageLongName**			
Package	M2::MSR::Documentation::TextModel::SingleLanguageData			
Note	SingleLanguageLongName			
Base	*ARObject*, *MixedContentForLongName*			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
–	–	–	–	–

Table 4.9: SingleLanguageLongName

Class	<<atpMixedString>> **LLongName**			
Package	M2::MSR::Documentation::TextModel::LanguageDataModel			
Note	MixedContentForLongNames in one particular language. The language is denoted in the attribute l.			
Base	*ARObject*, *LanguageSpecific*, *MixedContentForLongName*			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
blueprintValue	String	0..1	attr	This represents a description that documents how the value shall be defined when deriving objects from the blueprint. **Tags:** atp.Status=draft xml.attribute=true

Table 4.10: LLongName

Class	<<atpMixedString>> ***MixedContentForLongName*** (abstract)			
Package	M2::MSR::Documentation::TextModel::InlineTextModel			
Note	This is the model for titles and long-names. It allows some emphasis and index entries but no reference target (which is provided by the identifiable in question). It is intended that the content model can also be rendered as plain text. The abstract class can be used for single language as well as for multi language elements.			
Base	*ARObject*			
Subclasses	LLongName, SingleLanguageLongName			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
e	EmphasisText	1	aggr	This is emphasized text **Tags:**xml.sequenceOffset=40
ie	IndexEntry	1	aggr	This is an index entry. **Tags:**xml.sequenceOffset=70

▽

△

Class	<<atpMixedString>> ***MixedContentForLongName*** (abstract)			
sub	Superscript	1	attr	This is subscript text. **Tags:**xml.sequenceOffset=60
sup	Superscript	1	attr	This is superscript text. **Tags:**xml.sequenceOffset=50
tt	Tt	1	aggr	This is a technical term. **Tags:**xml.sequenceOffset=30

Table 4.11: MixedContentForLongName

see also `MultiLanguageOverviewParagraph` see also `LOverviewParagraph`

Class	***Referrable*** (abstract)			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Identifiable			
Note	Instances of this class can be referred to by their identifier (while adhering to namespace borders).			
Base	*ARObject*			
Subclasses	*AtpDefinition*, BswDistinguishedPartition, *BswModuleCallPoint*, BswModuleClientServerEntry, BswVariableAccess, CouplingPortTrafficClassAssignment, *DiagnosticEnvModeElement*, EthernetPriorityRegeneration, ExclusiveAreaNestingOrder, *HwDescriptionEntity*, *ImplementationProps*, LinSlaveConfigIdent, ModeTransition, *MultilanguageReferrable*, PncMappingIdent, *SingleLanguageReferrable*, SoConIPduIdentifier, SocketConnectionBundle, TimeSyncServerConfiguration, TpConnectionIdent			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
shortName	Identifier	1	attr	This specifies an identifying shortName for the object. It needs to be unique within its context and is intended for humans but even more for technical reference. **Stereotypes:** atpIdentityContributor **Tags:** xml.enforceMinMultiplicity=true xml.sequenceOffset=-100
shortNameFragment	ShortNameFragment	*	aggr	This specifies how the Referrable.shortName is composed of several shortNameFragments. **Tags:**xml.sequenceOffset=-90

Table 4.12: Referrable

Class	***MultilanguageReferrable*** (abstract)			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Identifiable			
Note	Instances of this class can be referred to by their identifier (while adhering to namespace borders). They also may have a longName. But they are not considered to contribute substantially to the overall structure of an AUTOSAR description. In particular it does not contain other Referrables.			
Base	*ARObject*, *Referrable*			
Subclasses	Caption, DefItem, DocumentationContext, *Identifiable*, SdgCaption, *TraceReferrable*, *Traceable*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
longName	MultilanguageLongName	0..1	aggr	This specifies the long name of the object. Long name is targeted to human readers and acts like a headline.

Table 4.13: MultilanguageReferrable

Class	**SingleLanguageReferrable** (abstract)			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Identifiable			
Note	Instances of this class can be referred to by their identifier (while adhering to namespace borders). They also may have a longName but in one language only. Specializations of this class only occur as inline elements in one particular language. Therefore they aggregate But they are not considered to contribute substantially to the overall structure of an AUTOSAR description. In particular it does not contain other Referrables.			
Base	*ARObject*, *Referrable*			
Subclasses	Std, Xdoc, Xfile, XrefTarget			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
longName1	SingleLanguageLongName	0..1	aggr	This specifies the long name of the object. The role is longName1 for compatibiilty to ASAM FSX

Table 4.14: SingleLanguageReferrable

Class	**ShortNameFragment**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Identifiable			
Note	This class describes how the Referrable.shortName is composed of several shortNameFragments.			
Base	*ARObject*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
fragment	Identifier	1	attr	This specifies a single shortName (fragment) which is part of the composed shortName. **Tags:**xml.sequenceOffset=20
role	String	1	attr	This specifies the role of fragment to define e.g. the order of the fragments. **Tags:**xml.sequenceOffset=10

Table 4.15: ShortNameFragment

4.3.1 Name spaces and uniqueness of `shortName`

The `shortName` in `Referrable` contributes to the identification of specializations of `Referrable`, in particular `Identifiable`.

[TPS_GST_00018] Name Space established by `Identifiable` ⌈`Identifiable` thereby establishes a name space in which the contained `Identifiable`s respectively `Referrable`s can unambiguously identified by their `shortName`. In some cases there are meta-classes derived from `Identifiable` that aggregate meta-classes that are not derived from `Identifiable` but they aggregate in turn a meta-class that **is** derived from `Referrable`. In other words, there are cases where `Identifiable`s are aggregated only indirectly.

Anyhow the rules for interpreting the established name space apply in this case as well. In particular, for a given instance of `Referrable`, the name space is established by the nearest **ancestor** (not only parent) element which is an `Identifiable`. This expressed by the phrase "within a given `Identifiable`" in [constr_2508].⌋*()*

[TPS_GST_00019] `Referrable` does not establish Name Space ⌈Note that `Referrable` does **not** establish a name space. Beyond `Identifiable` there is no use case where `Referrable` aggregates another `Referrable`. But if it would, the `shortName` of the aggregated one would need to be unique within the nearest ancestor `Identifiable` (not only the nearest ancestor `Referrable`).⌋*()*

Note that it is possible to extend the name space for particular elements in order to support global references. See [constr_2538] in Chapter 4.2 for more details.

[TPS_GST_00021] Case Sensitivity of `shortName` ⌈`shortName` shall be

- basically interpreted as case sensitive (see [TPS_GST_00020])
- but checked for uniqueness as not case sensitive ([constr_2508])

⌋*()*

[constr_2508] The `shortName` shall be unique in its name space ⌈The content of `shortName` needs to be unique (case insensitive) within a the parent given name space.

Note that the check for uniqueness of `shortName` shall be performed case insensitively. This supports the good practice that names should not differ in upper / lower case only which would cause a lot of confusion.

The term “case insensitive” indicates that the characters in the sets

```
{a b c d e f g h i j k l m n o p q r s t u v w x y z}
{A B C D E F G H I J K L M N O P Q R S T U V W X Y Z}
```

are respectively considered to be the same. In other words case-insensitive check for uniqueness of `shortName`s results in the fact that e.g. elements with `shortName` "X" and "x" are considered the same and shall not exist in the same name space.⌋*()*

The listing 4.2 illustrates the valid case: SR_1 and SR_2 in the name space SwComp.

Listing 4.2: Example for valid Name Space Use Case

```
<AR-PACKAGE>
  <SHORT-NAME>SwComp</SHORT-NAME>
  <ELEMENTS>
    <APPLICATION-SW-COMPONENT-TYPE>
      <SHORT-NAME>SwComp1</SHORT-NAME>
      <PORTS>
        <P-PORT-PROTOTYPE>
          <SHORT-NAME>PPortA</SHORT-NAME>
          <PROVIDED-INTERFACE-TREF DEST="SENDER-RECEIVER-INTERFACE">/
              SwComp/SR_1</PROVIDED-INTERFACE-TREF>
        </P-PORT-PROTOTYPE>
        <R-PORT-PROTOTYPE>
          <SHORT-NAME>RPortB</SHORT-NAME>
          <REQUIRED-INTERFACE-TREF DEST="SENDER-RECEIVER-INTERFACE">/
              SwComp/SR_2</REQUIRED-INTERFACE-TREF>
        </R-PORT-PROTOTYPE>
      </PORTS>
```

```
      </APPLICATION-SW-COMPONENT-TYPE>
      <SENDER-RECEIVER-INTERFACE>
        <SHORT-NAME>SR_1</SHORT-NAME>
        <IS-SERVICE>false</IS-SERVICE>
      </SENDER-RECEIVER-INTERFACE>
      <SENDER-RECEIVER-INTERFACE>
        <SHORT-NAME>SR_2</SHORT-NAME>
        <IS-SERVICE>false</IS-SERVICE>
      </SENDER-RECEIVER-INTERFACE>
    </ELEMENTS>
  </AR-PACKAGE>
```

The listing 4.3 illustrates the invalid case: Although the references to SR_1 and Sr_1 are unique according to [TPS_GST_00167], the short names in the name space SwComp violate the [constr_2508].

Listing 4.3: Example for invalid Name Space Use Case

```
<AR-PACKAGE>
 <SHORT-NAME>SwComp</SHORT-NAME>
 <ELEMENTS>
   <APPLICATION-SW-COMPONENT-TYPE>
     <SHORT-NAME>SwComp1</SHORT-NAME>
     <PORTS>
       <P-PORT-PROTOTYPE>
         <SHORT-NAME>PPortA</SHORT-NAME>
         <PROVIDED-INTERFACE-TREF DEST="SENDER-RECEIVER-INTERFACE">/
             SwComp/SR_1</PROVIDED-INTERFACE-TREF>
       </P-PORT-PROTOTYPE>
       <R-PORT-PROTOTYPE>
         <SHORT-NAME>RPortB</SHORT-NAME>
         <REQUIRED-INTERFACE-TREF DEST="SENDER-RECEIVER-INTERFACE">/
             SwComp/Sr_1</REQUIRED-INTERFACE-TREF>
       </R-PORT-PROTOTYPE>
     </PORTS>
   </APPLICATION-SW-COMPONENT-TYPE>
   <SENDER-RECEIVER-INTERFACE>
     <SHORT-NAME>SR_1</SHORT-NAME>
     <IS-SERVICE>false</IS-SERVICE>
   </SENDER-RECEIVER-INTERFACE>
   <SENDER-RECEIVER-INTERFACE>
     <SHORT-NAME>Sr_1</SHORT-NAME>
     <IS-SERVICE>false</IS-SERVICE>
   </SENDER-RECEIVER-INTERFACE>
 </ELEMENTS>
</AR-PACKAGE>
```

4.4 Administrative Data

AdminData is used to denote administrative meta data to objects. It covers various aspects:

- **[TPS_GST_00105] Control of the Document Language by `AdminData`** ⌈control of the languages in the document.⌋*()* (see chapter 3 and chapter 9.6).

  [constr_2572] Unique Control of Document Languages ⌈The settings for multiple languages are specified in the top-Level `AdminData` only⌋*()*

- **[TPS_GST_00106] Version Management** ⌈Version and change management is performed using `DocRevision`. Note that entry for the current revision is the first one. Information about previous revisions can be provided as change history.⌋*()*

  [TPS_GST_00107] Merge Operations in Version Management ⌈`DocRevision` allows to specify the revision label of its predecessors in order to document merge operations.⌋*()*

  [TPS_GST_00108] Special Information in Version Management ⌈`Sdg` allows to denote specific information in addition to the AUTOSAR standardized model. The usage of `Sdg` shall be mutually agreed between the involved parties.⌋*()*

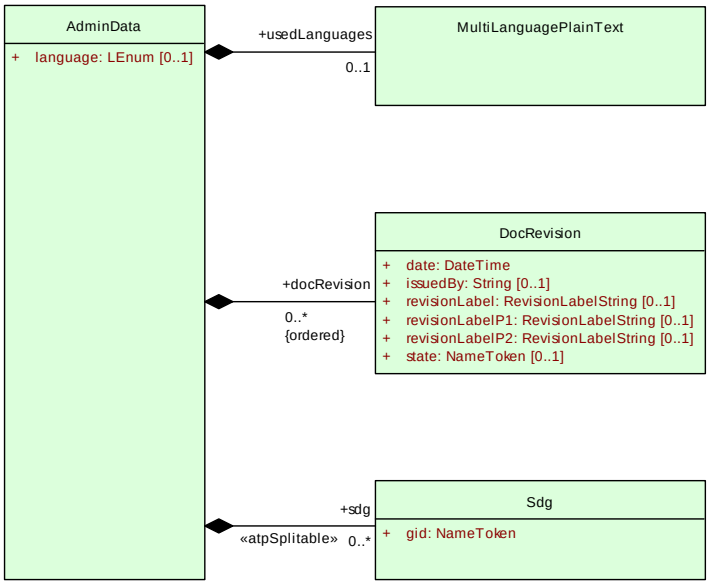


Figure 4.4: AdminData

Class	**AdminData**			
Package	M2::MSR::AsamHdo::AdminData			
Note	AdminData represents the ability to express administrative information for an element. This administration information is to be treated as meta-data such as revision id or state of the file. There are basically four kinds of meta-data • The language and/or used languages. • Revision information covering e.g. revision number, state, release date, changes. Note that this information can be given in general as well as related to a particular company. • Document meta-data specific for a company			
Base	*ARObject*			
Attribute	*Type*	*Mult.*	*Kind*	*Note*

▽

△

Class	**AdminData**			
docRevision (ordered)	DocRevision	*	aggr	This allows to denote information about the current revision of the object. Note that information about previous revisions can also be logged here. The entries shall be sorted descendant by date in order to reflect the history. Therefore the most recent entry representing the current version is denoted first. **Tags:** xml.roleElement=true xml.roleWrapperElement=true xml.sequenceOffset=50 xml.typeElement=false xml.typeWrapperElement=false
language	LEnum	0..1	attr	This attribute specifies the master language of the document or the document fragment. The master language is the one in which the document is maintained and from which the other languages are derived from. In particular in case of inconsistencies, the information in the master language is priority. **Tags:**xml.sequenceOffset=20
sdg	Sdg	*	aggr	This property allows to keep special data which is not represented by the standard model. It can be utilized to keep e.g. tool specific data. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=sdg, sdg.variationPoint.shortLabel xml.roleElement=true xml.roleWrapperElement=true xml.sequenceOffset=60 xml.typeElement=false xml.typeWrapperElement=false
usedLanguages	MultiLanguagePlainText	0..1	aggr	This property specifies the languages which are provided in the document. Therefore it should only be specified in the top level admin data. For each language provided in the document there is one entry in MultilanguagePlain Text. The content of each entry can be used for illustration of the language. The used language itself depends on the language attribute in the entry. **Tags:**xml.sequenceOffset=30

Table 4.16: AdminData

Class	**DocRevision**			
Package	M2::MSR::AsamHdo::AdminData			
Note	This meta-class represents the ability to maintain information which relates to revision management of documents or objects.			
Base	*ARObject*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
date	DateTime	1	attr	This specifies the date and time, when the object in question was released **Tags:**xml.sequenceOffset=80
issuedBy	String	0..1	attr	This is the name of an individual or an organization who issued the current revision of the document or document fragment. **Tags:**xml.sequenceOffset=60

▽

△

Class	**DocRevision**			
modification	Modification	*	aggr	This property represents one particular modification in comparison to its predecessor. **Tags:** xml.roleElement=true xml.roleWrapperElement=true xml.sequenceOffset=100 xml.typeElement=false xml.typeWrapperElement=false
revisionLabel	RevisionLabelString	0..1	attr	This attribute represents the version number of the object. **Tags:**xml.sequenceOffset=20
revisionLabelP1	RevisionLabelString	0..1	attr	This attribute represents the version number of the first predecessor of the object. **Tags:**xml.sequenceOffset=30
revisionLabelP2	RevisionLabelString	0..1	attr	This attribute represents the version number of the second predecessor of the object. This attribute is used if the object is the result of a merge process in which two branches are merged in to one new revision. **Tags:**xml.sequenceOffset=40
state	NameToken	0..1	attr	The attribute state represents the current state of the current file according to the configuration management plan. It is a NameToken until possible states are standardized. **Tags:**xml.sequenceOffset=50

Table 4.17: DocRevision

Class	**Modification**			
Package	M2::MSR::AsamHdo::AdminData			
Note	This meta-class represents the ability to record what has changed in a document in comparison to its predecessor.			
Base	*ARObject*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
change	MultiLanguageOverview Paragraph	1	aggr	This property denotes the one particular change which was performed on the object. **Tags:**xml.sequenceOffset=20
reason	MultiLanguageOverview Paragraph	0..1	aggr	This property represents the rationale for the particular change. **Tags:**xml.sequenceOffset=30

Table 4.18: Modification

4.5 Special Data - Extension Mechanism

4.5.1 Special Data

Special data groups (Sdgs) provide a standardized mechanism to store arbitrary data for which no other element exists of the data model.

[TPS_GST_00223] Use Cases for Sdg ⌈Intended use cases for special data are such as

- Representing proprietary information within an AUTOSAR model without the need for "sidecar" files.
- Representing information to implement workarounds for deficiencies or features missing in the AUTOSAR meta-model.
- Representing use case at dedicated places in the model e.g. BuildAction-IoElement).

Application of Sdg depends on mutual agreement between the involved parties.⌋()

[TPS_GST_00224] Applicable modeling support in Special Data ⌈Basically Sdg allows a generic representation of arbitrary models (similar to "well formed XML") without the need of an explicit meta-model. It provides the following approaches:

- Sdg aggregated outside of SdgContents represents the root container of special data.
- Sdg in the role SdgContents.sdg represents a container for structured information. It can contain an arbitrary mix of sdg, sd and sdx.
- Sd in the role Sdg.sd represents primitive information. In opposite to TagWith-OptionalValue the value is represented implicitly and therefore can not be optional (see [TPS_GST_00358]).
- Sdg.sdx (a reference to Referrable) represents an association.

⌋()

[TPS_GST_00356] Application of Sdg ⌈There is a slight difference in application of Sdg:

- AdminData.sdg is used for pure proprietary purposes
- Dedicated aggregations in other meta-classes for which the purpose is described in the respective template.

⌋()

[TPS_GST_00225] Specifiation of roles in Special Data ⌈The particular role of elements in special data is denoted in the attribute gid of Sd respectively of Sdg. Note that sdx does not have such an attribute. If specific roles need to be denoted, then an extra container sdg needs to be wrapped around the sdx.⌋()

The following hints apply:

- Create a separate sdg per mutual agreement. This may be denoted in gid by the name an organization, or a name indicating a particular semantics.
- Choose distinctive names for gid since Sdg does **not** create name spaces.

- Chose a proper layout of containers (sdg): not one container per sd neither one "mega container" for all sd.
- Do not use sdg to represent information which is already supported by the meta-model.
- Be aware that sd preserves white space.

Example 4.4 illustrates a use case "tracing-info" as a more complete example. This specifies some properties of a document with respect to requirements tracing. It is a simple container with a bunch of attributes.

Listing 4.4: Simple Example for Special Data

```
<SDG GID="TRACING-INFO">
  <SD GID="ITEM-PREFIX">BswM</SD>
  <SD GID="CONTRIBUTES-TO-GENERAL">SRS_BSWGeneral</SD>
  <SD GID="CONTRIBUTES-TO">SRS_ModeManagement</SD>
  <SD GID="INCLUDES">SWS_BSWGeneral</SD>
</SDG>
```

Example 4.5 illustrates a use case "Relation" as a simple example. This example illustrates how the relationship of a requirement and related development objects could be represented using Sdg. The example shows in particular:

- an Sdg with a caption (sdgCaption) which contains shortName and desc
- arbitrary associations represented by sdx
- that the role of the sdx referencing /My/Mypackage is not explicitly specified.
- the semantics of these associations to be expressed by a surrounding sdg
- the arbitrary mix of sdg and sd sdx in SdgContents

Listing 4.5: Example for Special Data with References

```
<SDG GID="EXAMPLE-RELATION">
  <SDG-CAPTION>
    <SHORT-NAME>Rel_90</SHORT-NAME>
    <LONG-NAME>
      <L-4 L="AA">This is the relationship no 90</L-4>
    </LONG-NAME>
    <DESC>
      <L-2 L="AA">This relationship specifies objects related to
        requirement 90.</L-2>
    </DESC>
  </SDG-CAPTION>
  <SDX-REF DEST="AR-PACKAGE">/My/Mypackage</SDX-REF>
  <SD GID="SEVERITY">significant</SD>
  <SDG GID="SOURCE">
    <SDX-REF DEST="TRACEABLE" BASE="ArTrace">RS_MyReq_0090</SDX-REF>
  </SDG>
  <SDG GID="TARGETS">
    <SDX-REF DEST="SW-COMPONENT-TYPE" BASE="MyComponent">SwComponentTypes
      /MyComponent</SDX-REF>
```

```
    <SDX-REF DEST="COMPU-METHOD" BASE="MyComponent">CompuMethods/MyMethod
      </SDX-REF>
    <SDX-REF DEST="AUTOSAR-DATA-TYPE" BASE="MyComponent">
      ImplementationDataTypes/MyType</SDX-REF>
  </SDG>
  <SD GID="Path">maintenance</SD>
</SDG>
```

To express variability in special data group see listing 4.6.

Listing 4.6: Example for Special Data with Port-Reference and Variant

```
<SDG GID="EXAMPLE-PORT-REF">
  <SDXF>
    <REFERRABLE-REF DEST="P-PORT-PROTOTYPE">SwComponentTypes/MyComponent/
      PData1</REFERRABLE-REF>
      <VARIATION-POINT>
        <SW-SYSCOND BINDING-TIME="PRE-COMPILE-TIME">
          <SYSC-REF DEST="SW-SYSTEMCONST" BASE="MySwSystemconsts">
            SwSystemconsts/ScVar1</SYSC-REF> &gt;= 1
        </SW-SYSCOND>
      </VARIATION-POINT>
  </SDXF>
  <SDF GID="EXAMPLE-VARIANT-SD">
    <VALUE><SYSC-REF DEST="SW-SYSTEMCONST" BASE="MySwSystemconsts">
      SwSystemconsts/ScVar2</SYSC-REF>+4</VALUE>
  </SDF>
  <SDG GID="EXAMPLE-VARIANT-SDG">
    <SD GID="Path">maintenance</SD>
    <VARIATION-POINT>
      <SW-SYSCOND BINDING-TIME="PRE-COMPILE-TIME">
        <SYSC-REF DEST="SW-SYSTEMCONST" BASE="MySwSystemconsts">
          SwSystemconsts/ScVar3</SYSC-REF> &lt; 1
      </SW-SYSCOND>
    </VARIATION-POINT>
  </SDG>
</SDG>
```

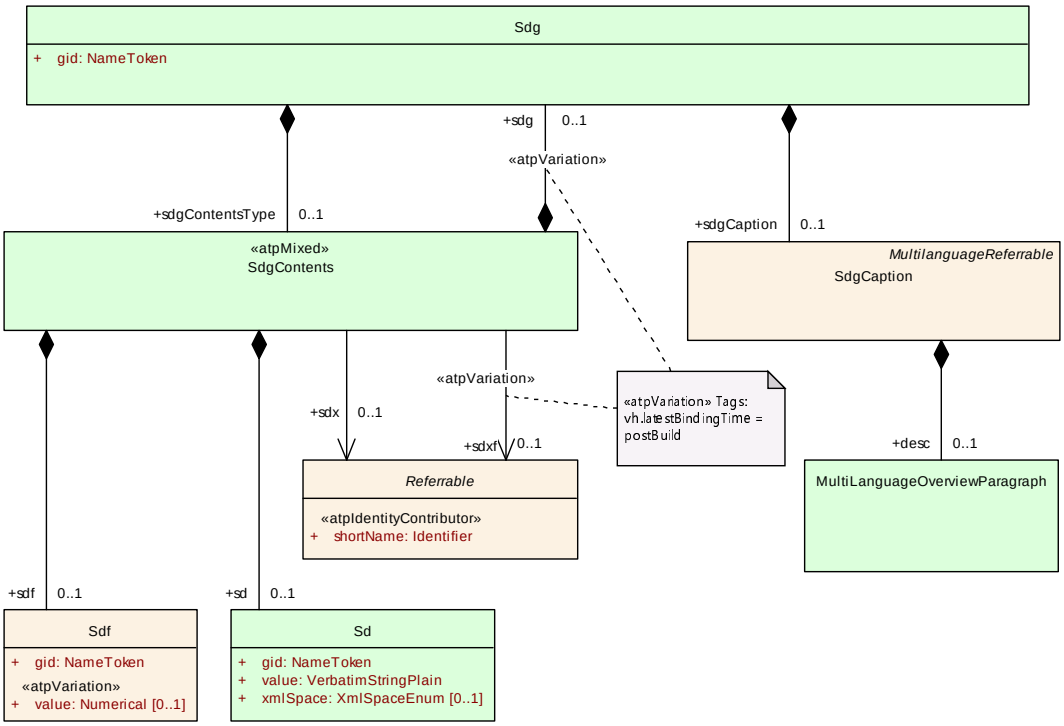


Figure 4.5: SpecialData

Class	**Sdg**			
Package	M2::MSR::AsamHdo::SpecialData			
Note	Sdg (SpecialDataGroup) is a generic model which can be used to keep arbitrary information which is not explicitly modeled in the meta-model. Sdg can have various contents as defined by sdgContentsType. Special Data should only be used moderately since all elements should be defined in the meta-model. Thereby SDG should be considered as a temporary solution when no explicit model is available. If an sdg Caption is available, it is possible to establish a reference to the sdg structure.			
Base	*ARObject*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
gid	NameToken	1	attr	This attributes specifies an identifier. Gid comes from the SGML/XML-Term "Generic Identifier" which is the element name in XML. The role of this attribute is the same as the name of an XML - element. **Tags:**xml.attribute=true
sdgCaption	SdgCaption	0..1	aggr	This aggregation allows to assign the properties of Identifiable to the sdg. By this, a shortName etc. can be assigned to the Sdg. **Tags:**xml.sequenceOffset=20
sdgContents Type	SdgContents	0..1	aggr	This is the content of the Sdg. **Tags:** xml.roleElement=false xml.roleWrapperElement=false xml.sequenceOffset=30 xml.typeElement=false xml.typeWrapperElement=false

Table 4.19: Sdg

Class	<<atpMixed>> **SdgContents**			
Package	M2::MSR::AsamHdo::SpecialData			
Note	This meta-class represents the possible contents of a special data group. It can be an arbitrary mix of references, of primitive special data and nested special data groups.			
Base	*ARObject*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
sd	Sd	0..1	aggr	This is one particular special data element. **Tags:**xml.sequenceOffset=40
sdf	Sdf	0..1	aggr	This is one particular special data element. **Tags:**xml.sequenceOffset=60
sdg	Sdg	0..1	aggr	This aggregation allows to express nested special data groups. By this, any structure can be represented in SpeicalData. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=postBuild xml.sequenceOffset=50
sdx	Referrable	0..1	ref	Reference to any identifiable element. This allows to use Sdg even to establish arbitrary relationships.
sdxf	Referrable	0..1	ref	Additional reference with variant support. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=postBuild

Table 4.20: SdgContents

Class	**SdgCaption**			
Package	M2::MSR::AsamHdo::SpecialData			
Note	This meta-class represents the caption of a special data group. This allows to have some parts of special data as identifiable.			
Base	*ARObject*, *MultilanguageReferrable*, *Referrable*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
desc	MultiLanguageOverview Paragraph	0..1	aggr	This represents a general but brief (one paragraph) description what the special data in question is about. It is only one paragraph! Desc is intended to be collected into overview tables. This property helps a human reader to identify the special data in question.

Table 4.21: SdgCaption

Class	**Sd**			
Package	M2::MSR::AsamHdo::SpecialData			
Note	This class represents a primitive element in a special data group.			
Base	*ARObject*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
gid	NameToken	1	attr	This attributes specifies an identifier. Gid comes from the SGML/XML-Term "Generic Identifier" which is the element name in XML. The role of this attribute is the same as the name of an XML - element. **Tags:**xml.attribute=true

▽

△

Class	Sd			
value	VerbatimStringPlain	1	attr	This is the value of the special data. **Tags:** xml.roleElement=false xml.roleWrapperElement=false xml.typeElement=false xml.typeWrapperElement=false
xmlSpace	XmlSpaceEnum	0..1	attr	This attribute is used to signal an intention that in that element, white space should be preserved by applications. It is defined according to xml:space as declared by W3C. **Tags:** xml.attribute=true xml.attributeRef=true xml.enforceMinMultiplicity=true xml.name=space xml.nsPrefix=xml

Table 4.22: Sd

Class	Sdf			
Package	M2::MSR::AsamHdo::SpecialData			
Note	This class represents a numerical value in a special data group which may be subject to variability.			
Base	*ARObject*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
gid	NameToken	1	attr	This attributes specifies an identifier. Gid comes from the SGML/XML-Term "Generic Identifier" which is the element name in XML. The role of this attribute is the same as the name of an XML - element. **Tags:**xml.attribute=true
value	Numerical	0..1	attr	This is the value of the special data. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime

Table 4.23: Sdf

4.5.2 Special Data Definitions

[TPS_GST_00374] Purpose of `SdgDef` ⌈The Special Data Group Definition `SdgDef` specifies the structure of `Sdg`s that are accepted in `AUTOSAR Model`s. The structure is defined by specifying a `Meta-Model` using `SdgClass`es and its `SdgAttribute`s (`SdgPrimitiveAttribute`s, `SdgPrimitiveAttributeWithVariation`s, `SdgReference`s, `SdgAggregationWithVariation`s, `SdgForeignReference`s and `SdgForeignReferenceWithVariation`s).⌋*()*

The `SdgDef` is the AUTOSAR counterpart of a UML Profile.

[TPS_GST_00421] Categories of `SdgPrimitiveAttribute` and `SdgPrimitiveAttributeWithVariation` ⌈The standardized categories of `SdgPrimitiveAttribute` are INTEGER, BOOLEAN, FLOAT, STRING, ENUMERATION. The

standardized categories of SdgPrimitiveAttributeWithVariation are one of INTEGER, BOOLEAN, FLOAT, ENUMERATION.⌋()

Listing 4.8 illustrates a definition of the enumeration attributes "severity". The modelling of enumeration attributes is done using SdgPrimitiveAttribute.pattern which defines the valid enumeration literals.

[TPS_GST_00375] Purpose of SdgClass ⌈The SdgClass specifies the structure of a Sdg with a specific gid.⌋()

The SdgClass is the AUTOSAR counterpart of a UML Stereotype.

[TPS_GST_00422] Specification of the value of SdgClass.extendsMetaClass and SdgAbstractForeignReference.destMetaClass ⌈The string entered as the value of SdgClass.extendsMetaClass and SdgAbstractForeignReference.destMetaClass shall follow the format of a name of a M2 class as defined in the XML schema and the referenced class needs to be derived (directly or indirectly) from Referrable.⌋()

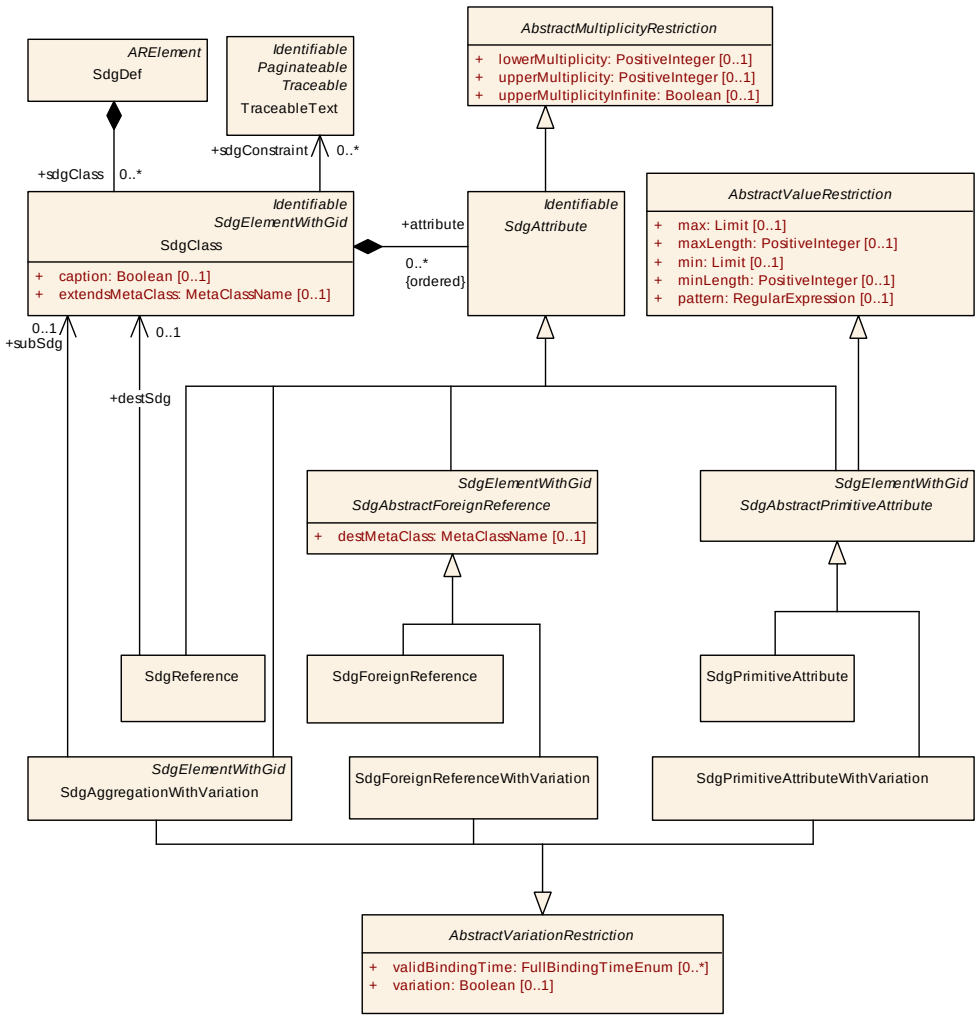


Figure 4.6: Special Data Group Definition

Example 4.7 specifies a special data group definition SdgDef that accepts simple extensions as described in example 4.4.

Listing 4.7: Example of a Special Data Definition

```
<SDG-DEF>
  <SHORT-NAME>TracingInfoExtension</SHORT-NAME>
  <SDG-CLASSES>
    <SDG-CLASS>
      <SHORT-NAME>TracingInfo</SHORT-NAME>
      <GID>TRACING-INFO</GID>
      <EXTENDS-META-CLASS>Documentation</EXTENDS-META-CLASS>
      <ATTRIBUTES>
        <SDG-PRIMITIVE-ATTRIBUTE>
          <SHORT-NAME>ItemPrefix</SHORT-NAME>
          <GID>ITEM-PREFIX</GID>
        </SDG-PRIMITIVE-ATTRIBUTE>
        <SDG-PRIMITIVE-ATTRIBUTE>
          <SHORT-NAME>ContributesToGeneral</SHORT-NAME>
          <GID>CONTRIBUTES-TO-GENERAL</GID>
        </SDG-PRIMITIVE-ATTRIBUTE>
        <SDG-PRIMITIVE-ATTRIBUTE>
          <SHORT-NAME>ContributesTo</SHORT-NAME>
          <GID>CONTRIBUTES-TO</GID>
        </SDG-PRIMITIVE-ATTRIBUTE>
        <SDG-PRIMITIVE-ATTRIBUTE>
          <SHORT-NAME>Includes</SHORT-NAME>
          <GID>INCLUDES</GID>
        </SDG-PRIMITIVE-ATTRIBUTE>
      </ATTRIBUTES>
    </SDG-CLASS>
  </SDG-CLASSES>
</SDG-DEF>
```

Example 4.8 specifies a special data group definition SdgDef that accepts extensions with nested Sdg containers and references to objects of a AUTOSAR model. An example of an accepted Sdg is provided in 4.5.

Listing 4.8: Example of a Special Data Definition that specifies a structure of Special Data with sub groups and references

```
<SDG-DEF>
  <SHORT-NAME>ExampleSdgDefWithForeignReferencesAndSubSdgs</SHORT-NAME>
  <SDG-CLASSES>
    <SDG-CLASS>
      <SHORT-NAME>ExampleRelation</SHORT-NAME>
      <GID>EXAMPLE-RELATION</GID>
      <EXTENDS-META-CLASS>ARPackage</EXTENDS-META-CLASS>
      <CAPTION>true</CAPTION>
      <ATTRIBUTES>
        <SDG-FOREIGN-REFERENCE>
          <SHORT-NAME>packageRef</SHORT-NAME>
          <UPPER-MULTIPLICITY>1</UPPER-MULTIPLICITY>
          <DEST-META-CLASS>ArPackage</DEST-META-CLASS>
        </SDG-FOREIGN-REFERENCE>
        <SDG-PRIMITIVE-ATTRIBUTE>
          <SHORT-NAME>severity</SHORT-NAME>
```

```
        <UPPER-MULTIPLICITY>1</UPPER-MULTIPLICITY>
        <GID>SEVERITY</GID>
        <PATTERN>critical|significant|minor|low</PATTERN>
      </SDG-PRIMITIVE-ATTRIBUTE>
      <SDG-AGGREGATION-WITH-VARIATION>
        <SHORT-NAME>source</SHORT-NAME>
        <LOWER-MULTIPLICITY>1</LOWER-MULTIPLICITY>
        <UPPER-MULTIPLICITY>1</UPPER-MULTIPLICITY>
        <GID>SOURCE</GID>
        <VARIATION>false</VARIATION>
        <SUB-SDG-REF DEST="SDG-CLASS" BASE="Extensions">
            ExampleRelationExtension/SourceRefContainer</SUB-SDG-REF>
      </SDG-AGGREGATION-WITH-VARIATION>
      <SDG-AGGREGATION-WITH-VARIATION>
        <SHORT-NAME>target</SHORT-NAME>
        <LOWER-MULTIPLICITY>1</LOWER-MULTIPLICITY>
        <UPPER-MULTIPLICITY>1</UPPER-MULTIPLICITY>
        <GID>TARGETS</GID>
        <VARIATION>false</VARIATION>
        <SUB-SDG-REF DEST="SDG-CLASS" BASE="Extensions">
            ExampleRelationExtension/TargetsRefContainer</SUB-SDG-REF>
      </SDG-AGGREGATION-WITH-VARIATION>
      <SDG-PRIMITIVE-ATTRIBUTE>
        <SHORT-NAME>path</SHORT-NAME>
        <LOWER-MULTIPLICITY>0</LOWER-MULTIPLICITY>
        <UPPER-MULTIPLICITY>1</UPPER-MULTIPLICITY>
        <GID>Path</GID>
      </SDG-PRIMITIVE-ATTRIBUTE>
    </ATTRIBUTES>
  </SDG-CLASS>
  <SDG-CLASS>
    <SHORT-NAME>SourceRefContainer</SHORT-NAME>
    <GID>SOURCE-REF-CONTAINER</GID>
    <EXTENDS-META-CLASS>ARPackage</EXTENDS-META-CLASS>
    <CAPTION>true</CAPTION>
    <ATTRIBUTES>
      <SDG-FOREIGN-REFERENCE>
        <SHORT-NAME>sourceRef</SHORT-NAME>
        <LOWER-MULTIPLICITY>1</LOWER-MULTIPLICITY>
        <UPPER-MULTIPLICITY>1</UPPER-MULTIPLICITY>
        <DEST-META-CLASS>Traceable</DEST-META-CLASS>
      </SDG-FOREIGN-REFERENCE>
    </ATTRIBUTES>
  </SDG-CLASS>
  <SDG-CLASS>
    <SHORT-NAME>TargetsRefContainer</SHORT-NAME>
      <GID>STARGET-REF-CONTAINER</GID>
    <EXTENDS-META-CLASS>ARPackage</EXTENDS-META-CLASS>
    <CAPTION>true</CAPTION>
    <ATTRIBUTES>
      <SDG-FOREIGN-REFERENCE>
        <SHORT-NAME>targetRef</SHORT-NAME>
        <LOWER-MULTIPLICITY>1</LOWER-MULTIPLICITY>
        <UPPER-MULTIPLICITY-INFINITE>true</UPPER-MULTIPLICITY-INFINITE>
        <DEST-META-CLASS>Referrable</DEST-META-CLASS>
      </SDG-FOREIGN-REFERENCE>
```

```
      </ATTRIBUTES>
    </SDG-CLASS>
  </SDG-CLASSES>
</SDG-DEF>
```

Example 4.9 specifies a special data group definition SdgDef that accepts extensions with variation. An example of an accepted Sdg is provided in 4.6.

Listing 4.9: Example of a Special Data Definition that specifies a structure of Special Data with variation

```
<SDG-DEF>
  <SHORT-NAME>ExampleSdgDefWithVariation</SHORT-NAME>
  <SDG-CLASSES>
    <SDG-CLASS>
      <SHORT-NAME>ExampleSdgWithVariation</SHORT-NAME>
      <GID>EXAMPLE-PORT-REF</GID>
      <ATTRIBUTES>
        <SDG-FOREIGN-REFERENCE-WITH-VARIATION>
          <SHORT-NAME>exampleVariantSfxf</SHORT-NAME>
          <LOWER-MULTIPLICITY>1</LOWER-MULTIPLICITY>
          <UPPER-MULTIPLICITY>1</UPPER-MULTIPLICITY>
          <DEST-META-CLASS>PortPrototype</DEST-META-CLASS>
          <VARIATION>true</VARIATION>
          <VALID-BINDING-TIMES>
            <VALID-BINDING-TIME>PRE-COMPILE-TIME</VALID-BINDING-TIME>
            <VALID-BINDING-TIME>SYSTEM-DESIGN-TIME</VALID-BINDING-TIME>
          </VALID-BINDING-TIMES>
        </SDG-FOREIGN-REFERENCE-WITH-VARIATION>
        <SDG-PRIMITIVE-ATTRIBUTE-WITH-VARIATION>
          <SHORT-NAME>ExampleVariantSd</SHORT-NAME>
          <LOWER-MULTIPLICITY>1</LOWER-MULTIPLICITY>
          <UPPER-MULTIPLICITY>1</UPPER-MULTIPLICITY>
          <GID>EXAMPLE-VARIANT-SD</GID>
          <VARIATION>true</VARIATION>
          <VALID-BINDING-TIMES>
            <VALID-BINDING-TIME>PRE-COMPILE-TIME</VALID-BINDING-TIME>
            <VALID-BINDING-TIME>SYSTEM-DESIGN-TIME</VALID-BINDING-TIME>
          </VALID-BINDING-TIMES>
        </SDG-PRIMITIVE-ATTRIBUTE-WITH-VARIATION>
        <SDG-AGGREGATION-WITH-VARIATION>
          <SHORT-NAME>ExampleVariantSdg</SHORT-NAME>
          <LOWER-MULTIPLICITY>1</LOWER-MULTIPLICITY>
          <UPPER-MULTIPLICITY>1</UPPER-MULTIPLICITY>
          <GID>EXAMPLE-VARIANT-SDG</GID>
          <VARIATION>true</VARIATION>
          <VALID-BINDING-TIMES>
            <VALID-BINDING-TIME>PRE-COMPILE-TIME</VALID-BINDING-TIME>
            <VALID-BINDING-TIME>SYSTEM-DESIGN-TIME</VALID-BINDING-TIME>
          </VALID-BINDING-TIMES>
          <SUB-SDG-REF DEST="SDG-CLASS" BASE="Extensions">
              ExampleSdgWithVariation/ExampleVariantSdgContainer</SUB-SDG-
              REF>
        </SDG-AGGREGATION-WITH-VARIATION>
      </ATTRIBUTES>
    </SDG-CLASS>
```

```
    <SDG-CLASS>
      <SHORT-NAME>ExampleVariantSdgContainer</SHORT-NAME>
      <ATTRIBUTES>
        <SDG-PRIMITIVE-ATTRIBUTE>
          <SHORT-NAME>Path</SHORT-NAME>
          <GID>Path</GID>
        </SDG-PRIMITIVE-ATTRIBUTE>
      </ATTRIBUTES>
    </SDG-CLASS>
  </SDG-CLASSES>
</SDG-DEF>
```

[TPS_GST_00425] Set `SdgClass.caption` = true if the `Sdg` on the value side shall be referrable ⌈If the `Sdg` defined by a `SdgClass` shall be referrable, then the attribute `SdgClass.caption` shall be set true. In this case, it is mandatory for a valid `Sdg` instance to have a `SdgCaption`. If the attribute `SdgClass.caption` is not set, or the set value is false, then the usage of a `SdgCaption` is optional.⌋*()*

The [TPS_GST_00425] has to differentiate between the definition side (Example 4.10) and the value side (Example 4.11). The existence of the caption is a mechanism to reference a `Sdg` to be used in case the standardized referencing cannot be applied.

Listing 4.10: Example of a Special Data definition side part

```
<SDG-DEF>
  <SHORT-NAME>InstanceExtensions</SHORT-NAME>
  <SDG-CLASSES>
    <SDG-CLASS>
      <SHORT-NAME>ProvidedUserDefinedServiceInstance</SHORT-NAME>
      <GID>acme:instanceExtensions</GID>
      <EXTENDS-META-CLASS>ProvidedUserDefinedServiceInstance</EXTENDS-META
          -CLASS>
      <CAPTION>true</CAPTION>
      <ATTRIBUTES>
        <SDG-FOREIGN-REFERENCE>
          <SHORT-NAME>SdServerTimeConfig</SHORT-NAME>
          <LOWER-MULTIPLICITY>0</LOWER-MULTIPLICITY>
          <UPPER-MULTIPLICITY>1</UPPER-MULTIPLICITY>
          <GID>acme:sdServerTimeConfig</GID>
          <DEST-META-CLASS>SomeipSdServerServiceInstanceConfig</DEST-META-
              CLASS>
        </SDG-FOREIGN-REFERENCE>
      </ATTRIBUTES>
    </SDG-CLASS>
  </SDG-CLASSES>
</SDG-DEF>
```

Listing 4.11: Example of a Special Data value side part

```
<PROVIDED-USER-DEFINED-SERVICE-INSTANCE>
  <SHORT-NAME>UDSI</SHORT-NAME>
  <ADMIN-DATA>
    <SDGS>
      <SDG GID="acme:instanceExtensions">
        <SDG GID="acme:sdServerTimeConfig">
          <SDG-CAPTION>
```

```
            <SHORT-NAME>MyConfigRef</SHORT-NAME>
          </SDG-CAPTION>
          <SDX-REF DEST="SOMEIP-SD-SERVER-SERVICE-INSTANCE-CONFIG">/SD/
              MyConf</SDX-REF>
        </SDG>
      </SDG>
    </SDGS>
  </ADMIN-DATA>
</PROVIDED-USER-DEFINED-SERVICE-INSTANCE>
```

The class SdgReference can be used on the definition side to model a reference to a Sdg on the value side. On the definition side, the attribute SdgReference.destSdg refers to a SdgClass which describes the destination type of the referenced Sdg. Note, on the value side, the reference is realized by means of the originating Sdg defining an sdx attribute which refers to the sdgCaption of the referenced Sdg. To enforce the existence of an SdgCaption on the value side, the attribute SdgClass.caption must be set true for the SdgClass referenced by SdgReference.destSdg, see [TPS_GST_00425].

Assume a SWC (Example 4.12) where some custom safety guidelines and a reference to a safety level shall be specified as special data.

Listing 4.12: Example of a Special Data usage in a SWC

```
<AR-PACKAGE>
  <SHORT-NAME>SwComponentTypes</SHORT-NAME>
  <ELEMENTS>
    <APPLICATION-SW-COMPONENT-TYPE>
      <SHORT-NAME>SwcFoo</SHORT-NAME>
      <ADMIN-DATA>
        <SDGS>
          <SDG GID="CustomSafetyGuidelines">
            <SDX-REF DEST="SDG-CAPTION">/SwComponentTypes/SwcFoo/ASIL_B</
                SDX-REF>
            <SD GID="Rule">R00042</SD>
          </SDG>
          <SDG GID="SafetyLevel">
            <SDG-CAPTION>
              <SHORT-NAME>ASIL_B</SHORT-NAME>
            </SDG-CAPTION>
            <SD GID="ASIL">B</SD>
          </SDG>
        </SDGS>
      </ADMIN-DATA>
    </APPLICATION-SW-COMPONENT-TYPE>
  </ELEMENTS>
</AR-PACKAGE>
```

This can be modeled (Example 4.13) using the following Sdg definition:

Listing 4.13: Example of the corresponding Special Data definition

```
<AR-PACKAGE>
  <SHORT-NAME>SdgDefs</SHORT-NAME>
  <ELEMENTS>
```

```
<SDG-DEF>
  <SHORT-NAME>CustomSafetyExtensionDef</SHORT-NAME>
  <SDG-CLASSES>
    <SDG-CLASS>
      <SHORT-NAME>CustomSafetyGuidelines</SHORT-NAME>
      <GID>CustomSafetyGuidelines</GID>
      <ATTRIBUTES>
        <SDG-REFERENCE>
          <SHORT-NAME>SafetyLevelRef</SHORT-NAME>
          <DEST-SDG-REF DEST="SDG-CLASS">/SdgDefs/
              CustomSafetyExtensionDef/SafetyLevelDef</DEST-SDG-REF>
        </SDG-REFERENCE>
        <SDG-PRIMITIVE-ATTRIBUTE>
          <SHORT-NAME>RuleDef</SHORT-NAME>
          <CATEGORY>STRING</CATEGORY>
          <UPPER-MULTIPLICITY-INFINITE>true</UPPER-MULTIPLICITY-
              INFINITE>
          <GID>Rule</GID>
        </SDG-PRIMITIVE-ATTRIBUTE>
      </ATTRIBUTES>
    </SDG-CLASS>
    <SDG-CLASS>
      <SHORT-NAME>SafetyLevelDef</SHORT-NAME>
      <GID>SafetyLevel</GID>
      <CAPTION>true</CAPTION>
      <ATTRIBUTES>
        <SDG-PRIMITIVE-ATTRIBUTE>
          <SHORT-NAME>AsilDef</SHORT-NAME>
          <CATEGORY>ENUMERATION</CATEGORY>
          <UPPER-MULTIPLICITY>1</UPPER-MULTIPLICITY>
          <GID>ASIL</GID>
          <PATTERN>A|B|C|D</PATTERN>
        </SDG-PRIMITIVE-ATTRIBUTE>
      </ATTRIBUTES>
    </SDG-CLASS>
  </SDG-CLASSES>
</SDG-DEF>
</ELEMENTS>
</AR-PACKAGE>
```

Class	**SdgDef**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::SpecialDataDef			
Note	A SdgDef groups several SdgClasses which belong to the same extension. The concept of an SdgDef is similiar to an UML Profile. **Tags:**atp.recommendedPackage=SdgDefs			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
sdgClass	SdgClass	*	aggr	The owned sdgClasses which define the structure of the Sdgs **Tags:**xml.namePlural=SDG-CLASSES

Table 4.24: SdgDef

Class	***SdgElementWithGid*** (abstract)			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::SpecialDataDef			
Note	A special data group element with gid is an abstract element that shall have a name (gid, "Generic Identifier").			
Base	*ARObject*			
Subclasses	*SdgAbstractForeignReference*, *SdgAbstractPrimitiveAttribute*, SdgAggregationWithVariation, SdgClass			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
gid	NameToken	0..1	attr	Specifies the name that identifies the element.

Table 4.25: SdgElementWithGid

Class	**SdgClass**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::SpecialDataDef			
Note	An SdgClass specifies the name and structure of the SDG that may be used to store proprietary data in an AUTOSAR model. The SdgClass is similar to an UML stereotype.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *SdgElementWithGid*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
attribute (ordered)	SdgAttribute	*	aggr	Defintion of the structure of the Sdg **Tags:**xml.sequenceOffset=30
caption	Boolean	0..1	attr	Specifies if a caption is required. Note: only Sdgs that have a caption can be referenced **Tags:**xml.sequenceOffset=20
extendsMeta Class	MetaClassName	0..1	attr	The AUTOSAR Meta-Class that may be extended by this SdgClass. **Tags:**xml.sequenceOffset=10
sdgConstraint	TraceableText	*	ref	Semantic constraints that restrict the structure of the special data group. **Tags:**xml.sequenceOffset=40

Table 4.26: SdgClass

Class	***SdgAttribute*** (abstract)			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::SpecialDataDef			
Note	Describes the attributes of an Sdg.			
Base	*ARObject*, *AbstractMultiplicityRestriction*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	*SdgAbstractForeignReference*, *SdgAbstractPrimitiveAttribute*, SdgAggregationWithVariation, Sdg Reference			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table 4.27: SdgAttribute

Class	***SdgAbstractPrimitiveAttribute*** (abstract)
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::SpecialDataDef
Note	Describes primitive attributes of a special data group.

△

Class	**SdgAbstractPrimitiveAttribute** (abstract)			
Base	*ARObject*, *AbstractMultiplicityRestriction*, *AbstractValueRestriction*, *Identifiable*, *Multilanguage Referrable*, *Referrable*, *SdgAttribute*, *SdgElementWithGid*			
Subclasses	SdgPrimitiveAttribute, SdgPrimitiveAttributeWithVariation			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table 4.28: SdgAbstractPrimitiveAttribute

Class	**SdgPrimitiveAttribute**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::SpecialDataDef			
Note	Describes primitive special data attributes without variation. This class accepts a special data "sd" attribute.			
Base	*ARObject*, *AbstractMultiplicityRestriction*, *AbstractValueRestriction*, *Identifiable*, *Multilanguage Referrable*, *Referrable*, *SdgAbstractPrimitiveAttribute*, *SdgAttribute*, *SdgElementWithGid*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table 4.29: SdgPrimitiveAttribute

Class	**SdgPrimitiveAttributeWithVariation**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::SpecialDataDef			
Note	Describes a primitive numerical special data attribute with variation. This class accepts a special data "sdf" element.			
Base	*ARObject*, *AbstractMultiplicityRestriction*, *AbstractValueRestriction*, *AbstractVariationRestriction*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *SdgAbstractPrimitiveAttribute*, *SdgAttribute*, *Sdg ElementWithGid*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table 4.30: SdgPrimitiveAttributeWithVariation

Class	**SdgAggregationWithVariation**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::SpecialDataDef			
Note	Describes that the Sdg may contain another Sdg. The gid of the nested Sdg is defined by subSdg. Represents 'sdg'.			
Base	*ARObject*, *AbstractMultiplicityRestriction*, *AbstractVariationRestriction*, *Identifiable*, *Multilanguage Referrable*, *Referrable*, *SdgAttribute*, *SdgElementWithGid*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
subSdg	SdgClass	0..1	ref	Supported sub Sdg Class

Table 4.31: SdgAggregationWithVariation

Class	**SdgReference**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::SpecialDataDef
Note	Describes an attribute of a SdgClass which is used on the definition side to model a reference from one Sdg to another Sdg on the value side.

▽

△

Class	SdgReference			
Base	ARObject, AbstractMultiplicityRestriction, Identifiable, MultilanguageReferrable, Referrable, SdgAttribute			
Attribute	Type	Mult.	Kind	Note
destSdg	SdgClass	0..1	ref	Refers to a SdgClass which is used on the definition side to model the destination type of the referenced Sdg. On the value side the reference is realized by means of the originating Sdg defining an sdgx attribute which refers to the sdgCaption of the referenced Sdg.

Table 4.32: SdgReference

Class	*SdgAbstractForeignReference* (abstract)			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::SpecialDataDef			
Note	An abstract reference that can point to any referrable object in an AUTOSAR Model.			
Base	ARObject, AbstractMultiplicityRestriction, Identifiable, MultilanguageReferrable, Referrable, SdgAttribute, SdgElementWithGid			
Subclasses	SdgForeignReference, SdgForeignReferenceWithVariation			
Attribute	Type	Mult.	Kind	Note
destMetaClass	MetaClassName	0..1	attr	specifies the destination meta-class of the reference.

Table 4.33: SdgAbstractForeignReference

Class	SdgForeignReference			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::SpecialDataDef			
Note	A reference without variation support that can point to any referrable object in an AUTOSAR Model. This class accepts the special data "Sdx" reference.			
Base	ARObject, AbstractMultiplicityRestriction, Identifiable, MultilanguageReferrable, Referrable, SdgAbstract ForeignReference, SdgAttribute, SdgElementWithGid			
Attribute	Type	Mult.	Kind	Note
–	–	–	–	–

Table 4.34: SdgForeignReference

Class	SdgForeignReferenceWithVariation			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::SpecialDataDef			
Note	A reference with variation support that can point to any referrable object in an AUTOSAR Model. This class accepts the special data "Sdxf" reference.			
Base	ARObject, AbstractMultiplicityRestriction, AbstractVariationRestriction, Identifiable, Multilanguage Referrable, Referrable, SdgAbstractForeignReference, SdgAttribute, SdgElementWithGid			
Attribute	Type	Mult.	Kind	Note
–	–	–	–	–

Table 4.35: SdgForeignReferenceWithVariation

Primitive	MetaClassName
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes

▽

△

Primitive	**MetaClassName**
Note	Name of a class in the AUTOSAR Meta-Model. **Tags:** xml.xsd.customType=META-CLASS-NAME xml.xsd.pattern=[A-Z][a-zA-Z0-9_]* xml.xsd.type=string

Table 4.36: MetaClassName

4.6 Model Restriction Types

[TPS_GST_00376] Purpose of Model Restriction Types ⌈Model Restriction Types specify rules that restrict the content of AUTOSAR models. Those restrictions are e.g. used in the context of Special Data Group Definitions or Data Exchange Points [2].⌋*()* Corresponding details to Special Data Group Definitions are given in 4.5.2.

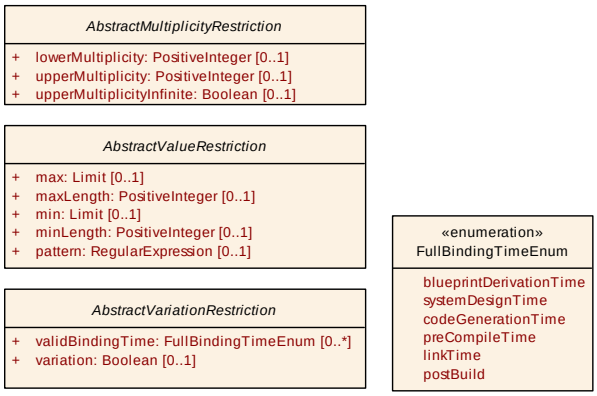


Figure 4.7: Model Restriction Types

4.6.1 Restriction of Simple Primitive Values

[TPS_GST_00377] Purpose of `AbstractValueRestriction` ⌈ The `AbstractValueRestriction` defines constraints on the value space of a simple primitive data type. The attributes of this class represent constraining facets according to the XML Schema Specification [11]. A value is valid if it is valid according to all defined constraints.⌋*()*

Class	***AbstractValueRestriction*** (abstract)
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::ModelRestrictionTypes
Note	Restricts primitive values. A value is valid if all rules that are defined by this restriction evaluate to true.
Base	*ARObject*
Subclasses	PrimitiveAttributeCondition, *SdgAbstractPrimitiveAttribute*, ValueRestrictionWithSeverity

▽

△

Class	***AbstractValueRestriction*** (abstract)			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
max	Limit	0..1	attr	Specifies the upper bounds for numeric values.
maxLength	PositiveInteger	0..1	attr	Specifies the maximum number of characters of textual values.
min	Limit	0..1	attr	Specifies the lower bounds for numeric values.
minLength	PositiveInteger	0..1	attr	Specifies the minimal number of characters of textual values.
pattern	RegularExpression	0..1	attr	Defines the exact sequence of characters that are acceptable.

Table 4.37: AbstractValueRestriction

4.6.2 Restriction of Multiplicities

[TPS_GST_00378] Purpose of `AbstractMultiplicityRestriction` ⌈ The `AbstractMultiplicityRestriction` specifies how often an element may occur. With the two attributes `lowerMultiplicity` and `upperMultiplicity` the minimum and maximum occurrence of the configuration element is specified.⌋ *()*

[TPS_GST_00380] Countably infinite number of elements ⌈ To express a countable infinite number of occurrences of elements the `upperMultiplicityInfinite` element shall exist and shall be set to `true`.⌋ *()*

[constr_2606] Existence of `upperMultiplicityInfinite` and `upperMultiplicity` of `AbstractMultiplicityRestriction` is mutually exclusive ⌈ The existence of the elements `upperMultiplicityInfinite` and `upperMultiplicity` of `AbstractMultiplicityRestriction` shall be mutually exclusive.⌋ *()*

[constr_2607] `lowerMultiplicity` of `AbstractMultiplicityRestriction` shall be smaller or equal to `upperMultiplicity` ⌈ `lowerMultiplicity` of `AbstractMultiplicityRestriction` shall be smaller or equal to `upperMultiplicity`.⌋ *()*

4.6.3 Restriction of use of Variation

[TPS_GST_00379] Purpose of `AbstractVariationRestriction` ⌈ The `AbstractVariationRestriction` defines constraints on the usage of variation and on the valid binding times.⌋ *()*

Class	***AbstractVariationRestriction*** (abstract)
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::ModelRestrictionTypes
Note	Defines constraints on the usage of variation and on the valid binding times.

▽

△

Class	***AbstractVariationRestriction*** (abstract)			
Base	*ARObject*			
Subclasses	SdgAggregationWithVariation, SdgForeignReferenceWithVariation, SdgPrimitiveAttributeWithVariation, VariationRestrictionWithSeverity			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
validBinding Time	FullBindingTimeEnum	*	attr	List of valid binding times. **Tags:**xml.sequenceOffset=20
variation	Boolean	0..1	attr	Defines if the AUTOSAR model may define a Variation Point at this location. **Tags:**xml.sequenceOffset=10

Table 4.38: AbstractVariationRestriction

Enumeration	**FullBindingTimeEnum**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::ModelRestrictionTypes
Note	This enumeration specifies the BindingTimes that can be used in AUTOSAR models.
Literal	***Description***
blueprintDerivation Time	The point in time when an object is created from a blueprint. **Tags:**atp.EnumerationLiteralIndex=0
codeGeneration Time	• Coding by hand, based on requirements document. • Tool based code generation, e.g. from a model. • The model may contain variants. • Only code for the selected variant(s) is actually generated. **Tags:**atp.EnumerationLiteralIndex=2
linkTime	Configure what is included in object code, and what is omitted Based on which variant(s) are selected E.g. for modules that are delivered as object code (as opposed to those that are delivered as source code) **Tags:**atp.EnumerationLiteralIndex=4
postBuild	PostBuild is the binding time which is bound latest at startup of the ECU. In other words this is everything between creation of the executable program and startup of the ECU. **Tags:**atp.EnumerationLiteralIndex=5
preCompileTime	This is typically the C-Preprocessor. Exclude parts of the code from the compilation process, e.g., because they are not required for the selected variant, because they are incompatible with the selected variant, because they require resources that are not present in the selected variant. Object code is only generated for the selected variant(s). The code that is excluded at this stage code will not be available at later stages. **Tags:**atp.EnumerationLiteralIndex=3
systemDesignTime	• Designing the VFB. • Software Component types (PortInterfaces). • SWC Prototypes and the Connections between SWCprototypes. • Designing the Topology • ECUs and interconnecting Networks • Designing the Communication Matrix and Data Mapping **Tags:**atp.EnumerationLiteralIndex=1

Table 4.39: FullBindingTimeEnum

4.7 Primitive Types

This chapter describes the primitive types which are used in the AUTOSAR M2 model. These primitives are shown in the class tables below. In addition to these primitives some packages may define some own local primitives. Primitive types of MSR are included in the separate package due to historical reason.

Note that the AUTOSAR meta-model does not use the built in primitives provided by UML.

Note further that some of these primitives also have attributes. Such attributes result in xml attributes of xml elements representing the primitive.

In former releases of the GST strings as "0b" were accepted as valid due to 0[bB][0-1]*. The change of the binary part to 0[bB][0-1]+ in the regular expression intents on that the given string is valid if the regular expression has exactly one match which covers the whole string. Non meaningful inputs (e.g. "0b") will now be identified as invalid binary number. This also applies to other regular expressions containing the binary part.

[constr_2628] Representation of xml.xsd.type=double data types ⌈All data types with xml.xsd.type=double shall comply with IEEE 754 and are limited to what can be expressed by a 64 bit binary representation.⌋*()*

Note for tool implementers: The regex pattern must match the whole string representation of the primitive value from start to end, i.e. for tool implementation with a typical regex library, the pattern should be enclosed by ^ (first character) and $ (last character) to match the whole input string.

[constr_2534] Limits of unlimited Integer ⌈Practically `UnlimitedInteger` shall be limited such that it fits into 64 bit.

If a signed value is represented the min value can be down to -9223372036854775808 (0x800000000000000014) and the max value can be up to 9223372036854775807 (`0x7fffffffffffffff`).

If an unsigned value is represented the min value can be down to 0 and the max value can be up to 18446744073709551615 (`0xffffffffffffffff`).⌋*()*

[TPS_GST_02501] Compatibility of Numerical Values ⌈Compatibility of numerical values (in particular `Float`, `Numerical`, `PositiveInteger`, `UnlimitedInteger`) is defined independent of the representation (float, integer.octal/hex/binary/decimal) as:

$v1$ and $v2$ are compatible if and only if $abs(v1 - v2) < epsilon$⌋*()*

[constr_2631] Usage of value ANY for `AnyServiceInstanceId` ⌈The value of a given `AnyServiceInstanceId` shall not be set to ANY.⌋*()*

Note: The value ANY for primitive meta-class `AnyServiceInstanceId` is only included in the xml.xsd.pattern for backwards-compatibility reasons. The same applies to the naming of the meta-class, i.e. the prefix "Any". It would, from the perspective

of the AUTOSAR meta-model, have been cleaner to set the atp.Status of the existing meta-class to removed and create a new one that properly advertises the "ALL" semantics. But this change would have an impact on existing ARXML files and require a model migration.

The following list is collecting all those meta-classes which are not explicitly mentioned in other sections of this document.

Primitive	**Address**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	This is used to specify an address within the CPU. **Tags:** xml.xsd.customType=ADDRESS xml.xsd.pattern=0x[0-9a-z]* xml.xsd.type=string

Table 4.40: Address

Primitive	**AlignmentType**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	This primitive represents the alignment of objects within a memory section. The value is in number of bits or UNKNOWN (deprecated), 8 , 16, 32 UNSPECIFIED, BOOLEAN, or PTR. Typical values for numbers are 8, 16, 32. **Tags:** xml.xsd.customType=ALIGNMENT-TYPE xml.xsd.pattern=[1-9][0-9]*|0[xX][0-9a-fA-F]*|0[bB][0-1]+|0[0-7]*|UNSPECIFIED|UNKNOWN|BOOLEAN|PTR xml.xsd.type=string

Table 4.41: AlignmentType

Primitive	**AnyVersionString**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	**Tags:** xml.xsd.customType=ANY-VERSION-STRING xml.xsd.pattern=[0-9]+|ANY xml.xsd.type=string

Table 4.42: AnyVersionString

Primitive	**AsamRecordLayoutSemantics**
Package	M2::MSR::DataDictionary::RecordLayout
Note	This meta-class is used to denote the semantics in particular in terms of the corresponding A2L-Keyword. This is to support the mapping of the more general record layouts in AUTOSAR/MSR to the specific A2L keywords. It is possible to express the specific semantics of A2l RecordLayout keywords in SwRecordlayoutGroup but not always vice versa. Therefore the mapping is provided in this optional attribute. It is specified as NMTOKEN to reduce the direct dependency of ASAM an AUTOSAR standards. **Tags:** xml.xsd.customType=ASAM-RECORD-LAYOUT-SEMANTICS xml.xsd.type=NMTOKEN

Table 4.43: AsamRecordLayoutSemantics

Primitive	**AxisIndexType**
Package	M2::MSR::DataDictionary::RecordLayout
Note	This meta-class specifies an axis in a curve/map data object. The index satisfies the following convention: • 0 output "axis" • 1 input axis 1 (X input axis e.g. of a CURVE) • 2 input axis 2 (Y input axis e.g. of a MAP) • 3 input axis 3 (Z input axis e.g. of a CUBOID) • 4 input axis 3 (Z4 input axis e.g. of a CUBE_4) • 5 input axis 3 (Z5 input axis e.g. of a CUBE_5) • 6..9 etc. The output "axis" provides access to the output value of the parameter. Note that this access is usually performed via an index according to the input axis. In addition to this, the Values STRING and ARRAY support specific iterations. **Tags:** xml.xsd.customType=AXIS-INDEX-TYPE xml.xsd.pattern=[0-9]+|STRING|ARRAY xml.xsd.type=string

Table 4.44: AxisIndexType

Primitive	**BaseTypeEncodingString**
Package	M2::MSR::AsamHdo::BaseTypes
Note	This is the string denotion of a BaseType encoding. It may be refined by specific use-cases. **Tags:** xml.xsd.customType=BASE-TYPE-ENCODING-STRING xml.xsd.type=string

Table 4.45: BaseTypeEncodingString

Primitive	**CIdentifier**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes			
Note	This datatype represents a string, that follows the rules of C-identifiers. **Tags:** xml.xsd.customType=C-IDENTIFIER xml.xsd.pattern=[a-zA-Z_][a-zA-Z0-9_]* xml.xsd.type=string			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
blueprintValue	String	1	attr	This represents a description that documents how the value shall be defined when deriving objects from the blueprint. **Tags:** atp.Status=draft xml.attribute=true
namePattern	String	0..1	attr	This attribute represents a pattern which shall be used to define the value of the identifier if the CIdentifier in question is part of a blueprint. For more details refer to TPS_StandardizationTemplate. **Tags:**xml.attribute=true

Table 4.46: CIdentifier

Primitive	**CIdentifierWithIndex**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	This datatype represents a string, that follows the rules of C-identifiers with an index. **Tags:** xml.xsd.customType=C-IDENTIFIER-WITH-INDEX xml.xsd.pattern=[a-zA-Z_][a-zA-Z0-9_]*\[[0-9]+\] xml.xsd.type=string

Table 4.47: CIdentifierWithIndex

Primitive	**CategoryString**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	This represents the pattern applicable to categories. It is basically the same as Identifier but has a different semantics. Therefore it is modeled as a primitive of its own. **Tags:** xml.xsd.customType=CATEGORY-STRING xml.xsd.pattern=[a-zA-Z][a-zA-Z0-9_]* xml.xsd.type=string

Table 4.48: CategoryString

Primitive	**DateTime**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	A datatype representing a timestamp. The smallest granularity is 1 second. This datatype represents a timestamp in the format yyyy-mm-dd followed by an optional time. The lead-in character for the time is "T" and the format is hh:mm:ss. In addition, a time zone designator shall be specified. The time zone designator can either be "Z" (for UTC) or the time offset to UTC, i.e. (+|-)hh:mm. Examples: 2009-07-23 2009-07-23T14:38:00+01:00 2009-07-23T13:38:00Z **Tags:** xml.xsd.customType=DATE xml.xsd.pattern=([0-9]{4}-[0-9]{2}-[0-9]{2})(T[0-9]{2}:[0-9]{2}:[0-9]{2}(Z|([+\-][0-9]{2}:[0-9]{2})))? xml.xsd.type=string

Table 4.49: DateTime

Primitive	**DiagRequirementIdString**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	This string denotes an Identifier for a requirement. **Tags:** xml.xsd.customType=DIAG-REQUIREMENT-ID-STRING xml.xsd.pattern=[0-9a-zA-Z_\-]+ xml.xsd.type=string

Table 4.50: DiagRequirementIdString

Primitive	**DisplayFormatString**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	This is a display format specifier for the display of values e.g. in documents or in measurement and calibration systems. The display format specifier is a subset of the ANSI C printf specifiers with the following form: % [flags] [width] [.prec] type character For more details refer to "ASAM-HarmonizedDataObjects-V1.1.pdf" chapter 13.3.2 DISPLAY OF DATA. Due to the numerical nature of value settings, only the following type characters are allowed: • d: Signed decimal integer • i: Signed decimal integer • o: Unsigned octal integer • u: Unsigned decimal integer • x: Unsigned hexadecimal integer, using "abcdef" • X: Unsigned hexadecimal integer, using "ABCDEF" • e: Signed value having the form [-]d.dddd e [sign]ddd where d is a single decimal digit, dddd is one or more decimal digits, ddd is exactly three decimal digits, and sign is + or - • E: Identical to the e format except that E rather than e introduces the exponent • f: Signed value having the form [-]dddd.dddd, where dddd is one or more decimal digits; the number of digits before the decimal point depends on the magnitude of the number, and the number of digits after the decimal point depends on the requested precision • g: Signed value printed in f or e format, whichever is more compact for the given value and precision; trailing zeros are truncated, and the decimal point appears only if one or more digits follow it • G: Identical to the g format, except that E, rather than e, introduces the exponent (where appropriate) **Tags:** xml.xsd.customType=DISPLAY-FORMAT-STRING xml.xsd.pattern=%[\-+#]?[0-9]*(\.[0-9]+)?[diouxXfeEgGcs] xml.xsd.type=string

Table 4.51: DisplayFormatString

Primitive	**Ip4AddressString**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	This is used to specify an IP4 address. Notation: 255.255.255.255 **Tags:** xml.xsd.customType=IP4-ADDRESS-STRING xml.xsd.pattern=(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)|ANY xml.xsd.type=string

Table 4.52: Ip4AddressString

Primitive	**Ip6AddressString**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes

▽

△

Primitive	**Ip6AddressString**
Note	This is used to specify an IP6 address. Notation: FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF Alternative notations, short-cuts with duplicate colons like ::, etc. or mixtures using colons and dots, are not allowed. **Tags:** xml.xsd.customType=IP6-ADDRESS-STRING xml.xsd.pattern=[0-9A-Fa-f]{1,4}(:[0-9A-Fa-f]{1,4}){7,7}|ANY xml.xsd.type=string

Table 4.53: Ip6AddressString

Primitive	**MacAddressString**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	This primitive specifies a Mac Address. Notation: FF:FF:FF:FF:FF:FF Alternative notations, e.g. using dash instead of colon, or another grouping of numbers, is not allowed. **Tags:** xml.xsd.customType=MAC-ADDRESS-STRING xml.xsd.pattern=([0-9a-fA-F]{2}:){5}[0-9a-fA-F]{2} xml.xsd.type=string

Table 4.54: MacAddressString

Primitive	**McdIdentifier**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	This primitive denotes a name used for measurement and calibration systems and shall follow the restrictions for an ASAM ASAP2 ident. For detailed syntax see the xsd.pattern. The size limitations are not captured. McdIdentifiers are random names which may contain characters A through Z, a through z, underscore (_), numerals 0 through 9, points ('.') and brackets ('[',']'). However, the following limitations apply: the first character shall be a letter or an underscore, brackets shall occur in pairs at the end of a partial string and shall contain a number or an alpha-numerical string (description of the index of an array element). **Tags:** xml.xsd.customType=MCD-IDENTIFIER xml.xsd.pattern=[a-zA-Z_][a-zA-Z0-9_]*(\[([a-zA-Z_][a-zA-Z0-9_]*|[0-9]+)\])*(\.[a-zA-Z_][a-z A-Z0-9_]*(\[([a-zA-Z_][a-zA-Z0-9_]*|[0-9]+)\])*)* xml.xsd.type=string

Table 4.55: McdIdentifier

Primitive	**MimeTypeString**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	This primitive denotes the an Internet media type, originally called a MIME type after MIME and sometimes a Content-type after the name of a header in several protocols whose value is such a type, is a two-part identifier for file formats on the Internet. **Tags:** xml.xsd.customType=MIME-TYPE-STRING xml.xsd.type=string

Table 4.56: MimeTypeString

Primitive	**NameTokens**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	This is a white-space separated list of name tokens. **Tags:** xml.xsd.customType=NMTOKENS-STRING xml.xsd.type=NMTOKENS

Table 4.57: NameTokens

Primitive	**NativeDeclarationString**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	This string contains a native data declaration of a data type in a programming language. It is basically a string, but white-space shall be preserved. **Tags:** xml.xsd.customType=NATIVE-DECLARATION-STRING xml.xsd.type=string xml.xsd.whiteSpace=preserve

Table 4.58: NativeDeclarationString

Primitive	**PositiveUnlimitedInteger**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	This is a positive unlimited integer which can be denoted in decimal, binary, octal and hexadecimal. **Tags:** xml.xsd.customType=POSITIVE-UNLIMITED-INTEGER xml.xsd.pattern=0|[\+]?[1-9][0-9]*|0[xX][0-9a-fA-F]+|0[bB][0-1]+|0[0-7]+ xml.xsd.type=string

Table 4.59: PositiveUnlimitedInteger

Primitive	**PrimitiveIdentifier**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	This meta-class has the ability to contain a string. Please note that this meta-class has only been introduced to fix an issue with the generation of attributes on primitives in context with [TR_APRXML_00024]. **Tags:** xml.xsd.customType=PRIMITIVE-IDENTIFIER xml.xsd.maxLength=128 xml.xsd.pattern=[a-zA-Z]([a-zA-Z0-9]|_[a-zA-Z0-9])*_? xml.xsd.type=string

Table 4.60: PrimitiveIdentifier

Primitive	**RecordLayoutIteratorPoint**
Package	M2::MSR::DataDictionary::RecordLayout

△

Primitive	**RecordLayoutIteratorPoint**
Note	This meta-class denotes a start / endpoint for the iteration of a SwRecordLayoutGroup. It can be an integer or one of the keywords MAX-TEXT-SIZE|ARRAY-SIZE. Note that negative numbers are counted backwards. Therefore e.g. -1 refers to the last value. **Tags:** xml.xsd.customType=RECORD-LAYOUT-ITERATOR-POINT xml.xsd.pattern=-?([0-9]+|MAX-TEXT-SIZE|ARRAY-SIZE) xml.xsd.type=string

Table 4.61: RecordLayoutIteratorPoint

Primitive	**RegularExpression**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	This is a regular expression as defined in http://www.w3.org/TR/xmlschema-2 As of now it is still produced as a string in XSD. **Tags:** xml.xsd.customType=REGULAR-EXPRESSION xml.xsd.type=string

Table 4.62: RegularExpression

Primitive	**RevisionLabelString**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	This primitive represents an internal AUTOSAR revision label which identifies an engineering object. It represents a pattern which • supports three integers representing from left to right MajorVersion, MinorVersion, PatchVersion. • may add an application specific suffix separated by one of ".", "_", ";". Legal patterns are for example: • 4.0.0 • 4.0.0.1234565 • 4.0.0_vendor specific;13 • 4.0.0;12 **Tags:** xml.xsd.customType=REVISION-LABEL-STRING xml.xsd.pattern=[0-9]+\.[0-9]+\.[0-9]+([\._;].*)? xml.xsd.type=string

Table 4.63: RevisionLabelString

Primitive	**SectionInitializationPolicyType**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes

▽

△

Primitive	**SectionInitializationPolicyType**
Note	SectionInitializationPolicyType describes the intended initialization of MemorySections. The following values are standardized in AUTOSAR Methodology: • **INIT**: To be used for (explicitly or not explicitly) initialized variables. • **CLEARED**: To be used for not explicitly initialized variables. • **POWER-ON-CLEARED**: To be used for variables that are not explicitly initialized (cleared) during normal start-up. Instead these are cleared only after power on reset. Please note that the values are defined similar to the representation of enumeration types in the XML schema to ensure backward compatibility. **Tags:** xml.xsd.customType=SECTION-INITIALIZATION-POLICY-TYPE xml.xsd.type=NMTOKEN

Table 4.64: SectionInitializationPolicyType

Primitive	**String**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	This represents a String in which white-space shall be normalized before processing. For example: in order to compare two Strings: • leading and trailing white-space needs to be removed • consecutive white-space (blank, cr, lf, tab) needs to be replaced by one blank. **Tags:** xml.xsd.customType=STRING xml.xsd.type=string

Table 4.65: String

Primitive	**StrongRevisionLabelString**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	This primitive represents a revision label which identifies an object under version control. It represents a pattern which requires three integer numbers separated by a dot, representing from left to right Major Version, MinorVersion, PatchVersion and additional labels for pre-release version and build metadata. Legal patterns are for example: 1.0.0-alpha+001 1.0.0+20130313144700 1.0.0-beta+exp.sha.5114f85 **Tags:** atp.Status=draft xml.xsd.customType=STRONG-REVISION-LABEL-STRING xml.xsd.pattern=(0|[1-9]\d*)\.(0|[1-9]\d*)\.(0|[1-9]\d*)(-((0|[1-9]\d*|\d*[a-zA-Z-][0-9a-z A-Z-]*)(\.(0|[1-9]\d*|\d*[a-zA-Z-][0-9a-zA-Z-]*))*))?(\+([0-9a-zA-Z-]+(\.[0-9a-zA-Z-]+)*))? xml.xsd.type=string

Table 4.66: StrongRevisionLabelString

Primitive	**SymbolString**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	This meta-class has the ability to contain a string plus an additional namePattern. Please note that this meta-class has only been introduced to fix an issue with the backwards compatibility between R4.0.3 and R4.1.1 in the context of McDataInstance **Tags:** xml.xsd.customType=SYMBOL-STRING xml.xsd.type=string

▽

△

Primitive	**SymbolString**			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
namePattern	String	1	attr	This attribute represents a pattern which shall be used to define the value of the identifier if the CIdentifier in question is part of a blueprint. For more details refer to TPS_StandardizationTemplate. **Tags:**xml.attribute=true

Table 4.67: SymbolString

Primitive	**UriString**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	A Uniform Resource Identifier (URI), is a compact string of characters used to identify or name a resource. **Tags:** xml.xsd.customType=URI-STRING xml.xsd.type=string

Table 4.68: UriString

Primitive	**VerbatimString**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes			
Note	This primitive represents a string in which white-space needs to be preserved. **Tags:** xml.xsd.customType=VERBATIM-STRING xml.xsd.type=string xml.xsd.whiteSpace=preserve			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
blueprintValue	String	0..1	attr	This represents a description that documents how the value shall be defined when deriving objects from the blueprint. **Tags:** atp.Status=draft xml.attribute=true
xmlSpace	XmlSpaceEnum	0..1	attr	This attribute is used to signal an intention that in that element, white space should be preserved by applications. It is defined according to xml:space as declared by W3C. **Tags:** xml.attribute=true xml.attributeRef=true xml.name=space xml.nsPrefix=xml

Table 4.69: VerbatimString

Primitive	**VerbatimStringPlain**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes

▽

△

Primitive	**VerbatimStringPlain**
Note	This primitive represents a string in which white-space needs to be preserved. This primitive is applied in cases where xml:space attribute cannot be provided by the primitive type but needs to be provided by the container class. This is in particular the case in applications of TR_APRXML_00024. **Tags:** xml.xsd.customType=VERBATIM-STRING-PLAIN xml.xsd.type=string xml.xsd.whiteSpace=preserve

Table 4.70: VerbatimStringPlain

4.8 Formula Language

This chapter details the introduction of a general purpose formula language. The formula language can be used in different processing steps in the methodology, e.g. XML-processors, C preprocessor, Modeling tools.

Core features of formula language are:

- The formula language always yields numerical.
- It is formaly defined in ANTLR [12].
- Provides well-defined extension points.
- The formula language can handle:
  - numerical literals, string literals
  - values provided by references to autosar model element e.g. sysc, ecuc, supported references as specified in the metamodel
  - commonly used operators $+$, $-$, etc. (see section 4.8.2.1)
  - hardcoded function `sin`, etc. (extension point, see section 4.8.2.2)
  - string - comparisons see [TPS_GST_00146], [TPS_GST_00147]
  - special values such as `undefined` [TPS_GST_00010] `infinite` [TPS_GST_00275]

4.8.1 Applying Formula Language

Until Release 3 the AUTOSAR artifacts could not express dependencies, i.e. calculate the value of one parameter based on other parameter values, or define values based on variant information. Each of these use cases is represented as a specialization of the abstract meta-class `FormulaExpression` as shown in figure 4.8.

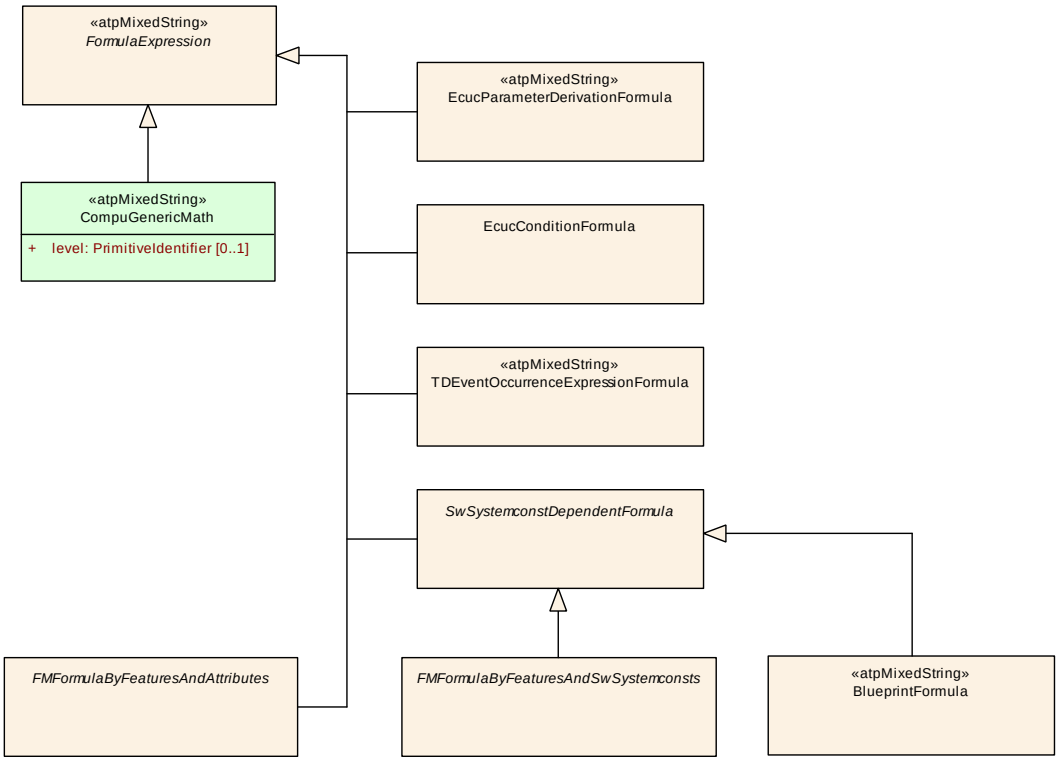


Figure 4.8: Formula language class hierarchy

[TPS_GST_00355] Specialization of `FormulaExpression` ⌈These specializations represent three kinds of extensions:

- The applicable operands are specified as associations in the subclasses (see [TPS_GST_00001] for more details). The valid `reference` in the grammar below are taken from the role names in the meta-model. Maintaining the references as formal associations allows to retrieve dependencies even without parsing the formula expressions.
- Additional functions are represented by the specialization and documented in the context of the same (see [TPS_GST_00293] for more details). It is not possible to extend the language by further operators.
- The particular application constrains the expected result. E.g. in `AttributeValueVariationPoint`, the result is given by the type of the variant attribute. Another example is `swSyscond` which expects a boolean result.

⌋*()* An example of applicable operands is given in figure 4.10.

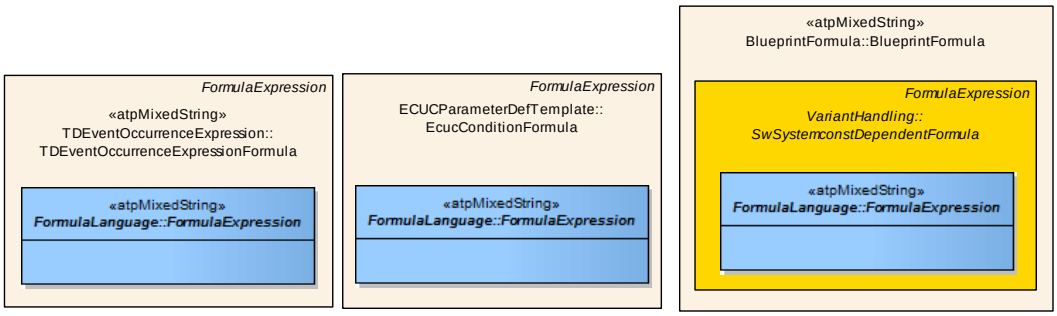


Figure 4.9: Formula language family

The formula language defined here is the core part of the specialization, blue part in figure 4.9.

Please note that `FormulaExpression` is ≪`atpMixedString`≫ (see Chapter 2.3.1). Therefore one expression can e.g. be dependent on multiple `SwSystemconst`s.

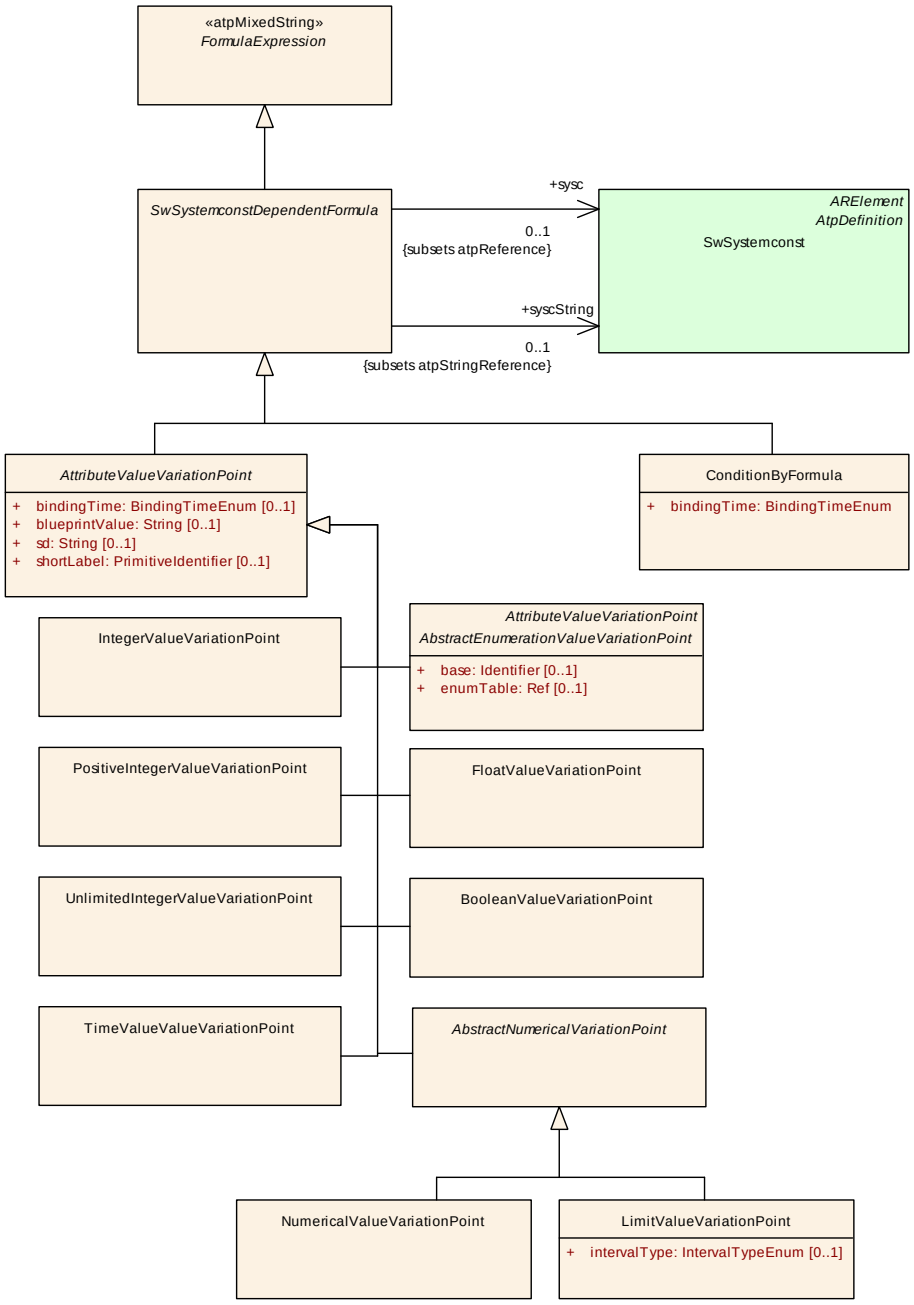


Figure 4.10: Formula depending on `SwSystemconst`

4.8.2 Formula Language Definition

The connection between the formula and referenced meta-model objects acting as operands is established by associations in the meta-model and represented in the grammar as rule named `reference` and `stringReference` (see [TPS_GST_00001]). Therefore `FormulaExpression`s are «`atpMixedString`>`s` when serialized in ARXML. The grammar for the expressions, however, is defined after the processing of the mixed strings by a XML Parser, i.e. they no longer contain any

XML-Tags, -Entities, -Comments, etc.. This is done to not "mimic" XML in the grammar but to focus on the expressions itself.

The following rules shall be applied to come from the ARXML representation to the string that is then subject to the definition by the grammar:

- XML-Entities are replaced by the UTF characters they stand for (e.g. "<" is replaced by "<")
- XML-Comments are removed
- ARXML References are replaced by:
  - `reference`("{value of the DEST attribute}:{text content of the tag}") if they are subsets of the `+/atpReference` Association between `FormulaExpression` and `Referrable`
  - `stringReference`("{value of the DEST attribute}:{text content of the tag}") if they are subsets of the `+/atpStringReference` Association between `FormulaExpression` and `Referrable`

E.g. consider the following ARXML snippet:

```
<VARIATION-POINT>
  <SW-SYSCOND>
    defined(<SYSC-REF DEST="SYSTEM-CONSTANT">SY_COUNT<SYSC-REF>)
    <!-- this is a comment -->
    &amp;&amp; <SYSC-REF DEST="SYSTEM-CONSTANT">SY_COUNT<SYSC-REF> &lt; 10
  </SW-SYSCOND>
<VARIATION-POINT>
```

The `FormulaExpression` is serialized in ARXML in the «`atpMixedStrig`» Tag `<SW-SYSCOND>`. It is transformed by the above rules into:

```
   defined(reference("SW-SYSTEMCONST:SY_COUNT") &&
           reference("SW-SYSTEMCONST:SY_COUNT") < 10
```

This `FormulaExpression` is then considered as well-formed according to the syntax defined by the formula language grammar.

Use case specific extensions may be applied by redefining particular token definitions according to [TPS_GST_00293].

[TPS_GST_00012] AUTOSAR Formula language ⌈The AUTOSAR formula language uses the following syntax described in Listing 4.14 defined according to [12].⌋*()*

Listing 4.14: AUTOSAR Formula language

```
grammar autosarFormulaLanguage;

expr
   : atomExpr ;

atomExpr
```

```
    : condExpr ;

condExpr
    : orExpr (CondOperator orExpr AltOperator orExpr)? ;

orExpr
    : xorExpr (OrOperator xorExpr)* ;

xorExpr
    : andExpr (XorOperator andExpr)* ;

andExpr
    : bitOrExpr (AndOperator bitOrExpr)* ;

bitOrExpr
    : bitXorExpr (BitOrOperator bitXorExpr)* ;

bitXorExpr
    : bitAndExpr (BitXorOperator bitAndExpr)* ;

bitAndExpr
    : compEqExpr (BitAndOperator compEqExpr)* ;

compEqExpr
    : compExpr (CompEqOperator compExpr)* ;

compExpr
    : shiftExpr (CompOperator shiftExpr)* ;

shiftExpr
    : sumExpr (ShiftOperator sumExpr)* ;

sumExpr
    : mulExpr (SumOperator mulExpr)* ;

mulExpr
    : powExpr (MulOperator powExpr)* ;

powExpr
    : unaryExpr (PowOperator unaryExpr)? ;

unaryExpr
    : unaryOperator? atom;

atom
    : DecIntegerLiteral
    | reference
    | BooleanLiteral
    | HexIntegerLiteral
    | OctIntegerLiteral
    | BinIntegerLiteral
    | DecimalLiteral
    | DoubleLiteral
    | ArgumentOperand
    | DefinedFuncName LPAREN (reference | stringReference)  RPAREN
    | StringFuncName LPAREN stringArg COMMA stringArg RPAREN
```

```
    | BinaryFuncName LPAREN atomExpr COMMA atomExpr RPAREN
    | UnaryFuncName LPAREN atomExpr RPAREN
    | Keyword
    | (LPAREN atomExpr RPAREN)

    // Here the supported extension points are listed:
    // Depending on which of the extension points are defined in a
    // spcialisation of the formula language, the following one,
    // two or all of the following alternatives are added to this rule:
    // | ExtBinaryFuncName LPAREN atomExpr COMMA atomExpr RPAREN
    // | ExtUnaryFuncName LPAREN atomExpr RPAREN
    // | ExtKeyword
    // end of extension points
    ;

stringArg
    : stringReference | StringLiteral ;

unaryOperator
    : SumOperator | OtherUnaryOperator ;

//-----------------------------------------------------------------------
// these rules represent the reference to operands
// in an XML-Arttifact, this is represented as XML-Artifact
//
// Applicable RefFuncNames depend on the associations in the
// particular specialization of FormulaLanguage in the
// metamodel
// see [TPS_GST_00001]

stringReference
    : StringReferenceFuncName LPAREN StringLiteral RPAREN ;

reference
    : ReferenceFuncName LPAREN StringLiteral RPAREN ;

// argumentOperand is valid only in formula derived from CompuGenericMath

ArgumentOperand
    : 'X' ('1' .. '9')('0' .. '9')* ;


// Tokens
OrOperator
    : '||';

XorOperator
    : '^^' ;

AndOperator
    : '&&';

BitOrOperator
    : '|';

BitXorOperator
```

```
    : '^';

BitAndOperator
    : '&';

CompEqOperator
    : '==' | '!=';

CompOperator
    : '<=' | '<' | '>=' | '>' ;

ShiftOperator
    : '<<' | '>>' ;

SumOperator
    : '+' | '-' ;

OtherUnaryOperator
    : '!' | '~' ;

MulOperator
    : '*' | '/' | '%' ;

PowOperator
    : '**' ;

CondOperator
    : '?';

AltOperator
    : ':';

DefinedFuncName
    : 'defined' ;

StringFuncName
    : 'streqcs' | 'streqci';

UnaryFuncName
    : 'round'
    | 'ceil'
    | 'floor'
    | 'abs'
    | 'log'
    | 'exp'
    | 'sin'
    | 'cos'
    | 'asin'
    | 'acos'
    | 'atan'
    | 'sinh'
    | 'cosh'
    | 'tan'
    | 'tanh'
    | 'sqrt'
    | 'log10'
```

```
    | 'sgn'
    ;


ReferenceFuncName : 'reference' ;

StringReferenceFuncName : 'stringReference' ;

BinaryFuncName
    : 'max'
    | 'min'
    | 'pow'
    ;

Keyword
   : 'epsilon'
   | 'undefined'
     ;

////////////////////////////////////////////////////////////////////
//Here the definitions for the extenstion points would be included
////////////////////////////////////////////////////////////////////


BooleanLiteral
     : 'true'
     | 'false'
     ;

LPAREN : '(' ;
RPAREN : ')' ;
COMMA  : ',' ;


//
    ------------------------------------------------------------------------------


DecIntegerLiteral
    : '0'
    | ('1'..'9')('0'..'9')*
    ;

HexIntegerLiteral
    : '0' ('x' | 'X') (('0'..'9') | ('a'..'f') | ('A'..'F'))+ ;

OctIntegerLiteral
    : '0'('0'..'7')+ ;

BinIntegerLiteral
    : '0' ('b' | 'B') ('0' ..'1')+ ;

DecimalLiteral
    :  ('0'? '.' ('0'..'9')+) |  (('1'..'9')('0'..'9')* ('.' ('0'..'9')*)) ;

DoubleLiteral
```

```
    :  ((('0'? '.' ('0'..'9')+) |  (('1'..'9')('0'..'9')* ('.' ('0'..'9')*)
       ?))
       ('e' | 'E') ('+' | '-')? ('0'..'9')+)
    | 'NaN'
    | 'INF'
    ;

// The next rule defines StringLiterals as double quoted sequences of
// (almost) arbitrary characters with the backslash as escape character.
// Please notice that the tilde is used in the ANTLR syntax to denote
// negation and it is necessary to escape the backslash to match itself.

StringLiteral
    : '"' ((~('"'|'\\') | '\\' ~('\\') | '\\\\')*) '"'
    ;

WS : (' ' | '\t' | '\r' | '\n' )+ {$channel=HIDDEN;}
    ;
```

4.8.2.1 Operators in arithmetic expressions

The following operators are supported in arithmetic expressions:

- **[TPS_GST_00111] Negation Operator** ⌈(yields boolean, bit-wise negation)⌋*()*
  Symbol: !, ~ Return Types see [TPS_GST_00039]
- **[TPS_GST_00112] Exponentiation Operator** ⌈yields operand 1 power operand 2⌋*()*
  Symbol: ** Return Types see [TPS_GST_00035]
- **[TPS_GST_00113] Multiplicative Operator / division** ⌈yields Multiplication, division, modulo⌋*()*
  Symbol: *, /, % Return Types see [TPS_GST_00035]
- **[TPS_GST_00114] Additive Operator** ⌈Yields Addition, subtraction, sign⌋*()*
  Symbol: +, - Return Types see [TPS_GST_00034]
- **[TPS_GST_00115] Shift Operator** ⌈Yields bit-wise shift (left, right) note that the ends are filled with "0".⌋*()*
  Symbol: <<,>> Return Types see [TPS_GST_00037]
- **[TPS_GST_00116] Ranking Operator** ⌈Yields comparison: less than, less than or equal to, greater than, greater than or equal to⌋*()*
  Symbol: <, <=, >, >= Return Types see [TPS_GST_00036]
- **[TPS_GST_00117] Comparison: equality** ⌈Yields equal, unequal to⌋*()*
  Symbol: ==, != Return Types see [TPS_GST_00036]
- **[TPS_GST_00118] Bit-wise AND** ⌈Yields the bit wise and of the operators⌋*()*
  Symbol: & Return Types see [TPS_GST_00037]

- **[TPS_GST_00119] Bit-wise XOR** ⌈Yields the bit wise xor of the operators⌋*()*
  Symbol:ˆReturn Type see [TPS_GST_00037]
- **[TPS_GST_00120] Bit-wise OR** ⌈Yields the bit wise or of the operators⌋*()*
  Symbol: |Return Types see [TPS_GST_00037]
- **[TPS_GST_00121] Boolean AND** ⌈Yields a boolean AND with operand value 0 -> false, others -> true⌋*()*
  Symbol: && Return Types see [TPS_GST_00036]
- **[TPS_GST_00122] Boolean XOR** ⌈Yields a boolean XOR with operand value 0 -> false, others -> true⌋*()*
  Symbol:ˆˆReturn Types see [TPS_GST_00036]
- **[TPS_GST_00123] Boolean OR** ⌈Yields a boolean OR with operand value 0 -> false, others -> true⌋*()*
  Symbol: ||Return Types see [TPS_GST_00036]

4.8.2.2 Mathematical functions in arithmetic expressions

The following mathematical functions are supported in arithmetic expressions:

- **[TPS_GST_00124] Round Function** ⌈This rounds positive and negative numbers to the nearest whole number. Note that when processing such expressions, that `round` is defined for the value ranges -2147483648 to +4294967295.⌋*()*
  Function: *round* Param.: *1* Type of result: *integer*
- **[TPS_GST_00125] Round Up Function** ⌈This rounds positive and negative numbers up to the next whole number. Note that when processing such expressions, that `ceil` is defined for the value ranges -2147483648 to +4294967295.⌋*()*
  Function: *ceil* Param.: *1* Type of result: *integer*
- **[TPS_GST_00126] Round Down Function** ⌈This rounds positive and negative numbers down to the next whole number. Note that when processing such expressions, that `floor` is defined for the value ranges -2147483648 to +4294967295.⌋*()*
  Function: *floor* Param.: *1* Type of result: *integer*
- **[TPS_GST_00127] Absolute Value** ⌈This yields the absolute value of the operand.⌋*()*
  Function: *abs* Param.: *1* Type of result: *like operand*
- **[TPS_GST_00128] Natural Logarithm** ⌈This yields the natural logarithm (base e) of the argument.⌋*()*
  Function: *log* Param.: *1* Type of result: *float*

- **[TPS_GST_00129] Decimal Logarithm** ⌈This yields the logarithm base 10 - provided for A2L 1.6.⌋*()*
  Function: *log10* Param.: *1* Type of result: *float*
- **[TPS_GST_00130] Square Root** ⌈This yields the square root - provided for A2L 1.6.⌋*()*
  Function: *sqrt* Param.: *1* Type of result: *float*
- **[TPS_GST_00131] Sinus** ⌈This yields sinus - provided for A2L 1.6.⌋*()*
  Function: *sin* Param.: *1* Type of result: *float*
- **[TPS_GST_00132] Arcus Sinus** ⌈This yields arcus sinus - provided for A2L 1.6.⌋*()*
  Function: *asin* Param.: *1* Type of result: *float*
- **[TPS_GST_00133] Cosinus** ⌈This yields cosinus - provided for A2L 1.6.⌋*()*
  Function: *cos* Param.: *1* Type of result: *float*
- **[TPS_GST_00134] Arcus Cosinus** ⌈This yields arcus cosinus - provided for A2L 1.6.⌋*()*
  Function: *acos* Param.: *1* Type of result: *float*
- **[TPS_GST_00135] Sinus Hyperbolicus** ⌈This yields sinus hyperbolicus - provided for A2L 1.6.⌋*()*
  Function: *sinh* Param.: *1* Type of result: *float*
- **[TPS_GST_00136] Cosinus Hyperbolicus** ⌈This yields cosinus hyperbolicus - provided for A2L 1.6.⌋*()*
  Function: *cosh* Param.: *1* Type of result: *float*
- **[TPS_GST_00137] Tangens** ⌈This yields tangens - provided for A2L 1.6.⌋*()*
  Function: *tan* Param.: *1* Type of result: *float*
- **[TPS_GST_00138] Arcus Tangens** ⌈This yields arcus tangens - provided for A2L 1.6.⌋*()*
  Function: *atan* Param.: *1* Type of result: *float*
- **[TPS_GST_00139] Tangens Hyperbolicus** ⌈This yields tangens hyperbolicus - provided for A2L 1.6.⌋*()*
  Function: *tanh* Param.: *1* Type of result: *float*
- **[TPS_GST_00140] Exponential** ⌈This yields exponential function (base e).⌋*()*
  Function: *exp* Param.: *1* Type of result: *float*
- **[TPS_GST_00141] Is Defined** ⌈This checks whether the reference given as the argument is defined.⌋*()*
  Function: *defined* Param.: *1* Type of result: *0 or 1*
- **[TPS_GST_00142] Signum** ⌈This yields signum, result is one of -1, 0, +1.⌋*()*
  Function: *sgn* Param.: *1* Type of result: *integer*

- **[TPS_GST_00143] Maximum Value** ⌈This finds the maximum value of the arguments.⌋*()*
  Function: *max* Param.: *2* Type of result: *depends on operands*
- **[TPS_GST_00144] Minium Value** ⌈This finds the minimum value of the arguments.⌋*()*
  Function: *min* Param.: *2* Type of result: *depends on operands*
- **[TPS_GST_00145] Power Function** ⌈This yields argument 1 power argument 2 - provided for A2L 1.6. This is equivalent to [TPS_GST_00112].⌋*()*
  Function: *pow* Param.: *2* Type of result: *depends on arguments*
  Result type follows [TPS_GST_00035].
- **[TPS_GST_00146] Case Sensitive String Compare** ⌈This compares two strings in case sensitive manner.⌋*()*
  Function: *streqcs* Param.: *2* Type of result: *0 or 1*
- **[TPS_GST_00147] Non Case Insensitive String Compare** ⌈This compares two strings in non case sensitive manner.⌋*()*
  Function: *streqci* Param.: *2* Type of result: *0 or 1*

4.8.2.3 Implementation details of a Formula Processor

The following implementation details apply:

- **[TPS_GST_00001] Connection between Formula and Model Elements** ⌈The formula language mentioned above has production rules (**`reference`**, **`stringReference`**) which are defined as

  ```
  stringReference
     :      'stringReference' LPAREN StringLiteral RPAREN;

  reference
     :      'reference' LPAREN StringLiteral RPAREN;
  ```

  This production indicates, that at this point, a reference to a model element needs to be resolved. It is not allowed to accept AUTOSAR artifacts which contain such a textual representation.⌋*()*
- **[TPS_GST_00293] Use Case Specific Extension of Formula Language** ⌈A use case specific extension is indicated in the meta-model by a specialization of `FormulaExpression`. The semantics of the extension is specified in the context of this specialization as a fragment of an ANTLR [12] specification redefining the following Tokens:
  - `ExtKeyword` - this provides additional keywords similar to `epsilon`
  - `ExtUnaryFuncName` - this provides additional names for unary functions
  - `ExtBinaryFuncName` - this provides additional names for binary functions

It is not possible to extend the formula language by additional operators.⌋()

Thus the list of functions in chapter 4.8.2.2 is extended. An example for the definition of additional Keywords based on an unary function is given in Listing 4.15.

Listing 4.15: AUTOSAR Formula language extension

```
ExtUnaryFuncName  :  'TIMEX_value' |
                     'TIMEX_occurs' |
                     'TIMEX_hasOccurred' |
                     'TIMEX_timeSinceLastOccurrence' |
                     'TIMEX_angleSinceLastOccurrence'
                     ;
```

- **[TPS_GST_00015] result of reference/stringReference** ⌈
  - `reference` shall yield a numerical / boolean value.
  - `stringReference` shall yield a string value.

  ⌋()
- **[TPS_GST_00002] aborting logical expressions** ⌈The calculation of expressions with boolean AND or OR is aborted if the first operand has produced a result so that the total result cannot be changed anymore by the second operand (as in C).⌋()

  Note: This is useful e.g. in the context of `SwSystemconstDependentFormula`, a specialization of the formula language. This behavior gives meaningful results in expressions such as:

```
defined(reference("SW-SYSTEMCONST:SY_COUNT")) &&
        reference("SW-SYSTEMCONST:SY_COUNT") > 1
```

  since here, if `SY_COUNT` is not defined, the check for "`> 1`" is not carried out and an unwanted error message is thus avoided.
- **[TPS_GST_00003] true and false** ⌈Like in C a integer "0" respectively floating point "0.0" and "-0.0" are interpreted as false. Every other value is treated as true within boolean expressions.

  The language also provides the literals "`true`" yielding `1` respectively "`false`" yielding `0` to express literal boolean values in expressions.⌋()

  It is strongly recommended not to apply the Boolean operations ||, &&, ^^, ==, != to floating-point values. Caused by the limited ability of float operands to hold exact integer (in the mathematical meaning) numbers the result of these operations may have (as implementation specifics have a huge impact here) values that are hard to predict.

- **[TPS_GST_00004] Priority of Operations** ⌈
  The priority rules of C++[1] apply and are modeled in the grammar: multiplication and division take precedence over addition and subtraction. The exponent operator (`**`) has priority over these other mathematical operators.

  The unary minus has greater precedence than all other operators. For a complete list of the priorities please refer to [13].

  Example:

  - `-2**3` becomes `-8 = (-2)**3`
  - `-log(2.718281828)**2` corresponds to `(-log(2.718281828))**2` and therefore is equal to `+1`.

  ⌋*()*

- **[TPS_GST_00005] left-to-right evaluation** ⌈Binary operators shall be evaluated from left to right (left-to-right associativity). For example 2 == 2 == 2 shall be evaluated as (2 == 2) == 2.⌋*()*

- **[TPS_GST_00006] Associativity of XOR** ⌈Boolean XOR operator ^^ is an additional operator which has no counterpart in C. The result of a boolean XOR is "1" if one of the operands is interpreted as false and the other as true. The result of a boolean XOR is "0" if both operands have the same value independent of whether the value is true or false. Other than boolean AND (&&) and boolean OR (||), boolean XOR always evaluates both operands. This is because the result of a boolean XOR cannot be determined by only evaluating e.g. the first operand.

  Hint: In hardware circuits sometimes an XOR gate with more than two inputs is used. For such a hardware XOR gate the output is "1" if and only if one of the inputs is "1" and all other inputs are "0". The XOR operator within arithmetic expressions is a binary operator and therefore behaves different than hardware XOR gates. This means in particular that according to left-to-right associativity e.g.
  1 ^^ 1 ^^ 1 is interpreted as (1 ^^ 1) ^^ 1 which yields "1".⌋*()*

- **[TPS_GST_00007] Shift operation** ⌈
  When the shift operation is performed, the first or the last bit (depending on the direction of the shift) is filled with "0".

  Shift left means that the bits are shifted towards the higher values. Shift right means that the bits are shifted towards the lower values.

  Shift is performed as long as we are in the range defined by [TPS_GST_00008]⌋*()*

  Implementers shall apply a mask on the intended size number of bits on the result. For example if `0b1111 « 1` shall still yield 4 bits, then user shall write

  ```
  (0b1111 « 1) & 0b1111
  ```

[1] Note that XOR is not defined in C++.

to get the intended result of `0b1110`. Without the mask, the result would be `0b11110` which is 5 bit.

Example based on 32 bit integer implementation of the formula processor (note that not all leading zero digits are shown):

- `0b0001 << 1` returns `0b00010`
- `0b1111 >> 1` returns `0b0111`
- `0b1111 << 1` returns `0b11110`
- `0b1111 << 2` returns `0b111100`

- **[TPS_GST_00008] Types in Formula Expressions** ⌈
The type of an arithmetic expression is one of
  - Integer in the range (32 bit implementation)

    ```
    0x80000000 .. 0xffffffff
    -2147483648 .. +4294967295
    ```

    respectively (in a 64 bit implementation)

    ```
    0x8000000000000000 .. +0xffffffffffffffff
    -9223372036854775808 .. +18446744073709551615
    ```

  - Float (internally represented by double)

  ⌋*()* For the result type of a function or operand see 4.8.2.4.

  Note: The ranges of the 32-bit implementation are the set union of signed INT (-2147483648 .. 2147483647) and unsigned INT (0 .. 4294967295) resulting in (-2147483648 .. 4294967295). The same procedure is applied for the 64-bit implementation.

- **[TPS_GST_00359] Handling of the Sign** ⌈
In consequence of [TPS_GST_00008], implementations of a formula processor need to provide an extra handling of the sign in case of integer values.

  ```
  -2147483648 < value < 0          => signed INT (sign "-")
  -2147483648 < value < 2147483647 => signed INT (sign "-" or "+")
  -0                               => signed INT (sign "-")
  +0                               => signed INT (sign "+")
   0          < value < 2147483647 => unsigned INT
  2147483647  < value < 4294967295 => unsigned INT.
  ```

  ⌋*()*

- **[TPS_GST_00009] Keyword 'epsilon'** ⌈
`epsilon` represents an implementation specific constant intended to support the comparison of float values. It represents the smallest increment which can be expressed by the given implementation.

For example instead of comparing a float with zero, one should use

```
abs(sysc(x)) < epsilon ? 0 : 1
```

⌋()

- **[TPS_GST_00276] Power of Null** ⌈
  `pow(0, 0)` respectively `0**0` is undefined and shall raise an error (see also [TPS_GST_00014]).

  ⌋()

- **[TPS_GST_00010] Keyword `undefined`** ⌈
  `undefined` represents a sub term which is undefined. It is subject to be replaced in further process steps. The main purpose is to denote blueprints of expressions. The result of `undefined` is the same as an undefined operand. Usually it yields a runtime error. The following expressions hold true:

  ```
  defined(undefined) = false
  true && undefined = error
  false && undefined = false
  undefined = error
  ```

  And consequently for OR it is

  ```
  true || undefined = true
  false || undefined = error
  ```

  ⌋()

- **[TPS_GST_00275] Float Literals `INF`, `NaN`** ⌈In order to maintain consistency with `Float`, formula expressions also supports the special float literals `INF` and `NaN`.

  The support of these literals is in particular:

  – `INF` is allowed only in the context of operators `"=="`, `"!="`, unary `"-"`, result of `condExpr`

  – `NaN` is allowed only in the context of operators `"=="`, `"!="`, result of `condExpr`

  – `INF`, `NaN` can only be obtained by literal specification (e.g. in a `SwSystemconstValue`) but not as the result of an arithmetic operation (In particular [TPS_GST_00014] is not affected by these literals).

  ⌋()

  Note that there is no specific literal `-INF` since this is supported via the unary operator `"-"`.

  An example from the context of `SwSystemconstDependentFormula` for an expression yielding `INF` is

```
reference("SW-SYSTEMCONST:unlimited") == true ? INF : 10000
```

- **[TPS_GST_00011] Functions `round`, `ceil`, `floor`** ⌈
  These act as 'Integer-Cast' with rounding. The following applies:

  - **`round`**
    rounds positive and negative numbers to the next whole number.

```
Ex. : round(4.4) = 4      round(-4.4) = -4
      round(4.7) = 5      round(-4.7) = -5
      round(4.5) = 5      round(-4.5) = -5
```

  - **`floor`**
    rounds positive and negative numbers down to the next whole number.

```
Ex. : floor(4.4) = 4      floor(-4.4) = -5
      floor(4.7) = 4      floor(-4.7) = -5
      floor(4.0) = 4      floor(-4.0) = -4
```

  - **`ceil`**
    rounds positive and negative numbers up to the next whole number.

```
Ex. : ceil(4.4)  = 5      ceil(-4.4)  = -4
      ceil(4.7)  = 5      ceil(-4.7)  = -4
      ceil(4.0)  = 4      ceil(-4.0)  = -4
```

  ⌋()

- **[TPS_GST_00013] Function `defined`** ⌈`defined(reference)` returns `1` if the reference passed as a parameter is defined. If a `reference(...)` is defined by a formula `defined(reference(...))` returns 1. The expression `defined(stringReference(...))` yields 1 if the reference passed as parameter can yield a string. In all other cases it returns 0.⌋()

  Note that for example `SwSystemconstValue` can be defined using a formula. Such a 'redefined' `SwSystemconst` is always treated as 'defined' even if its formula refers to an undefined `SwSystemconst`. A `reference` can return a string if for example, in case of a `SwSystemconstDependentFormula`, the referenced `SwSystemconst` refers a `CompuMethod` of category `TEXTTABLE`.

  Example:

```
suppose
   reference("SW-SYSTEMCONST:ZYLZA")  is set to 4
   reference("SW-SYSTEMCONST:ZYLZA2") is set to reference("SW-
SYSTEMCONST:ZYLZA")+2
   reference("SW-SYSTEMCONST:TURBO")  is left undefined
   reference("SW-SYSTEMCONST:TURBO2") is set to reference("SW-
SYSTEMCONST:TURBO")+2
then
   defined(reference("SW-SYSTEMCONST:ZYLZA"))  yields 1
```

```
defined(reference("SW-SYSTEMCONST:ZYLZA2")) yields 1
defined(reference("SW-SYSTEMCONST:TURBO"))  yields 0
defined(reference("SW-SYSTEMCONST:TURBO2")) yields 1 (TURBO2 is defined, even
                                                      though the referenced
                                                          TURBO is undefined.)
```

- **[TPS_GST_00014] Error handling in Formula Evaluator** ⌈
  Error messages shall be exposed by an evaluator
  - if the arithmetic expression is syntactically incorrect
  - in case of division by 0
  - in case `0**0` respectively `pow(0, 0)` (see [TPS_GST_00276])
  - if the definition range of a function is violated
  - if the function value of a function is outside the range which can be represented (for example, this applies to `floor(10E22)`)
  - if the evaluation of an operand fails (see [TPS_GST_00002], [TPS_GST_00010])
  - if the range that can be represented is exceeded when using the basic arithmetic operators `'-', '+', '*', '/'`

  ⌋()

4.8.2.4 Resulting Data Types of Formula Expressions

The following return types apply for operator/operands:

[TPS_GST_00034] Return Types for Additive 0perators ⌈

```
+ - / *
      Integer, Integer  -> Integer (this is an integer division)

      Integer, Float    -> Float
      Float, Integer    -> Float
      Float, Float      -> Float
```

⌋()

[TPS_GST_00035] Return Types for Multiply Operators ⌈

```
**, pow
      Integer, Integer  -> Integer

      Integer, Float    -> Float
      Float, Integer    -> Float
      Float, Float      -> Float
```

⌋()

[TPS_GST_00036] Return Types for Logical Operators ⌈

```
||, &&,^^, ==, <=, <, >, >=, !=
      Integer, Integer  -> Integer representing boolean (0: false,  1: true)

      Integer, Float    -> Integer representing boolean (0: false,  1: true)
      Float, Integer    -> Integer representing boolean (0: false,  1: true)
      Float, Float      -> Integer representing boolean (0: false,  1: true)
```

⌋()

[TPS_GST_00037] Return Types for Bitwise Operators ⌈

```
|, &, ^, <<, >>         bit-wise operators always render Integer
                        with value >= 0
      Integer, Integer  -> Integer
      Integer, Float    -> Fault
      Float, Integer    -> Fault
      Float, Float      -> Fault
                         using an operand of type Float
                         or negative Integer leads to incorrect
                         arithmetic expression and an error message
```

⌋()

[TPS_GST_00038] Return Types for Binary Function ⌈

```
min
      Integer, Integer  -> Integer

      Float, Integer    -> Float
      Integer, Float    -> Float
      Float, Float      -> Float

max
      Integer, Integer  -> Integer
      Integer, Float    -> Float
      Float, Integer    -> Float
      Float, Float      -> Float
```

⌋()

[TPS_GST_00039] Return Types for Negation Operators ⌈

```
~, !
      Integer           -> Integer
      Float             -> Fault
                           using an operand of type Float
                           or negative Integer leads to incorrect
                           arithmetic expression and an error message
```

⌋()

[TPS_GST_00040] Return Types for Define ⌈

```
defined                 -> Integer representing boolean (0: false,  1: true)
```

⌋()

[TPS_GST_00041] Return Types for Unary Functions ⌈

```
log, log10, exp
      Integer           -> Float
      Float             -> Float

abs
      Integer           -> Integer
      Float             -> Float

sgn
      Integer           -> Integer (the result is one of -1, 0, 1)
      Float             -> Integer (the result is one of -1, 0, 1)

ceil, floor, round
      Integer           -> Integer
      Float             -> Integer
```

⌋()

[TPS_GST_00042] Return Types for Keywords ⌈

```
epsilon                 -> Float

true, false             -> Integer representing boolean (0: false,  1: true)
```

⌋()

[TPS_GST_00208] Representation of return type in float ⌈The serialisation of the final return value of type float shall be in float format and even if the result has no decimal places one decimal place shall be emitted (e.g. 5.0).⌋()

4.8.2.5 Examples for the Formula Language expressions

Examples of correct arithmetic expressions are[2]:

```
<APPLICATION-VALUE-SPECIFICATION>
  <SW-VALUE-CONT>
    <SW-VALUES-PHYS>
      <VF>1</VF>
    </SW-VALUES-PHYS>
  </SW-VALUE-CONT>
</APPLICATION-VALUE-SPECIFICATION>

...
```

[2]Note that the example does not reflect the BINDING-TIME

```
      <VF>2.0</VF>
...

...
      <VF>1.5E5</VF>
...

...
      <VF>2.0 * (1 + 2)</VF>
...

...
      <VF>2 > 1</VF>
...
```

Example without entities. The grammar is defined for strings AFTER replacement of the entities.

```
...
      <VF>2 &gt; 1</VF>
...
```

Example with entities. When looking in real ARXML-Files, e.g. with a text editor, one may find entities in the formula expression. Entities are things like

```
&gt;
```

standing for "'>"'.

Additional formula language expression are available as test cases in [14] [AUTOSAR_TP_FormulaLanguage_TestCase_Blueprint.arxml]. For further examples with references see content below [TPS_GST_00265].

4.9 AUTOSAR Model Query Language (ARMQL)

This chapter describes ARMQL, the AUTOSAR Model Query Language [3]. Because this language is currently only used in the context of blueprints, see [2] chapter "The Principles of Blueprint", all examples are taken from this application field. Within a blueprint, ARMQL is used within the `expression` inside a `formalBlueprintGenerator`, which resides inside a `VariationPoint`.

```
<VARIATION-POINT>
  <FORMAL-BLUEPRINT-GENERATOR>
    <EXPRESSION>
        ...
    </EXPRESSION>
  </FORMAL-BLUEPRINT-GENERATOR>
</VARIATION-POINT>
```

[3] The language design was inspired by XQuery, functional programming as well as template engines. Core of the language is the untyped λ-calculus with a non-strict evaluation strategy. It embraces principles like immutability, referential transparency and static scoping.

The main idea is, that an formalBlueprintGenerator "generates" an ordered (possibly empty) list of sets of name-value assignments. For each of these assignment sets the enclosing element of the VariationPoint is "printed out", while recursively applying the same approach for any descendants of the enclosing element of the VariationPoint. The assignment-sets generated by these formalBlueprint-Generators deeper down in the ARXML containment hierarchy are joined with the assignment sets from formalBlueprintGenerators higher in the hierarchy, so that the assignment-sets form a hierarchical scope. Keys in assignments on a lower hierarchy level have precedence over assignments to the same key on a higher level. For this process, elements without a formalBlueprintGenerator are considered to have implicitly an "empty" formalBlueprintGenerator, i.e. one, that produces an empty assignment set.

During "printing out" an element, the current assignment set on the respective scope is used to interpolate variables in certain Strings.

A consequence of this is, that an element with an explicit formalBlueprintGenera-tor can potentially be derived into zero, one or many elements of the same meta-class.

4.9.1 Applying ARMQL

In this section the use of ARMQL within a FORMAL-BLUEPINT-GENERATOR tag as well as the interplay with other parts of ARXML is demonstrated with the help of an example. The definition of the ARMQL follows in section 4.9.2.

4.9.1.1 LET Block

Consider the following part of a blueprint:

```
...
<CLIENT-SERVER-INTERFACE>
  <SHORT-NAME BLUEPRINT-VALUE="IF_{Name}_{Id}">Interface</SHORT-NAME>
  ...
  <VARIATION-POINT>
    <FORMAL-BLUEPRINT-GENERATOR>
      <EXPRESSION>
        LET Name = "Example";
            Id   = 1;
      </EXPRESSION>
    </FORMAL-BLUEPRINT-GENERATOR>
  </VARIATION-POINT>
    ...
</CLIENT-SERVER-INTERFACE>
...
```

During blueprint derivation, it will be derived into the following ARXML-Code:

```
...
<CLIENT-SERVER-INTERFACE>
```

```
  <SHORT-NAME>IF_Example_1</SHORT-NAME>
    ...
</CLIENT-SERVER-INTERFACE>
...
```

We're going through this example line by line but leave out lines that only contain closing tags (we assume familiarity with ARXML) or the ellipsis (. . . - the ellipsis is used to denote arbitrary ARXML code which is not necessary for the understanding of the example, but could be there and might even be necessary to make the example schema valid).

```
<CLIENT-SERVER-INTERFACE>
```

This is the meta-class of the of element that will be derived from the blueprint. During the derivation, this tag remains unchanged. Remark: We're using a `CLIENT-SERVER-INTERFACE` here, because it is a common candidate for blueprinting - but everything shown here would work with any meta-class that is blueprintable.

```
<SHORT-NAME BLUEPRINT-VALUE="IF_{Name}_{Id}">Interface</SHORT-NAME>
```

The `shortName` will be modified during the blueprint derivation, because it contains the attribute `blueprintValue`. The value of this attribute, after an variable interpolation, will become the new value of the tag. The original content of the tag (in this example `Interface`) is discarded.

Variable interpolation means here, that the parts that are enclosed in curly braces (in this example `Name` and `Id`) are interpreted as names of variables and are then replaced by the value of these variables. The assignments of values to names happens inside a `formalBlueprintGenerator`. We're coming back to the derivation of the value once we had a look at the `formalBlueprintGenerator`.

```
<VARIATION-POINT>
  <FORMAL-BLUEPRINT-GENERATOR>
    <EXPRESSION>
```

A `formalBlueprintGenerator` resides inside a `VariationPoint`. A `VariationPoint` defines the variability of its direct ancestor element, i.e. of the `ClientServerInterface` in this example. This is important to understand the scope of the assignments made inside the `formalBlueprintGenerator`.

This tag `FORMAL-BLUEPRINT-GENERATOR` denotes, that this `VariationPoint` is used for a formal blueprint derivation. Inside the `EXPRESSION` tag, the generation of the assignment sets is described. In addition to the `EXPRESSION` tag it is possible that e.g. notes for a human reader are also put into the `formalBlueprintGenerator` using the `introduction` sub element.

```
  LET Name = "Example";
      Id   = 1;
```

The content of the `EXPRESSION` needs to be formulated in ARMQL. In this example, the expression consists only of a `LET` block which defines simple mappings of names to values. A `LET` block starts with the keyword `LET` (`Let` or `let` can also be used) and

can contain one or more assignments. Each assignment is terminated by a semicolon. In this case the name `Name` is mapped to the value `"Example"` and the name `Id` is mapped to the value `1`.

Consider again the `blueprintValue` attribute above. In this example, the string `"IF_{Name}_{Id}"` is derived to `"IF_Example_1"` because `Name` is replaced by `"Example"` and `{Id}` is replaced by `1`, because these are the values that were assigned to these names inside the `FORMAL-BLUEPRINT-GENERATOR` of the `CLIENT-SERVER-INTERFACE`.

4.9.1.2 Expressions

The language supports basic arithmetic operations like addition, subtraction, multiplication and division. Furthermore the standard operator precedence rules are obeyed, e.g. multiplication has a higher precedence then addition. So in the following listing the value `7` is assigned to the name `Id`.

```
LET Id   = 1 + 2 * 3;
```

You can apply functions within expressions. In the following listing the predefined function `startsWith()` is called with two arguments, namely `"Example"` and `"Ex"`. This function call returns the Boolean value `True`, because the string `"Example"` starts with the string `"Ex"`. This value `True` is then assigned to the name `isMatch`.

```
LET isMatch = startsWith("Example", "Ex");
```

Functions can also be defined in a `LET` block. In the following the function `genId()` is defined in the first line and is then called in the second line. The function takes one parameter (named `s`). The function definition of `genId()` has the following meaning: If the argument given to `genId()` is a string and starts with `"Ex"`, `genId()` returns the integer value `"24"`, if the argument is a string but does not start with `"Ex"`, then the integer value `"42"` is returned. If the argument is not a string, then `genId()` the returned value is `undefined`. Please note that `undefined` is considered to be a special value of a special type and not considered to be e.g. a special integer number.

```
LET genId(s) = if( startsWith(s, "Ex") , 24, 42);
    Id       = genId( "Example" );
```

To get rid of a lot of parentheses and to improve readability another way to call functions was included into the language: The first argument of a function could be prefixed, followed by a point, followed by the function name, followed by possibly remaining arguments in parentheses. E.g. in the first line of the following example, the expression `s.startsWith("Ex")` is equivalent to `startsWith(s,"Ex")`. The second line of the example starts with `//`, the line comment token, which turns this line into a comment. This comment illustrates another way of expressing the first line, i.e. by also prefixing the first argument of the `if()` function. The third line of the example shows, that this way of calling functions also applies to newly defined functions, like `genId()` in this case. Note: This syntax mimics a method invocation in object oriented languages, but should not be mistaken for it, because it is really just another syntax for a function call.

```
  LET genId(s) = if( s.startsWith("Ex") , 24, 42);
 //or genId(s) = s.startsWith("Ex").if(24, 42);
      Id       = "Example".genId();
```

4.9.1.3 FOR Block

The FOR block is used to iterate (or to "map") over a list of values. Consider the following example:

```
...
<CLIENT-SERVER-INTERFACE>
  <SHORT-NAME BLUEPRINT-VALUE="IF_{Name}_{Id}">Interface</SHORT-NAME>
  ...
  <VARIATION-POINT>
    <FORMAL-BLUEPRINT-GENERATOR>
      <EXPRESSION>
        FOR Id : [1,3];
        LET Name = "Example";
      </EXPRESSION>
    </FORMAL-BLUEPRINT-GENERATOR>
  </VARIATION-POINT>
    ...
</CLIENT-SERVER-INTERFACE>
...
```

During blueprint derivation it will be derived to the following ARXML-Code (Note: The expression `[1,3]` expands to a list consisting of the integer values 1, 2 and 3). The LET block will be executed for each element in the list, leading to a set of three (Name, Id) pairs, which are applied to the BlueprintValue attributes of the ARElement containing the VariationPoint.

```
...
<CLIENT-SERVER-INTERFACE>
  <SHORT-NAME>IF_Example_1</SHORT-NAME>
  ...
</CLIENT-SERVER-INTERFACE>
<CLIENT-SERVER-INTERFACE>
  <SHORT-NAME>IF_Example_2</SHORT-NAME>
  ...
</CLIENT-SERVER-INTERFACE>
<CLIENT-SERVER-INTERFACE>
  <SHORT-NAME>IF_Example_3</SHORT-NAME>
  ...
</CLIENT-SERVER-INTERFACE>
...
```

4.9.1.4 WHERE Block

The WHERE block is used to filter out certain parts of the output during blueprint derivation. After the WHERE keyword there is exactly one condition, i.e. an expression that

shall evaluate to a Boolean value. During blueprint derivation the output is only generated with the current assignment set, if this condition evaluates to `True`.

Consider the following example:

Note: The operator `!=` means "is not equal to".

```
...
<CLIENT-SERVER-INTERFACE>
  <SHORT-NAME BLUEPRINT-VALUE="IF_{Name}_{Id}">Interface</SHORT-NAME>
  ...
  <VARIATION-POINT>
    <FORMAL-BLUEPRINT-GENERATOR>
      <EXPRESSION>
        FOR   Id : [1,3];
        LET   Name = "Example";
        WHERE Id != 2;
      </EXPRESSION>
    </FORMAL-BLUEPRINT-GENERATOR>
  </VARIATION-POINT>
    ...
</CLIENT-SERVER-INTERFACE>
...
```

During blueprint derivation it will be derived to the following ARXML-Code:

```
...
<CLIENT-SERVER-INTERFACE>
  <SHORT-NAME>IF_Example_1</SHORT-NAME>
  ...
</CLIENT-SERVER-INTERFACE>
<CLIENT-SERVER-INTERFACE>
  <SHORT-NAME>IF_Example_3</SHORT-NAME>
  ...
</CLIENT-SERVER-INTERFACE>
...
```

4.9.2 ARMQL Definition

This subsection defines the language used inside a `formalBlueprintGenerator`.

4.9.2.1 Grammar

[TPS_GST_00184] Grammar of ARMQL ⌈The syntax of ARMQL is defined by the following grammar.⌋*()*

Listing 4.16: ARMQL ANTLR 4 Grammar (Parser)

```
flws   :   ( stmt1 | stmt2 | stmt3 )*
       ;
```

```
stmt1 :    FOR    fors+
         ( LET    lets+   )?
         ( WHERE  where   )?
      ;

stmt2 :    LET    lets+
         ( WHERE  where   )?
      ;

stmt3 :    WHERE  where
      ;

fors  :  ID IN expr SEMI
      ;

lets  :  ID (LPAREN ID (COMMA ID)* RPAREN)? ASSIGN expr SEMI
      ;

where :  expr SEMI
      ;

expr  :  expr DOT ID LPAREN expr? ( COMMA expr )* RPAREN  # Method
      |  expr op=(MUL|DIV|MOD) expr                       # MulDiv
      |  expr op=(ADD|SUB) expr                           # AddSub
      |  expr op=(EQ|NE|GT|LT|GE|LE) expr                 # Compare
      |  NOT expr                                         # Not
      |  expr op=(AND|OR) expr                            # AndOr
      |  LAMBDA ID DOT expr                               # Lambda
      |  FLOAT                                            # Float
      |  INTEGER                                          # Integer
      |  BOOL                                             # Bool
      |  STRING                                           # String
      |  ID                                               # Id
      |  ID LPAREN expr? ( COMMA expr )* RPAREN           # Fun
      |  LPAREN expr RPAREN                               # Parens
      |  LBRACK expr COMMA expr RBRACK                    # Range
      ;
```

Listing 4.17: ARMQL ANTLR 4 Grammar (Lexer)

```
// Comments

COMMENTS
      : ( BlockComment
        |  LineComment
        ) -> channel(HIDDEN)
      ;
fragment BlockComment
      : '/*'  .*? ('*/' | EOF)          ;
fragment LineComment
      : '//' ~[\r\n]*                   ;


// -------------------------
// Numbers
```

```
FLOAT :   DecDigits '.' DecDigits? ExponentPart?
      |   '.' DecDigits ExponentPart?
      |   DecDigits ExponentPart
      ;

fragment ExponentPart
      :   [eE] [+-]? DecDigits
      ;

INTEGER
      : DecimalNumeral
      | HexNumeral
      | OctalNumeral
      | BinaryNumeral
      ;

fragment HexNumeral
      :   '0' [xX] HexDigits
      ;

fragment OctalNumeral
      :   '0' '_' OctalDigits
      ;

fragment DecimalNumeral
      :   '0'
      |   [1-9] DecDigit*
      ;

fragment BinaryNumeral
      :   '0' [bB] BinaryDigits
      ;


fragment HexDigits      : HexDigit+     ;
fragment DecDigits      : DecDigit+     ;
fragment OctalDigits    : OctalDigit+   ;
fragment BinaryDigits   : BinaryDigit+  ;

fragment HexDigit       : [0-9a-fA-F]   ;
fragment DecDigit       : [0-9]         ;
fragment OctalDigit     : [0-7]         ;
fragment BinaryDigit    : [01]          ;



// -------------------------
// Strings

STRING: '"' ( EscSeq | ~["\r\n\\] )* '"'
      ;

// Any kind of escaped character that we can embed within literal strings.
fragment EscSeq
      :   '\\'
          ( [tnr"\\] // Std escaped character set (tab, newline, etc.)
```

```
          | .         // Invalid escape character
          | EOF       // Incomplete at EOF
          )
      ;


// -------------------------
// Bools

BOOL  :  'true'    | 'True'    | 'TRUE'
      |  'false'   | 'False'   | 'FALSE'
      ;


// -------------------------
// Reserved Words and Operators

FOR     :   'for'    | 'For'    | 'FOR'  ;
LET     :   'let'    | 'Let'    | 'LET'  ;
WHERE   :   'where'  | 'Where'  | 'WHERE';
IN      :   'in'     | 'In'     | 'IN'   ;

LT      :   'lt'     | 'Lt'     | 'LT'   ; //less than
GT      :   'gt'     | 'Gt'     | 'GT'   ; //greater than
LE      :   'le'     | 'Le'     | 'LE'   ; //less or equal
GE      :   'ge'     | 'Ge'     | 'GE'   ; //greater or equal

AND     :   'and'    | 'And'    | 'AND'  ;
OR      :   'or'     | 'Or'     | 'OR'   ;
NOT     :   'not'    | 'Not'    | 'NOT'  ;

NE      :   '!='                         ; //not equal
EQ      :   '=='                         ; //equal

COMMA   :   ','                          ;
SEMI    :   ';'                          ;
LPAREN  :   '('                          ;
RPAREN  :   ')'                          ;

ASSIGN  :   '='                          ;
LBRACK  :   '['                          ;
RBRACK  :   ']'                          ;

MUL     :   '*'                          ;
DIV     :   '/'                          ;
MOD     :   '%'                          ;
ADD     :   '+'                          ; //also string concatenation
SUB     :   '-'                          ;

LAMBDA  :   '\\'                         ;
DOT     :   '.'                          ;
COLON   :   ':'                          ;


// -------------------------
// Identifiers
```

```
ID      : NameStartChar NameChar*       ;

fragment NameStartChar
        :   'A'..'Z'
        |   'a'..'z'
        |   '_'
        ;

fragment NameChar
        :   NameStartChar
        |   '0'..'9'
        |   '_'
        ;


// ------------------------
// Whitespace and Errors

WS      : ( '[ \t]' | '\r'? [\n] )+  -> channel(HIDDEN)
        ;

ERRCHAR: .   -> channel(HIDDEN)          ;
```

4.9.2.2 Types and Values

[TPS_GST_00388] Types and Values in ARMQL ⌈While ARMQL is not statically typed, the following types are distinguished at run-time:

- Floating-point Numbers (e.g. `1.415e-10`, always 64bit)
- Integer Numbers (e.g. `-37`, arbitrary length, ”BigInteger”)
- Boolean Values (e.g. `true` or `false`)
- Strings (e.g. `"this is a string"`)
- Functions (e.g. `x.y.y+x`)
- Opaque Objects (e.g. `ECV`, used enrich the language by abstract data types)
- Lists (e.g. $[1, 4]$);
- Undefined (e.g. $1/0$, two undefined values are considered to never be equal)

⌋*()*

[TPS_GST_00389] Type Coercion Rules ⌈

- There is no automatic conversion from integer to floating point numbers, this is to be done manually by using the functions `int()` or `float()`
- All arithmetic operations are homogeneous regarding to the numeric type, i.e no automatic coercion but returning `undefined` at run-time. E.g. addition is either

floating point only or integer only. This rule also applies to division of integers, which will result in an integer, so `7 / 2` will result in the integer number `3`.

- Literals for floating point numbers shall contain a decimal point. Numeric literals without a decimal point are interpreted as integers.

⌋()

[TPS_GST_00390] Operators and Expressions ⌈The following expressions and operators are available:⌋()

Listing 4.18: ARMQL ANTLR 4 Grammar (Expressions)

```
|  expr op=(MUL|DIV|MOD) expr                    # MulDiv
|  expr op=(ADD|SUB) expr                        # AddSub
|  expr op=(EQ|NE|GT|LT|GE|LE) expr             # Compare
|  NOT expr                                     # Not
|  expr op=(AND|OR) expr                        # AndOr
```

Listing 4.19: ARMQL ANTLR 4 Grammar (Operators)

```
LT      :   'lt'    | 'Lt'    | 'LT'   ; //less than
GT      :   'gt'    | 'Gt'    | 'GT'   ; //greater than
LE      :   'le'    | 'Le'    | 'LE'   ; //less or equal
GE      :   'ge'    | 'Ge'    | 'GE'   ; //greater or equal

AND     :   'and'   | 'And'   | 'AND'  ;
OR      :   'or'    | 'Or'    | 'OR'   ;
NOT     :   'not'   | 'Not'   | 'NOT'  ;

NE      :   '!='                       ; //not equal
EQ      :   '=='                       ; //equal

COMMA   :   ','                        ;
SEMI    :   ';'                        ;
LPAREN  :   '('                        ;
RPAREN  :   ')'                        ;

ASSIGN  :   '='                        ;
LBRACK  :   '['                        ;
RBRACK  :   ']'                        ;

MUL     :   '*'                        ;
DIV     :   '/'                        ;
MOD     :   '%'                        ;
ADD     :   '+'                        ; //also string concatenation
SUB     :   '-'                        ;
```

[TPS_GST_00391] Lambda Abstraction, Function and Pseudo-Method Invocation ⌈Anonymous functions can be defined with the help of the λ operator .⌋()

The result of the following expression is a function that takes one argument and adds `3` to this argument:

```
1    \ a . a + 3
```

Some functions use anonymous functions as arguments, e.g. `filter()`. Anonymous functions can also be given a name and can than be applied to arguments. So the `result` in the following example is `7`. Please note that the comment shows an equivalent way to define `add3`.

```
1    LET add3     = \ a . a + 3;
2     // add3(a) = a + 3
3        result   = add3(4);
```

Furthermore, a function with two arguments can be applied to just one argument. The result of this function application is a anonymous function with one argument. So the following is yet another way to define `add3()`:

```
1    LET add(a,b) = a + b;
2     // add      = \ b . \ a . a + b;
3        add3     = add(3);
```

There is also some syntactic sugar for prefixing the first argument of a function call which resembles a method invocation in object oriented languages and drastically reduces the amount of parentheses needed.

[TPS_GST_00392] The Integer Range Expression ⌈

If `n` and `m` are integers, then the expression `[n,m]` expands to the list of all integers i for which the following holds: `n` $\leq i \leq$ `m`. If one of `n` or `m` (or both) do not denote integers, then `[n,m]` is undefined.⌋*()*

In other words, `[n,m]` denotes a closed interval of integers. Please note that a consequence of the definition is, that the interval is empty, if `n` $\geq$ `m`.

4.9.2.3 LET Block

[TPS_GST_00393] Assigning Names to Values ⌈The `LET` block is used to give names to expressions. One `LET` block allows the arbitrary naming of many values.⌋ *()*

Example with a single name assignment:

```
1  LET var = "value";
```

Here the name `var` is given to the String value `"value"`.

Example with multiple name assignments in one LET block:

```
1  LET str = "a␣string";
2      num = 42;
3      boo = true;
4      exp = 17.3 * 2 + num;
```

Here the names `str`, `num`, `boo`, `exp` are defined. The definition of `exp` illustrates, that expressions can be named and that the previously defined names can be used within these expressions (namely `num` is used in the definition of `exp`). Within the LET block the sequence of statements is arbitrary.

[constr_2627] No reassigning of the same name within one LET Block ⌈Within one `LET` block one name shall be assigned to an value at most once.⌋*()*

[TPS_GST_00394] Definition of functions ⌈The `LET` block supports the definition of functions.⌋*()*

Example for function definitions within a LET block.

```
1 LET fun1a(a)    =      a + 3;
2     fun1b       = \a . a + 3;
3     fun2a(a,b)  = a + b;
4     fun2b       = \b . \a . a + b;
```

Here, the function `fun1a` and `fun2a` are considered to be directly defined in the `LET` block. Direct function definition is syntactic sugar for a special kind of lambda expressions. I.e. `fun1a` is equivalent to `fun1b` and `fun2a` is equivalent to `fun2b`.

4.9.2.4 FOR Block

The `FOR` block is used to iterate (or to ”map”) over a list of values.

[TPS_GST_00395] Single Line FOR Blocks ⌈A single line `FOR` block has the following form:

`FOR <name> : <list>; <remainder>`

The `<name>` is bound in turn to each of the values in the `<list>`. A `FOR` block starts a new hierarchy level. The order of the values in `<list>` is kept. With each assignment `<remainder>` is evaluated.⌋*()*

Please note, that `<remainder>` could only consist of the expansion of the blueprinted ARXML, i.e. access to the value bound to `<name>` within a `formalBlueprintGenerator` attribute. `<remainder>` could even be empty which means that the assignment of the `FOR` is never used.

Example:

```
...
<CLIENT-SERVER-INTERFACE>
  <SHORT-NAME BLUEPRINT-VALUE="IF_{Name}_{Id}">Interface</SHORT-NAME>
  ...
  <VARIATION-POINT>
    <FORMAL-BLUEPRINT-GENERATOR>
      <EXPRESSION>
        FOR Id : [1,3];
        LET Name = "Example";
      </EXPRESSION>
    </FORMAL-BLUEPRINT-GENERATOR>
  </VARIATION-POINT>
    ...
</CLIENT-SERVER-INTERFACE>
...
```

[TPS_GST_00396] Multi Line FOR Blocks ⌈

Multiline `FOR` blocks has the following form:

```
FOR <name1> : <list1>; <name2> : <list2>; ... <nameN> : <listN>;
<remainder>
```

Multiline `FOR` blocks are syntactic sugar only and are directly equivalent to a sequence of single line `FOR` blocks:

```
FOR <name1> : <list1>; FOR <name2> : <list2>; ... FOR <nameN> :
<listN>; <remainder>
```

⌋()

This means that multiline `FOR` blocks evaluate to nested `FOR`s, i.e. `FOR Id1 : [1,3], Id2 : [5,6];` evaluates to the cross product of `[1,3]` and `[5,6]` with the first component bound to `Id1` and the second bound to `Id2`.

Note: A consequence of this is, `<name1>` could be used in the description of `<list2>` (or `<listN>`, but `<name2>` could not be used in the description of `<list1>`.

```
...
<CLIENT-SERVER-INTERFACE>
  <SHORT-NAME BLUEPRINT-VALUE="IF_{Name}_{Id1}_{Id2}">Interface</SHORT-NAME
     >
  ...
  <VARIATION-POINT>
    <FORMAL-BLUEPRINT-GENERATOR>
      <EXPRESSION>
        FOR Id1 : [1,3];
            Id2 : [5,6];
        LET Name = "Example";
      </EXPRESSION>
    </FORMAL-BLUEPRINT-GENERATOR>
  </VARIATION-POINT>
    ...
</CLIENT-SERVER-INTERFACE>
...
```

During blueprint derivation this will produce the following:

```
...
<CLIENT-SERVER-INTERFACE>
  <SHORT-NAME>IF_Example_1_5</SHORT-NAME>
    ...
</CLIENT-SERVER-INTERFACE>
<CLIENT-SERVER-INTERFACE>
  <SHORT-NAME>IF_Example_1_6</SHORT-NAME>
    ...
</CLIENT-SERVER-INTERFACE>
<CLIENT-SERVER-INTERFACE>
  <SHORT-NAME>IF_Example_2_5</SHORT-NAME>
    ...
</CLIENT-SERVER-INTERFACE>
<CLIENT-SERVER-INTERFACE>
```

```
  <SHORT-NAME>IF_Example_2_6</SHORT-NAME>
    ...
</CLIENT-SERVER-INTERFACE>
<CLIENT-SERVER-INTERFACE>
  <SHORT-NAME>IF_Example_3_5</SHORT-NAME>
    ...
</CLIENT-SERVER-INTERFACE>
<CLIENT-SERVER-INTERFACE>
  <SHORT-NAME>IF_Example_3_6</SHORT-NAME>
    ...
</CLIENT-SERVER-INTERFACE>
...
```

4.9.2.4.1 Alternating LET and FOR Blocks

FOR blocks and LET blocks could alternate to allow the assignment of meaningful names also for the description of a list in a FOR block.

```
...
<CLIENT-SERVER-INTERFACE>
  <SHORT-NAME BLUEPRINT-VALUE="IF_{Name}_{Id1}_{Id2}">Interface</SHORT-NAME
     >
  ...
  <VARIATION-POINT>
    <FORMAL-BLUEPRINT-GENERATOR>
      <EXPRESSION>
        FOR Id1  : [1,3];
        LET Low  = Id1 + 3;
            High = Id1 + 5;
        FOR Id2  : [Low, High];
        LET Name = "Example";
      </EXPRESSION>
    </FORMAL-BLUEPRINT-GENERATOR>
  </VARIATION-POINT>
    ...
</CLIENT-SERVER-INTERFACE>
...
```

During blueprint derivation this will result in the following 9 assignment sets:

```
[Id1=1, Low=4, High=6, Id2=4, Name="Example"]
[Id1=1, Low=4, High=6, Id2=5, Name="Example"]
[Id1=1, Low=4, High=6, Id2=6, Name="Example"]
[Id1=2, Low=5, High=7, Id2=5, Name="Example"]
[Id1=2, Low=5, High=7, Id2=6, Name="Example"]
[Id1=2, Low=5, High=7, Id2=7, Name="Example"]
[Id1=3, Low=6, High=8, Id2=6, Name="Example"]
[Id1=3, Low=6, High=8, Id2=7, Name="Example"]
[Id1=3, Low=6, High=8, Id2=8, Name="Example"]
```

and will produce the following:

```
...
```

```
<CLIENT-SERVER-INTERFACE>
  <SHORT-NAME>IF_Example_1_4</SHORT-NAME>
    ...
</CLIENT-SERVER-INTERFACE>
<CLIENT-SERVER-INTERFACE>
  <SHORT-NAME>IF_Example_1_5</SHORT-NAME>
    ...
</CLIENT-SERVER-INTERFACE>
<CLIENT-SERVER-INTERFACE>
  <SHORT-NAME>IF_Example_1_6</SHORT-NAME>
    ...
</CLIENT-SERVER-INTERFACE>
<CLIENT-SERVER-INTERFACE>
  <SHORT-NAME>IF_Example_2_5</SHORT-NAME>
    ...
</CLIENT-SERVER-INTERFACE>
<CLIENT-SERVER-INTERFACE>
  <SHORT-NAME>IF_Example_2_6</SHORT-NAME>
    ...
</CLIENT-SERVER-INTERFACE>
<CLIENT-SERVER-INTERFACE>
  <SHORT-NAME>IF_Example_2_7</SHORT-NAME>
    ...
</CLIENT-SERVER-INTERFACE>
<CLIENT-SERVER-INTERFACE>
  <SHORT-NAME>IF_Example_3_6</SHORT-NAME>
    ...
</CLIENT-SERVER-INTERFACE>
<CLIENT-SERVER-INTERFACE>
  <SHORT-NAME>IF_Example_3_7</SHORT-NAME>
    ...
</CLIENT-SERVER-INTERFACE>
<CLIENT-SERVER-INTERFACE>
  <SHORT-NAME>IF_Example_3_8</SHORT-NAME>
    ...
</CLIENT-SERVER-INTERFACE>
...
```

4.9.2.5 WHERE Block

`WHERE` blocks are used to filter assignments before they are used for the blueprint expansion.

[TPS_GST_00397] Semantics of WHERE Blocks ⌈`WHERE` blocks are always single line and have the following form:

```
WHERE <condition>; <remainder>
```

If `<condition>` evaluates to the boolean value `true` with the current assignment, then the current assign is used to evaluate `<remainder>`. Else `<remainder>` is not evaluated with the current assignment.⌋*()*

Please note, that <remainder> could possibly be the expansion of blueprinted ARXML only, i.e. access to the names in the current assignment within a formalBlueprint-Generator attribute. <remainder> could be also further ARMQL expressions.

```
...
<CLIENT-SERVER-INTERFACE>
  <SHORT-NAME BLUEPRINT-VALUE="IF_{Name}_{Id1}_{Id2}">Interface</SHORT-NAME
     >
  ...
  <VARIATION-POINT>
    <FORMAL-BLUEPRINT-GENERATOR>
      <EXPRESSION>
        FOR Id1  : [1,3];
        LET Low  = Id1 + 3;
            High = Id1 + 5;
        FOR Id2  : [Low, High];
        LET Name = "Example";
        WHERE Id1 + Id2 lt 10;
      </EXPRESSION>
    </FORMAL-BLUEPRINT-GENERATOR>
  </VARIATION-POINT>
    ...
</CLIENT-SERVER-INTERFACE>
...
```

During blueprint derivation this will result in the following 7 assignment sets:

```
[Id1=1, Low=4, High=6, Id2=4, Name="Example"] // 1 + 4 < 10
[Id1=1, Low=4, High=6, Id2=5, Name="Example"] // 1 + 5 < 10
[Id1=1, Low=4, High=6, Id2=6, Name="Example"] // 1 + 6 < 10
[Id1=2, Low=5, High=7, Id2=5, Name="Example"] // 2 + 5 < 10
[Id1=2, Low=5, High=7, Id2=6, Name="Example"] // 2 + 6 < 10
[Id1=2, Low=5, High=7, Id2=7, Name="Example"] // 2 + 7 < 10
[Id1=3, Low=6, High=8, Id2=6, Name="Example"] // 3 + 6 < 10
// for the remaining Id1 + Id2 >= 10
```

and will produce the following:

```
...
<CLIENT-SERVER-INTERFACE>
  <SHORT-NAME>IF_Example_1_4</SHORT-NAME>
    ...
</CLIENT-SERVER-INTERFACE>
<CLIENT-SERVER-INTERFACE>
  <SHORT-NAME>IF_Example_1_5</SHORT-NAME>
    ...
</CLIENT-SERVER-INTERFACE>
<CLIENT-SERVER-INTERFACE>
  <SHORT-NAME>IF_Example_1_6</SHORT-NAME>
    ...
</CLIENT-SERVER-INTERFACE>
<CLIENT-SERVER-INTERFACE>
  <SHORT-NAME>IF_Example_2_5</SHORT-NAME>
    ...
</CLIENT-SERVER-INTERFACE>
<CLIENT-SERVER-INTERFACE>
```

```
  <SHORT-NAME>IF_Example_2_6</SHORT-NAME>
    ...
</CLIENT-SERVER-INTERFACE>
<CLIENT-SERVER-INTERFACE>
  <SHORT-NAME>IF_Example_2_7</SHORT-NAME>
    ...
</CLIENT-SERVER-INTERFACE>
<CLIENT-SERVER-INTERFACE>
  <SHORT-NAME>IF_Example_3_6</SHORT-NAME>
    ...
</CLIENT-SERVER-INTERFACE>
...
```

4.9.2.6 Predefined Functions and Objects

The following functions are available

4.9.2.6.1 Type-Predicates

[TPS_GST_00398] Type predicates ⌈Type predicates are unary functions returning `true` if their argument has a certain runtime type. All type predicates are strict, i.e. they return `undefined` if their argument is `undefined`. In all other cases they return `false`. The following type predicates are available:

- `isNumber(a) : Bool`
- `isBool(a) : Bool`
- `isString(a) : Bool`
- `isObject(a) : Bool`
- `isSeq(a) : Bool`
- `isLambda(a) : Bool`

⌋*()*

Note: The type `Object` is used for extension of the language with abstract data types, e.g. for accessing ECUC Values or model parameters (see section 4.9.2.6.2 and section 4.9.2.6.3).

[TPS_GST_00399] Checking if an expression is defined ⌈The function `defined(a) : Bool` returns `false` if its argument is `undefined`. In all other cases it returns `true`⌋ *()*

[TPS_GST_00400] General Functions ⌈The following general functions are defined:

- `if(condition, a, b)` returns `a` if `condition` evaluates to `true`, `b` if `condition` evaluates to `false`, returns `undefined` in all other cases.

- `default(a,d)` returns `a` if `a` is defined, otherwise it returns `d`.

⌋()

[TPS_GST_00401] String predicates ⌈String predicates are functions taking a String as first argument and returning a `Bool` value if certain conditions are met. All string predicates are strict, i.e. they return `undefined` if their argument is `undefined`. The following string predicates are available:

- `matches(s : String, regExp : String) : Bool` Returns true if the first argument (a string) matches the second argument (a regular expression). The function has the same definition as the JAVA method "String.matches()".
- `contains(s : String, contained : String) : Bool` Returns true if the first argument (a string) contains the second argument (also a string). The function has the same definition as the JAVA method "String.contains()".
- `endsWith(s : String, endsWith : String) : Bool` Returns true if the first argument (a string) ends with second argument (also a string). The function has the same definition as the JAVA method "String.endsWith()".
- `startsWith(s : String, startsWith : String) : Bool` Returns true if the first argument (a string) starts with the second argument (also a string). The function has the same definition as the JAVA method "String.startsWith()".

⌋()

[TPS_GST_00402] List processing functions ⌈List processing functions are functions taking a `Seq` as first argument. All list processing functions are at least strict in their first argument, i.e. they return `undefined` if their first argument, the `Seq`, is `undefined`. The following string predicates are available:

- `count(l : Seq) : Integer` returns the number of elements in `l`.
- `filter(l : Seq, pred : Lambda) : Seq` – The function `pred` shall be an unary function returning a `Bool` value for each element of `l`. `filter()` then returns a `Seq` consisting of all elements of `l` for which `pred` returned true, keeping the order of `l`. `filter()` return `undefined` if `pred` returns `undefined` for any elements contained in `l`.
- `map(l : Seq, fun : Lambda) : Seq a` – `map()` returns a `Seq` consisting of the results of applying `fun` to each of the elements of `l` while keeping the order.
- `fold(l : Seq, start : a, fun : Lambda) : b` – The function `fun` shall be able to accept two arguments. `fold` accumulates the elements of `l` by recursively applying `fun` to the already accumulated results (as first argument) and the first not already accumulated element in `l` (as second argument). The result of of this is then considered to be the next accumulated result and the used element of `l` is then considered to be accumulated. `start` is considered to be the first accumulated result. When all elements of `l` are accumulated, the last accumulated result is returned. The strictness of `fold()` in its second and third argument depends fully on the strictness of `fun`.

- `flatten(x : Seq) : Seq` – If all elements of `x` have the runtime type `Seq`, then `flatten()` returns a `Seq` consisting of the all elements of the elements of `x` while keeping the order, i.e. it concatenates all `Seq`s that are contained in `x`. Otherwise `flatten()` returns `undefined`. Please note: The function `flatten()` is not recursive, i.e. only one level of `Seq`s is flattened.

⌋()

[TPS_GST_00403] Type coercion functions ⌈Type coercion functions take a number with a certain runtime type and return a number with a certain other runtime type. If their argument is not of the expected runtime type, they return `undefined`. The following type coercion functions are available:

- `int(f : Float) : Integer` casts a `Float` to an `Integer`, truncating decimals.
- `float(i : Integer) : Float` casts an `Integer` to a `Float`. If `i` is too large to fit into a `Float`, then `undefined` is returned.

⌋()

4.9.2.6.2 ECUC Value Access

The ARMQL is enhanced by an abstract datatype for access to values coming from `ECUC` values. Remark: For access to model values another, more specialized abstract data type exists, see 4.9.2.6.3.

This abstract data type will be called `EObj` (coming from ”ECUC value access object”) in this subsection. Furthermore a type called `DefinitionRefPath` is used. The type is a normal `String` with a certain inner structure and semantic which is described below.

[TPS_GST_00404] Acccess to ECUC Values ⌈The abstract data type `EObj` consists of the following:

- `ECV : EObj` is a predefined Object. It forms the root to access configuration values coming from the ECUC.
- `subElt(e : EObj, p: DefinitionRefPath) : EObj` is a function, which returns the `EObj`, which is a descendent of `e` and reachable via the relative path (starting from `e`) given by `p`. If this evaluates not to exactly one `EObj`, then the result is `undefined`
- `subEltList(e : EObj, p: DefinitionRefPath) : Seq<EObj>` is a function, which returns a list if `EObj`s which are descendants of `e` and reachable via the relative path (starting from `e`) given by `p`. The order of the `EObj`s in this list is not defined. Note: If no descendent is found an empty list is returned.
- `hasValue(EObj) : Bool` returns `true`, if the `EObj` contains a `Value`, else `false`.

- `hasShortname(EObj) : Bool` returns `true`, if the `EObj` contains a `Shortname`, else `false`.
- `hasValueRef(EObj) : Bool` returns `true`, if the `EObj` contains a `ValueRef`, else `false`.
- `value(e : EObj) : t` returns the `Value` of `e`. Depending on `e`, the type `t` of the returned value is the following
  - `String`
  - `Integer`
  - `Float`
  - `Boolean`

  If `e` does not contain a value, the result is `undefined`
- `shortname(e : EObj) : String` returns the `Shortname` of `e`. If `e` does not contain a `Shortname`, the result is `undefined`.
- `valueRef(e : EObj) : String` returns the `ValueRef` of `e`. If `e` does not contain a `ValueRef`, the result is `undefined`.
- `deref(e : EObj) : EObj` returns the `EObj` to which `e` is referring. If `e` does not contain a `ValueRef`, the result is `undefined`.
- `definitionRef(e : EObj) : String` returns the ECUC Parameter Definition path of `e` starting with the module name (e.g. Csm, NvM, ...). If `e` does not have an ECUC Parameter Definition path, the result is `undefined`.
- `definitionRefName(e : EObj) : String` returns the last part of the ECUC Parameter Definition path of `e`, i.e. the string after the last slash of the ECUC Parameter Definition path. If `e` does not have an ECUC Parameter Definition path, the result is `undefined`.

⌋()

[TPS_GST_00405] DefinitionRefPath ⌈

DefinitionRefPath is a string which contains a path separated by "/", and can contain wildcards:

```
DefinitionRefPath ::=  "/"? PathPart ("/" PathPart)* "/"?
PathPart      ::= ( "*" | "**" | "?" | [a-zA-Z0-9_] )+
*  --> Any Sequence of [a-zA-Z0-9_]
** --> Any Sequence of ("/" | [a-zA-Z0-9_])
?  --> Any ONE of [a-zA-Z0-9_]
```

⌋()

The following expression iterates over all EcuC-Values with the Definition-Ref "NvM/NvMCommon/NvMApiConfigClass" and matches if their value is "NVM_API_CONFIG_CLASS_3".

```
FOR
    configClass : ECV.subEltList("NvM/NvMCommon/NvMApiConfigClass");
LET
    isConfigClass3 = configClass.value() == "NVM_API_CONFIG_CLASS_3";
WHERE
    isConfigClass3;
```

4.9.2.6.3 Model Value Access

The ARMQL is enhanced by an abstract datatype for access to values coming from an model described in ARXML. Remark: For access to `ECUC` values another, more specialized abstract data type exists.

This abstract data type will be called `MObj` (coming from ”model access object”) in this subsection. It consists of the following:

[TPS_GST_00406] Acccess to Model Values ⌈

- `MODEL : MObj` is a predefined Object. It forms the root to access configuration values coming from a ARXML model.
- `getAttribute(a: MObj, b: String) : MObj` is a function, which returns the `MObj`, which is a direct sub-element of `a` and has the name or role `b`
- `filterByType(a: MObj, b: String) : Seq` is a function, which returns a `Seq` of all `MObj`s, which are descendants of `a` and have the type `b`
- `mdlDeref(a: MObj) : MObj` dereferences `a`, i.e. if `a` is a reference to an `MObj` in the `MODEL`, it returns this `MObj`. In all other cases, the result is `undefined`.

⌋*()*

4.9.2.7 Non-Strict Evaluation in every parameter

[TPS_GST_00407] Non-Strict evaluation ⌈Expressions are only evaluated, if the value is really used. A value is only ”used” if it is needed to interpolate a `formal-BlueprintGenerator` attribute, or for the calculation of such a value.⌋*()*

4.9.2.8 Scoping and Visibility Rules

The language uses hierarchical, static scoping and visibility. Three kinds of scoping rules exist: General scoping and visibility rules, hierarchy levels within one `<EXPRESSION>` tag, hierarchy levels between different `<EXPRESSION>`.

[TPS_GST_00408] General scoping rules ⌈

Hierarchy levels define the scoping and visibility of the names in ARMQL. The hierarchy level of ARMQL form a tree with the hierarchy level of all predefined names as its root. The scope of a name spans the hierarchy level it is defined in, as well as all sub hierarchy levels of this hierarchy level. Names can be redefined, i.e. within the scope of a name, a different value can be assigned to the name (exception: no redefinition within one `LET` block). This ”hides” the previously defined name, i.e. the visibility of the previously defined name ends and so its value cannot be accessed any more in this hierarchy level and sub hierarchy levels of this hierarchy level. If a name is not redefined, its visibility corresponds to its scope.⌋*()*

[TPS_GST_00409] Hierarchy levels within one EXPRESSION tag ⌈Each `FOR` block as well as each definition within a `FOR` block starts a new sub-hierarchy level. Each `LET` block starts a new sub-hierarchy level. In contrast to `FOR` blocks, all definitions inside one `LET` block are considered to be in the same hierarchy level. Each `WHERE` block starts a new hierarchy level.⌋*()*

[TPS_GST_00410] Hierarchy levels between EXPRESSION tags ⌈The hierarchy levels between different `<EXPRESSION>` tags corresponds to the ARXML inclusion hierarchy of the parent element of the variation point inside which the ARMQL expressions reside. ARMQL expressions deeper inside the ARMXL inclusion hierarchy (as described above) are considered to be on sub hierarchy levels of this ARMQL expression.⌋*()*

4.9.2.8.1 Examples for scoping and visibility

This example shows relevant parts of the definition of a `ClientServerInterface` with two `ClientServerOperation`s. `Name`, `Id` and the function `genId()` is defined on highest hierarchy-level of this definition, so their scope extends over the complete `ClientServerInterface`.

The example illustrates three things:

- Access to a value: The value named `Name` is accessed within the `ClientServerOperation`s, but defined on a higher hierarchy-level.
- Hiding of a definition: The `Id` is redefined within the `ClientServerOperation`s, hiding the definition of the name `Id` defined on the level of the `ClientServerInterface`.
- Access to a function: The function `genId()` is defined on the level of the `ClientServerInterface` but accessed on the level of the `ClientServerOperation`s.

```
...
<CLIENT-SERVER-INTERFACE>
  <SHORT-NAME BLUEPRINT-VALUE="IF_{Name}_{Id}">Interface</SHORT-NAME>
  ...
  <VARIATION-POINT>
```

```
    <FORMAL-BLUEPRINT-GENERATOR>
      <EXPRESSION>
        LET Name      = "Example";
            Id        = 1;
            genId(s)  = s.startsWith("Ex").if(24, 42);
      </EXPRESSION>
    </FORMAL-BLUEPRINT-GENERATOR>
  </VARIATION-POINT>
    ...
  <OPERATIONS>
    <CLIENT-SERVER-OPERATION>
      <SHORT-NAME BLUEPRINT-VALUE="Op_{Name}_{Id}">RequestResults</SHORT-
        NAME>
      ...
      <VARIATION-POINT>
        <FORMAL-BLUEPRINT-GENERATOR>
          <EXPRESSION>
            LET OpName = "ExampleOp";
                Id     = OpName.genId();
          </EXPRESSION>
        </FORMAL-BLUEPRINT-GENERATOR>
      </VARIATION-POINT>
      ...
    </CLIENT-SERVER-OPERATION>
    <CLIENT-SERVER-OPERATION>
      <SHORT-NAME BLUEPRINT-VALUE="Op_{Name}_{Id}">RequestResults</SHORT-
        NAME>
      ...
      <VARIATION-POINT>
        <FORMAL-BLUEPRINT-GENERATOR>
          <EXPRESSION>
            LET OpName = "AnotherExampleOp";
                Id     = OpName.genId();
          </EXPRESSION>
        </FORMAL-BLUEPRINT-GENERATOR>
      </VARIATION-POINT>
      ...
    </CLIENT-SERVER-OPERATION>
    ...
  </OPERATIONS>
</CLIENT-SERVER-INTERFACE>
...
```

During blueprint derivation the above will be derived to the following:

```
...
<CLIENT-SERVER-INTERFACE>
  <SHORT-NAME>IF_Example_1</SHORT-NAME>
    ...
  <OPERATIONS>
    <CLIENT-SERVER-OPERATION>
      <SHORT-NAME>Op_ExampleOp_24</SHORT-NAME>
      ...
    </CLIENT-SERVER-OPERATION>
    <CLIENT-SERVER-OPERATION>
      <SHORT-NAME>Op_AnotherExampleOp_42</SHORT-NAME>
```

```
      ...
    </CLIENT-SERVER-OPERATION>
    ...
  </OPERATIONS>
</CLIENT-SERVER-INTERFACE>
...
```

4.9.2.9 Blueprint Derivation and Variable Interpolation

During blueprint derivation the constructed assignment sets are used to interpolate variables in certain Strings.

[TPS_GST_00411] Multiplicity of Derived Elements ⌈An element containing a `VariationPoint` containing a `formalBlueprintGenerator` could potentially be derived in zero, one or many elements of the same meta-class during blueprint derivation.⌋*()*

Note: Special care has to be taken to ensure, that the ARXML that is produced during the blueprint derivation process will be schema conform.

[TPS_GST_00412] Variable Interpolation ⌈The content of a tag that contains the attribute `blueprintValue` will be modified during blueprint derivation. The value of this attribute after an variable interpolation will become the new value of the tag. The original content of the tag is discarded. Variable interpolation means, that the parts that are enclosed in curly braces are interpreted as names of values and will be replaced by the corresponding values. All other parts are copied as is. The run-time type of a value bound to a name used for variable interpolation shall be `String` or `Integer`.⌋*()*

In the example of section 4.9.2.8.1,

- the original shortName of the C/S Interface "Interface" is replaced by the resulting blueprintValue "IF_Example_1" and
- the original shortName of the C/S operation "RequestResults" is replaced by the resulting blueprintValue "Op_ExampleOp_24".

4.10 EngineeringObject

While developing AUTOSAR based systems, it is necessary to refer to physical files. These files can be artifacts in which an AUTOSAR model is stored, but may also be source files, diagrams etc. AUTOSAR M1 models may need to refer to such files.

[TPS_GST_00109] Abstraction of Artifacts from Physical File Systems ⌈It is required to keep AUTOSAR XML files independent of the physical layout of hard drives. Therefore, references to physical file such is abstracted as an `AutosarEngineeringObject`.⌋*()*

This follows the approach in [15].

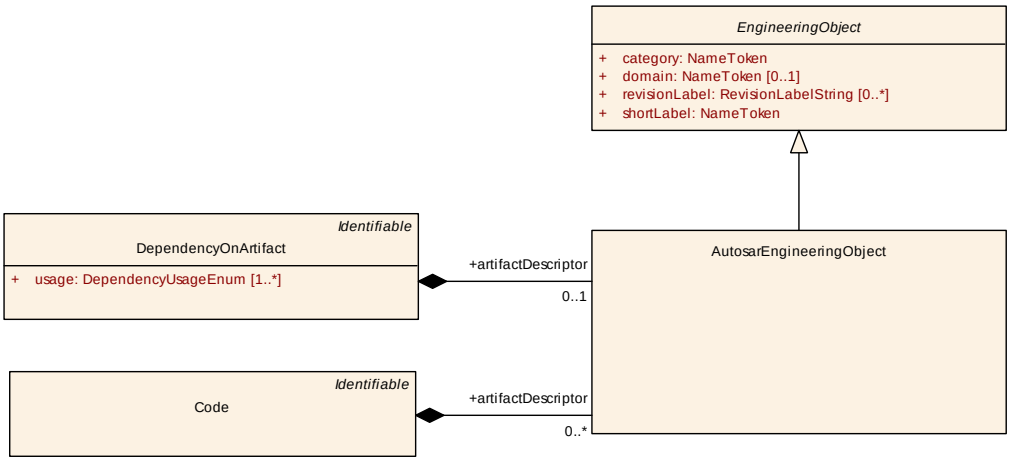


Figure 4.11: Engineering Object

Class	***EngineeringObject*** (abstract)			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::EngineeringObject			
Note	This class specifies an engineering object. Usually such an object is represented by a file artifact. The properties of engineering object are such that the artifact can be found by querying an ASAM catalog file. The engineering object is uniquely identified by domain+category+shortLabel+revisionLabel.			
Base	*ARObject*			
Subclasses	AutosarEngineeringObject, BuildEngineeringObject, Graphic			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
category	NameToken	1	attr	This denotes the role of the engineering object in the development cycle. Categories are such as • SWSRC for source code • SWOBJ for object code • SWHDR for a C-header file Further roles need to be defined via Methodology. **Tags:**xml.sequenceOffset=20
domain	NameToken	0..1	attr	This denotes the domain in which the engineering object is stored. This allows to indicate various segments in the repository keeping the engineering objects. The domain may segregate companies, as well as automotive domains. Details need to be defined by the Methodology. Attribute is optional to support a default domain. **Tags:**xml.sequenceOffset=40
revisionLabel	RevisionLabelString	*	attr	This is a revision label denoting a particular version of the engineering object. **Tags:**xml.sequenceOffset=30
shortLabel	NameToken	1	attr	This is the short name of the engineering object. Note that it is modeled as NameToken and not as Identifier since in ASAM-CC it is also a NameToken. **Tags:**xml.sequenceOffset=10

Table 4.71: EngineeringObject

Class	**AutosarEngineeringObject**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::EngineeringObject			
Note	This denotes an engineering object being part of the process. It is a specialization of the abstract class EngineeringObject for usage within AUTOSAR.			
Base	*ARObject*, *EngineeringObject*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table 4.72: AutosarEngineeringObject

The following example illustrates the usage of an `EngineeringObject` to refer to a physical file.

Listing 4.20: Example for an artifact description

```
<?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns="http://autosar.org/schema/r4.0"
         xsi:schemaLocation="http://autosar.org/schema/r4.0␣AUTOSAR_4-1-3.
            xsd">
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>demo</SHORT-NAME>
      <ELEMENTS>
        <SWC-IMPLEMENTATION>
          <SHORT-NAME>foo</SHORT-NAME>
          <ANNOTATIONS>
            <ANNOTATION>
                <ANNOTATION-TEXT>
                  <P>
                  </P>
                  <FORMULA>
                  </FORMULA>
                  <VERBATIM>
                  </VERBATIM>
                  <P>
                  </P>
                </ANNOTATION-TEXT>
            </ANNOTATION>
          </ANNOTATIONS>
          <REQUIRED-ARTIFACTS>
            <DEPENDENCY-ON-ARTIFACT>
              <SHORT-NAME>Foo</SHORT-NAME>
              <ARTIFACT-DESCRIPTOR>
                <SHORT-LABEL>FOO</SHORT-LABEL>
                <CATEGORY>SWSRC</CATEGORY>
                <DOMAIN>AUTOSAR</DOMAIN>
              </ARTIFACT-DESCRIPTOR>
              <USAGES>
                <USAGE>COMPILE</USAGE>
              </USAGES>
            </DEPENDENCY-ON-ARTIFACT>
          </REQUIRED-ARTIFACTS>
        </SWC-IMPLEMENTATION>
```

```
      </ELEMENTS>
    </AR-PACKAGE>
  </AR-PACKAGES>
</AUTOSAR>
```

[TPS_GST_00110] ⌈EngineeringObject can be resolved via a container catalog as defined in [16] in order to find the physical File.⌋*()*

The `artifactDescriptor` in the example above describes the artifact "FOO" which is of category "SWSRC". Using this information, the path to the physical file can be resolved via the following catalog example. There it is the first ABLOCK.

Listing 4.21: Example for an ASAM catalog

```
<?xml version = "1.0" encoding = "utf-8"?>
<CATALOG xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:noNamespaceSchemaLocation="catalog_V3_0_0.ml.xsd">
  <SHORT-NAME>sample</SHORT-NAME>
  <ABLOCKS>
    <ABLOCK>
      <SHORT-NAME>FOO</SHORT-NAME>
      <CATEGORY>SWSRC</CATEGORY>
      <DOMAIN>AUTOSAR</DOMAIN>
      <FILES>
        <FILE>source/c/foo.c</FILE>
      </FILES>
    </ABLOCK>
    <ABLOCK>
      <SHORT-NAME>FOO</SHORT-NAME>
      <CATEGORY>SWCT</CATEGORY>
      <DOMAIN>AUTOSAR</DOMAIN>
      <FILES>
        <FILE>AUTOSAR/xml/foo.arxml</FILE>
      </FILES>
    </ABLOCK>
    <ABLOCK>
      <SHORT-NAME>FOO</SHORT-NAME>
      <CATEGORY>ECUC</CATEGORY>
      <DOMAIN>AUTOSAR</DOMAIN>
      <FILES>
        <FILE>AUTOSAR/ecuc/foo.ecucvalues.arxml</FILE>
      </FILES>
    </ABLOCK>
  </ABLOCKS>
</CATALOG>
```

4.11 Annotations

[TPS_GST_00148] Annotation ⌈In the development process it is often required to place annotation (a kind of yellow pads) to the model. In order to support this in a generic way, the abstract meta-class `GeneralAnnotation` is applied and specialized according to the particular use case.

If no further attributes are required, the concrete meta-class Annotation is applied.⌋ *()*

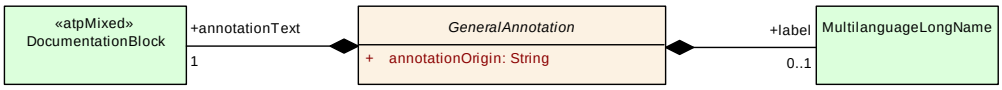


Figure 4.12: General Annotation

Class	***GeneralAnnotation*** (abstract)			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::GeneralAnnotation			
Note	This class represents textual comments (called annotations) which relate to the object in which it is aggregated. These annotations are intended for use during the development process for transferring information from one step of the development process to the next one. The approach is similar to the "yellow pads" ... This abstract class can be specialized in order to add some further formal properties.			
Base	*ARObject*			
Subclasses	Annotation, ClientServerAnnotation, DelegatedPortAnnotation, IoHwAbstractionServerAnnotation, Mode PortAnnotation, NvDataPortAnnotation, ParameterPortAnnotation, *SenderReceiverAnnotation*, Trigger PortAnnotation			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
annotation Origin	String	1	attr	This attribute identifies the origin of the annotation. It is an arbitrary string since it can be an individual's name as well as the name of a tool or even the name of a process step. **Tags:**xml.sequenceOffset=30
annotationText	DocumentationBlock	1	aggr	This is the text of the annotation. **Tags:**xml.sequenceOffset=40
label	MultilanguageLong Name	0..1	aggr	This is the headline for the annotation. **Tags:**xml.sequenceOffset=20

Table 4.73: GeneralAnnotation

Class	**Annotation**			
Package	M2::MSR::Documentation::Annotation			
Note	This is a plain annotation which does not have further formal data.			
Base	*ARObject*, *GeneralAnnotation*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table 4.74: Annotation

4.12 MultiDimensionalTime

[TPS_GST_00149] Usage of MultiDimensionalTime ⌈From timing point of view, it is important to specify a clear semantics for the timing properties (e.g. if a property has as unit seconds, or angular degrees). With the model element MultidimensionalTime, this can be done by using ASAM CSE code types (Codes for Scaling Units) as defined in [17].⌋ *()*

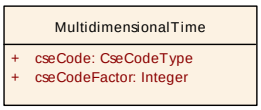


Figure 4.13: MultiDimensionalTime

Class	**MultidimensionalTime**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::MultidimensionalTime			
Note	This is used to specify a multidimensional time value based on ASAM CSE codes. It is specified by a code which defined the basis of the time and a scaling factor which finally determines the time value. If for example the cseCode is 100 and the cseCodeFactor is 360, it represents 360 angular degrees. If the cseCode is 0 and the cseCodeFactor is 50 it represents 50 microseconds.			
Base	*ARObject*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
cseCode	CseCodeType	1	attr	Specifies the time base by means of CSE codes.
cseCodeFactor	Integer	1	attr	The scaling factor for the time value based on the specified CSE code.

Table 4.75: MultidimensionalTime

Primitive	**CseCodeType**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	This primitive represents an ASAM CSE (Codes for Scaling Units) based on the definition in the ASAM-MCD-2MC-ASAP2 specification. The particular semantics is specified in [TPS_GST_00354]. **Tags:** xml.xsd.customType=CSE-CODE-TYPE-STRING xml.xsd.type=unsignedInt

Table 4.76: CseCodeType

[TPS_GST_00354] Semantics of `CseCodeType` ⌈The semantics of `CseCodeType` are defined due to Time Domain in Table 4.77, due to Angle Domain in Table 4.78 and due to Other Domain in Table 4.79.⌋ *()*

CSE-Code	Semantics Time Domain
0	1 microsec (microsecond)
1	10 microsec (microsecond)
2	100 microsec (microsecond)
3	1 msec (millisecond)
4	10 msec (millisecond)
5	100 msec (millisecond)
6	1 sec (second)
7	10 sec (second)
8	1 min (minute)
9	1 hour (hour)
10	1 day (day)
20	1 fs (femtosecond)
21	10 fs (femtosecond)
22	100 fs (femtosecond)
23	1 ps (picosecond)
24	10 ps (picosecond)
25	100 ps (picosecond)

CSE-Code	Semantics Time Domain
26	1 ns (nanosecond)
27	10 ns (nanosecond)
28	100 ns (nanosecond)

Table 4.77: `CseCodeType` in Time Domain

CSE-Code	Semantics Angle Domain
100	Angular degrees
101	Revolutions (1=360 degrees)
102	Cycle (1=720 degrees) e.g. in case of IC engines
103	Cylinder segment e.g. in case of IC engines
104	0.1 angular degree
105	0.01 angular degree
106	0.001 angular degree

Table 4.78: `CseCodeType` in Angle Domain

CSE-Code	Semantics Other Domain
997	Computing Cycle
998	When frame available; Time Source defined in the ASAP 2 keyword FRAME
999	Always if there is new value Calculation of a new upper range limit after receiving a new partial value, e.g. when calculating a complex trigger condition
1000	Non deterministic Without fixed scaling

Table 4.79: `CseCodeType` in Other Domain

4.13 TagWithOptionalValue

[TPS_GST_00358] Purpose of TagWithOptionalValue ⌈The `TagWithOptionalValue` provides the possibility to attach tags with values and tags without values to an element. Note that in opposite to `Sd` the `TagWithOptionalValue` has the ability to attach a tag to an object which has no value (see [TPS_GST_00224]).⌋*()*

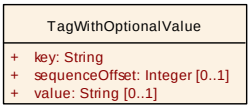


Figure 4.14: TagWithOptionalValue

Class	**TagWithOptionalValue**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::TagWithOptionalValue			
Note	A tagged value is a combination of a tag (key) and a value that gives supplementary information that is attached to a model element. Please note that keys without a value are allowed.			
Base	*ARObject*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***

▽

△

Class	**TagWithOptionalValue**			
key	String	1	attr	Defines a key.
sequenceOffset	Integer	0..1	attr	The sequenceOffset attribute supports the use case where TagWithOptionalValue is aggregated as splitable. If multiple aggregations define the same value of attribute key then the order in which the value collection is merged might be significant. As an example consider the modeling of the $PATH environment variable by means of a meta class TagWithOptionalValue. The sequenceOffset describes the relative position of each contribution in the concatenated value. The contributions are sorted in increasing integer order.
value	String	0..1	attr	Defines the corresponding value.

Table 4.80: TagWithOptionalValue

5 AbstractStructure

Abstract structures are used to define a kind of pattern which is applied by specialization. Abstract structures are established by

- abstract meta-classes
- relations between these meta-classes
  - marked as ≪atpAbstract≫ (see [TPS_GST_00022]) and/or ≪atpDerived≫ , (see [TPS_GST_00023]),
  - if it is ≪atpDerived≫ the target of the relation is marked as **derived**. This is shown in the diagrams by a slash preceding the role name.

    [TPS_GST_00150] Derived Attributes Do not Appear in the XML Schema ⌈Derived means that the attribute is not directly in the model but somehow calculated from other information in the model. As an example, base in AtpInstanceRef is calculated as the container of the first atpContext.

    In consequence of this, derived relations do not appear in the XML schema.⌋ *()*.

    [TPS_GST_00151] Specializations of Derived Relations ⌈Specializations of **derived** relations are **derived** only if this is explicitly noted. ≪atpDerived≫ means an implicit relationship which is not explicitly expressed in the model (e.g. in the as element in the xml-schema). Rationale of non abstract specializations of **derived** relations shall be well documented.⌋ *()*

    [TPS_GST_00152] Derived Union ⌈Optionally the target of the relation can be marked as **derived union**. In this case the attribute is calculated as union of all concrete relations. This is shown in diagrams at the relation end in curly brackets. Note for such relations the upper multiplicity obviously needs to be greater than one.⌋ *()*

Abstract structures are applied by

[TPS_GST_00153] Applying Abstract Structures ⌈

- subclasses of the abstract meta-classes mentioned before.
- relations between these subclasses. These relations specialize the relationships between the abstract meta-classes.

⌋ *()*

[TPS_GST_00154] Specialization of Relations ⌈

There are two kinds of specialization:

- **redefines**
  redefine replaces the abstract relationship entirely
- **subsets**
  subset contributes to the abstract relation such that it can be derived by building the union of all subsets.

The specialization is shown in diagrams at the relation end in curly brackets. Note that relations of upper multiplicity equal 1 can only be “redefined” but not “subsetted”. On the other hand, relations with upper multiplicity greater than 1 can only be “subsetted” but not “redefined”.⌋()

Figure 5.1 illustrates the approach.

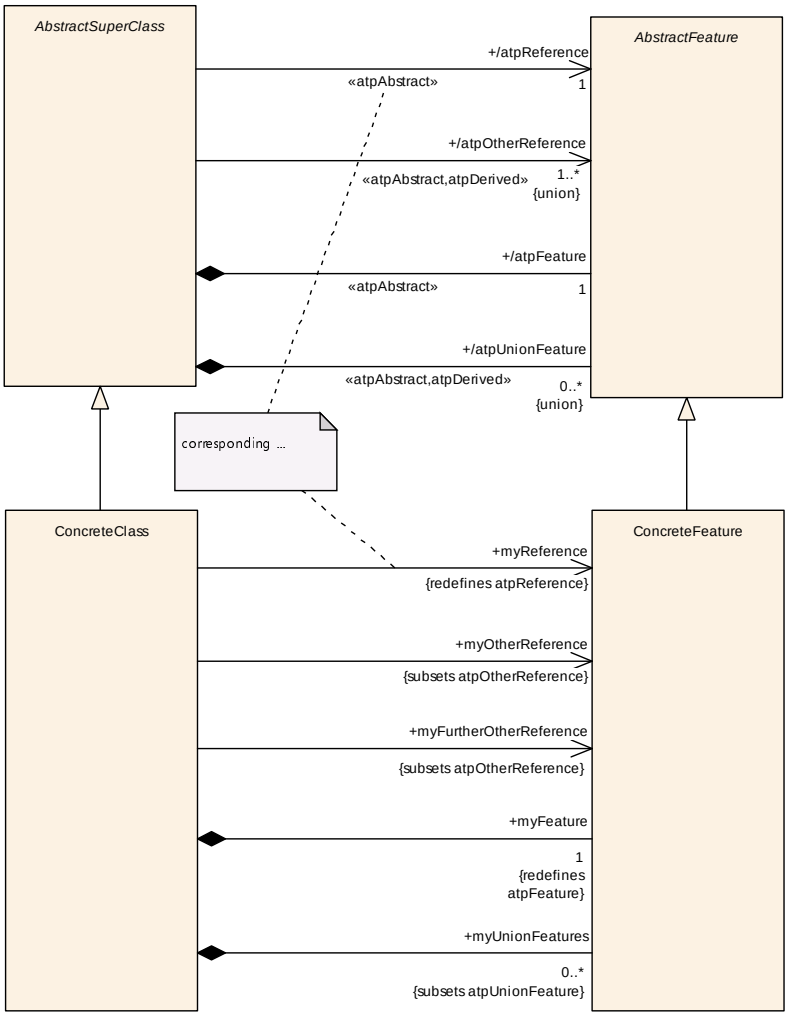


Figure 5.1: Definition and usage of abstract structures

5.1 Reusable Structural Hierarchies

5.1.1 Motivation

When designing a system it is often the case that elements in the runtime space share the same structure. A well-known example domain is object-oriented programming, where objects instantiated from the same class all have the same structure specified by that class. The ability to specify a structure once and then use it in multiple places in the design is also useful in the automotive domain. To account for this, the concepts of *types* and *prototypes* have been introduced into the AUTOSAR metamodel. A type represents a reusable structure and a prototype represents a use of such structure in a certain *role* within a type.

Consider the M1 model in Fig. 5.2. It shows an application component type ”WindowControllerType” with a port prototype ”ctrl” typed by ”ControlInterface”, and a composition type ”PowerWindowType” which has two component prototypes by the names ”leftController” and ”rightController”, both typed by ”WindowControllerType”. Hence the type ”WindowControllerType” is used twice in the ”PowerWindowType” composition - once in the role of left and once in the role of right controller. Note that though the port ”ctrl” appears graphically twice within ”PowerWindowType” it is in fact specified only once, as part of the structure of ”WindowControllerType”.

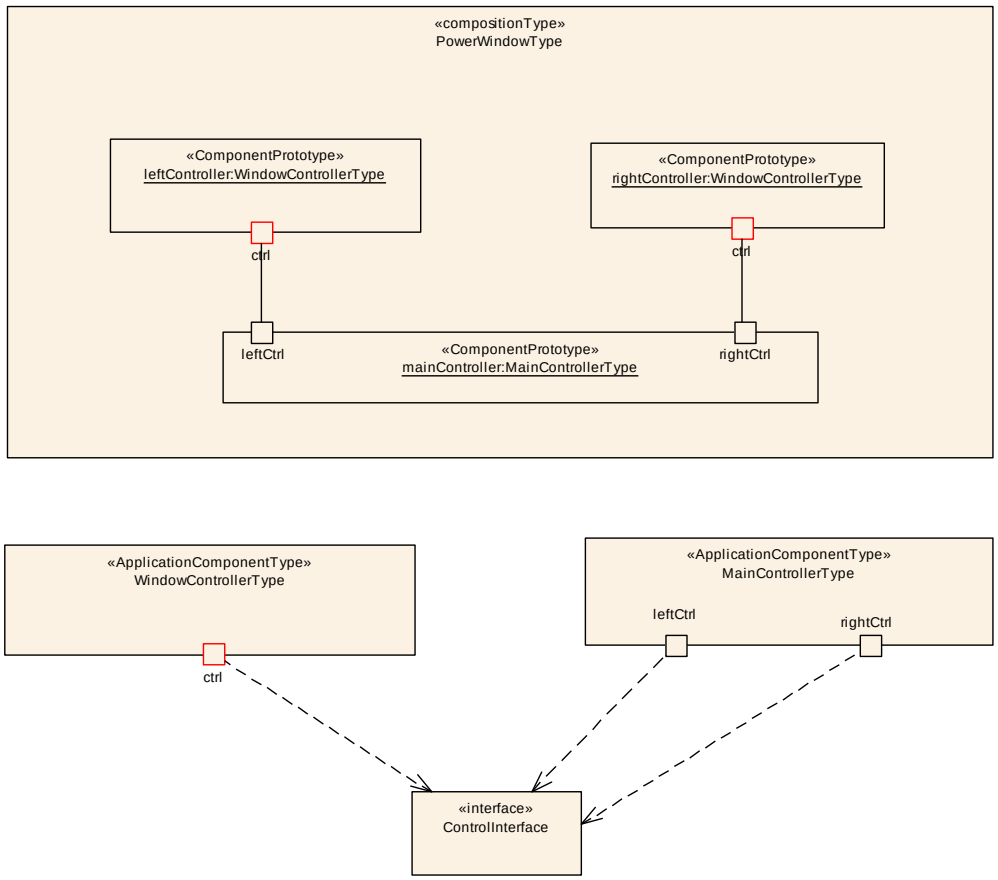


Figure 5.2: Reusable type example

The concept of reusable types results in a situation where a flat M1 model specifies deep, tree-like M0 instances. The structure of M0 instances of ”PowerWindowType” is (partly) shown in 5.3. As can be seen, there are two instances corresponding to the ”ctrl” port, defined once in M1.

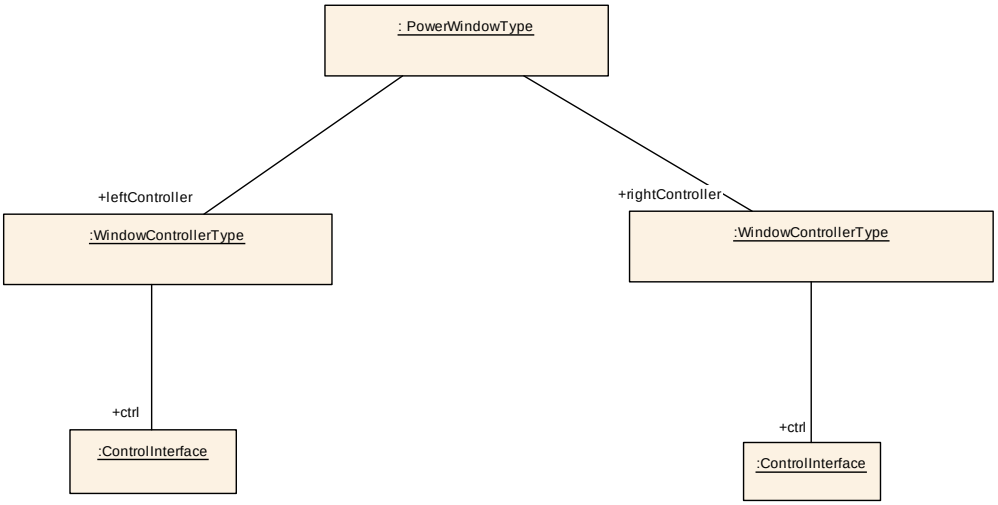


Figure 5.3: M0 instances of reusable types

It turns out that this duplication of structure in different roles also has consequences in the level of M1 models. Returning to Fig. 5.2, the ”PowerWindowType” composition also contains a component prototype ”mainController” typed by ”MainControllerType” which has left and right control ports. The connectors inside the composition connect the port of the left window controller to the left port of the main controller and the port of the right window controller to the right port of the main controller.

Recall that though the ”ctrl” port of the left and right controller appear graphically twice in the figure they in fact appear only once in the M1 specification - in the type ”WindowControllerType”. But in order to well-define the connector (e.g. in the XML description) there shall be a way to distinguish *in the M1 model specification* between those two future M0 instances. This is because we need to attach the left instance to the ”leftCtrl” port of the main controller and the right to the ”rightCtrl” port. So the problem is how to refer to distinct would-be M0 instances which originate from the same M1 model element. This is addressed by the concept of *instance refs*.

The next section introduces the abstract layer for types, prototypes, and structure elements, and provides a more detailed account of these concepts. The next one introduces the abstract layer for instance refs and explains this concept in more detail.

5.1.2 Types, Prototypes and Structure elements

Figure 5.4 shows the abstract layer for elements with internal structure.

[TPS_GST_00155] Representation of Classifier and Feature ⌈A *classifier* classifies instances according to their *features*. Here a ”classifier” means an M1 instance of (a

concrete subclass of) the M2 meta-model class AtpClassifier, and ”features” are instances of (concrete subclasses of) the M2 meta-model class AtpFeature.⌋()

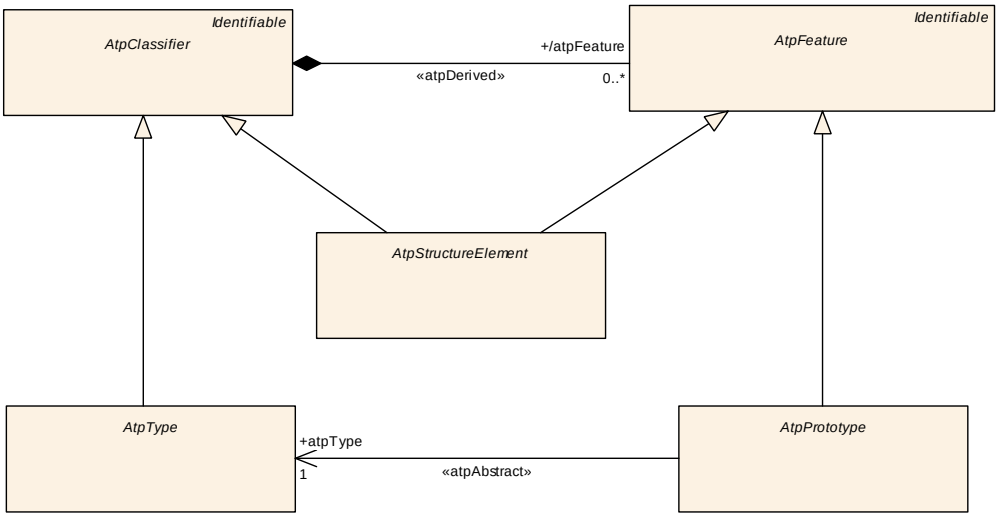


Figure 5.4: Abstract structure

An example of a classifier is some given component type and an example of a feature is some given port. So for example, the component type ”WindowControllerType” from Fig. 5.2 has a feature ”ctrl”, which is a port through which control signals arrive. Those elements are M1 elements and what they do is characterize M0 elements in the ”runtime” system. So the runtime system will have several instances of ”WindowControllerType” (two in each instance of ”PowerWindowType”) each of which will have a control port.

The set of all M0 instances of a given system may be partitioned, or classified, according to the features of each instance. A classifier represents an assembly of such features into a meaningful whole.

[TPS_GST_00156] Purpose of AtpClassifier ⌈The M2 meta-class AtpClassifier is a ”vertical” concept in that its semantics, or meaning, cuts through layers of abstraction: the meaning of this M2 class is that M1 instances of it classify the M0 instance space.⌋()

The interplay of classifiers and features is such that the way by which a feature contributes to the specification of the classifier of which it is a part is via another classifier which specifies the structure of the feature.

[TPS_GST_00157] Purpose of AtpPrototype ⌈The meta-class AtpPrototype stands for features whose structure is given by another classifier, which types them. The meta-class AtpType stands for classifiers which type prototypes.⌋()

Some classifiers do not need to be reusable. This case is captured by the concept of *structure elements*.

[TPS_GST_00158] Purpose of AtpStructureElement ⌈A structure element is a feature which is **also** a classifier and hence specifies its own structure instead of referencing to a type.

The abstract class for this kind of element is AtpStructureElement. Structure elements are simpler to define because a single element does the job of both type and prototype.⌋()

Both types and structure elements are classifiers, i.e. have M0 instances. The difference is that types are reusable within an M1 model: a given type, e.g. ”WindowControllerType”, may be used to type many prototypes within a given model. Those prototypes represent different *roles* that instances of ”WindowLifterType” - window controller components - play in the containing composition. For example, one instance may play the role of ”leftLifter” and the other of ”rightLifter”.

The meta-classes AtpType, AtpPrototype, and AtpStructureElement are abstract. They are used in the meta-model as parent classes for concrete meta-classes. Figure 5.5 shows an example where composition types are defined as containers of component prototypes which in turn are typed by component types.

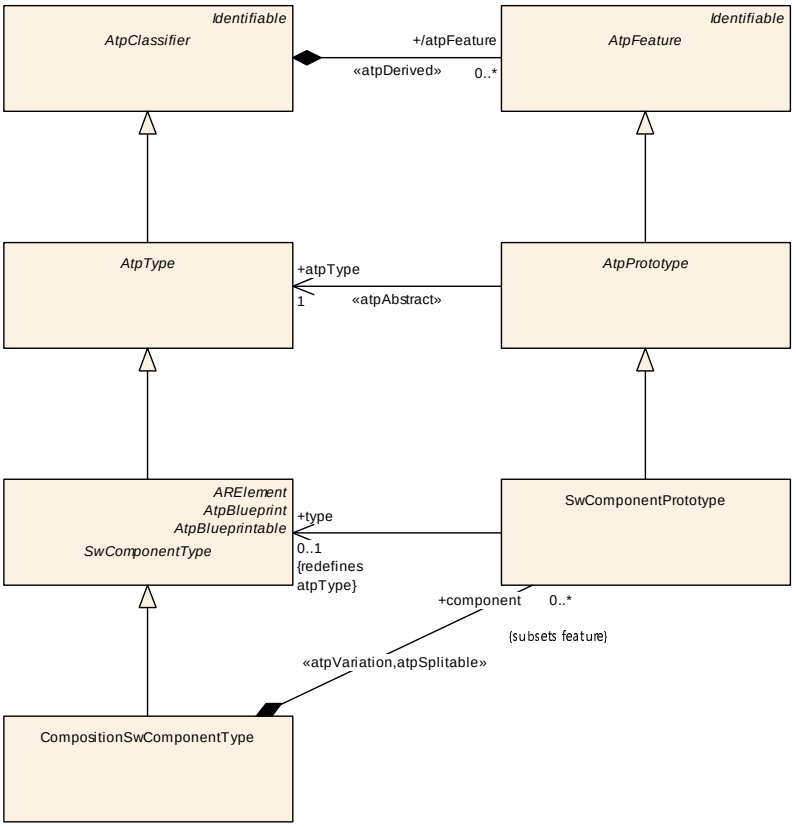


Figure 5.5: Concrete type-prototype

The structure of components as shown in the figure is a specialization of the general structure shown in figure 5.4.

[TPS_GST_00159] Deriving features in abstract structures ⌈In addition to specializing the classes, the association roles ”atpType” and ”atpFeature” are also specialized. ”atpType is redefined whereas ”atpFeature” is subsetted, in accordance with the fact that the first is abstract and the latter is a derived union.

The concrete ”type” association redefines the abstract ”atpType” one.

The union of the concrete feature associations (notice the plural) results in the derived atpFeature.⌋()

So for example the features of a given component type include all its component prototypes **and** its ports. For technical reasons, the sub-setting of features is not indicated in the meta-model diagrams.

Figure 5.6 shows an example of a concrete structure element.

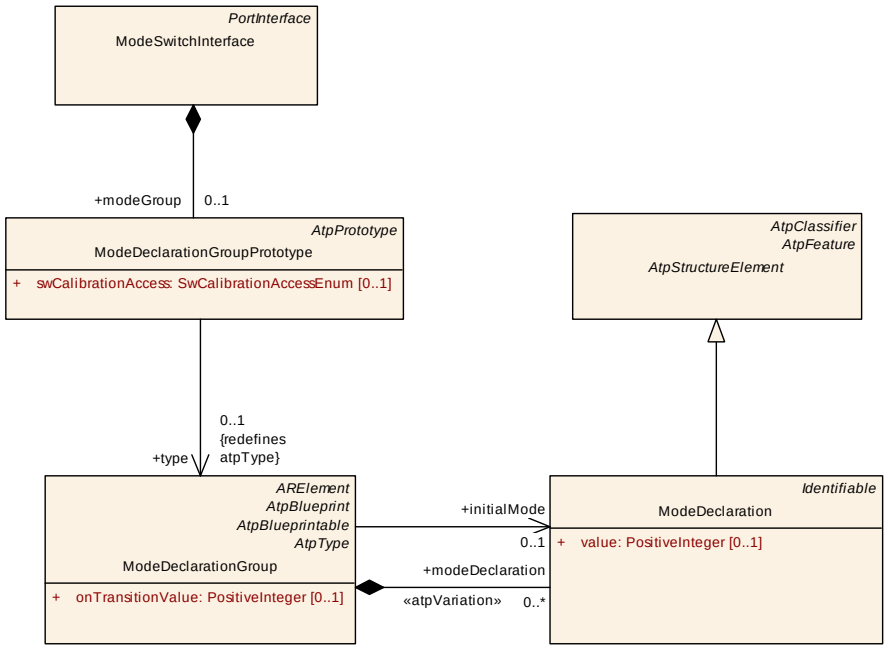


Figure 5.6: Concrete structure element

Class	***AtpClassifier*** (abstract)			
Package	M2::AUTOSARTemplates::GenericStructure::AbstractStructure			
Note	A classifier classifies M0 instances according to their features. Or: a classifier is something that has instances - an M1 classifier has M0 instances.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	*AtpStructureElement*, *AtpType*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
atpFeature	AtpFeature	*	aggr	This is a feature of the classifier. **Stereotypes:** atpDerived

Table 5.1: AtpClassifier

Class	***AtpFeature*** (abstract)
Package	M2::AUTOSARTemplates::GenericStructure::AbstractStructure
Note	Features are properties via which a classifier classifies instances. Or: a classifier has features and every M0 instance of it will have those features.
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*
Subclasses	*AtpPrototype*, *AtpStructureElement*

△

Class	***AtpFeature*** (abstract)			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table 5.2: AtpFeature

Class	***AtpInstanceRef*** (abstract)			
Package	M2::AUTOSARTemplates::GenericStructure::AbstractStructure			
Note	An M0 instance of a classifier may be represented as a tree rooted at that instance, where under each node come the sub-trees representing the instances which act as features under that node. An instance ref specifies a navigation path from any M0 tree-instance of the base (which is a classifier) to a leaf (which is an instance of the target).			
Base	*ARObject*			
Subclasses	AnyInstanceRef, ApplicationCompositeElementInPortInterfaceInstanceRef, ComponentInComposition InstanceRef, ComponentInSystemInstanceRef, *DataPrototypeInPortInterfaceInstanceRef*, Data PrototypeInSystemInstanceRef, InnerDataPrototypeGroupInCompositionInstanceRef, InnerPortGroupIn CompositionInstanceRef, InnerRunnableEntityGroupInCompositionInstanceRef, InstanceEventIn CompositionInstanceRef, ModeDeclarationGroupPrototypeInSystemInstanceRef, *ModeGroupInAtomic SwcInstanceRef*, ModeInBswModuleDescriptionInstanceRef, ModeInSwcInstanceRef, Operation ArgumentInComponentInstanceRef, *OperationInAtomicSwcInstanceRef*, OperationInSystemInstance Ref, PModeInSystemInstanceRef, ParameterDataPrototypeInSystemInstanceRef, ParameterInAtomicS WCTypeInstanceRef, PortGroupInSystemInstanceRef, *PortInCompositionTypeInstanceRef*, RModeIn AtomicSwcInstanceRef, RteEventInCompositionInstanceRef, RteEventInEcuInstanceRef, RteEventIn SystemInstanceRef, RunnableEntityInCompositionInstanceRef, SwcServiceDependencyInSystem InstanceRef, *TriggerInAtomicSwcInstanceRef*, TriggerInSystemInstanceRef, VariableAccessInEcu InstanceRef, VariableDataPrototypeInCompositionInstanceRef, VariableDataPrototypeInSystemInstance Ref, VariableInAtomicSWCTypeInstanceRef, *VariableInAtomicSwcInstanceRef*, VariableInComponent InstanceRef			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
atpBase	AtpClassifier	1	ref	This is the base from which the navigaion path starts. **Stereotypes:** atpAbstract; atpDerived
atpContext Element (ordered)	AtpPrototype	*	ref	This is one particular step in the navigation path. **Stereotypes:** atpAbstract
atpTarget	AtpFeature	1	ref	This is the target of the instance ref. In other words it is the terminal of the navigation path. **Stereotypes:** atpAbstract

Table 5.3: AtpInstanceRef

Class	***AtpPrototype*** (abstract)			
Package	M2::AUTOSARTemplates::GenericStructure::AbstractStructure			
Note	A prototype is a typed feature. A prototype in a classifier indicates that instances of that classifier will have a feature, and the structure of that feature is given by the its type. An instance of that type will play the role indicated by the feature in the owning classifier. A feature is not an instance but an indication of an instance-to-be.			
Base	*ARObject*, *AtpFeature*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	*DataPrototype*, ModeDeclarationGroupPrototype, *PortPrototype*, RootSwCompositionPrototype, Sw ComponentPrototype			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
atpType	AtpType	1	ref	This is the type of the feature. **Stereotypes:** atpAbstract

Table 5.4: AtpPrototype

Class	***AtpStructureElement*** (abstract)			
Package	M2::AUTOSARTemplates::GenericStructure::AbstractStructure			
Note	A structure element is both a classifier and a feature. As a feature, its structure is given by the feature it owns as a classifier.			
Base	*ARObject*, *AtpClassifier*, *AtpFeature*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	*AbstractAccessPoint*, *AbstractImplementationDataTypeElement*, AsynchronousServerCallResultPoint, BswModuleDescription, BulkNvDataDescriptor, ClientServerOperation, DataPrototypeGroup, *Ident Caption*, *InternalBehavior*, InternalTriggeringPoint, ModeDeclaration, ModeDeclarationMapping, Mode SwitchPoint, ModeTransition, NvBlockDescriptor, ParameterAccess, PerInstanceMemory, PortGroup, PortPrototypeBlueprint, *RTEEvent*, RunnableEntity, RunnableEntityGroup, *ServerCallPoint*, *Sw Connector*, SwcBswMapping, SwcServiceDependency, System, Trigger, VariableAccess			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table 5.5: AtpStructureElement

Class	***AtpType*** (abstract)			
Package	M2::AUTOSARTemplates::GenericStructure::AbstractStructure			
Note	A type is a classifier that may serve to type prototypes. It is a reusable classifier.			
Base	*ARObject*, *AtpClassifier*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	*AutosarDataType*, ModeDeclarationGroup, ModeDeclarationMappingSet, *PortInterface*, *SwComponent Type*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table 5.6: AtpType

5.1.3 Instance Refs

[TPS_GST_00160] Instance Reference ⌈Instance refs are M1 elements which define a particular navigation within future M0 instance trees of M1 classifiers.⌋*()*

Figure 5.7 shows an M1 instance ref called ”ctrlInRightControllerInPowerWindowType”. In each M0 instance of ”PowerWindowType”, which has the structure shown in Fig. 5.3, this instance ref identifies the bottom most instance on the right side of that figure. It has the composition type ”PowerWindowType” as *base*, the component prototype ”rightController” as *context*, and the port ”ctrl” as *target*. The context servers to navigate the instance tree in a particular path, the right path in this example.

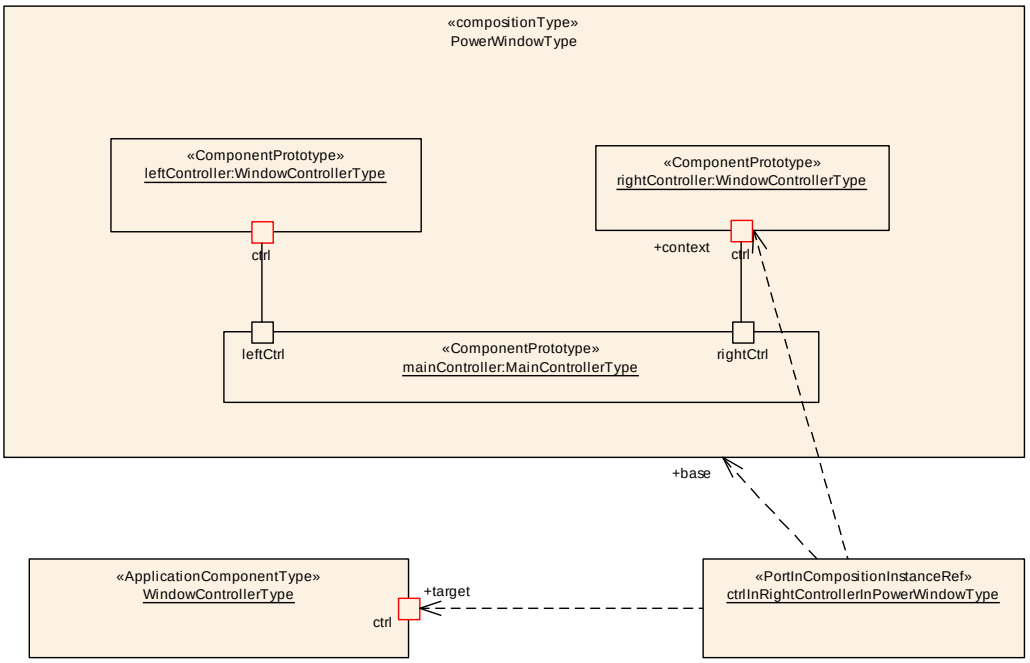


Figure 5.7: M1 Instance Ref

An M1 instance ref is an instance of a concrete subclass of the meta-class `AtpInstanceRef`. Figure 5.8 shows the abstract layer for instance refs.

[TPS_GST_00161] Definition of an instance ref ⌈Each instance ref is defined with respect to a *base* which is a classifier. What the instance ref does is specify a particular navigation leading from the root of any M0 instance of the base to an inner instance in the tree. One instance ref, i.e. one navigation, works for *any* M0 instance of the base. The navigation is specified via a series of *context elements*, which are features, and a *target* which is also a feature.⌋*()*

[TPS_GST_00162] Context path in instance ref ⌈The ordered set of context features plus the target constitutes the *path* leading from the root to the specified inner instance. In other words, the context starts with the first element of the InstanceRef. The target is always the last element in the InstanceRef-class.⌋*()*

[TPS_GST_00386] `atpContextElement`s of `InstanceRefs` shall be consistent ⌈The first `atpContextElement` in the path shall be an `atpFeature` of the `atpBase`. For all subsequent `atpContextElement`s, they shall be an `atpFeature` of the `atpType` of the previous element (which is an `AtpPrototype`).⌋*()*

[constr_2626] `atpTarget` of InstanceRefs shall be consistent ⌈The `atpTarget` of an instance ref shall either

- be an `atpFeature` owned by the `atpType` of the last `atpContextElement` or
- be an `atpFeature` owned by an `AtpStructureElement` owned by the `atpType` of the last `atpContextElement`.

⌋*()*

[TPS_GST_00387] `AtpInstanceRef` shall be close to the base ⌈An `AtpInstanceRef` shall be aggregated such that its relationship to the `AtpClassifier` referenced in the role `atpBase` is unambiguous. This is the case in one of the following situations:

- The `AtpInstanceRef` is aggregated within the `AtpFeature` referenced in the role `atpBase`.
- The `atpBase` is the root of the instance tree. It is the `AtpClassifier` which is aggregating the first `AtpFeature` representing the first (outermost) `atpContextElement`.

⌋()

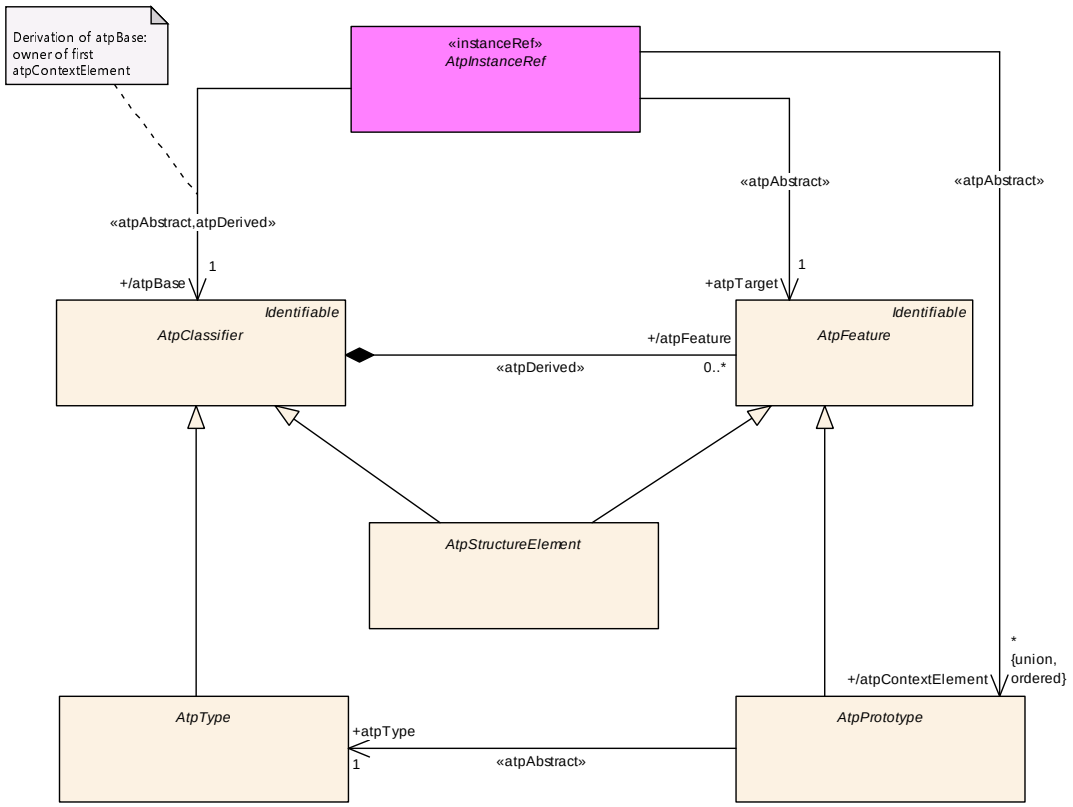


Figure 5.8: Abstract instance Refs

Figure 5.9 shows a concrete subclass of `AtpInstanceRef`, named `RPortInCompositionInstanceRef`, used to define instance refs which navigate to a port within a composition type via an inner component prototype. This concrete meta-class specifies that the base shall be a composition type, that there shall be exactly one context element which is a component prototype, and that the target is a port. The instance ref "ctrlInRightControllerInPowerWindowType" is an instance of `RPortInCompositionInstanceRef`.

Figure 5.9 also illustrates how such an instance ref is applied in the meta-model:

[TPS_GST_00043] Application of Instance Ref ⌈Instance refs are applied in the meta-model by two representations which shall exist together:

- A **dependency** with stereotype ≪instanceRef≫ which represents the intention and contributes to documentation and classtables.
- A corresponding **aggregation** of the concrete subclass of AtpInstanceRef in the source of the reference which represents the implementation and therefore contributes to the xml schema.

Note that both representations are relevant and shall exist in the meta-model.⌋()

[TPS_GST_00044] Identification of corresponding Instance Ref representations ⌈The target role name of the **dependency** and the target role name of the corresponding **aggregation** used to model an instance ref (as described in [TPS_GST_00043]) shall be identical.⌋()[1]

Please find an example of the application of [TPS_GST_00044] in Figure 5.9. The target role name of the dependency from AssemblySwConnector to AbstractRequiredPortPrototype (i.e. requester) is identical to the target role name of the aggregation of RPortInCompositionInstanceRef at AssemblySwConnector.

Listing 5.1 illustrates the ARXML representation of a scenario according to Figure 5.9, in particular the instance refs in the AssemblySwConnector at the end of the example.

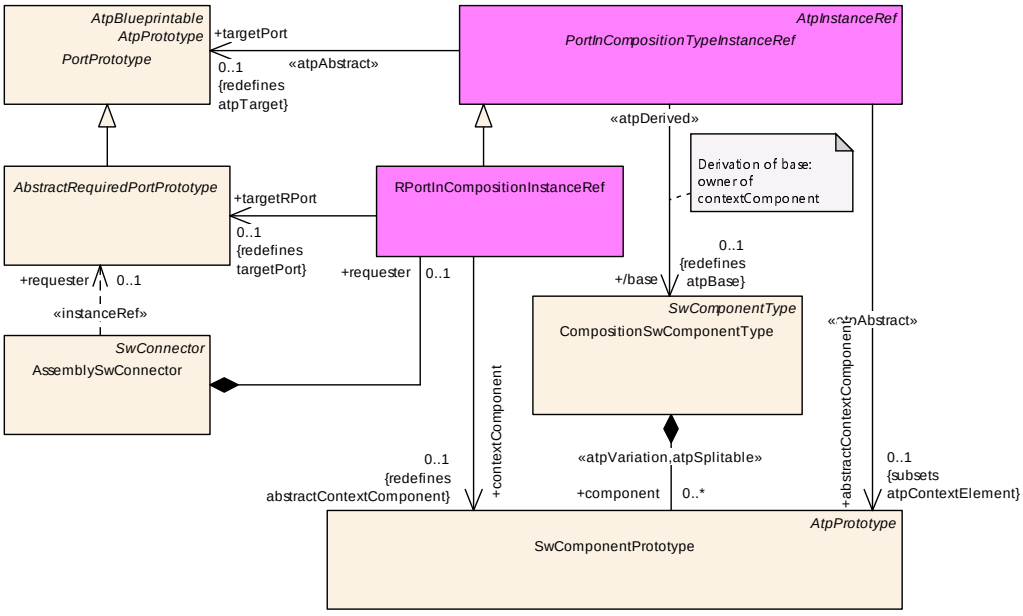


Figure 5.9: Application of an instance Ref

Listing 5.1: Application of an instance Ref

```
<AUTOSAR xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http:
    //autosar.org/schema/r4.0" xsi:schemaLocation="http://autosar.org/schema
    /r4.0␣AUTOSAR_4-1-3.xsd">
  <ADMIN-DATA>
    <USED-LANGUAGES>
```

[1]Note that the UML-tool used to generate the diagrams allows to specify role names on dependencies, even if it is not supported by UML.

```
      <L-10 xml:space="preserve" L="EN">English</L-10>
    </USED-LANGUAGES>
  </ADMIN-DATA>
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>Example</SHORT-NAME>
      <INTRODUCTION>
        <P>
          <L-1 L="EN">This example fragment illustrates an instance ref </L
              -1>
        </P>
      </INTRODUCTION>
      <ELEMENTS>
        <APPLICATION-SW-COMPONENT-TYPE>
          <SHORT-NAME>MainControllerType</SHORT-NAME>
          <PORTS>
            <P-PORT-PROTOTYPE>
              <SHORT-NAME>leftCtrl</SHORT-NAME>
            </P-PORT-PROTOTYPE>
            <P-PORT-PROTOTYPE>
              <SHORT-NAME>rightCtrl</SHORT-NAME>
            </P-PORT-PROTOTYPE>
          </PORTS>
        </APPLICATION-SW-COMPONENT-TYPE>
        <APPLICATION-SW-COMPONENT-TYPE>
          <SHORT-NAME>WindowControllerType</SHORT-NAME>
          <PORTS>
            <R-PORT-PROTOTYPE>
              <SHORT-NAME>ctrl</SHORT-NAME>
            </R-PORT-PROTOTYPE>
          </PORTS>
        </APPLICATION-SW-COMPONENT-TYPE>
        <COMPOSITION-SW-COMPONENT-TYPE>
          <SHORT-NAME>PowerWindowType</SHORT-NAME>
          <COMPONENTS>
            <SW-COMPONENT-PROTOTYPE>
              <SHORT-NAME>rightController</SHORT-NAME>
              <TYPE-TREF DEST="APPLICATION-SW-COMPONENT-TYPE">/Example/
                  WindowControllerType</TYPE-TREF>
            </SW-COMPONENT-PROTOTYPE>
            <SW-COMPONENT-PROTOTYPE>
              <SHORT-NAME>mainController</SHORT-NAME>
              <TYPE-TREF DEST="APPLICATION-SW-COMPONENT-TYPE">/Example/
                  MainControllerType</TYPE-TREF>
            </SW-COMPONENT-PROTOTYPE>
          </COMPONENTS>
          <CONNECTORS>
            <ASSEMBLY-SW-CONNECTOR>
              <SHORT-NAME>rightControl</SHORT-NAME>
              <PROVIDER-IREF>
                <CONTEXT-COMPONENT-REF DEST="SW-COMPONENT-PROTOTYPE">/
                    Example/PowerWindowType/mainController</CONTEXT-
                    COMPONENT-REF>
                <TARGET-P-PORT-REF DEST="P-PORT-PROTOTYPE">/Example/
                    MainControllerType/rightCtrl</TARGET-P-PORT-REF>
              </PROVIDER-IREF>
```

```
            <REQUESTER-IREF>
              <CONTEXT-COMPONENT-REF DEST="SW-COMPONENT-PROTOTYPE">/
                 Example/PowerWindowType/rightController</CONTEXT-
                 COMPONENT-REF>
              <TARGET-R-PORT-REF DEST="R-PORT-PROTOTYPE">/Example/
                 WindowControllerType/ctrl</TARGET-R-PORT-REF>
            </REQUESTER-IREF>
          </ASSEMBLY-SW-CONNECTOR>
        </CONNECTORS>
      </COMPOSITION-SW-COMPONENT-TYPE>
    </ELEMENTS>
  </AR-PACKAGE>
</AR-PACKAGES>
</AUTOSAR>
```

5.1.4 Any Instance Refs

The meta-class AnyInstanceRef provides a generic ability to describe an InstanceRef to any target being an AtpFeature. Nevertheless the M1 model of such an InstanceRef needs to follow the rules described in section 5.1.3.

see also AtpInstanceRef

A System could be used as an AtpFeature for example as the first contextElement in the AnyInstanceRef. Deriving the base of the AnyInstanceRef would then return the owner of System, an ARPackage. But an ARPackage is not an AtpClassifier and this would contradict the specification of the AnyInstanceRef which defines that the derived base reference of the AnyInstanceRef shall return an AtpClassifier. To avoid this inconsistency [constr_2587] applies.

[constr_2587] No System in AnyInstanceRef ⌈In consequence of [TPS_GST_00387] System shall not be contextElement nor target of an AnyInstanceRef. Otherwise atpBase would not be determined.⌋*()*

5.1.4.1 AnyInstanceRef applied to ImplementationDataTypeElement

In case the AnyInstanceRef references as a target an ImplementationDataTypeElement further constraints apply in order to ensure a consistent model. In this case it can be required to provide additional contexts even if no type-prototype sequence occurs.

[constr_2602] Completeness of AnyInstanceRef referencing ImplementationDataTypeElement ⌈If the target references an ImplementationDataTypeElement the AnyInstanceRef shall define a contextElement reference for

1. each leaf ImplementationDataTypeElement in a chain of referencing ImplementationDataTypes which is not the target

2. and each ImplementationDataTypeElement of category ARRAY in a chain of referencing ImplementationDataTypes

Thereby the contexts are created according [TPS_GST_00162] from the root to the leaf ImplementationDataTypeElement which is either typed (directly or indirectly via ImplementationDataType of category TYPE_REFERENCE) or owns the target.⌋()

Note: technically, it would be possible to avoid the context for a one-dimensional array in the hierarchy. The context is still required because then the rule for the existence of contexts becomes much simpler.

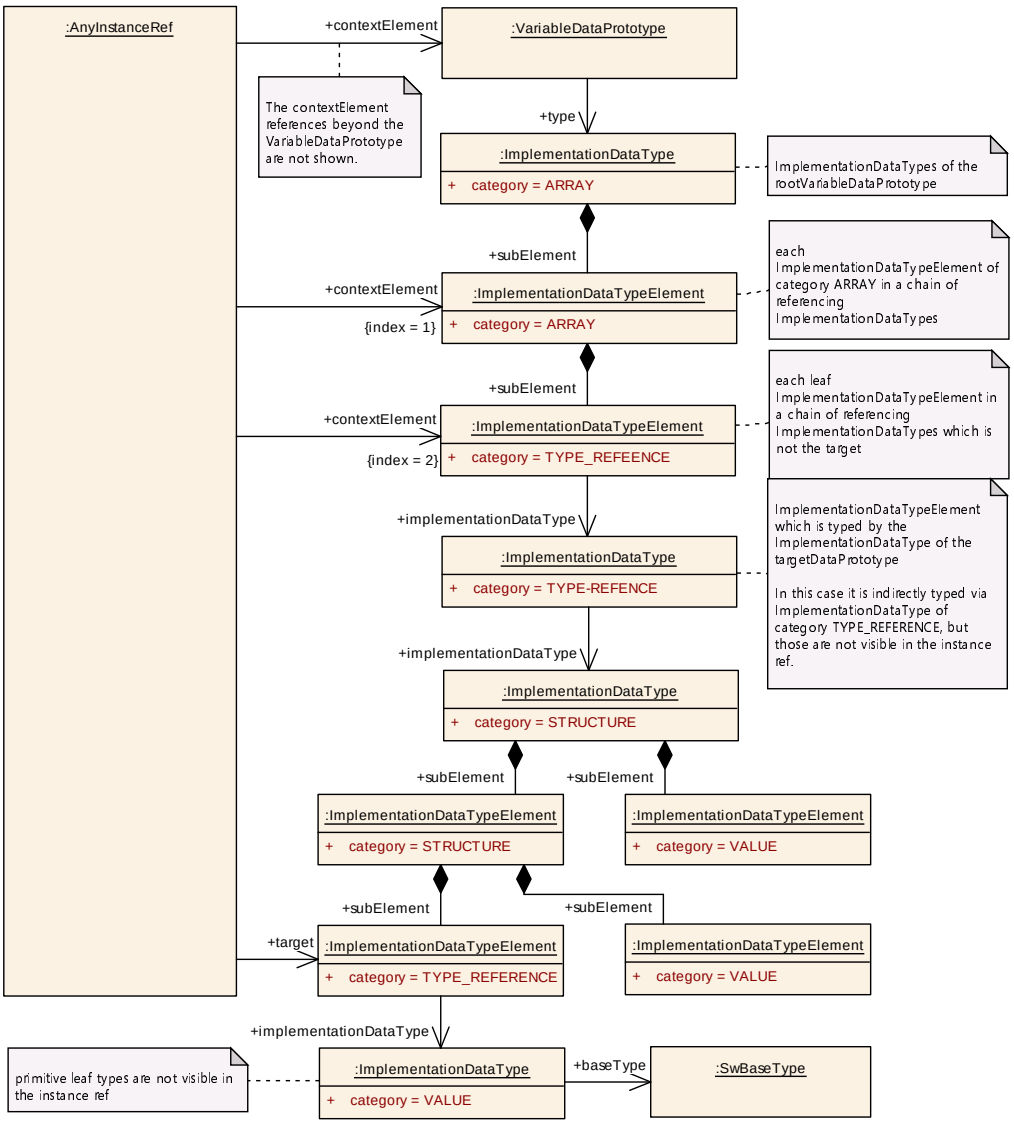


Figure 5.10: AnyInstanceRef applied to ImplementationDataTypeElement

6 Metamodeling Patterns and Model Transformation

A metamodeling pattern is a parameterized structure which, when applied to actual parameters, yields a regular, non-parameterized structure. A *structure* is just a collection of meta-classes related by associations and aggregations. The benefit of patterns is that they allow recurring structures to be used over and over again without the need to repeat their definitions. This chapter describes the concept of metamodeling patterns as well as their use and notation in the AUTOSAR Metamodel. Another advantage is that the original structure of the meta-model is preserved and not blurred with implementation details.

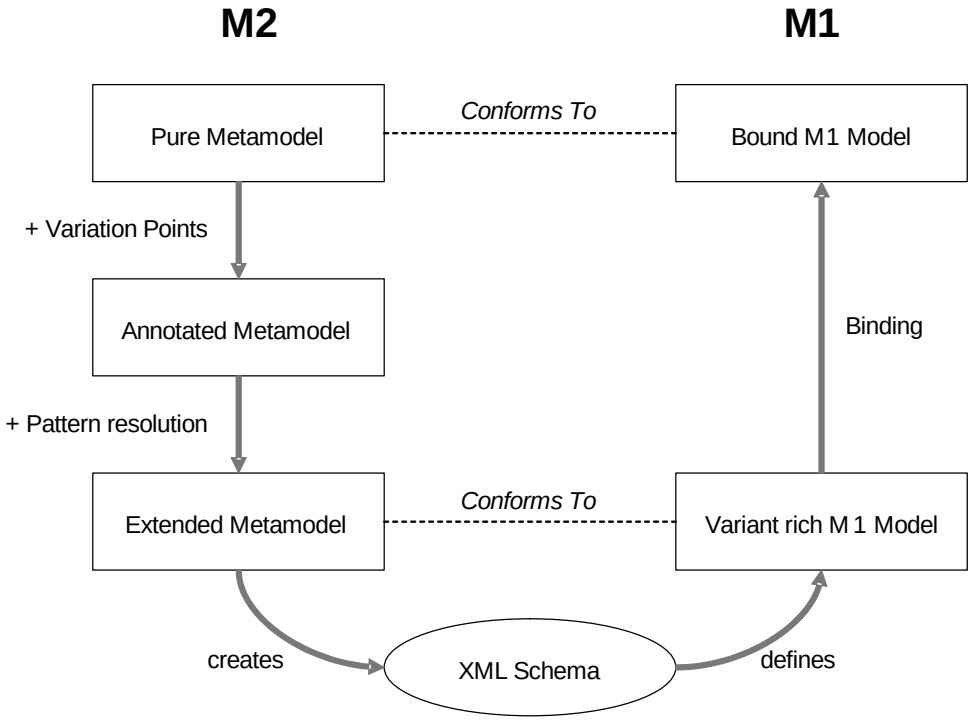


Figure 6.1: Metamodeling Patterns and Model Transformation Overview

Figure 6.1 presents a quick overview of the approach. Thereby it uses as an example the variant handling in AUTOSAR[1]:

1. **[TPS_GST_00197] Pure meta-model** ⌈AUTOSAR primarily defines a meta-model representing the general approach without annotations and transformations (e.g. without support for variation).⌋*()*
2. **[TPS_GST_00163] Annotated meta-model** ⌈The pure meta-model is then annotated using stereotypes and UML tags to describe intended transformations (e.g. ≪`atpVariation`≫ and UML tag (`vh.latestBindingTime`).

[1]Note that “model” in this context is a conceptual entity regardless of its representation. In that sense, XML, C and PDF etc. are considered as representations of the “model”. The term “model” relates to the AUTOSAR related content, not physical entities.

Note that finally only the annotated meta-model is maintained manually and part of AUTOSAR deliverables.⌋()

3. **[TPS_GST_00164] Extended meta-model** ⌈A **model transformation** converts the *annotated meta-model* into the *extended meta-model*, which is then used to generate the schema.

   The *extended meta-model* differs from the *annotated meta-model* such that it adds several more elements that provide all the information which is necessary to fully describe e.g. a variation point. It also introduces additional constraints e.g. to support the variation points.

   These additional elements are generated by applying patterns to those locations in the *annotated meta-model* that are annotated e.g. as variation points.⌋()

To illustrate consider the pattern `VHUnboundedAggregationPattern` in Fig. 6.2. It shows a parameterized structure consisting of four classes and some aggregations between them. The parameters, shown in the figure between curly brackets ({}) are:

- `WholeClass`, which is a class.
- `PartClass`, which an aggregated class.
- `partRole`, which is the role of the aggregated class.
- `vh.latestBindingTime`, which is a value of the enumeration type `BindingTimeEnum`.

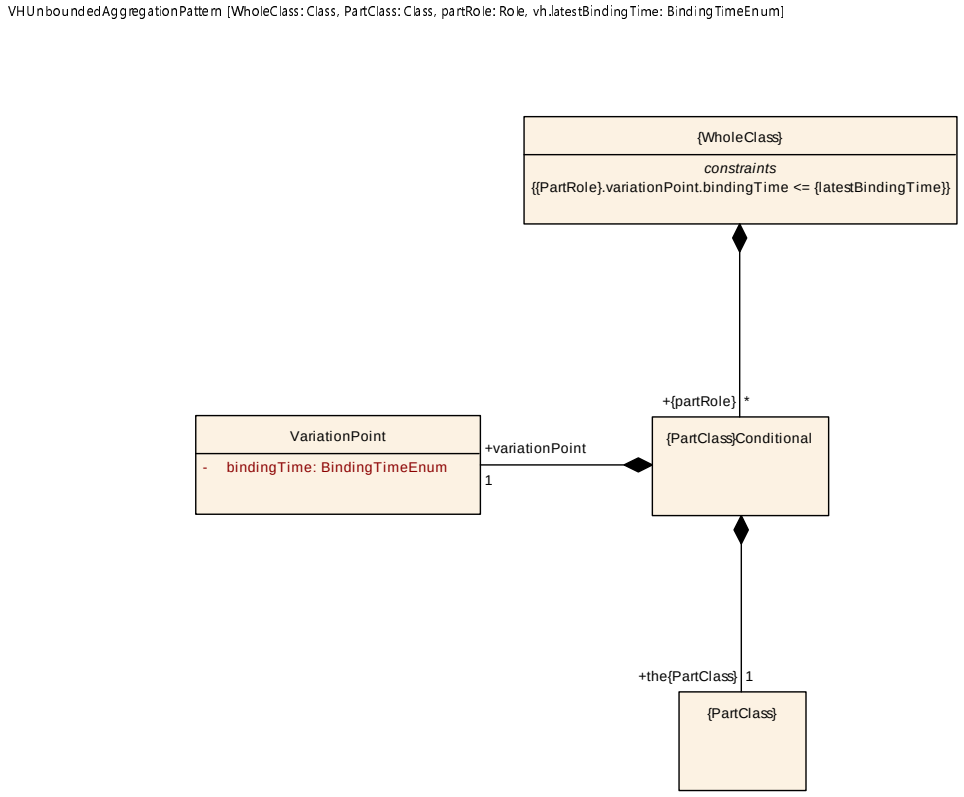


Figure 6.2: Pattern Example (VHUnboundedAggregationPattern)

This pattern may now be applied to actual parameters to yield a non-parameterized structure. For example, by assigning the actual parameters

- WholeClass = SwComponentTypes
- PartClass = PortPrototype
- partRole = port
- vh.latestBindingTime = systemDesignTime

we obtain the structure in Fig. 6.3. Many different structures may be obtained by applying the same pattern to different parameters.

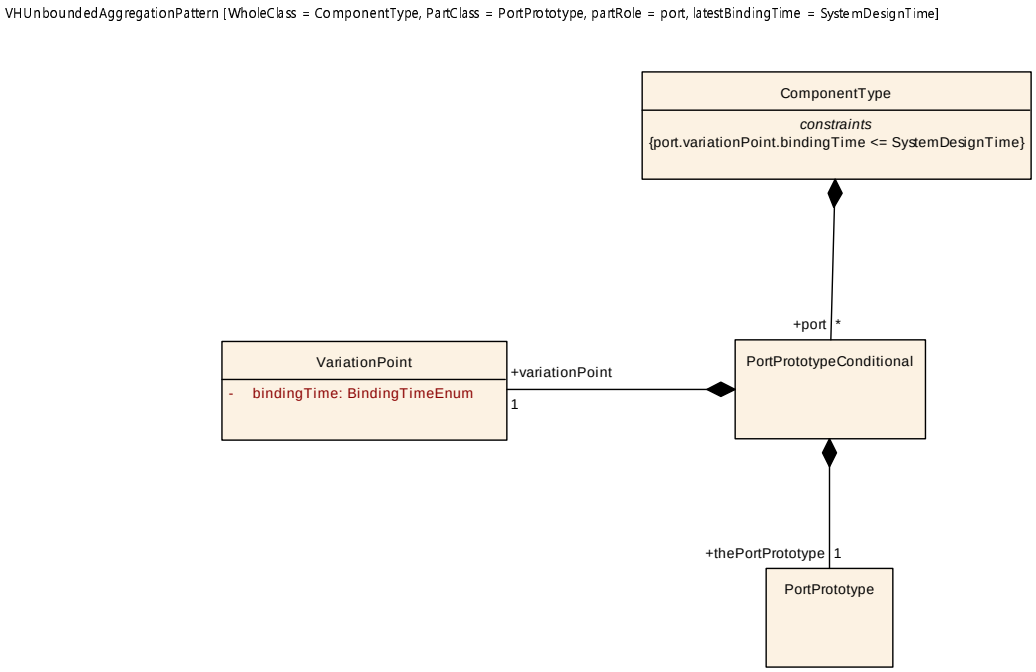


Figure 6.3: Pattern Application Result Example

6.1 Notation for Pattern Application

A recurring structure which has been abstracted into a pattern usually has a well-defined semantics explained in terms of the roles played by the parameters. This semantics will usually suggest an intuitive notation for the application of the pattern, which is of course specific to the pattern at hand.

Figure 6.4 shows a notation for the application of the VHUnboundedAggregation-Pattern on component types and ports. The notation uses an aggregation arrow between the classes playing the role of *WholeClass* and *PartClass*, decorated with the ≪atpVariation≫ stereotype. This notation suggests that the class SwCompo-nentTypes "semantically aggregates" the class PortPrototype in role port while allowing for variations. This diagram is a notation for a pattern application which results in the diagram in Fig. 6.3

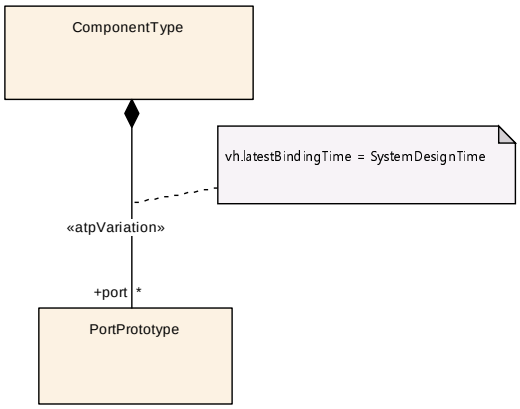


Figure 6.4: Pattern Application Notation Example

A similar notation is used for all variation-related patterns. The particular patterns are described below along with their notations. A major benefit of this kind of notation is that it is similar to a variation-free aggregation. This makes it possible to specify the variations "on top" of the regular, variation-free design instead of meddling with the conceptual design itself. It is however important to keep in mind that aggregation arrows decorated by ≪atpVariation≫ such as the one in Fig. 6.4 are not real aggregations: they are a notation for the application of a pattern (the result of which includes real aggregations).

6.2 Pattern Specification

[TPS_GST_00165] specification of a transformation pattern ⌈The specification of a pattern includes:

- The name of the pattern
- The list of parameters
- The parameterized structure
- The notation for the pattern application

⌋()

The above elements are all specified via a single diagram as illustrated in Fig. 6.5 for the unbounded aggregation pattern.

VHUnboundedAggregationPattern [WholeClass: Class, PartClass: Class, partRole: Role, vh.latestBindingTime: BindingTimeEnum]

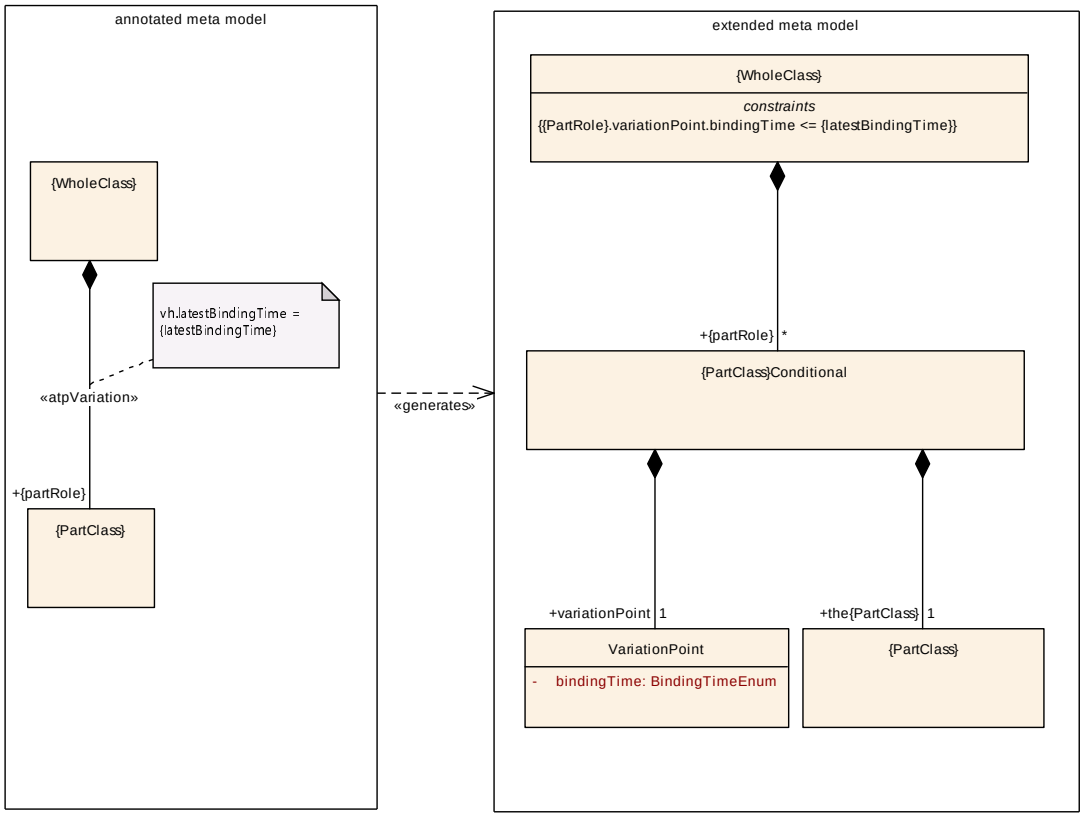


Figure 6.5: Pattern Specification Example

6.3 Model Transformations applied in the Meta-Model

The model transformation pattern mechanism is applied in the meta-model for:

- Variant Handling (chapter 7)
- Other specific purposes such as references (chapter 6.3)

6.3.1 Implementing ≪primitive≫s

This section illustrates the implementation of primitives even if it is not yet (R4.0) implemented as a true model transformation[2].

[TPS_GST_00166] Model Transformation for Primitives ⌈A meta-class of ≪primitive≫ is converted to the following elements.

1. another primitive with the same name enhanced by "_simple". For xml, this primitive results in a simple type in the generated schema and specifies the implemen-

[2]This approach is implemented in the schema generator.

tation details, such as patterns and facets. This meta-class is used for attributes tagged with `xml.attribute=true` but also as primitive type for attributes tagged with `xml.roleElement=false`, `xml.roleWrapperElement=false`, `xml.typeElement=false`, `xml.typeWrapperElement=false`.

2. a meta-class with the same name. As all other meta-classes, this meta-class inherits from ≪`atpObject`≫. Thus UML-attributes in the meta-model are finally implemented as aggregation of this meta-class unless tagged with `xml.attribute=true`.

⌋() Details to ≪`atpObject`≫ are described in chapter 6.3.3.

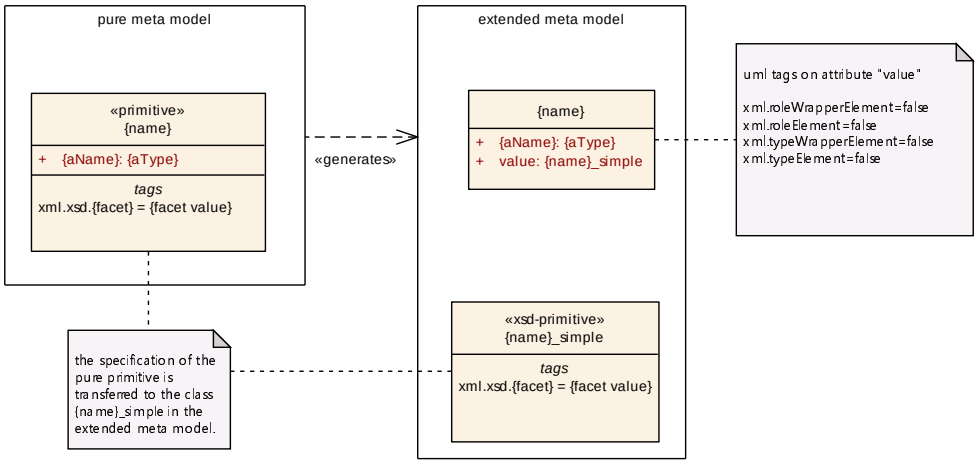


Figure 6.6: Pattern to implement Primitives

6.3.2 Implementing Associations as References

This section illustrates the implementation of associations even if it is not yet (R4.0) implemented as a true model transformation.

[TPS_GST_00020] Establishing References ⌈References between meta-classes are represented as associations. Referenced meta-classes are derived from `Referrable`, mostly being `Identifiable`. Thereby they define a `shortName` which shall be unique within its name space (see [constr_2508]). Therefore references (associations in the meta-model) are expressed by

1. by specifying a **case-sensitive** path of `shortName`s (absolute or relative)
2. the destination type of the reference
3. the **case-sensitive** name of the reference base in case of a relative reference

The name space hierarchy is defined in the Meta-Model by composite association of classes derived from `Identifiable`. Each `Identifiable` is name space of its directly or indirectly associated (composite association) classes.⌋()

[TPS_GST_00167] Case Sensitivity of References ⌈Note that The term “case-sensitive” indicates that the characters in the sets

```
{a b c d e f g h i j k l m n o p q r s t u v w x y z}
{A B C D E F G H I J K L M N O P Q R S T U V W X Y Z}
```

are respectively considered to be different. Any usage of short name shall use the same case sensitivity as the actual definition of the short name.

⌋*()*

[TPS_GST_00168] Representation of Type Reference ⌈Note that for association stereotyped with ≪`isOfType`≫ the role of the reference class is `theRoleTref`.⌋ *()*

[TPS_GST_00351] Model Transformation on Associations ⌈An association is transformed to an aggregation of an anonymous[3] meta-class with the property `dest`. This meta-class inherits from `Ref` after the transformation according to [TPS_GST_00166] was performed. `Ref` thus provides `value` which denotes the `shortName` path to the referenced object.⌋*()* Figure 6.7 illustrates the equivalent pattern.

[3]This shortcut was possible since the described approach is not implemented as a model transformation but in the schema generator. If a model transformation were applied, a particular meta-class would need to be created for each particular reference.

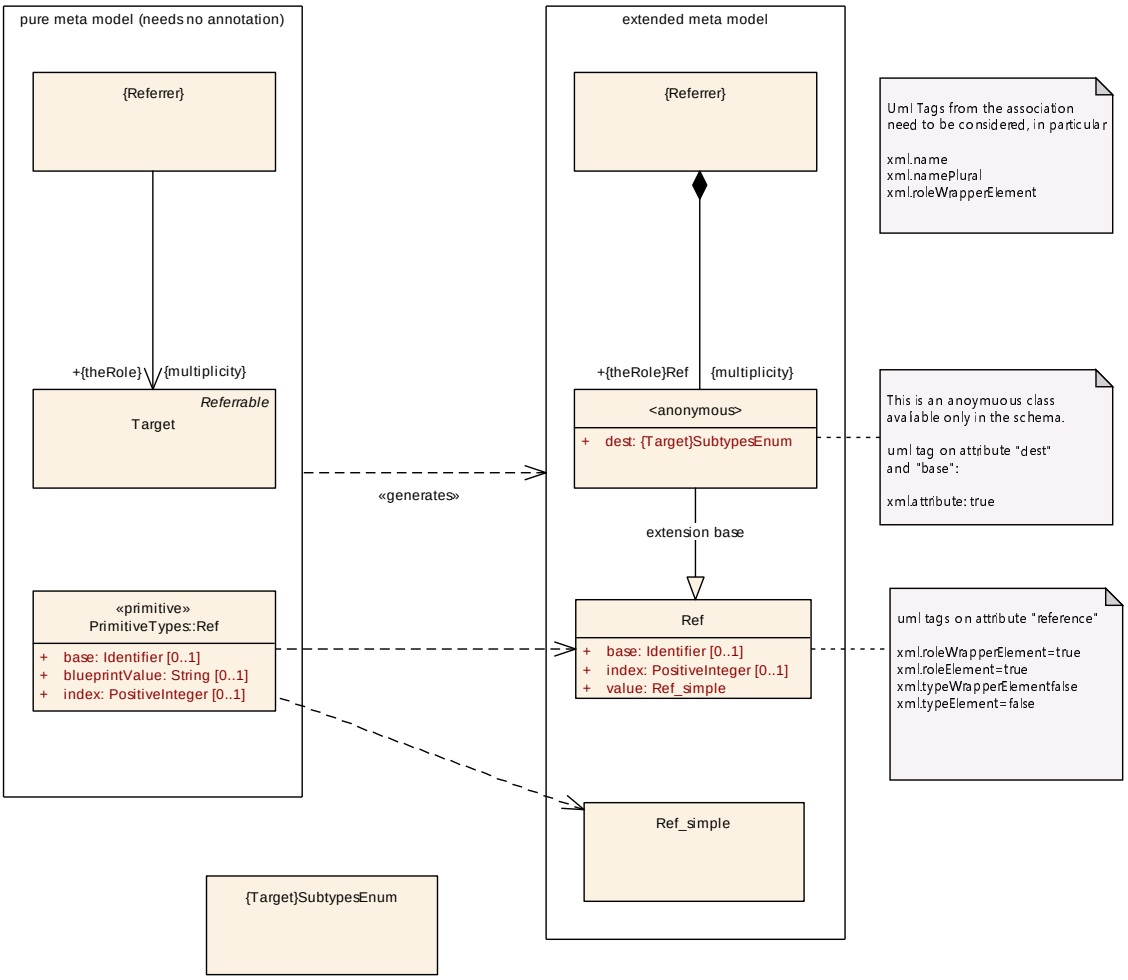


Figure 6.7: Pattern to implement references

6.3.2.1 Absolute `ShortName`-path

`ShortName` paths are composed of sequences of `shortName`s separated by '/'. The following rules apply to `shortName` paths used in AUTOSAR:

[TPS_GST_00169] Absolute `shortName`-Path ⌈

1. An absolute path is calculated by collecting the `shortName` of the model elements on the containment path from root element of the model to the referenced element.
2. Absolute paths begin with the character '/'.
3. The attribute `base` is ignored in case of an absolute reference path.

⌋()

Note that an absolute path can be converted to a relative one by removing the `shortName`-path specified by the `base`.

6.3.2.2 Relative ShortName-path

[TPS_GST_00170] Relative shortName-path ⌈A relative reference path does not start with the character '/'. A relative reference path can be converted to an absolute path by adding the appropriate shortName-path of the ReferenceBase in front of the relative shortName-path.⌋*()*

[TPS_GST_00171] Identifying the ReferenceBase of a Relative Reference ⌈The appropriate ReferenceBase is identified by

- the attribute base. This denotes the first containing ARPackage visible (the first ancestor package from the reference to the root of the model) from the reference which has a referenceBase with shortLabel equal to the base. In other words: as packages are nested the appropriate referenceBase is searched bottom up.
- alternatively the innermost package which has a referenceBase with default set to "true".

⌋*()*

[constr_2511] Named reference bases shall be available ⌈If there is a relative references, then one of the containing packages shall have a referenceBase with a shortLabel equal to the base of the reference.⌋*()*

[TPS_GST_00172] ReferenceBase in Partial Models ⌈Note that ReferenceBase is ≪atpSplitable≫. Therefore it is necessary to search for the appropriate ReferenceBase in the entire model. In other words the ReferenceBase shall be searched in all partial models.⌋*()*

Listing 6.1 illustrates the most simple form of relative references. In this case, relative references within a package shall be independent of the package name. Please note the callouts specified as xml comments <!- 1 ->:

1. <!- 1 ->: this a base of relative references. Note that isDefault denotes the fact that it is the default.
2. <!- 2 ->: this is a relative reference. Since the base attribute is not specified, the reference is resolved using the default base specified in <!- 1 ->.

Listing 6.1: Relative Reference with default base

```
<?xml version="1.0"?>
<AUTOSAR xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http:
    //autosar.org/schema/r4.0" xsi:schemaLocation="http://autosar.org/schema
    /r4.0␣AUTOSAR_4-1-3.xsd">
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>MyComponent</SHORT-NAME>
      <REFERENCE-BASES>
        <!--  1 -->
        <REFERENCE-BASE>
          <SHORT-LABEL>default</SHORT-LABEL>
          <IS-DEFAULT>true</IS-DEFAULT>
```

```
          <IS-GLOBAL>false</IS-GLOBAL>
          <BASE-IS-THIS-PACKAGE>false</BASE-IS-THIS-PACKAGE>
          <PACKAGE-REF DEST="AR-PACKAGE">/MyComponent</PACKAGE-REF>
        </REFERENCE-BASE>
      </REFERENCE-BASES>
      <ELEMENTS>
        <SENDER-RECEIVER-INTERFACE>
          <SHORT-NAME>MyInterface</SHORT-NAME>
          <DATA-ELEMENTS>
            <VARIABLE-DATA-PROTOTYPE>
              <SHORT-NAME>MyData</SHORT-NAME>
              <CATEGORY>VALUE</CATEGORY>
            </VARIABLE-DATA-PROTOTYPE>
          </DATA-ELEMENTS>
        </SENDER-RECEIVER-INTERFACE>
        <APPLICATION-SW-COMPONENT-TYPE>
          <SHORT-NAME>MyComponent</SHORT-NAME>
          <PORTS>
            <P-PORT-PROTOTYPE>
              <SHORT-NAME>MyPort</SHORT-NAME>
              <!-- 2 -->
              <!-- /MyComponent/MyInterface -->
              <PROVIDED-INTERFACE-TREF DEST="SENDER-RECEIVER-INTERFACE">
                 MyInterface</PROVIDED-INTERFACE-TREF>
            </P-PORT-PROTOTYPE>
          </PORTS>
        </APPLICATION-SW-COMPONENT-TYPE>
      </ELEMENTS>
    </AR-PACKAGE>
  </AR-PACKAGES>
</AUTOSAR>
```

Listing 6.2 illustrates that there can be multiple reference bases. In this case, in addition to the previous example another base is given for CompuMethods, since those live in a separate package.

1. `<!- 1.1 ->`: this another base of relative references. Note that `isDefault` attribute is missing. It is the same as if it is specified as `false`.
2. `<!- 2.1 ->`: this is a relative reference. The attribute `base` denotes that the base with `shortLabel` `compum` (defined in `<!- 1.1 ->` shall be used fore the relative reference. The corresponding absolute reference is `/CompuMethods/MyCompu`.

Listing 6.2: Relative Reference with explicit base

```
<?xml version="1.0"?>
<AUTOSAR xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http:
    //autosar.org/schema/r4.0" xsi:schemaLocation="http://autosar.org/schema
    /r4.0␣AUTOSAR_4-1-3.xsd">
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>MyComponent</SHORT-NAME>
      <REFERENCE-BASES>
        <!--  1 -->
```

```
      <REFERENCE-BASE>
        <SHORT-LABEL>default</SHORT-LABEL>
        <IS-DEFAULT>true</IS-DEFAULT>
        <IS-GLOBAL>false</IS-GLOBAL>
        <BASE-IS-THIS-PACKAGE>false</BASE-IS-THIS-PACKAGE>
        <PACKAGE-REF DEST="AR-PACKAGE">/MyComponent</PACKAGE-REF>
      </REFERENCE-BASE>
      <REFERENCE-BASE>
        <SHORT-LABEL>compum</SHORT-LABEL>
        <IS-DEFAULT>false</IS-DEFAULT>
        <IS-GLOBAL>false</IS-GLOBAL>
        <BASE-IS-THIS-PACKAGE>false</BASE-IS-THIS-PACKAGE>
        <!-- 1.1 -->
        <PACKAGE-REF DEST="AR-PACKAGE">/CompuMethods</PACKAGE-REF>
      </REFERENCE-BASE>
    </REFERENCE-BASES>
    <ELEMENTS>
      <SENDER-RECEIVER-INTERFACE>
        <SHORT-NAME>MyInterface</SHORT-NAME>
        <DATA-ELEMENTS>
          <VARIABLE-DATA-PROTOTYPE>
            <SHORT-NAME>MyData</SHORT-NAME>
            <CATEGORY>VALUE</CATEGORY>
            <SW-DATA-DEF-PROPS>
              <SW-DATA-DEF-PROPS-VARIANTS>
                <SW-DATA-DEF-PROPS-CONDITIONAL>
                  <!-- 2.1 -->
                  <!-- /CompuMethods/MyCompu -->
                  <COMPU-METHOD-REF DEST="COMPU-METHOD" BASE="compum">
                    MyCompu</COMPU-METHOD-REF>
                </SW-DATA-DEF-PROPS-CONDITIONAL>
              </SW-DATA-DEF-PROPS-VARIANTS>
            </SW-DATA-DEF-PROPS>
          </VARIABLE-DATA-PROTOTYPE>
        </DATA-ELEMENTS>
      </SENDER-RECEIVER-INTERFACE>
      <APPLICATION-SW-COMPONENT-TYPE>
        <SHORT-NAME>MyComponent</SHORT-NAME>
        <PORTS>
          <P-PORT-PROTOTYPE>
            <SHORT-NAME>MyPort</SHORT-NAME>
            <!-- 2 -->
            <!-- /MyComponent/MyInterface -->
            <PROVIDED-INTERFACE-TREF DEST="SENDER-RECEIVER-INTERFACE">
              MyInterface</PROVIDED-INTERFACE-TREF>
          </P-PORT-PROTOTYPE>
        </PORTS>
      </APPLICATION-SW-COMPONENT-TYPE>
    </ELEMENTS>
  </AR-PACKAGE>
  <AR-PACKAGE>
    <SHORT-NAME>CompuMethods</SHORT-NAME>
    <ELEMENTS>
      <COMPU-METHOD>
        <SHORT-NAME>MyCompu</SHORT-NAME>
        <CATEGORY>RATFUNC</CATEGORY>
```

```
        </COMPU-METHOD>
      </ELEMENTS>
    </AR-PACKAGE>
  </AR-PACKAGES>
</AUTOSAR>
```

Listing 6.3 illustrates that reference bases can be nested. In this case, the base for the CompuMethods is defined relative to the default base. This allows to maintain the entire package relationships at the beginning of a package.

It further on shows that the reference base needs to be search from inside out.

1. <!- 1.1.1 ->: this another base of relative references. Note that base attribute is now explicitly specified. It would also have been possible to relate to the default mechanism here.
2. <!- 1.2 ->: This is the base for units. Note that this base is also defined relative to the default.
3. <!- 2.2 ->: this is a relative reference. The attribute base denotes that the base with shortLabel compum (defined in <!- 1.1.1 -> shall be used fore the relative reference. The corresponding absolute reference is /MyComponent/CompuMethods/MyCompu.
4. <!- 3 ->: This is the package with the CompuMethods. it now lives as nested package in /MyComponent
5. <!- 3.1 ->: This is a reference to a Unit which is relative to the base named units which is defined in <!- 1.2 ->. Note that the reference lives in an inner package which has no package bases.

   This illustrates, that the relevant reference base is in one of the outer packages, but not necessarily in the directly containing package.
6. <!- 4 ->: This is the package with the Units. It lives in a sub package of /MyComponent.

Listing 6.3: Relative Reference with multiple and nested bases

```
<?xml version="1.0"?>
<AUTOSAR xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http:
    //autosar.org/schema/r4.0" xsi:schemaLocation="http://autosar.org/schema
    /r4.0␣AUTOSAR_4-1-3.xsd">
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>MyComponent</SHORT-NAME>
      <REFERENCE-BASES>
        <!--  1 -->
        <REFERENCE-BASE>
          <SHORT-LABEL>default</SHORT-LABEL>
          <IS-DEFAULT>true</IS-DEFAULT>
          <IS-GLOBAL>false</IS-GLOBAL>
          <BASE-IS-THIS-PACKAGE>false</BASE-IS-THIS-PACKAGE>
          <PACKAGE-REF DEST="AR-PACKAGE">/MyComponent</PACKAGE-REF>
        </REFERENCE-BASE>
```

```
    <!-- 1.1 -->
    <REFERENCE-BASE>
      <SHORT-LABEL>compum</SHORT-LABEL>
      <IS-DEFAULT>false</IS-DEFAULT>
      <IS-GLOBAL>false</IS-GLOBAL>
      <BASE-IS-THIS-PACKAGE>false</BASE-IS-THIS-PACKAGE>
      <!-- 1.1.1 -->
      <PACKAGE-REF BASE="default" DEST="AR-PACKAGE">NestedCompuMethods<
          /PACKAGE-REF>
    </REFERENCE-BASE>
    <REFERENCE-BASE>
      <SHORT-LABEL>unit</SHORT-LABEL>
      <IS-DEFAULT>false</IS-DEFAULT>
      <IS-GLOBAL>false</IS-GLOBAL>
      <BASE-IS-THIS-PACKAGE>false</BASE-IS-THIS-PACKAGE>
      <!-- 1.2 -->
      <PACKAGE-REF DEST="AR-PACKAGE">/MyUnits</PACKAGE-REF>
    </REFERENCE-BASE>
  </REFERENCE-BASES>
  <ELEMENTS>
    <SENDER-RECEIVER-INTERFACE>
      <SHORT-NAME>MyInterface</SHORT-NAME>
      <DATA-ELEMENTS>
        <VARIABLE-DATA-PROTOTYPE>
          <SHORT-NAME>MyData</SHORT-NAME>
          <CATEGORY>VALUE</CATEGORY>
          <SW-DATA-DEF-PROPS>
            <SW-DATA-DEF-PROPS-VARIANTS>
              <SW-DATA-DEF-PROPS-CONDITIONAL>
                <!-- 2.2 -->
                <!-- /MyComponent/NestedCompuMethods/MyCompu  -->
                <COMPU-METHOD-REF DEST="COMPU-METHOD" BASE="compum">
                    MyCompu</COMPU-METHOD-REF>
              </SW-DATA-DEF-PROPS-CONDITIONAL>
            </SW-DATA-DEF-PROPS-VARIANTS>
          </SW-DATA-DEF-PROPS>
        </VARIABLE-DATA-PROTOTYPE>
      </DATA-ELEMENTS>
    </SENDER-RECEIVER-INTERFACE>
    <APPLICATION-SW-COMPONENT-TYPE>
      <SHORT-NAME>MyComponent</SHORT-NAME>
      <PORTS>
        <P-PORT-PROTOTYPE>
          <SHORT-NAME>MyPort</SHORT-NAME>
          <!-- 2 -->
          <!-- /MyComponent/MyInterface -->
          <PROVIDED-INTERFACE-TREF DEST="SENDER-RECEIVER-INTERFACE">
              MyInterface</PROVIDED-INTERFACE-TREF>
        </P-PORT-PROTOTYPE>
      </PORTS>
    </APPLICATION-SW-COMPONENT-TYPE>
  </ELEMENTS>
  <AR-PACKAGES>
    <AR-PACKAGE>
      <!-- 3 -->
      <SHORT-NAME>NestedCompuMethods</SHORT-NAME>
```

```
          <ELEMENTS>
            <COMPU-METHOD>
              <SHORT-NAME>MyCompu</SHORT-NAME>
              <CATEGORY>RATFUNC</CATEGORY>
              <!-- 3.1 -->
              <!-- /MyUnits/kmh -->
              <UNIT-REF BASE="unit" DEST="UNIT">kmh</UNIT-REF>
            </COMPU-METHOD>
          </ELEMENTS>
        </AR-PACKAGE>
      </AR-PACKAGES>
    </AR-PACKAGE>
    <!-- 4 -->
    <AR-PACKAGE>
      <SHORT-NAME>MyUnits</SHORT-NAME>
      <ELEMENTS>
        <UNIT>
          <SHORT-NAME>kmh</SHORT-NAME>
        </UNIT>
      </ELEMENTS>
    </AR-PACKAGE>
  </AR-PACKAGES>
</AUTOSAR>
```

Listing 6.4 illustrates a relative reference with a global target.

1. `<!- 1 ->` denotes the `referenceBase`. Thereby it declares that via this base `Chapter` and `TextualConstraint`s are globally unique.
2. `<!- 2 ->` "ChR_4711" is a globally referable target
3. `<!- 3 ->` refers to a textual constraint ("Constr_0815") via the base in `<!- 1 ->`
4. `<!- 5 ->` "Constr_0815" is a globally referable `TextualConstraint`
5. `<!- 6 ->` refers to a chapter ("ChR_4711") via the base in `<!- 1 ->`

Listing 6.4: Relative Reference with global reference

```
<?xml version="1.0"?>
<AUTOSAR xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http:
    //autosar.org/schema/r4.0" xsi:schemaLocation="http://autosar.org/schema
    /r4.0_AUTOSAR_4-1-3.xsd">
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>Demo</SHORT-NAME>
      <REFERENCE-BASES>
        <!--  1 -->
        <REFERENCE-BASE>
          <SHORT-LABEL>globals</SHORT-LABEL>
          <IS-DEFAULT>false</IS-DEFAULT>
          <IS-GLOBAL>true</IS-GLOBAL>
          <BASE-IS-THIS-PACKAGE>true</BASE-IS-THIS-PACKAGE>
          <GLOBAL-ELEMENTS>
            <GLOBAL-ELEMENT>TRACEABLE</GLOBAL-ELEMENT>
            <GLOBAL-ELEMENT>CHAPTER</GLOBAL-ELEMENT>
```

```
      </GLOBAL-ELEMENTS>

    </REFERENCE-BASE>
  </REFERENCE-BASES>
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>mySupPackage</SHORT-NAME>
      <ELEMENTS>
        <DOCUMENTATION>
          <SHORT-NAME>myDoc</SHORT-NAME>
          <DOCUMENTATION-CONTENT>
            <!-- 2 -->
            <CHAPTER>
              <SHORT-NAME>ChR_4711</SHORT-NAME>
              <LONG-NAME>
                <L-4 L="EN">More details on Requirement 4711</L-4>
              </LONG-NAME>
              <P>
                <L-1 L="EN">Lorem ipsum dolor sit amet, consetetur
                    sadipscing elitr, sed diam nonumy eirmod tempor
                    invidunt ut labore et dolore magna aliquyam erat,<
                    XREF>
                  <!-- 3 -->
                  <!-- /Demo/mySubPackage/myDoc/Ch_001/Constr_0815
                       global: /Demo@/Const_0815 -->
                  <REFERRABLE-REF BASE="globals" DEST="TRACEABLE">
                     Constr_0815</REFERRABLE-REF></XREF>autem vel eum
                     iriure dolor in hendrerit in vulputate velit esse
                     molestie consequat, vel illum dolore eu</L-1>
              </P>
            </CHAPTER>
            <CHAPTER>
              <SHORT-NAME>Ch_001</SHORT-NAME>
              <LONG-NAME>
                <L-4 L="EN">an implementation</L-4>
              </LONG-NAME>
              <!-- 5 -->
              <TRACE>
                <SHORT-NAME>Constr_0815</SHORT-NAME>
                <LONG-NAME>
                  <L-4 L="EN">This is my specific constraint</L-4>
                </LONG-NAME>
                <P>
                  <L-1 L="EN">Lorem ipsum dolor sit amet, consetetur
                      sadipscing elitr, sed diam nonumy eirmod tempor
                      invidunt<XREF>
                    <!-- 6 -->
                    <!-- /Demo/mySupPackage/myDoc/ChR_4711
                         global: global: /Demo@/ChR_4711 -->
                    <REFERRABLE-REF BASE="globals" DEST="CHAPTER">
                       ChR_4711</REFERRABLE-REF></XREF>autem vel eum
                       iriure dolor in hendrerit in vulputate velit
                       esse molestie consequat, vel illum dolore eu</L
                       -1>
                </P>
              </TRACE>
```

```
            </CHAPTER>
          </DOCUMENTATION-CONTENT>
        </DOCUMENTATION>
      </ELEMENTS>
    </AR-PACKAGE>
  </AR-PACKAGES>
</AR-PACKAGE>
</AR-PACKAGES>
</AUTOSAR>
```

6.3.2.3 Destination Type

[TPS_GST_00173] Destination Type ⌈The destination type (specified as attribute `dest` in the reference elements) defines the type of the referenced object. The destination type can also reference subclasses (abstract and concrete). The value of this attribute is:

- If the reference in the meta-model points to an abstract class, the value of `dest` is the XML-name of the abstract class or any subclass of the same. Even if `dest` is the XML-name of an abstract class, the target of the reference can be an instance of any concrete class derived from the denoted abstract class.
- If the reference in the meta-model points to a concrete class, the value of `dest` is the name of this class. The target of the reference can only be an instance of the denoted class.

⌋()

The destination type improves the robustness of the XML descriptions such as:

- A tool can find references which refer to objects of the wrong type.
- If the referenced object is not available, the tool can indicate the correct type resp. instantiate a proper proxy.

But if the possible values of `dest` would not include abstract classes, this would cause the problem:

- It requires maintenance of the references if the reference target is changed to another subclass, even if the reference would not care about this.
- It does not propagate information of the meta-model to the XML schema

Figure 6.8 illustrates the implementation of an association.

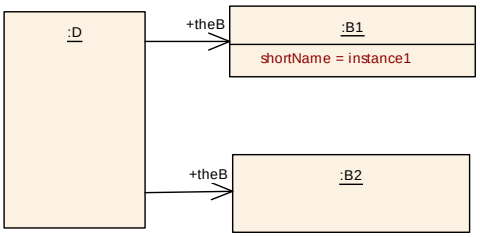


Figure 6.8: Example for linking

```
<D>
  <THE-B-REFS>
    <THE-B-REF DEST="B-1">/package1/package2/instance1</THE-B-REF>
    <THE-B-REF DEST="B-2">/package1/package2/instance2</THE-B-REF>
  </THE-B-REFS>
</D>
```

6.3.3 ≪atpObject≫ARObject

[TPS_GST_00198] Attributes for all meta-classes ⌈In the AUTOSAR meta-model there shall be attributes which are applicable to all concrete meta-classes. This can be considered as a model transformation which applies all meta-classes of ≪atpObject≫ (in particular ARObject) as superclass to all base classes.⌋*()*

See figure 6.9 for details of this transformation pattern.

Class	***ARObject*** (abstract)			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::ArObject			
Note	Implicit base class of all classes in meta-model.			
Base				
Subclasses	–all concrete metaclasses–			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
checksum	String	0..1	attr	Checksum calculated by the user's tool environment for an ArObject. May be used in an own tool environment to determine if an ArObject has changed. The checksum has no semantic meaning for an AUTOSAR model and there is no requirement for AUTOSAR tools to manage the checksum. **Tags:** xml.attribute=true xml.name=S
timestamp	DateTime	0..1	attr	Timestamp calculated by the user's tool environment for an ArObject. May be used in an own tool environment to determine the last change of an ArObject. The timestamp has no semantic meaning for an AUTOSAR model and there is no requirement for AUTOSAR tools to manage the timestamp. **Tags:** xml.attribute=true xml.name=T

Table 6.1: ARObject

Note that the pattern does not really require arguments.

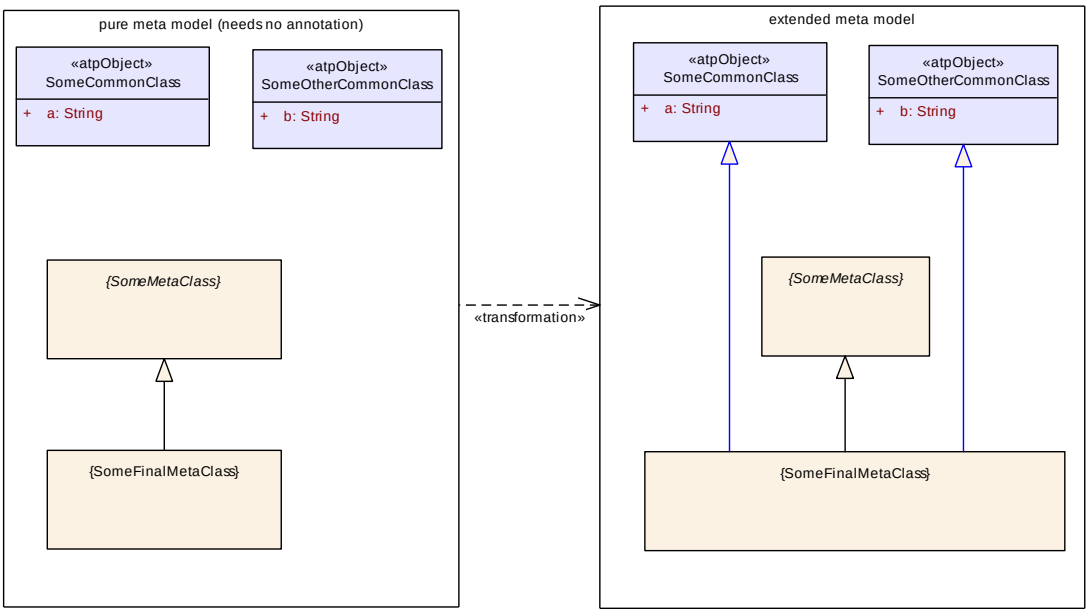


Figure 6.9: Pattern for ≪atpObject≫

7 Variant Handling

7.1 Introduction

The motivation for *Variant Handling* in AUTOSAR are to build a bridge between OEM's and suppliers, to avoid redundancy between artifacts, and to provide a basis for expressing basic product lines in AUTOSAR.

Of course, variant handling concepts do already exist at most companies, but they are typically not standardized (beyond company borders), and thus it is difficult for OEM's and suppliers to talk to each other on this subject.

Consider the following example. An OEM sends a model which contains variants to a supplier. The supplier generates code from this model, but does not resolve all variants. What the OEM gets back is object code with some variants bound, and other variants left "open" for binding at load time. This can only work if both parties speak the same language, and have the same understanding about variants. And quite often, more than two parties are involved.

Hence, variant handling in AUTOSAR is mostly about *documenting* variants:

[TPS_GST_00174] Variant Handling Terminology ⌈

- *Variation Points* are locations in the model that are variable. That is, they may not exist in all variants, or may have different characteristics in different variants.
- The *Binding Time* is the latest possible time when a variation point may be bound.
- *Binding Expressions* specify under which condition(s) a variable element exists, or determine certain variable characteristics.

⌋()

7.1.1 A Quick Overview

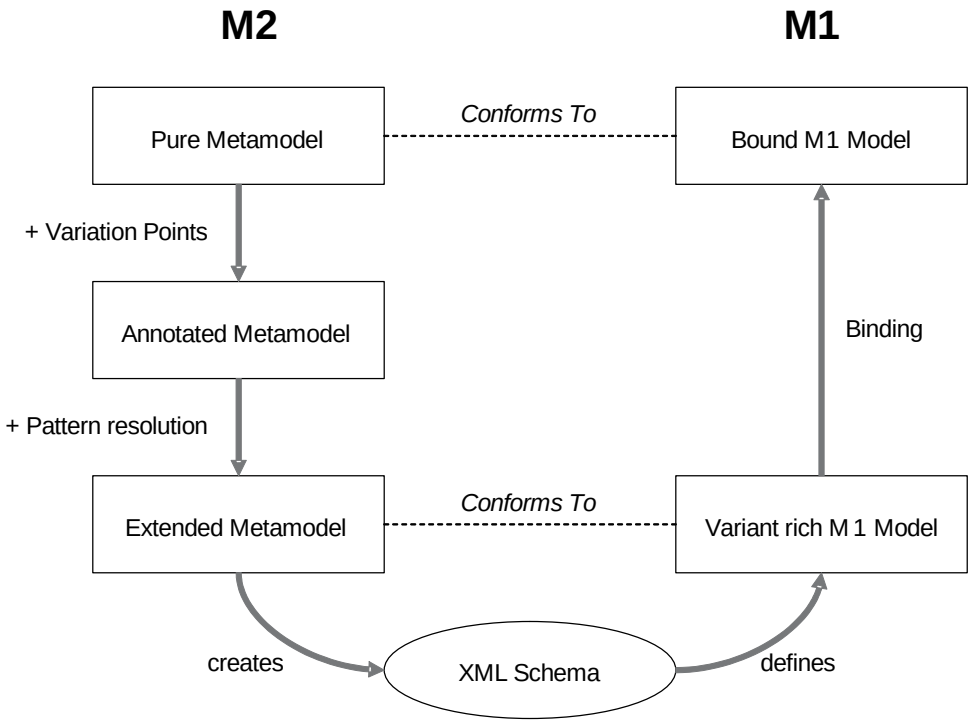


Figure 7.1: Quick Overview

In this section, we continue the meta-model transformation overview from Chapter 6.

1. (Recapitulation from Chapter 6.) AUTOSAR starts with the *pure meta-model*. The *annotated meta-model* has the same structure, but marks those locations that are variation points with a stereotype (≪`atpVariation`≫). In this section, we will define patterns that show to transform those locations into a structure that has all the information that is necessary to implement variation points.
2. All this happens on the M2 model. After the XML schema has been created, the focus shifts to M1. This is where the variants are bound.
3. **[TPS_GST_00175] Variant-rich M1 model** ⌈A *variant-rich M1 model* is a model (which shall conform to the rules defined by the *extended meta-model* [TPS_GST_00164]) which contains all supported variants simultaneously. Variation is thereby expressed by `VariationPoint` respectively `AttributeValueVariationPoint`.⌋*()*
4. **[TPS_GST_00176] Bound M1 model** ⌈The bound model represents one particular variant. It is created by from the variant-rich model which is transformed into one particular variant by binding the variation points.⌋*()*

   [constr_2503] Bound model shall be compliant to the pure meta-model ⌈The *completely bound M1 model*[1] shall adhere to the *pure meta-model* with respect

[1] Completely bound includes post build!

to consistency rules and semantic constraints defined in the related template specifications. Especially, the multiplicities in the bound model shall conform to the multiplicities and the constraints of the *pure meta-model*.⌋*()*

For example, in an invariant model, ports may have either 1:n or n:1 connections. On the other hand, this rule does not apply to a variant-rich model because the variants might overlap each other which results in m:n connections. After binding the variant-rich model and extracting one particular variant, the rules of an invariant model apply again.

As another example, the interface compatibility rules can only be applied to a particular variant.

Note that the existence of *PostBuild* variation points implies in practice that the completely bound M1 model not necessarily exists as an artifact. The reason is that the *PostBuild* variation points are bound at start-up of the ECU and obviously cannot be resolved in the M1 model. However, the resulting M0 model conforms to the bound M1 model.

[TPS_GST_00177] Remove Deselected Objects ⌈If the bound M1 model is saved as artifact, then deselected objects shall be removed.⌋*()*

[TPS_GST_00178] Remove Binding Function upon Binding ⌈It is up to the process whether the `VariationPoint` for the selected objects are left in the artifacts. If they are left in, the binding function shall be removed respectively replaced by the result in `AttributeValueVariationPoint`. By this the variation point does not look like an unbound variation point).⌋*()*

7.1.2 Variant Handling and Methodology

As shown before, variant handling takes place on different levels and in different stages in the methodology, and so may have impact on different work products. In fact, every work product can now contain variations.

[TPS_GST_00179] Scope of Variant Handling Specification ⌈In AUTOSAR, we specify only how variations are represented in ARXML. Everything else is implementation specific and not standardized on M2 level.⌋*()*

But in general, one could say that the methodology without variant handling can be extended to variant handling by introducing resolution of variation as first action of each activity.

For example, a variant rich software component description (represented by an ARXML file) can be preprocessed to a bound software component description by a variant resolving tool, and then handled as defined in the non-variant methodology.

[TPS_GST_00180] Resolving Variation Points along the Development Steps ⌈Every variation point starts in the XML representation, but is not necessarily resolved in the XML representation. If it is not resolved in a particular processing step, it needs

to be transferred to the output artifacts. For example, this means that generated C headers may contain `#ifdef` statements (for *PreCompile Time* variability).

Another example is *PostBuild* variation, where the generated object code contains conditional statements for implementing the *PostBuild* variation points, as specified in the XML representation.⌋*()*

For more details on relationship of binding time and methodology please refer to [TPS_METH_00001] (tasks), [TPS_METH_00002] (artifacts), [TPS_METH_00003] (task usage) in [18].

7.1.3 How Variant Handling is implemented in the meta-model

Variant Handling in AUTOSAR consists of the following steps[2]:

1. **[TPS_GST_00181] Annotated meta-model for Variant Handling** ⌈In the *annotated meta-model* (see [TPS_GST_00163]) – that is, on M2 level – all locations which may exhibit variability are annotated as such. This is realized by applying the stereotype ≪`atpVariation`≫.⌋*()*

   There are four kinds of locations in the meta-model which may exhibit variability:

   - Aggregations (see Section 7.2)
   - Associations (see Section 7.3)
   - Attribute Values (see Section 7.4)
   - Classes providing property sets (see Section 7.5)

   Each of the sections referenced above describes a *pattern* that is used to transform the stereotype ≪`atpVariation`≫ and its associated element into a structure which provides all the information that is necessary to describe a variation point. In other words, they describe the transformation from the *annotated meta-model* to the *enhanced meta-model*.

   The advantage of this approach is – besides not cluttering the meta-model with lots of variant handling related information – that the stereotype ≪`atpVariation`≫ now also serves as a way to document all sites in the meta-model where variations may occur.

   Note: at this point, we are still in the *annotated meta-model*. Hence, we do only define which variations are possible, but do not provide the means to specify a condition for this variation (i.e., *when* it occurs), or means to select a particular variant.

2. **[TPS_GST_00182] Notation of Latest Binding time on M2** ⌈In all of the variant handling related model transformation patterns, an UML tag

[2] In this section, and continuing in the reminder of this chapter, we are heavily borrowing terms from Section 6, especially Figure 6.1

vh.latestBindingTime (see [TPS_GST_00052]) is associated with the stereotype. More precisely, it is attached to the element that has the stereotype ≪atpVariation≫. In particular, this is the meta-class, an attribute or an aggregation/association (not the target end of the same).

Applicable values for vh.latestBindingTime are the values defined by both BindingTimeEnum and AdditionalBindingTimeEnum.⌋*()*

This tag is applied to M2 elements. It does *not* specify the binding time for the variation point, but puts an *upper limit* on the possible values for the binding time of the variation point.

[TPS_GST_00183] Representation of Binding Time ⌈The binding time of an individual variation point is specified in the attribute bindingTime of its ConditionByFormula or AttributeValueVariationPoint element. Hence the name *latest* the value stated defined with vh.latestBindingTime is the latest binding time that can be assigned to this element.⌋*()*

vh.latestBindingTime is described in more detail in Sections 7.1.6 and 7.6.4; the binding times are explained in [18].

3. **[TPS_GST_00185] Transformation on meta-model** ⌈Before the XML schema can be generated from the meta-model (for example, by the metamodel tool), all variation points are transformed into a structure that allows to specify detailed information ([TPS_GST_00164]) for each variation point, including its actual binding time, and the conditions for when the variation occurs.⌋*()*

   This is done by applying the patterns mentioned in step 1 to the *annotated meta model*. The result of this transformation is the *expanded meta-model*.

   In other words, individual variations are defined on M1 level using the means provided on M2 level.

   Details for the individual patterns are described in the respective Sections 7.2, 7.3, 7.4, and 7.5, while the class VariationPoint is explained in Section 7.6.

4. **[TPS_GST_00186] Description of Variation on M1** ⌈The model designer finally specifies the condition(s) and the latest binding time for this variation. This is done in the *variant-rich model* on M1 level.⌋*()*

   The condition specifies under what circumstances a particular variation becomes active. For example, it may select one of several alternatives; in this case, the conditions should better be mutually exclusive. Such a condition is, in a nutshell, an expression with system constants as operands.

   See Section 7.6.8 and Section 4.8 earlier in this document for more details on conditions.

5. The next step is to *choose* a particular variant. This may occur at any time before the binding time defined for this variation.

Conditions are, as already said in the step 4, essentially expressions using system constants as operands.

[TPS_GST_00187] Choosing a Particular Variant ⌈Hence, a particular variant of a system is chosen by assigning values to all system constants which are referenced by all variation points which are relevant for the given binding time.⌋*()*

It should be noted here that not all variants are chosen at the same time. Instead, there may (and typically will) be several "waves" of binding variations, each at a different time. Similarly, not all system constants need to be determined at the same time – the only restriction here is that their values shall be available at the time they are needed.

In addition, at this stage the model still contains information about possible variants. In the beginning, this are *all* variants. At a later stage, some of those variants are already bound (see Step 6) and the model only contains information on a subset of the original variants.

This step is described in detail in Section 7.8.

6. **[TPS_GST_00188] Resolving Variation Points** ⌈When a particular variant is chosen as described in [TPS_GST_00187] the selection is implemented and the variation is resolved.

   This means that all elements which aggregate a `VariationPoint` whose condition evaluates to *false* are removed from the model, and all `AttributeValueVariationPoint` elements get their `value` attribute fixed (see [TPS_GST_00178]).⌋*()*

As indicated in Section 6, not *all* variations are bound at the same (binding) time. At each binding step, unbound variations may still be in existence, and can be bound at a later (binding) time[3]. In other words, Steps 5 and 6 are usually iterated several times.

7.1.4 Not every element in the meta-model may be variant

[TPS_GST_00189] Variation is Restricted to Specific Elements. ⌈The stereotype ≪`atpVariation`≫ is only allowed for specific elements. These elements are defined by the respective AUTOSAR templates.⌋*()*

The reason for this is that when the stereotype ≪`atpVariation`≫ is attached to an element, there are consequences for other elements. For example, when a port is variable, all the connections from and to this port are also variable. The appropriate measures have to be described precisely in the associated AUTOSAR template. Hence, only a limited number of locations in the meta-model support variation.

The list of UML elements that allow variation points can be retrieved by querying the stereotypes ≪`atpVariation`≫ in the meta-model (see Section C).

[3]Of course, there is no binding time after *PostBuild*.

Technically, this restriction is implemented as follows. The XML schema is equivalent to the *extended meta-model*, and simply does not allow for attaching `VariationPoint` or other variant handling related elements to arbitrary locations. The schema makes sure that only those locations that are tagged with ≪`atpVariation`≫ in the *annotated meta-model* may contain such elements (see [TPS_GST_00195] and [TPS_GST_00199]).

7.1.5 Variation Points are optional, even for variant elements

The stereotype ≪`atpVariation`≫ marks a model element on M2 level as variable. However, the element is not necessarily variable *every time* when it is used on M1 level – on the contrary, variability should only be employed when needed.

This might imply that if a model element is potentially variable, there is a significant overhead on M1 level even if the element is used in a non-variant way. However, this is not the case: the `VariationPoint` element that serves as a starting point for variability information is always optional. Hence, it only needs to be provided only when there is need for it – if absent, there is no variability.

7.1.6 A note on Binding Times

In AUTOSAR, we handle binding times on three different levels:

1. There is the binding time that is specified on M1 level. This the value of the attribute `bindingTime` of the element `ConditionByFormula` (see Figure 7.8) and of `AttributeValueVariationPoint` (see Figure 7.4).

   [TPS_GST_00190] Semantic of `bindingTime` ⌈The value of `bindingTime` defines the *latest* binding time for this particular variation point. A variation may be chosen and bound earlier, but not later than `bindingTime`. In consequence, a variation may be bound as soon as all involved system constants have values.⌋ *()*

   On the other hand, a system constant shall have a value at the "earliest" binding time in which it is required. If we do not have this, then it might be the case that a system constant value is changed during the development cycle which again leads to inconsistencies.

2. **[TPS_GST_00221] Attachment of Latest Binding Time** ⌈ There are the individual applications of the stereotype ≪`atpVariation`≫ in the annotated meta-model on M2 level. For each such application, a tag `vh.latestBindingTime` is associated with the stereotype.⌋ *()*

   [TPS_GST_00220] Attachment of Binding Time ⌈An element that is tagged with the stereotype ≪`atpVariation`≫ is transformed into a more elaborated structure which provides all the information that is necessary to define variation points. This transformation always introduces a `ConditionByFormula` or

a AttributeValueVariationPoint, i.e., an element that has an attribute bindingTime.⌋()

[constr_2504] Constraint to bindingTime ⌈The tag vh.latestBindingTime *constraints* the value of the attribute bindingTime from [TPS_GST_00190]. Hence, it defines the latest point in methodology which is allowed as value for bindingTime of this particular application of ≪atpVariation≫.⌋()

3. There are the patterns that describe how the stereotype ≪atpVariation≫ is translated into more elaborated UML constructs which provide all the information that is necessary to define variation points.

   These patterns may again restrict the potential range of values for vh.latestBindingTime. These particular restrictions are defined in Sections 7.2.2, 7.3.2, 7.4.4, and 7.5.3.

7.1.7 A note on the impact of Variant Handling on the XML Schema

Naturally, variant handling requires some changes and extensions in the XML schema for M1 AUTOSAR models. In general, those changes are kept as simple as possible. With the exception of the *Property Set Pattern* (Section 7.5), these changes do not introduce significant structural changes in the XML, but rather add and sometimes rename elements.

Pattern specific issues of the respective XML code are documented in the respective subsections of the individual patterns (Sections 7.2.4, 7.3.4, 7.4.6, and 7.5.5).

7.1.8 Patterns are independent of each other

We define four patterns for defining variation points in aggregations, associations, attribute values and property sets. These patterns describe how the ≪atpVariation≫ translates an existing UML construct – usually a relation or an attribute – into one that provides support for describing variants.

[TPS_GST_00191] Variant Handling Patterns can be Mixed ⌈The model transformation patterns for variant handling can be combined in any fashion. For example, an element that has variant attributes may be part of a variant aggregation. Or, individual attributes in the property set pattern may be variant themselves.⌋()

7.1.9 A note on multiplicities in the Variant Handling Patterns

[TPS_GST_00192] Variant Handling Extends Upper Multiplicity ⌈The Variant Handling patterns need to transform the upper multiplicities for certain aggregations asso-

ciations within the pattern. Typically, the upper multiplicity of an aggregation is raised to $*$.

This leads to the following constellation:

1. In the *pure meta-model*, the multiplicity is $m \ldots n$.
2. In the *annotated meta-model*, the multiplicity is still $m \ldots n$.
3. In the *extended meta-model*, the multiplicity is increased to $m \ldots *$.

⌋()

Of these, the *extended meta-model* is the one that adds additional structure to describe *all* possible variants. The consequence is that the *variant-rich M1 model*, which corresponds to the *extended (M2) meta model*, may aggregate or associate more elements than there would be allowed in the original $m \ldots n$ multiplicity of the *pure meta-model*.

In other words, the *variant-rich M1 model* violates the multiplicity that is defined in the *pure meta-model*.

However, it is not as serious as it sounds, because the violation only takes place before the binding is completed. The *variant-rich M1 model* starts with all possible variants, but as we have described in Section 7.1.3, the "excess" variations are removed during the binding process.

After the binding has been completed – in the *bound M1 model* – the multiplicity shall again be within the limit as defined in the *pure meta-model*. This is ensured by semantic constraints that are part of the pattern ([constr_2503]).

7.1.10 A note on the application of the variant handling patterns

[TPS_GST_00193] Order of Pattern Resolution in Variant Handling ⌈The variant handling patterns are resolved by the model transformation on M2(e.g. by metamodel tool) in the following order:

1. Property Set Pattern
2. Aggregation Pattern
3. Association Pattern
4. Attribute Value Pattern

⌋()

It should be noted that this is not the sequence in which the patterns are presented in this document; namely, the *property set pattern* comes last. This is because our descriptions start with those patterns that are easiest to understand, and then proceed to the more complex ones.

[TPS_GST_00384] Naming conventions in variant handling patterns ⌈During the variant handling model transformation some connectors from the meta-classes participating in the variant handling pattern to other meta-classes are created. Those other meta-classes either exist already in the annotated meta-model or they are created during the transformation. The already existing meta-classes are the following:

- `VariationPoint` (for Property Set Pattern, Aggregation Pattern, and Association Pattern)
- `{Type}ValueVariationPoint` (for Attribute Value Pattern: `{Type}` as stated in [TPS_GST_00205])
- `AbstractEnumerationValueVariationPoint` (for Attribute Value Pattern)

During the transformation these meta-classes are connected by naming convention i.e. need to be found by their class name in the annotated metamodel. Therefore the transformation relies on the existence and uniqueness (by class name) of these meta-classes.⌋*()*

7.2 *Aggregation Pattern* for Variation Points

7.2.1 Description

Figure 7.2 illustrates how the metamodel tool transforms an aggregation into an M2 model with variation information.

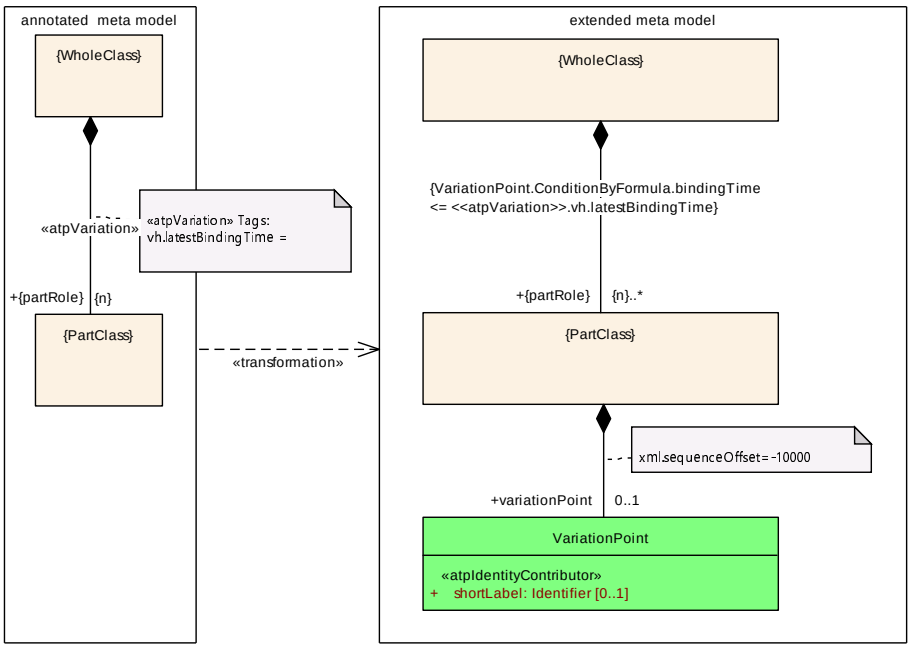


Figure 7.2: Variation Point in Aggregation

On the left side, {WholeClass} aggregates {PartClass}. This aggregation is subject to variation: the aggregation may or may not exist in the bound model. This is indicated by applying the stereotype ≪atpVariation≫ to the association.

Note that the left side only consists of {WholeClass}, ≪atpVariation≫ and {PartClass}, and does not include a condition that determines whether the variation exists. Such a condition is added as part of VariationPoint on the right side. In any case, it would not make sense to specify such a condition *here*, because the diagram is on M2 level, while the condition can only be filled in for a concrete variation point, e.g., on M1 level.

In a nutshell, the transformation works as follows:

[TPS_GST_00199] Transformation defined by Aggregation Pattern ⌈

1. {WholeClass} still aggregates {PartClass} directly.
2. The multiplicity of {partRole} changes from {n} to {n}...∗.

   The reason for the change in the multiplicity is that a model that contains variants naturally will contain more {PartClass} elements than a model in which the variants have already been bound. Note that the "excess" {PartClass} elements, i.e. the "unused" variants, are deleted in the binding process.

   This is indicated by the constraint [constr_2505].
3. {PartClass} aggregates a VariationPoint. VariationPoint aggregates further information about this variability, especially the binding time and the condition which guards the variation.

⌋() The structure of class VariationPoint is described in Section 7.6.1.

[TPS_GST_00194] Variation Points are Optional ⌈The multiplicity of VariationPoint is 0...1, meaning that the variation point is optional. If no variation point is given, then the aggregation from {WholeClass} to {PartClass} is invariant. This can be seen as equivalent to a variation point where the condition always evaluates to *true*.⌋()

[TPS_GST_00424] Variant aggregation of Abstract and Concrete subclasses ⌈If a class is aggregated as variant, no superclass of the same class shall be aggregated as variant.⌋()

[constr_2599] Maximum one VariationPoints in ≪atpMixed≫ ⌈In case an ≪atpMixed≫ meta-class is aggregated as ≪atpVariation≫ there shall not be more than one VariationPoint and the VariationPoint shall be the last aggregated element.⌋()

The reason for making VariationPoint optional is that the aggregation *may be*, but does not *have to be* variant. Hence, if the aggregation is used in a context where there is no variability – the aggregation is always there – then there is no need to add a variation point to the model (on M1 level). This helps to reduce the complexity of the resulting model, and trims the resulting XML representation.

[constr_2505] Multiplicity after binding ⌈

if Phase $\geq$ `{partRole}.BindingTime` *then* number of `{partRole}`'s = n

⌋()

where n is the final number of *PartRole* elements.

7.2.2 Binding Time

[constr_2577] Binding Time in Aggregation Pattern ⌈Within `VariationPoint`, the class `ConditionByFormula` has an attribute `bindingTime` which defines the *latest* binding time for this variation point. This binding time is further constrained by the UML tag `vh.latestBindingTime` that is attached to the aggregation see [TPS_GST_00190], [TPS_GST_00220], [TPS_GST_00221]):

`ConditionByFormula.bindingTime` $\leq$ *aggregation*.`vh.latestBindingTime`⌋()

This makes sure that the meta-model can define a restriction on M2 level. The actual binding time is specified on M1 level (when the value for the `bindingTime` attribute is fixed).

7.2.3 Multiplicity of `{PartClass}`

Table 7.1 shows how the multiplicity of `{PartClass}` changes during the transformation.

`{PartClass}` annotated meta-model	`{PartClass}` extended meta-model
$0 \ldots 1$	$0 \ldots *$
$0 \ldots n$	$0 \ldots *$
$0 \ldots *$	$0 \ldots *$
$1 \ldots n$	$1 \ldots *$
$1 \ldots *$	$1 \ldots *$
$m \ldots n$	$m \ldots *$
n	$n \ldots *$
$*$	$*$

Table 7.1: Multiplicity in the Association Pattern

The change in the multiplicity means that after the transformation, the model is *less strict* than it had been before. A multiplicity that had a fixed upper bound (or was a constant) before the transformation is replaced by one that has an unlimited upper bound.

The reason for this is that we need to provide several alternative `{PartClass}` elements and then choose one or more of them. Hence, we need to relax the original multiplicity – otherwise there would be no way to add the additional elements. And since

the number of additional {PartClass} elements to choose from cannot be known at this time, the upper multiplicity is always ∗.

7.2.4 XML Representation

The *aggregation pattern* has a low impact on the XML representation of a M1 model.

In fact, when a "normal" aggregation is made into a variation point, the only difference is that the XML element which corresponds to the {partRole} UML element gets an additional XML element named <VARIATION-POINT>.

An example for the XML code that is produced by the *aggregation pattern* can be found in Listing 7.5.

7.2.5 Notes and Restrictions

1. **[TPS_GST_00195] Annotated Meta-Model Defines Applicable Variation Points** ⌈If an association from {WholeClass} to {PartClass} is tagged with ≪atpVariation≫, then *any* occurrence of {PartClass} may aggregate a VariationPoint. In other words, {PartClass} may aggregate VariationPoint even in cases where it is not at the end of an association that is tagged with ≪atpVariation≫.

   However, using a variation in such a way would obviously not be compatible with the definition in the annotated meta-model⌋*()*.

   An example for this situation is that InternalBehavior aggregates ParameterDataPrototype in the role constantMemory with ≪atpVariation≫. In contrast to this, PortInterface also aggregates ParameterDataPrototype but not with ≪atpVariation≫. The XML Schema allows a variation point on both aggregations, while the annotated meta-model does not.

2. **[TPS_GST_00200] Schema Generator avoids duplicate VariationPoints** ⌈if {PartClass} has a subclass (say, SubPartClass) that is aggregated elsewhere (say, AnotherWholeClass) and this other aggregation is also tagged with ≪atpVariation≫, then SubPartClass would aggregate *two* variation points.

   To avoid this, the schema generator (i.e., the metamodel tool) uses a special processing step to clean up such duplicate VariationPoints.⌋*()*

3. **[TPS_GST_00201] Aggregation Pattern on Primitives** ⌈The *aggregation pattern* cannot be used for ≪primitive≫ elements. This is because of the special handling of ≪primitive≫ elements, as defined in section 4.1.1 of [3].⌋*()*

   The obvious workaround would be to make the element non-primitive.

4. **[TPS_GST_00202] Limitation of non post build** ⌈
   If `vh.latestBindingTime` is earlier than `PostBuild`, then `VariationPoint` cannot have a *PostBuild* branch, i.e. it cannot aggregate a `postBuildVariantCondition` (see [constr_2517]).⌋*()*

   This is explained in more detail in Section 7.6.6.

7.3 *Association Pattern* for Variation Points

Figure 7.2 shows how the metamodel tool transforms an association into an M2 model with variation information.

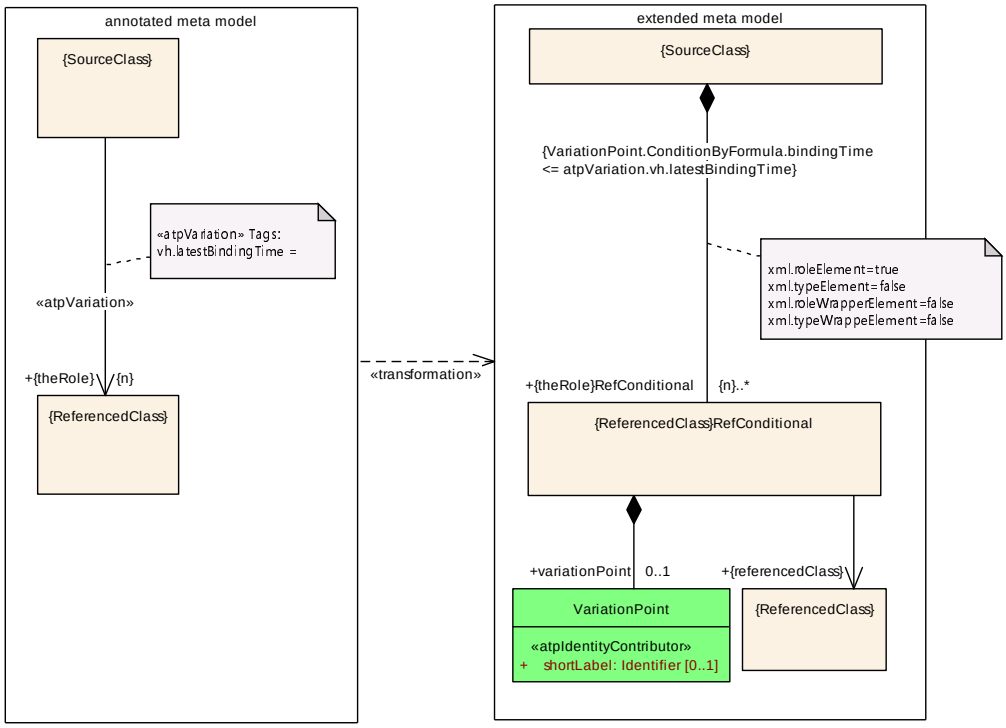


Figure 7.3: Variation Point in Association

The major difference between this pattern and the aggregation pattern (Section 7.2) is the addition of the `{ReferencedClass}RefConditional` class. The `VariationPoint` is now aggregated by `{ReferencedClass}RefConditional` instead of `{ReferencedClass}`.

7.3.1 Description

In a nutshell, the pattern for transforming an association works as follows:

[TPS_GST_00203] Transformation defined by Association Pattern ⌈

1. {SourceClass} aggregates a {ReferencedClass}RefConditional element. This is very similar to the *implementation* of nonvariant associations, which introduces a {ReferencedClass}Ref element that acts as a pointer (read: reference) to {ReferencedClass}.

2. {ReferencedClass}RefConditional aggregates a VariationPoint element. The VariationPoint controls whether the element {ReferencedClass}RefConditional exists.

   The multiplicity of this aggregation is $0 \ldots 1$. That is, VariationPoint is optional. If the VariationPoint element is omitted, then there is no variation, and {ReferencedClass}RefConditional always exists (this is equivalent to a variation point where the condition always evaluates to *true*.)

3. {ReferencedClass}RefConditional provides a reference to {ReferencedClass}.

   As said before, the nonvariant case would use {ReferencedClass}Ref instead, which also has a reference to {ReferencedClass}, but lacks the aggregated VariationPoint.

⌋()

7.3.2 Binding Time

The binding time for the *association pattern* follows the same schema as in the *aggregation pattern* (Section 7.2.2).

[constr_2578] Binding Time in Association Pattern ⌈Within VariationPoint, the class ConditionByFormula has an attribute bindingTime which defines the *latest* binding time for this variation point. This binding time is further constrained by the UML tag vh.latestBindingTime that is attached to the association (see [TPS_GST_00190], [TPS_GST_00220],[TPS_GST_00221]):

ConditionByFormula.bindingTime $\leq$ *association*.vh.latestBindingTime

⌋()

7.3.3 Multiplicity of {ReferencedClass}RefConditional

The multiplicity of {ReferencedClass}RefConditional in the *association pattern* follows the same scheme as the multiplicity of {PartClass} in the *aggregation pattern* (Section 7.2.3).

7.3.4 XML Representation

The *association pattern* has a low impact on the XML representation of a M1 model.

The element `{ReferencedClass}RefConditional` is visible on the XML level by its role name `{theRole}RefConditional`. This is only a slight difference from the non-variant case, which uses the role name `{theRole}Ref`. Both elements serve the same purpose: they are containers for the actual reference. The only difference is that the variant case adds an optional `<VARIATION-POINT>`.

An example for the XML code that is produced by the *association pattern* can be found in Figure 7.7.

7.3.5 Notes and Restrictions

[TPS_GST_00204] Handling of non variant associations ⌈The reason for making `VariationPoint` optional in `{ReferenceClass}RefConditional` is that the association *may be* variant, but does not have to be variant. Hence, if the association is used in a context where there is no variability – the association is always there – then there is no need to add a variation point to the model (on M1 level). This helps to reduce the complexity of the resulting model, and trims the resulting XML representation.⌋*()*

7.4 *Attribute Value Pattern* for Variation Points

Our first two patterns (Sections 7.2 and 7.3) dealt with the existence of a relationship between two elements. The pattern which will be described in this section implements a different kind of variation, namely a variation in the value of one or more attributes.

7.4.1 Description

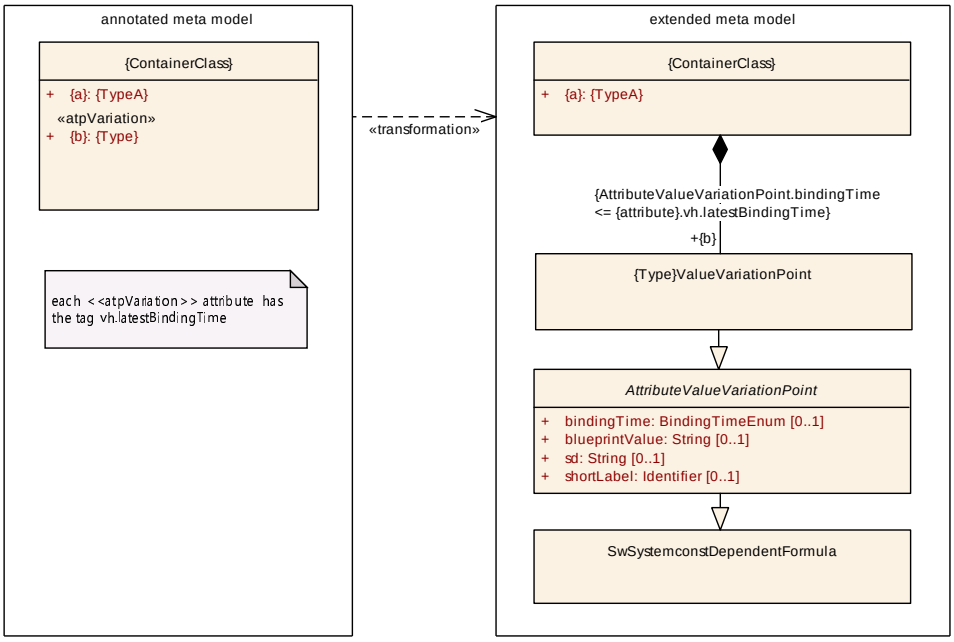


Figure 7.4: *attribute value pattern*

Class	<<atpMixedString>> ***AttributeValueVariationPoint*** (abstract)			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling::AttributeValueVariationPoints			
Note	This class represents the ability to derive the value of the Attribute from a system constant (by Sw SystemconstDependentFormula). It also provides a bindingTime.			
Base	*ARObject*, *FormulaExpression*, *SwSystemconstDependentFormula*			
Subclasses	*AbstractEnumerationValueVariationPoint*, *AbstractNumericalVariationPoint*, BooleanValueVariationPoint, FloatValueVariationPoint, IntegerValueVariationPoint, PositiveIntegerValueVariationPoint, TimeValue ValueVariationPoint, UnlimitedIntegerValueVariationPoint			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
bindingTime	BindingTimeEnum	0..1	attr	This is the binding time in which the attribute value needs to be bound. If this attribute is missing, the attribute is not a variation point. In particular this means that It needs to be a single value according to the type specified in the pure model. It is an error if it is still a formula. **Tags:**xml.attribute=true
blueprintValue	String	0..1	attr	This represents a description that documents how the value shall be defined when deriving objects from the blueprint. **Tags:**xml.attribute=true
sd	String	0..1	attr	This special data is provided to allow synchronization of Attribute value variation points with variant management systems. The usage is subject of agreement between the involved parties. **Tags:**xml.attribute=true

△

Class	<<atpMixedString>> ***AttributeValueVariationPoint*** (abstract)			
shortLabel	PrimitiveIdentifier	0..1	attr	This allows to identify the variation point. It is also intended to allow RTE support for CompileTime Variation points. **Tags:**xml.attribute=true

Table 7.2: AttributeValueVariationPoint

In this pattern, the stereotype ≪atpVariation≫ marks those attributes that are variant[4]. The transformation works as follows:

[TPS_GST_00205] Transformation defined by Attribute Value Pattern ⌈

1. {ContainerClass} is stripped of all variant attributes.
2. For each variant attribute, an element {Type}ValueVariationPoint is generated for the respective subclass of AttributeValueVariationPoint. If the attribute is an enumeration, {Type} shall be the name of the meta-class describing the enumeration.
3. {Type}ValueVariationPoint inherits from:
   - AttributeValueVariationPoint if {Type} is Integer, Float, Boolean, PositiveInteger, UnlimitedInteger, TimeValue or NameToken,
   - AbstractNumericalVariationPoint if {Type} is Numerical or Limit,
   - AbstractEnumerationValueVariationPoint if {Type} is the name of the meta-class describing the enumeration,

   whose attribute bindingTime specifies the binding time for the variant attribute (on M1 level).
4. AttributeValueVariationPoint in turn inherits from SwSystemconstDependentFormula, which implements a formula. This formula provides the value for the variant attribute.

⌋()

According to the pattern shown in Figure 7.4, step 1 leaves only attribute {a}. The class {Type}ValueVariationPoint mentioned in step 3 is described in Section 7.4.3. Inheritance of binding time is described in more detail in Section 7.4.4.

[4] In Figure 7.4, all attributes that are listed *below* ≪atpVariation≫ have this particular stereotype. In other words, {a} is a non-variant attribute, while {b} has the stereotype ≪atpVariation≫.

7.4.2 AttributeValueVariationPoint

AttributeValueVariationPoint contains four attributes, namely bindingTime, shortLabel, sd, and blueprintValue:

- bindingTime is described in Section 7.4.4.
- The shortLabel serves the same purpose as the shortLabel attribute of VariationPoint.

  [constr_2521] The shortLabel in AttributeValueVariationPoint shall be unique ⌈The shortLabel shall be unique (case insensitive) within the next enclosing Identifiable and is used to individually address variation points in the *variant-rich M1 model*.

  Note that the check for uniqueness of shortLabel shall be performed case insensitively. This supports the good practice that shortLabels should not differ in upper / lower case only which would cause a lot of confusion.

  The term 'case insensitive' indicates that the characters in the sets

  ```
  {a b c d e f g h i j k l m n o p q r s t u v w x y z}
  {A B C D E F G H I J K L M N O P Q R S T U V W X Y Z}
  ```

  are respectively considered to be the same. In other words case-insensitive check for uniqueness of shortLabel results in the fact that e.g. elements with shortLabel 'X' and 'x' are considered the same and shall not exist in the same context.⌋*()*

  See Section 7.6.2 for details.

- The sd attribute is a stripped down version of the sd member of a VariationPoint (see Section 7.6.3).

  sd is a string that may be used by an external application to add custom data.

  There are two reasons for not using a special data group like in VariationPoint. First, a variation point for an attribute value is not as structurally significant as one for full element, so it is conceivable that there less data are needed.

  Second, if AttributeValueVariationPoint would aggregate a special data group, then the resulting XML representation would require an additional wrapper – even though the aggregation is optional. This would be a significantly higher overhead than in the *aggregation pattern* and in the *association pattern*.

- The blueprintValue is used if the variation point is part of a blueprint. It contains a description which provides instructions how to derive appropriate objects from the blueprint. For more details on variation points in blueprints, see Section 7.6.11 and [2].

[constr_2567] Undefined Value in Attribute Value Blueprints ⌈If a blueprintValue is specified, then the value defined by the AttributeValueVariationPoint is not used and should therefore at least contain one term undefined which is to be refined when deriving objects from this blueprint.⌋*()*

Both shortLabel and sd are optional. bindingTime may be omitted under certain circumstances as described in Section 7.4.4.

[constr_2575] blueprintValue in blueprints only ⌈blueprintValue is only allowed in blueprints and may not be present in a system description.

⌋*()*

7.4.3 {Type}ValueVariationPoint

The figure 4.10 also reflects the modelling of {Type}ValueVariationPoint.

[TPS_GST_00206] Special meta-classes for AttributeValueVariationPoint ⌈The class AttributeValueVariationPoint comes in several different flavors, namely IntegerValueVariationPoint, FloatValueVariationPoint, BooleanValueVariationPoint, LimitValueVariationPoint, NumericalValueVariationPoint, PositiveIntegerValueVariationPoint, UnlimitedIntegerValueVariationPoint and TimeValueValueVariationPoint, as well as all concrete sub classes of the class AbstractEnumerationValueVariationPoint.⌋*()*

The concrete sub classes of the class AbstractEnumerationValueVariationPoint are automatically generated by following the AttributeValuePattern according to the (Figure 7.5).

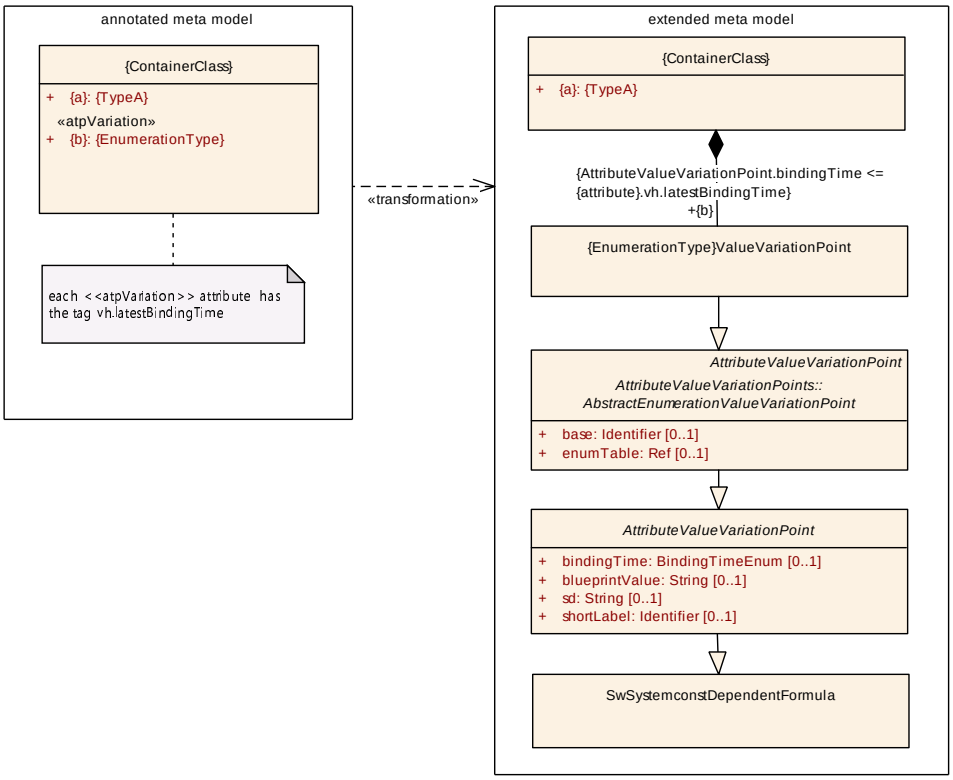


Figure 7.5: enumeration value pattern

[constr_2601] Value of `AbstractEnumerationValueVariationPoint` ⌈The formula of an `AbstractEnumerationValueVariationPoint` shall evaluate to a value for which a mapping is defined in the `EnumerationMappingTable` which is referenced by the attributes `base` and `enumTable`.⌋*()*

Note: Typically this constraint can only be checked in a complete model during binding of variability.

[TPS_GST_00373] Default `EnumerationMappingTable` ⌈If no values of `base` and `enumTable` are given for a sub class `{Type}ValueVariationPoint` of `AbstractEnumerationValueVariationPoint` then `base` shall be set to `EnumMappingTables` and `enumTable` shall be set to `{Type}`.⌋*()*

The default values of `EnumerationMappingTable` including the `EnumerationMappingEntry`s are available in `GeneralDefinitions` [7].

The reason for adding these extra classes is as follows. We could have defined the *attribute value pattern* without them, by just letting `{ContainerClass}` (Figure 7.4) aggregate a `AttributeValueVariationPoint`. But then, any trace of the *type* of the original attribute would have been lost.

7.4.4 Binding Time

[constr_2579] Binding Time in Attribute Value Pattern ⌈The meta-class AttributeValueVariationPoint has an attribute bindingTime which defines the *latest* binding time for this variation point. This binding time is further constrained by the UML tag vh.latestBindingTime that is attached to the attribute (see [TPS_GST_00190], [TPS_GST_00220], [TPS_GST_00221]):

AttributeValueVariationPoint.bindingTime $\leq$ *attribute*.vh.latestBindingTime⌋*()*

The binding time for attribute values is specified in the attribute bindingTime of AttributeValueVariationPoint. Each attribute may have its own binding time.

[TPS_GST_00207] No Binding time required for Constants ⌈The attribute bindingTime may be omitted if SwSystemconstDependentFormula specifies a single constant value for the attribute, and not an expression. That is, if the formula does not contain any references to other system constants, nor any functions or operators, just a single value. In this case, the value is fixed from the start. In all other cases, the attribute bindingTime shall be present.⌋*()*

[constr_2632] No postbuild variation for attribute values ⌈The tag vh.latestBindingTime is limited to preCompileTime and earlier binding times, i.e. (blueprintDerivationTime, systemDesignTime and codeGenerationTime) in the Attribute Value pattern.⌋*()*

7.4.5 Multiplicity of AttributeValueVariationPoint

In an M1 AUTOSAR model, one or more {Type}ValueVariationPoint elements shall be created for each variant attribute in the original {ContainerClass}. The exact number is determined as follows:

[TPS_GST_00210] Multiplicity of AttributeValueVariationPoint ⌈

- If the original attribute had a multiplicity of 1, then exactly one {Type}ValueVariationPoint is generated.
- If the original attribute was an array (i.e., it had a multiplicity of $[0 \ldots *]$, $[1 \ldots *]$ or more generally $[m \ldots n]$, then as many {Type}ValueVariationPoints shall be created as there are entries in the array.

⌋*()*

In this case, the *sequential order* of the corresponding XML elements may be significant and then shall correspond to the actual succession of elements in the array.

7.4.6 XML Representation

Without variability, both the attributes {a} and {b} in Figure 7.4 would be represented in the XML schema as individual elements, each holding a constant value.

The *attribute value pattern* replaces variant attributes with a system constant expression. If this expression is just a value, then the resulting XML has the same structure as in the non-variant case (except for the binding time, but even this attribute may be omitted for such expressions). A complicated expression may of course add significant overhead compared to the non-variant case.

The shortLabel (represented as optional XML attribute SHORT-LABEL) and sd (represented as optional XML attribute SD) and bindingTime (represented as optional XML attribute BINDING-TIME) are optional and do not add any overhead to the XML representation if they are not present.

An example for the XML code that is produced by the *attribute value pattern* can be found in Figure 7.9.

7.4.7 Notes and Restrictions

1. **[TPS_GST_00211] AttributeValueVariationPoint does not support PostBuild Variation** ⌈The binding time for an AttributeValueVariationPoint is at most *preCompileTime*. We do not support to use such a variation point with a postBuild binding time. This is because such a behavior is already covered by calibration parameters.⌋*()*

2. **[TPS_GST_00212] Existence of Attribute cannot be subject to Variation** ⌈It is not possible to model the *existence* of an attribute.

   The obvious way to do this would be to move the attribute in question to a separate element, aggregate this element and then make the aggregation a variation point.⌋*()*

   See *aggregation pattern*, Section 7.2.

3. **[TPS_GST_00213] Arrays should have the same Binding Time** ⌈Since each element of an array is specified with a separate AttributeValueVariationPoint element, it is no longer mandatory that all array elements have the same binding times. However, it is considered good practice to use identical binding times for all array elements.⌋*()*

4. **[TPS_GST_00214] Extending the Application of Attribute Value Pattern** ⌈The *attribute value pattern* is only defined for attributes of type Integer, Float, Boolean, Numerical, Limit PositiveInteger and UnlimitedInteger. Any extension to other types – as long as they are covered by the expression language – would require a change in the meta-model (and the meta-model tool).⌋ *()*

5. **[TPS_GST_00215] Rationale for BindingTime being optional in `AttributeValueVariationPoint`** ⌈The rationale for making `bindingTime` optional is that this makes the XML representation simpler. This is especially helpful for `EvaluatedVariantSet`s, which use the *attribute value pattern* to define values for system constants. In a typical use case, these values are constants, not formulas.⌋*()*

7.5 *Property Set Pattern* for Variation Points

Like the previous pattern, this one also deals with variations in attributes. However, the pattern introduced in Section 7.4 requires that every variant attribute is annotated with a stereotype ≪`atpVariation`≫. This also means that the M2 meta-model has to decide which attribute is variant, and which is not. Such a decision is not always practical, e.g. when there are a large number of attributes, each of which may be subject to variation. The pattern which we define in this section follows a different approach.

[TPS_GST_00216] Approach on Property Set Pattern ⌈By applying the stereotype ≪`atpVariation`≫ to the *meta-class* that contains the attributes (and not to individual attributes of the meta-class), we can define all attributes as potentially be subject to variation. Attributes are then partitioned into several sets of variant attributes, each of which has a variation point.⌋*()*

7.5.1 Example

Consider the following example. A class named `PropertiesClass` has six attributes: `attr1`, `attr2`, `attr3`, `attr4`, `attr5`, and `attr6`. Of these, the first four attributes are variant, while `attr5` and `attr6` are invariant.

Now, one solution would be to apply ≪`atpVariation`≫ to the `attr1`, `attr2`, `attr3`, and `attr4`. In this case, each attribute would be its own variation point, and could be varied independently.

However, it might be that there are three different sets of values for `attr1`, `attr2`, `attr3`, and `attr4`:

	attr1	attr2	attr3	attr4
Set1	1	1	2	2
Set2	2	3	4	1
Set3	3	1	6	3

Table 7.3: Example for *Property Set Pattern*

The *property set pattern* allows us to specify these three sets. When the stereotype ≪atpVariation≫ is applied to PropertiesClass (in the annotated meta-model), the class is transformed (in the extended meta-model) into a new PropertiesClass which has *no* attributes, but aggregates one or more classes named PropertiesClassConditional which now contain the attributes[5]. Each PropertiesClassConditional aggregates a variation point, very much like {PartClass} in the aggregation pattern (see Section 7.3).

The idea is that a concrete model will contain one instance of PropertiesClass (without attributes), which aggregates *four* (not three!) instances of PropertiesClassConditional. The first three instances hold Set1, Set2, and Set3, respectively. The remaining instance of PropertiesClassConditional contains the invariant attributes, namely attr5 and attr6. Their VariationPoints shall have appropriate conditions that make sure that exactly one of the first three instances PropertiesClassConditional can be selected. The last PropertiesClassConditional may omit its VariationPoint because the attributes that are defined there are invariant.

How can it be that the first three PropertiesClassConditional hold different attributes than the last one? Actually, PropertiesClassConditional contains a copy of all attributes that were in the original PropertiesClass (in the annotated meta-model), but the lower multiplicity of each attribute is reduced to 0 – that is, all attributes are optional. This way, each instance of PropertiesClassConditional may hold an arbitrary subset of the original attributes. The condition is that after the variation has been bound, the *sum* of all attributes shall be equal to the original set of attributes[6].

The property set pattern is actually a bit more flexible than the example shows. First, the variant attributes do not need to be in the same set; they can also be distributed over several PropertiesClassConditional classes. Second, the pattern not only applies to attributes, but also to aggregated and associated elements. Third, the pattern may be combined with the attribute value pattern (and others) to create even more powerful variant structures.

7.5.2 Description

The transformation that is involved with the *property set pattern* is more complex than the one for the previous patterns. This time, we apply the stereotype ≪atpVariation≫ to a class, not a relation or an attribute. Unlike the other patterns, a class may inherit its stereotype from another class (or may aggregate other classes), which requires special consideration.

Hence, we describe the *property set pattern* in two steps. First, we show how the transformation works for a class which is explicitly tagged in the meta-model with a

[5] As we will see in the next section, the actual transformation is a bit more involved, but the general idea is the same as outlined in this example.

[6] For completeness: attributes that were optional in the pure meta model may still be omitted.

stereotype ≪atpVariation≫ and does not derive from a class that has the same stereotype. In a second step, we extend this description to show what happens if a class is not explicitly tagged with ≪atpVariation≫, but is *derived* from such a class and therefore implicitly has the stereotype.

7.5.2.1 ≪atpVariation≫ Applied Directly to a Property Set Class

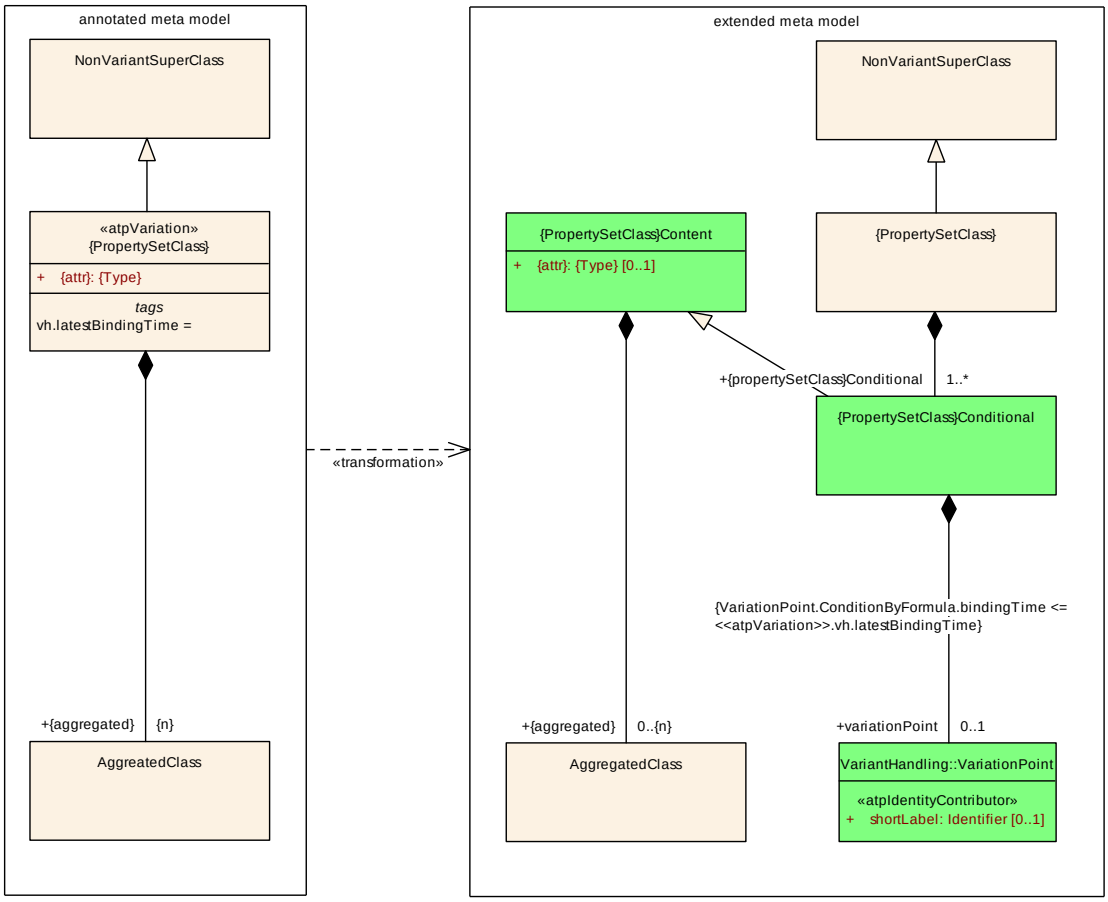


Figure 7.6: *Property Set Pattern* without inheritance in {PropertySetClass}

The annotated meta-model for this transformation is centered around {PropertySetClass}:

1. {PropertySetClass} is based on NonVariantSuperClass, which will not be changed by this pattern.
2. Class {PropertySetClass} has the stereotype ≪atpVariation≫ and contains an attribute named {attr}.
3. {PropertySetClass} aggregates AggregatedClass. AggregatedClass itself is not changed when the pattern is applied, although it will be aggregated by a different class afterwards.

[TPS_GST_00217] Transformation defined by Property Set Pattern ⌈The transformation works as follows:

1. All attributes and the stereotype ≪atpVariation≫, are removed from {PropertySetClass}.
2. The transformation generates a new class {PropertySetClass}Content which contains the original attribute(s) of {PropertySetClass}, but now with a lower multiplicity of 0.

   The change in the multiplicity stems from the fact that this class is designed to hold an arbitrary subset of the attributes that were originally in {PropertySetClass}.

   {PropertySetClass}Content can be seen as a kind of a clone of {PropertySetClass} that lacks the inheritance of NonVariantSuperClass and has a different multiplicity for its attribute.
3. A new class named {PropertySetClass}Conditional is generated. {PropertySetClass}Conditional derives from {PropertySetClass}Content, and aggregates a VariationPoint.

   Because it inherits from {PropertySetClass}Content, each instance of {PropertySetClass}Conditional holds a subset of the attributes that were originally in {PropertySetClass} in the *annotated meta model*.
4. {PropertySetClass} aggregates an arbitrary number of {PropertySetClass}Conditional elements. The idea is that a particular instance of {PropertySetClass}Conditional in the *extended meta-model* contains a subset of the original attributes of {PropertySetClass} in the *annotated meta-model*. But *after the binding*, the disjoint sum of all attributes given in {PropertySetClass}Conditional instances shall yield the full set of attributes that were in {PropertySetClass} in the *annotated meta model*.
5. The VariationPoint finally decides whether a {PropertySetClass}Conditional is included in a particular variant. That is, the aggregation from {PropertySetClass} to {PropertySetClass}Conditional is itself subject to variation.
6. AggregatedClass is now aggregated by {PropertySetClass}Content, with a lower multiplicity of 0. The reason for the change in the multiplicity is that aggregated classes are handled in the same way as attributes.
7. (Not shown in the above diagram) References to {PropertySetClass} in the annotated meta-model still point to {PropertySetClass} in the extended meta-model.

⌋()

7.5.2.2 ≪atpVariation≫ Applied to Superclass of a Property Set Class

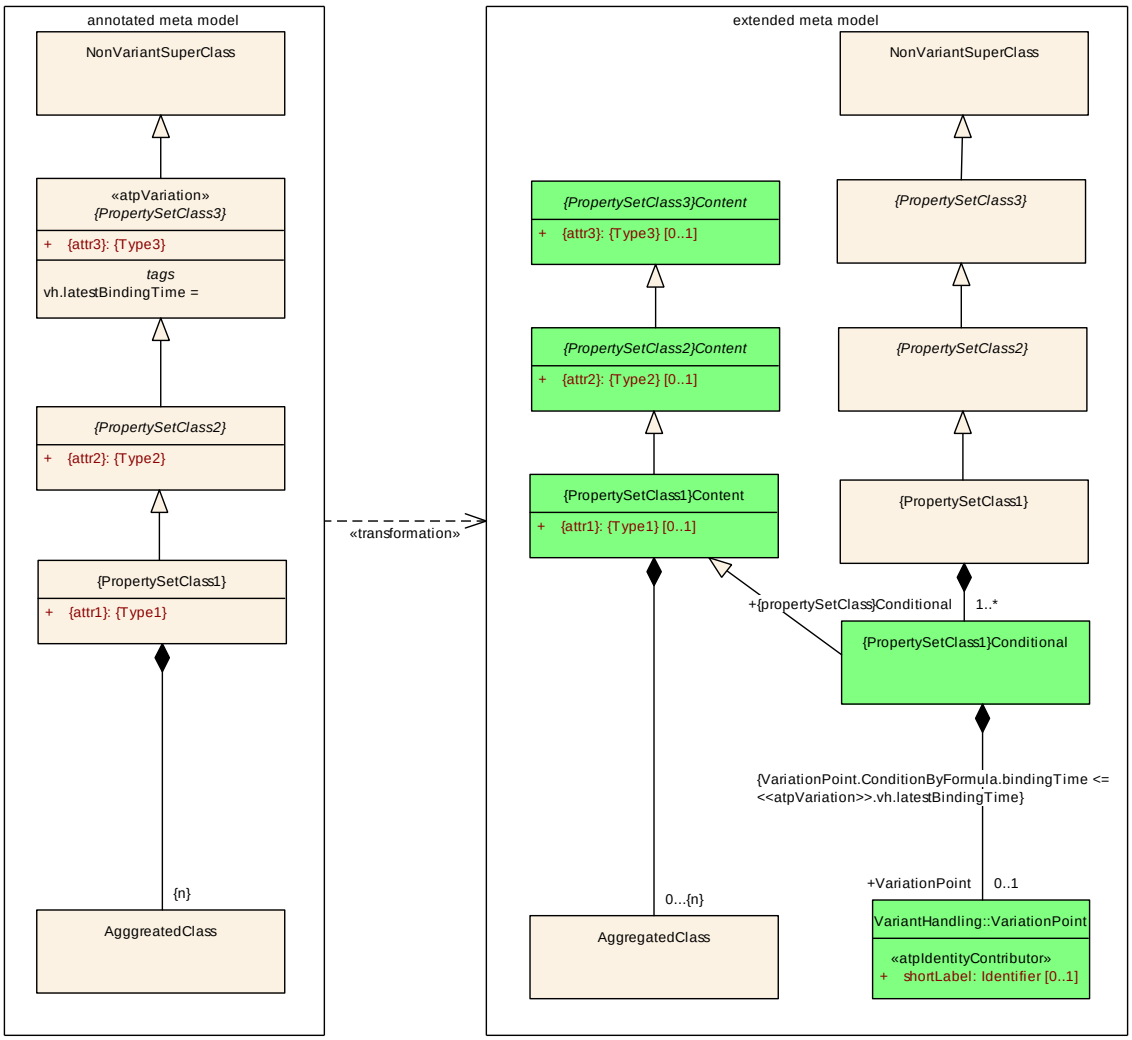


Figure 7.7: *Property Set Pattern* with inheritance in {PropertySetClass}

[TPS_GST_00218] Property Set pattern and Inheritance ⌈When the stereotype ≪atpVariation≫ is applied to a superclass of the property set class, then the situation becomes slightly more complex:

1. NonVariantSuperClass and AggregatedClass are the same as before.
2. {PropertySetClass1} plays the role that was occupied by {PropertySetClass} in the previous section. However, {PropertySetClass1} now derives from a class {PropertySetClass3}, which is also tagged with the stereotype ≪atpVariation≫. {PropertySetClass2} sits between those two classes.

⌋()

The actual transformation does not differ much between Figures 7.7 and 7.6. The only new aspect in Figure 7.7 is that for each property set class *up to* {PropertySetClass3}, we create a hierarchy of {PropertySetClass}Content classes. These

classes retain the attributes of the original classes, as well as their aggregations and association relations.

7.5.2.3 Constraints

In general several conditionals can exist side by side even though none of them has a variation point. In case variation is given the property set pattern needs a rule [TPS_GST_00433] and a constraint [constr_2634] that makes sure that for any variant, the set of all attribute values defined with this method is complete *and* there are no double definitions.

[TPS_GST_00433] Individual attributes in conditionals ⌈In the context of one variant and after all variation points are resolved, individual attributes shall only exist in one of the conditionals. If an individual attribute exists in multiple conditionals then the rules defined in [TPS_GST_00417] apply for merging the conditionals into one conditional.⌋ *()*

[constr_2634] Conditionals with ordered collections ⌈Ordered collections shall not be split over different conditionals.⌋*()*

The example listing 7.1 illustrates a set of two non-variant attributes and one variant attribute, these can all exist in the same property set. This example represents a valid case.

Listing 7.1: Example for two non-variant attributes and one variant attribute, valid case

```
<APPLICATION-PRIMITIVE-DATA-TYPE>
  <SHORT-NAME>MyDT</SHORT-NAME>
  <SW-DATA-DEF-PROPS>
    <SW-DATA-DEF-PROPS-VARIANTS>
      <SW-DATA-DEF-PROPS-CONDITIONAL>
        <SW-TEXT-PROPS>
          <ARRAY-SIZE-SEMANTICS>FIXED-SIZE</ARRAY-SIZE-SEMANTICS>
        </SW-TEXT-PROPS>
      </SW-DATA-DEF-PROPS-CONDITIONAL>
      <SW-DATA-DEF-PROPS-CONDITIONAL>
        <INVALID-VALUE>
          <APPLICATION-VALUE-SPECIFICATION>
            <SW-VALUE-CONT>
              <SW-VALUES-PHYS>
                <V>42</V>
              </SW-VALUES-PHYS>
            </SW-VALUE-CONT>
          </APPLICATION-VALUE-SPECIFICATION>
        </INVALID-VALUE>
      </SW-DATA-DEF-PROPS-CONDITIONAL>
      <SW-DATA-DEF-PROPS-CONDITIONAL>
        <BASE-TYPE-REF DEST="SW-BASE-TYPE">/A/B/C</BASE-TYPE-REF>
        <VARIATION-POINT>
          <SW-SYSCOND BINDING-TIME="CODE-GENERATION-TIME">
            <SYSC-REF DEST="SW-SYSTEMCONST">/J/K/L</SYSC-REF> == 1
          </SW-SYSCOND>
```

```
        </VARIATION-POINT>
      </SW-DATA-DEF-PROPS-CONDITIONAL>
    </SW-DATA-DEF-PROPS-VARIANTS>
  </SW-DATA-DEF-PROPS>
</APPLICATION-PRIMITIVE-DATA-TYPE>
```

The example listing 7.2 illustrates a set of three non-variant attributes (repetition of one attribute; equivalent values) and one variant attribute, these can all exist in the same property set. The repeated attribute can be merged due to equivalent values. This example represents a valid case.

Listing 7.2: Example for three non-variant attributes (equivalent values) and one variant attribute, valid case

```
<APPLICATION-PRIMITIVE-DATA-TYPE>
  <SHORT-NAME>MyDT</SHORT-NAME>
  <SW-DATA-DEF-PROPS>
    <SW-DATA-DEF-PROPS-VARIANTS>
      <SW-DATA-DEF-PROPS-CONDITIONAL>
        <SW-TEXT-PROPS>
          <ARRAY-SIZE-SEMANTICS>FIXED-SIZE</ARRAY-SIZE-SEMANTICS>
        </SW-TEXT-PROPS>
      </SW-DATA-DEF-PROPS-CONDITIONAL>
      <SW-DATA-DEF-PROPS-CONDITIONAL>
        <SW-TEXT-PROPS>
          <ARRAY-SIZE-SEMANTICS>FIXED-SIZE</ARRAY-SIZE-SEMANTICS>
        </SW-TEXT-PROPS>
        <INVALID-VALUE>
          <APPLICATION-VALUE-SPECIFICATION>
            <SW-VALUE-CONT>
              <SW-VALUES-PHYS>
                <V>42</V>
              </SW-VALUES-PHYS>
            </SW-VALUE-CONT>
          </APPLICATION-VALUE-SPECIFICATION>
        </INVALID-VALUE>
      </SW-DATA-DEF-PROPS-CONDITIONAL>
      <SW-DATA-DEF-PROPS-CONDITIONAL>
        <BASE-TYPE-REF DEST="SW-BASE-TYPE">/A/B/C</BASE-TYPE-REF>
        <VARIATION-POINT>
          <SW-SYSCOND
          BINDING-TIME="CODE-GENERATION-TIME">
          <SYSC-REF DEST="SW-SYSTEMCONST">/J/K/L</SYSC-REF>
          == 1
          </SW-SYSCOND>
        </VARIATION-POINT>
      </SW-DATA-DEF-PROPS-CONDITIONAL>
    </SW-DATA-DEF-PROPS-VARIANTS>
  </SW-DATA-DEF-PROPS>
</APPLICATION-PRIMITIVE-DATA-TYPE>
```

The example listing 7.3 illustrates a set of three non-variant attributes (repetition of one attribute; non-equivalent values) and one variant attribute, these can not exist in the same property set. The repeated attribute can not be merged due to non-equivalent values. This example represents a invalid case.

Listing 7.3: Example for three non-variant attributes (non-equivalent values) and one variant attribute, invalid case

```
<APPLICATION-PRIMITIVE-DATA-TYPE>
  <SHORT-NAME>MyDT</SHORT-NAME>
  <SW-DATA-DEF-PROPS>
    <SW-DATA-DEF-PROPS-VARIANTS>
      <SW-DATA-DEF-PROPS-CONDITIONAL>
        <SW-TEXT-PROPS>
          <ARRAY-SIZE-SEMANTICS>FIXED-SIZE</ARRAY-SIZE-SEMANTICS>
        </SW-TEXT-PROPS>
      </SW-DATA-DEF-PROPS-CONDITIONAL>
      <SW-DATA-DEF-PROPS-CONDITIONAL>
        <SW-TEXT-PROPS>
          <ARRAY-SIZE-SEMANTICS>VARIABLE-SIZE</ARRAY-SIZE-SEMANTICS>
        </SW-TEXT-PROPS>
        <INVALID-VALUE>
          <APPLICATION-VALUE-SPECIFICATION>
            <SW-VALUE-CONT>
              <SW-VALUES-PHYS>
                <V>42</V>
              </SW-VALUES-PHYS>
            </SW-VALUE-CONT>
          </APPLICATION-VALUE-SPECIFICATION>
        </INVALID-VALUE>
      </SW-DATA-DEF-PROPS-CONDITIONAL>
      <SW-DATA-DEF-PROPS-CONDITIONAL>
        <BASE-TYPE-REF DEST="SW-BASE-TYPE">/A/B/C</BASE-TYPE-REF>
        <VARIATION-POINT>
          <SW-SYSCOND
          BINDING-TIME="CODE-GENERATION-TIME">
          <SYSC-REF DEST="SW-SYSTEMCONST">/J/K/L</SYSC-REF>
          == 1
          </SW-SYSCOND>
        </VARIATION-POINT>
      </SW-DATA-DEF-PROPS-CONDITIONAL>
    </SW-DATA-DEF-PROPS-VARIANTS>
  </SW-DATA-DEF-PROPS>
</APPLICATION-PRIMITIVE-DATA-TYPE>
```

The same merging rules as used for the Splitkey (see 8.3) shall be applied.

7.5.3 Binding Time

[constr_2580] Binding Time in Property Set Pattern ⌈The meta-class `VariationPoint` has an attribute `bindingTime` which defines the *latest* binding time for this variation point. This binding time is further constrained by the UML tag `vh.latestBindingTime` that is attached to the meta-class which is marked as ≪`atpVariation`≫ (see [TPS_GST_00190], [TPS_GST_00220], [TPS_GST_00221]):

`VariationPoint.bindingTime` ≤ *meta class*.`vh.latestBindingTime`⌋*()*

[TPS_GST_00219] Binding Time for Property Set Pattern ⌈The latest binding time for the *property set pattern* is *PostBuild*.⌋*()*

7.5.4 Multiplicity of Attributes and aggregated elements

[TPS_GST_00222] Multiplicity in Property Set Pattern ⌈In the *property set pattern*, attributes (and aggregated elements) are moved from `{PropertySetClass}` in the *annotated meta-model* to `{PropertySetClass}Conditional` in the *extended meta-model*.

With this move, the lower multiplicity always changes to 0.⌋*()*

7.5.5 XML Representation

An example for the XML code that is produced by the *property set pattern* can be found in Figure 7.10.

Despite the perceived complexity of the pattern in Figures 7.6 and 7.7, the impact of the *property set pattern* on the XML code is rather limited. As Example 7.10 shows, the property set pattern adds a `-VARIANTS` wrapper around the attributes, and a `-CONDITIONAL` element for each (sub)set of attribute values. `-CONDITIONAL` also contains a `VARIATION-POINT` element.

So, the main impact on the XML code is the duplication of attribute values, but the overhead introduced by variant handling should only add a few elements.

7.5.6 Comparison with Other Patterns

Both this and the *attribute value pattern* (Section 7.4) are aimed at attributes, but with several differences:

- The *prototype set pattern* provides a way to *group* attributes that belong together.
- The *property set pattern* is more flexible in that variability is not restricted to those attributes for which the M2 meta-model "allows" variability. There is however a catch: because of the higher flexibility, it is not a priori clear which attributes will be invariant, and which not.
- The *attribute value pattern* may use an expression to define the value of an attribute, while the *property set pattern* can only use a fixed value (more precisely, a fixed value *per variant*). However, the *property set pattern* may be combined with the *attribute value pattern* to achieve this effect.
- The *attribute value pattern* is available for a limited number of data types only, namely `Integer`, `Float`, `Boolean`, `Numerical`, `PositiveInteger` and `UnlimitedInteger`. The *property set pattern* has no such limitation.

- If an attribute is optional, then the *property set pattern* may also decide whether an attribute exists or not. This is different from the *attribute value pattern*, which can only change the value of an attribute.

Furthermore, the *Property Set Pattern* differs from the *Aggregation Pattern* and the *Association Pattern* in that the former pattern works on a number of attributes, aggregations and associations at once, while the latter patterns determine the existence of a single aggregation or aggregation only.

7.5.7 Combining the *attribute value pattern* and the *property set pattern*s

In the previous section, we said that the *property set pattern* cannot specify an expression to define the value of an attribute. While this is true, there is a way to avoid this restriction, namely by combining the *property set pattern* with the *attribute value pattern*.

In this case, the *property set pattern* would allow to partition the complete set of attributes into several disjoint subsets. Each of these individual attributes may have the stereotype ≪atpVariation≫, which means that the *attribute value pattern* is applied, and the value of the attribute may be determined by an expression.

Furthermore, since an attribute may occur in multiple sets (on M1 level), there may be multiple expressions for determining the value of a particular attribute, each tailored for a particular variant.

7.6 VariationPoint

The structure of a VariationPoint is illustrated in Figure 7.8.

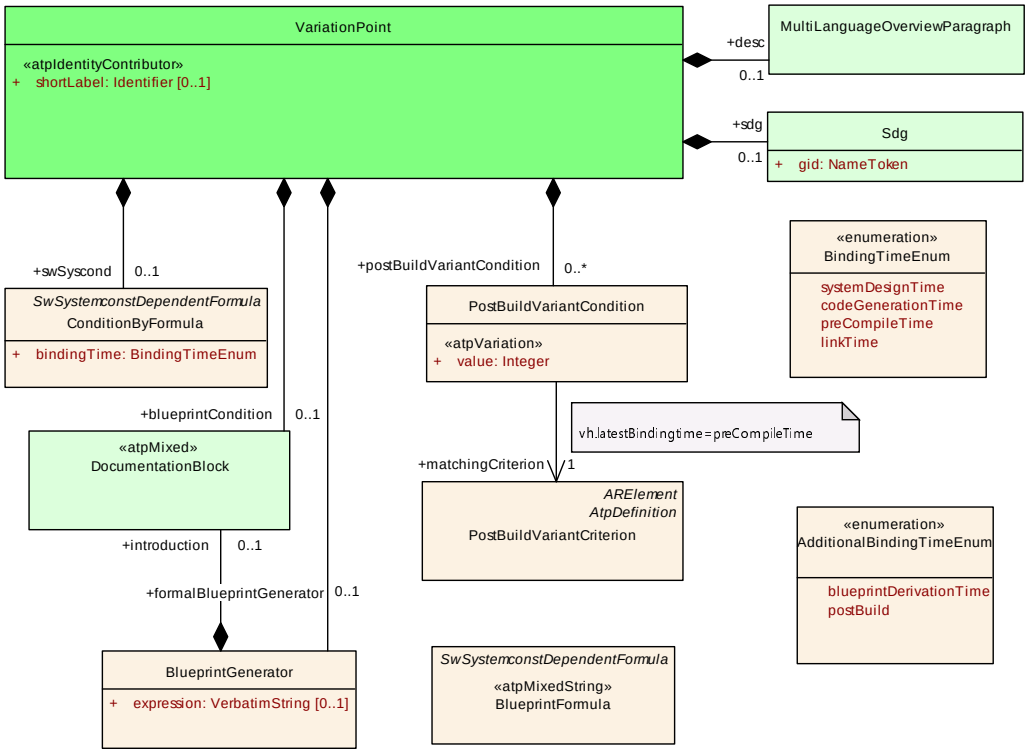


Figure 7.8: Variation Point

Class	**VariationPoint**			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling			
Note	This meta-class represents the ability to express a "structural variation point". The container of the variation point is part of the selected variant if swSyscond evaluates to true and each postBuildVariant Criterion is fulfilled.			
Base	*ARObject*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
blueprint Condition	DocumentationBlock	0..1	aggr	This represents a description that documents how the variation point shall be resolved when deriving objects from the blueprint. Note that variationPoints are not allowed within a blueprintCondition. **Tags:**xml.sequenceOffset=28
desc	MultiLanguageOverview Paragraph	0..1	aggr	This allows to describe shortly the purpose of the variation point. **Tags:**xml.sequenceOffset=20
formalBlueprint Generator	BlueprintGenerator	0..1	aggr	This represents a description that documents how the variation point shall be resolved when deriving objects from the blueprint by using ARMQL. Note that variationPoints are not allowed within a formal BlueprintGenerator. **Tags:** atp.Status=draft xml.sequenceOffset=30

△

Class	VariationPoint			
postBuildVariant Condition	PostBuildVariant Condition	*	aggr	This is the set of post build variant conditions which all shall be fulfilled in order to (postbuild) bind the variation point. **Tags:**xml.sequenceOffset=40
sdg	Sdg	0..1	aggr	An optional special data group is attached to every variation point. These data can be used by external software systems to attach application specific data. For example, a variant management system might add an identifier, an URL or a specific classifier. **Tags:**xml.sequenceOffset=50
shortLabel	Identifier	0..1	attr	This provides a name to the particular variation point to support the RTE generator. It is necessary for supporting splitable aggregations and if binding time is later than codeGenerationTime, as well as some RTE conditions. It needs to be unique with in the enclosing Identifiables with the same ShortName. **Stereotypes:** atpIdentityContributor **Tags:**xml.sequenceOffset=10
swSyscond	ConditionByFormula	0..1	aggr	This condition acts as Binding Function for the Variation Point. Note that the multiplicity is 0..1 in order to support pure postBuild variants. **Tags:**xml.sequenceOffset=30

Table 7.4: VariationPoint

7.6.1 The structure of class `VariationPoint`

The class `VariationPoint` holds information about a variation point in the *aggregation pattern* (Section 7.2), the *association pattern* (Section 7.3), and the *property set pattern* (Section 7.5)[7].

A `VariationPoint` aggregates a `ConditionByFormula`, a `PostBuildVariantCondition` and a `DocumentationBlock` in the role `blueprintCondition`. These three "branches" are independent of each other. As the multiplicities in Figure 7.8 shows, they are also all optional:

- **[TPS_GST_00245] PreBuild variation point** ⌈If a variation point aggregates `ConditionByFormula`, then this variation point is a *PreBuild* variation point.⌋*()* See Section 7.6.5 for details.
- **[TPS_GST_00246] PostBuild Variation Point** ⌈If a variation point aggregates a `PostBuildVariantCondition`, then this variation point is a *PostBuild* variation point.⌋*()* See Section 7.6.6 for details.
- **[TPS_GST_00247] BlueprintDerivation Variation Point** ⌈If a variation point aggregates a `blueprintCondition` or `formalBlueprintGenerator` then this variation shall be resolved when deriving objects.⌋*()* Refer to [2] for details.

[7] The *attribute value pattern* (Section 7.4) is simpler and does not make use of the class `VariationPoint`. Furthermore, its latest binding time is `CompileTime`, so many of the issues discussed in this section do not apply to this pattern.

- **[TPS_GST_00248] Combined PreBuild and PostBuild Variation Point** ⌈A variation point may also aggregate *both* `ConditionByFormula` and `PostBuildVariantCondition`. In this case, it is both a *PreBuild* and a *PostBuild* variation point.⌋*()* See Section 7.6.5 for details.
- **[TPS_GST_00249] Variation Point without Conditions** ⌈Technically, a variation point may also aggregate *none* of the above classes. In this case, there is no variation at all, and the element to which the variation point is attached to always exists.⌋*()*

  This is equivalent to a *PreBuild*-only variation point where `ConditionByFormula` has binding time `systemDesignTime`, and who's formula always evaluates to *true*.

[TPS_GST_00250] Multiplicity of `VariationPoint` ⌈In all patterns, the `VariationPoint` element has a multiplicity of $[0 \ldots 1]$, that is, it is optional. If the variation point is omitted, then there is no variation and the respective element always exists.⌋*()*

[constr_2557] No `VariationPoint`s where `vh.latestBindingTime` set to `BlueprintDerivationTime` in system configurations ⌈Blueprints are **not** part of a system configuration. In consequence of this, in a system configuration there shall be no `VariationPoint` where `vh.latestBindingTime` is restricted to `BlueprintDerivationTime` by the meta-model.⌋*()*

[constr_2558] If `vh.latestBindingTime` is `BlueprintDerivationTime` then there shall only be `blueprintCondition` or `formalBlueprintGenerator` respectively `blueprintValue` ⌈`VariationPoint`s with `vh.latestBindingTime` restricted to `BlueprintDerivation` shall not have `swSyscond` nor `postBuildVariantCondition`.⌋*()*

[constr_2559] No nested `VariationPoint` ⌈As `blueprintCondition` is a `DocumentationBlock` it could again contain `VariationPoint`s and therefore would allow nesting of `VariationPoint`s. This is not intended and shall not be used.⌋*()*

7.6.2 `shortLabel` in `VariationPoint`

`VariationPoint` has a single optional attribute `shortLabel` that implements a name for the variation point.

[constr_2514] `shortLabel` in `VariationPoint` shall be unique ⌈The combination of `shortName` and `shortLabel` shall be unique within the next enclosing `Identifiable` `{WholeClass}`. In case the `shortName` does not exist on the `{PartClass}` the `shortLabel` is unnecessary. In case the `shortName` of the `{PartClass}` is unique in the context of the `{WholeClass}` the `shortLabel` is unnecessary.⌋*()*

For example, in the aggregation pattern (Section 7.2), this enclosing `Identifiable` as usually `{WholeClass}`.

[TPS_GST_00251] Variant-Rich Model Violates [constr_2508] ⌈According to [TPS_GST_00097], AUTOSAR would use the attribute `shortName` of the next enclosing `Identifiable` as a unique name. This does not work with variation points. The reason for this is rooted in the difference between the *variant-rich M1 model* and the *bound M1 model*.

The *variant-rich M1 model* may define several alternative variants for one aggregation. As the term "alternatives" implies, only one of them is left in the *bound M1 meta-model* but all have the same `shortName`.

Therefore a (not-yet-bound) *variant-rich M1 model violates* AUTOSAR's consistency conditions ([constr_2508]) by having multiple elements with the same `shortName`. This is only feasible because we require that the *bound M1 model* (and the associated code) will eventually adhere to those consistency rules ([constr_2503]).

[constr_2508] is substituted by [constr_2512] for variant-rich models.⌋*()*

There are several situations where it is necessary to individually address variations in the *variant-rich M1 model* that have the same `shortName`:

- **[TPS_GST_00252] Split/Merge of Variant-Rich Model** ⌈If an aggregation has the stereotype ≪`atpSplitable`≫. The use case for this is that particular variants are held in a separate artifact. In order to merge such separate artefacts, it is necessary not only to consider `shortName` but also the `shortLabel` of the particular variants of an `Identifiable` ([constr_2512]).⌋*()* For more details about splitable elements refer to section 8.
- **[TPS_GST_00253] Distinguish `codeGenerationTime` Variation Points in RTE** ⌈If binding time is `codeGenerationTime` or later, the RTE needs to distinguish between the individual variants if `preCompileTime` variability is implemented.⌋*()*
- **[TPS_GST_00254] Referring to Variation Points from Outside** ⌈It is often necessary to refer to individual variation point from the *outside*. For example, a configuration management system might need to identify individual variation points for traceability.⌋*()*

Also, since `shortLabel` is an optional element, it has no impact on the size or complexity of the XML if it is not present.

[constr_2512] `shortName` uniqueness constraint for variants ⌈`shortName` + `shortLabel` of a variant element shall be unique within the name space established by the surrounding `Identifiable`.⌋*()*

The `shortLabel` in the `VariationPoint` is technically only required when `VariationPoint`s are used to switch between `Identifiable`s with identical `shortName`s, see listing F.1 and F.3. If additionally those `Identifiable`s are described in partial models the `shortLabel`s in the partial models indicates which elements belong together, see listing F.1 and F.2.

If the shortLabel in the Identifiable is used to vary the existence of the Identifiable without an equally named alternative the shortLabel in the VariationPoint is not required but may exists. If the shortLabel is defined and the Identifiable is described in partial models it is required to repeat the shortLabel consistently, see listing F.2. If the shortLabel is not defined it shall not occur in any of the partial models.

7.6.3 sdg in VariationPoint

The class VariationPoint aggregates an optional sdg object (see Section 4.5.1) which can be used by external software systems to attach application specific data to a variation point. For example, a variant management system might add an identifier, an URL or a specific classifier to a variation point.

Since such data is highly application and vendor specific, it cannot be standardized, and a special data group is necessary instead.

Also, since sdg is an optional element, it has not impact on the size or complexity of the XML representation if it is not present.

7.6.4 (Latest) Binding Time

In Section 7.1, we have seen that each variation point has a binding time. Binding times (see [18]) can be further categorized as *PreBuild* and *PostBuild* binding times:

- **[TPS_GST_00255] Definition of *PreBuild* Variation Point** ⌈This category contains the following binding times: systemDesignTime, codeGenerationTime, preCompileTime, and linkTime.

  A concrete variation point (i.e., on M1 level) is subject to *PreBuild* variation if it contains a ConditionByFormula element. Its binding time is specified in the bindingTime attribute of the ConditionByFormula element.⌋() For more details see Section 7.6.5.

- **[TPS_GST_00360] Definition of *PreBuild* Variation Point with Blueprint conditions** ⌈This category contains only a single binding time, namely blueprintDerivationTime. A concrete variation point (i.e., on M1 level) is subject to *PreBuild* variation with blueprintDerivationTime if it contains a blueprintCondition or formalBlueprintGenerator attribute.⌋() For more details see Section 7.6.11.2.

- **[TPS_GST_00256] Definition of *PostBuild* Variation Point** ⌈This category contains only a single binding time, namely PostBuild.

  A concrete variation point (i.e., on M1 level) is *PostBuild* if it contains a PostBuildVariantCondition element. Since there is only one binding time for

PostBuild, no particular attribute for specifying the binding time is necessary.⌋() For more details see Section 7.6.6.

The binding time is further constrained by the tag `vh.latestBindingTime` that was introduced in the patterns earlier in sections 7.2 to 7.5:

[TPS_GST_00257] BindingTime constrained by vh.latestBindingTime ⌈

- If `vh.latestBindingTime` = `PostBuild`, then a variation point on M1 level may have any binding time. It may be a *PreBuild* or a *PostBuild* variation point (or both, and may aggregate `ConditionByFormula` or `PostBuildVariantCondition`.
- If `vh.latestBindingTime` < `PostBuild`, then a variation point on M1 level can only be a *PreBuild*, but *not* a *PostBuild* variation point. Obviously, it may only aggregate a `ConditionByFormula` in this case.
- If `vh.latestBindingTime` = `BlueprintDerivationTime`, then a variation point on M1 level may only aggregate a `blueprintCondition` or `formalBlueprintGenerator`. See also [constr_2557] and [constr_2558].

⌋()

It is obvious that the binding time of a *PreBuild* variation point (that is, the value of the attribute `bindingTime` of `ConditionByFormula`), shall never exceed `vh.latestBindingTime`.

7.6.5 *PreBuild* Variation Points

Class	<<atpMixedString>> **ConditionByFormula**			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling			
Note	This class represents a condition which is computed based on system constants according to the specified expression. The expected result is considered as boolean value. The result of the expression is interpreted as a condition. • "0" represents "false"; • a value other than zero is considered "true"			
Base	*ARObject*, *FormulaExpression*, *SwSystemconstDependentFormula*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
bindingTime	BindingTimeEnum	1	attr	This attribute specifies the point in time when condition may be evaluated at earliest. At this point in time all referenced system constants shall have a value. **Tags:**xml.attribute=true

Table 7.5: ConditionByFormula

All the information that is necessary to implement a *PreBuild* variation point is provided by the class `ConditionByFormula`:

- `ConditionByFormula` derives from `SwSystemconstDependentFormula`. This class implements the (boolean) formula that determines whether the variation point is “on” or “off”.

  The formula language is defined in Section 4.8. See also Section 7.6.8 for further explanation how formulas are used in the variant handling concept.

- `ConditionByFormula` has a single attribute, `bindingTime`, which defines the latest binding time for this variation point. The binding times are described in more detail in [18].

[TPS_GST_00258] Binding `VariationPoint`s early ⌈A concrete software system *may* bind a variation point at an earlier binding time if this is technically feasible, and within contractual limits[8]. We define the additional restriction

```
Variation Point Binding Times later than System
Design Time are part of the contract.
```

According to this restriction, the RTE Generator is not allowed to resolve the variability in the application header file even if the variability is already chosen in the input of the RTE.

If the binding time is *systemDesignTime*, then the variability is not considered part of the contract phase, and it shall be bound properly during *systemDesignTime* before the contract.⌋*()*

7.6.6 *PostBuild* Variation Points

Class	**PostBuildVariantCondition**			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling			
Note	This class specifies the value which shall be assigned to a particular variant criterion in order to bind the variation point. If multiple criterion/value pairs are specified, they shall all match to bind the variation point. In other words binding can be represented by (criterion1 == value1) && (condition2 == value2) ...			
Base	*ARObject*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
matching Criterion	PostBuildVariant Criterion	1	ref	This is the criterion which needs to match the value in order to make the PostbuildVariantCondition to be true.
value	Integer	1	attr	This is the particular value of the post-build variant criterion. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime

Table 7.6: PostBuildVariantCondition

[8]For a definition of contract phases, see Chapter 3.1 in the RTE specification [19].

Class	**PostBuildVariantCriterion**			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling			
Note	This class specifies one particular PostBuildVariantSelector. **Tags:**atp.recommendedPackage=PostBuildVariantCriterions			
Base	*ARElement*, *ARObject*, *AtpDefinition*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
compuMethod	CompuMethod	1	ref	The compuMethod specifies the possible values for the variant criterion serving as an enumerator.

Table 7.7: PostBuildVariantCriterion

A *PostBuild* variation point contains multiple `PostBuildVariantCondition`s in the role `postBuildVariantCondition`, which in turn has a reference to a `PostBuildVariantCriterion`. Unlike a *PreBuild* variation point, which is governed by a formula defined in `ConditionByFormula`, a *PostBuild* variation point is governed by the combination of `PostBuildVariantCondition`s.

[constr_2517] `postBuildVariantCondition` only for PostBuild ⌈Aggregation of `PostBuildVariantCondition` in `VariationPoint` is only allowed if the annotated model states `vh.latestBindingTime` to `PostBuild`.⌋*()*

[TPS_GST_00260] PreBuild configuration of PostBuild criteria ⌈The attribute `value` of `PostBuildVariantCondition` is subject to *PreBuild* variation. It uses the *attribute value pattern*, hence its latest binding time is `preCompileTime`. That is, the value which will be compared with the contents of `PostBuildVariantCriterion` is computed[9] at `preCompileTime` (at most).

The actual comparison with the contents of `PostBuildVariantCriterion`, however, is done based on the result at start-up of the ECU.⌋*()*

[TPS_GST_00259] Evaluating `PostBuildVariantCondition` ⌈A `VariationPoint` element may aggregate any number of `PostBuildVariantCondition` in the role `postBuildVariantCondition`. A logical *and* is implied between all these elements: The *PostBuild* variation point is “enabled” if for all `postBuildVariantCondition`s the `value` defined in `PostBuildVariantCondition` matches the value provided by RTE for `PostBuildVariantCriterion` (see [SWS_Rte_06612])⌋*()*

[9] It may be an expression rather than a constant value or a single system constant.

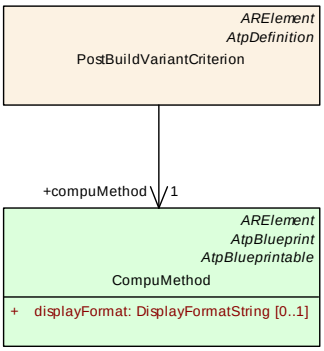


Figure 7.9: PostBuildVariantCriterion

[TPS_GST_00261] Possible Values for PostBuildVariantCriterion ⌈A PostBuildVariantCriterion also refers to a compuMethod which specifies the possible values for the criterion and the conversion between the physical and the internal representation of data (see the *Software Component Template* [20] [TPS_SWCT_01243], [TPS_SWCT_01278] for details).⌋*()*

The RTE is responsible for managing the PostBuildVariantCriterion values ([SWS_Rte_06612]).

7.6.7 System Constants

For *PreBuild* variation points, the binding function depends on SwSystemconst. Such a system constant is basically a name/value pair. shortName, dataConstr and compuMethod for a system constant are defined in SwSystemconst. Similar to data types an unit can be explicitly expressed by the unit reference.

dataConstrs are defined to describe limits to the applicable SwSystemconstValues. The compuMethod supports either the conversion between internal values and textual literals or the conversion between internal values and the physical meaning. Regardless of the existence of a compuMethod the evaluation of formula containing SwSystemconst is realized by internal values. Therefore also the SwSystemconstValues hold the internal values. Nevertheless the compuMethod can be used by tools to show the physical value.

[TPS_GST_00262] Representation of SwSystemconst ⌈compuMethod in the SwDataDefProps of SwSystemconst is intended only to support appropriate representation of the values in tools and documentation. The values shall always be set as internal representation.⌋*()*

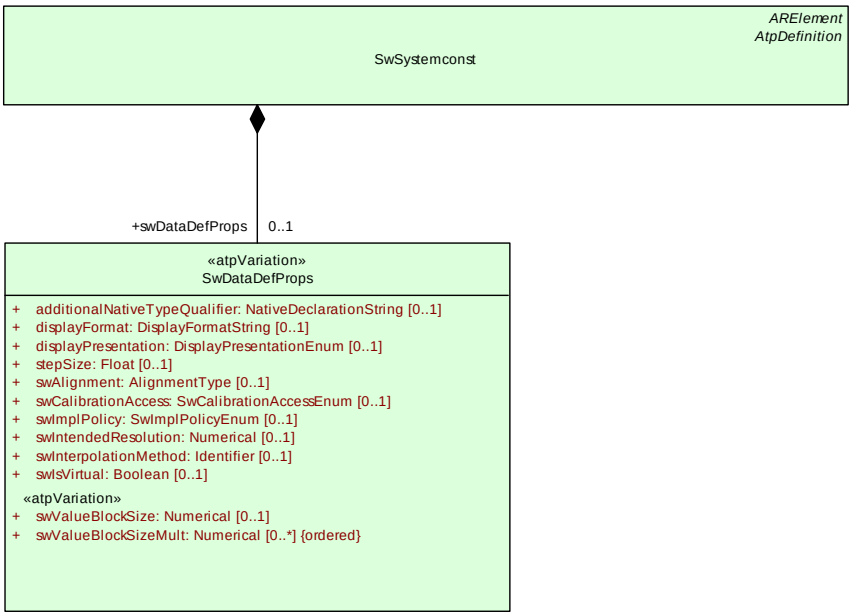


Figure 7.10: Defintion of a `SwSystemconst`

Class	**SwSystemconst**			
Package	M2::MSR::DataDictionary::SystemConstant			
Note	This element defines a system constant which serves an input to select a particular variation point. In particular a system constant serves as an operand of the binding function (swSyscond) in a Variation point. Note that the binding process can only happen if a value was assigned to to the referenced system constants. **Tags:**atp.recommendedPackage=SwSystemconsts			
Base	*ARElement*, *ARObject*, *AtpDefinition*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
swDataDef Props	SwDataDefProps	0..1	aggr	This denotes the data definition properties of the system constant. This supports to express the limits and optionally a conversion within the internal to physical values by a compu method. **Tags:**xml.sequenceOffset=40

Table 7.8: SwSystemconst

[TPS_GST_00263] Assigning values to `SwSystemconst` ⌈In order to choose variants, values need to be assigned to `SwSystemconst`. Note that the values shall always be specified as "internal values". This is done in `SwSystemconstValue`.⌋*()*

[constr_2594] Cyclic value assignments to `SwSystemconst` is not allowed ⌈It is explicitly forbidden to assign values to `SwSystemconst` which in turn depend directly or indirectly on this value assignment.⌋*()*

Cyclic value assignment to `SwSystemconst` can not be resolved due to the cyclic dependency of the values.

For more details refer to Section 7.8.

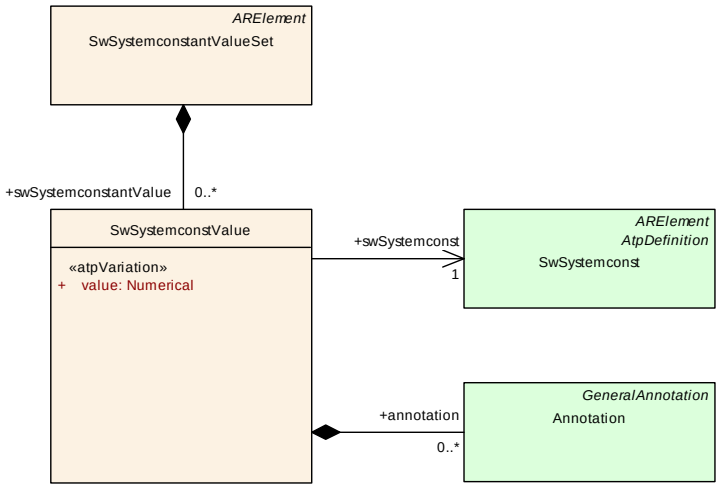


Figure 7.11: Assigning a value to a `SwSystemconst`

Class	**SwSystemconstValue**			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling			
Note	This meta-class assigns a particular value to a system constant.			
Base	*ARObject*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
annotation	Annotation	*	aggr	This provides the ability to add information why the value is set like it is. **Tags:**xml.sequenceOffset=30
swSystemconst	SwSystemconst	1	ref	This is the system constant to which the value applies. **Tags:**xml.sequenceOffset=10
value	Numerical	1	attr	This is the particular value of a system constant. It is specified as Numerical. Further restrictions may apply by the definition of the system constant. The value attribute defines the internal value of the Sw Systemconst as it is processed in the Formula Language. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=20

Table 7.9: SwSystemconstValue

7.6.8 Application of Formulas in Variation Points

Binding of variation points is performed by evaluating the formula in the variation point. These formula can be one of the subclasses according to Figure 4.10.

[TPS_GST_00264] Purpose of `SwSystemconstDependentFormula` ⌈A `SwSystemconstDependentFormula` element is a formula which uses system constants by the reference `sysc` to `SwSystemconst`) as operands. Note that the multiplicity of 1 in the diagram is a technicality (see [TPS_GST_00032]); a formula may actually use more than one system constants.⌋*()* (See Figure 4.10)

[TPS_GST_00265] System Constants in Formula ⌈SwSystemconstDependentFormula.sysc reflects the internal value of the SwSystemconst.

SwSystemconstDependentFormula.syscString reflects the string representation value of the SwSystemconst. This is in particular the symbol for representation of a CompuScale in C determined according to [TPS_SWCT_01431].⌋*()*

Examples of correct expressions with references are:

```
<APPLICATION-VALUE-SPECIFICATION>
  <SW-VALUE-CONT>
    <SW-VALUES-PHYS>
      <VF>
        <SYSC-REF DEST="SW-SYSTEMCONST">/S/SY_ZYLZA</SYSC-REF>
      </VF>
    </SW-VALUES-PHYS>
  </SW-VALUE-CONT>
</APPLICATION-VALUE-SPECIFICATION>
```

The Formula Language, which is serialized above inside the <VF>-Tag, is defined after the replacement of ARXML references, i.e. the ARXML above will be transformed into the following expression:

```
    reference("SW-SYSTEMCONST:/S/SY_ZYLZA")
```

which is then accepted as correct SwSystemconstDependentFormula.

```
...
      <VF>
        <SYSC-REF DEST="SW-SYSTEMCONST">/S/SY_ZYLZA</SYSC-REF>
      + <SYSC-REF DEST="SW-SYSTEMCONST">/S/SY_TURBO</SYSC-REF>
      </VF>
...
```

is transformed into:

```
    reference("SW-SYSTEMCONST:/S/SY_ZYLZA") + reference("SW-
SYSTEMCONST:/S/SY_TURBO")
```

```
...
      <VF>defined(<SYSC-REF DEST="SW-SYSTEMCONST">/S/SY_ZYLZA</SYSC-
REF>)</VF>
...
```

is transformed into:

```
    defined(reference("SW-SYSTEMCONST:/S/SY_ZYLZA"))
```

```
...
      <VF>1 / <SYSC-REF DEST="SW-SYSTEMCONST">/S/SY_ZYLZA</SYSC-REF></VF>
...
```

is transformed into:

```
1 / reference("SW-SYSTEMCONST:/S/SY_ZYLZA")
```

```
<VARIATION-POINT>
  <SW-SYSCOND>
   defined(<SYSC-REF DEST="SYSTEM-CONSTANT">SY_COUNT<SYSC-REF>)
   &amp;&amp; <SYSC-REF DEST="SYSTEM-CONSTANT">SY_COUNT<SYSC-REF> &lt; 10
  </SW-SYSCOND>
<VARIATION-POINT>
```

Please note, that this example shows that markup characters ("<" and "&") need to be represented as XML entities when the formula is serialized as ARXML. The above formula is transformed into:

```
defined(reference("SW-SYSTEMCONST:SY_COUNT") &&
        reference("SW-SYSTEMCONST:SY_COUNT") < 10
```

The formula language is described in detail in Section 4.8. In this section, we concentrate on the variant handling related classes that are derived from SwSystemconstDependentFormula, namely AttributeValueVariationPoint and ConditionByFormula:

ConditionByFormula is aggregated by VariationPoint and decides whether the element to which the VariationPoint is attached actually exists. This is used in all patterns except the *attribute value pattern*.

The return value of this formula is always interpreted as a boolean: 0 equals *false*, any other value is interpreted as *true*.

AttributeValueVariationPoint is primarily used to provide values in the *attribute value pattern* (Section 7.4). Since the *attribute value pattern* is implicitly used to define the condition of a *PostBuild* variation point, it may also be used in every other pattern.

AttributeValueVariationPoint further splits into eight subclasses, namely NumericalValueVariationPoint, FloatValueVariationPoint, IntegerValueVariationPoint, BooleanValueVariationPoint, TimeValueValueVariationPoint, PositiveIntegerValueVariationPoint and UnlimitedIntegerValueVariationPoint.

These subclasses provide information on the expected return type of the formula (see Chapter 4.8.2.4, and correspond to the AUTOSAR primitive types Numerical, Float, Integer, Boolean, PositiveInteger and UnlimitedInteger.

[constr_2516] Return type of an AttributeValueVariationPoint ⌈When such a formula is evaluated by a software tool, and the return value of the formula is shall be compatible to the type of the attribute in the pure meta-model.⌋*()*

Class	<<atpMixedString>> ***SwSystemconstDependentFormula*** (abstract)			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling			
Note	This class represents an expression depending on system constants.			
Base	*ARObject*, *FormulaExpression*			
Subclasses	*AttributeValueVariationPoint*, BlueprintFormula, ConditionByFormula, *FMFormulaByFeaturesAndSw Systemconsts*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
sysc	SwSystemconst	0..1	ref	This refers to a system constant. The internal (coded) value of the system constant shall be used. **Tags:**xml.sequenceOffset=50
syscString	SwSystemconst	0..1	ref	syscString indicates that the referenced system constant shall be evaluated as a string according to [TPS_SWCT_01431].

Table 7.10: SwSystemconstDependentFormula

see also `ConditionByFormula` see also `AttributeValueVariationPoint`

Class	<<atpMixedString>> ***AbstractNumericalVariationPoint*** (abstract)			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling::AttributeValueVariationPoints			
Note	This is an abstract NumericalValueVariationPoint. It is introduced to support the case that additional attributes are required for particular purposes.			
Base	*ARObject*, *AttributeValueVariationPoint*, *FormulaExpression*, *SwSystemconstDependentFormula*			
Subclasses	LimitValueVariationPoint, NumericalValueVariationPoint			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table 7.11: AbstractNumericalVariationPoint

Class	<<atpMixedString>> **BooleanValueVariationPoint**			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling::AttributeValueVariationPoints			
Note	This class represents an attribute value variation point for Boolean attributes. Note that this class might be used in the extended meta-model on			
Base	*ARObject*, *AttributeValueVariationPoint*, *FormulaExpression*, *SwSystemconstDependentFormula*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table 7.12: BooleanValueVariationPoint

Class	<<atpMixedString>> **FloatValueVariationPoint**			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling::AttributeValueVariationPoints			
Note	This class represents an attribute value variation point for Float attributes. Note that this class might be used in the extended meta-model only			
Base	*ARObject*, *AttributeValueVariationPoint*, *FormulaExpression*, *SwSystemconstDependentFormula*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table 7.13: FloatValueVariationPoint

Class	<<atpMixedString>> **IntegerValueVariationPoint**			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling::AttributeValueVariationPoints			
Note	This class represents an attribute value variation point for Integer attributes. Note that this class might be used in the extended meta-model only.			
Base	*ARObject*, *AttributeValueVariationPoint*, *FormulaExpression*, *SwSystemconstDependentFormula*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table 7.14: IntegerValueVariationPoint

Class	<<atpMixedString>> **LimitValueVariationPoint**			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling::AttributeValueVariationPoints			
Note	This class represents the ability to express a numerical limit. Note that this is in fact a NumericalValuation Point but has the additional attribute intervalType. Note that the xml.name is "LIMIT" for backward compatibility reasons. **Tags:**xml.name=LIMIT			
Base	*ARObject*, *AbstractNumericalVariationPoint*, *AttributeValueVariationPoint*, *FormulaExpression*, *Sw SystemconstDependentFormula*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
intervalType	IntervalTypeEnum	0..1	attr	This specifies the type of the interval. If the attribute is missing the interval shall be considered as "CLOSED". **Tags:**xml.attribute=true

Table 7.15: LimitValueVariationPoint

Class	<<atpMixedString>> **NumericalValueVariationPoint**			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling::AttributeValueVariationPoints			
Note	This class represents an attribute value variation point for Numerical attributes. Note that this class might be used in the extended meta-model only.			
Base	*ARObject*, *AbstractNumericalVariationPoint*, *AttributeValueVariationPoint*, *FormulaExpression*, *Sw SystemconstDependentFormula*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table 7.16: NumericalValueVariationPoint

Class	<<atpMixedString>> **PositiveIntegerValueVariationPoint**			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling::AttributeValueVariationPoints			
Note	This class represents an attribute value variation point for positive Integer attributes. Note that this class might be used in the extended meta-model only.			
Base	*ARObject*, *AttributeValueVariationPoint*, *FormulaExpression*, *SwSystemconstDependentFormula*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table 7.17: PositiveIntegerValueVariationPoint

Class	<<atpMixedString>> **UnlimitedIntegerValueVariationPoint**			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling::AttributeValueVariationPoints			
Note	This class represents an attribute value variation point for unlimited Integer attributes. Note that this class might be used in the extended meta-model only.			
Base	*ARObject*, *AttributeValueVariationPoint*, *FormulaExpression*, *SwSystemconstDependentFormula*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table 7.18: UnlimitedIntegerValueVariationPoint

Class	<<atpMixedString>> **TimeValueValueVariationPoint**			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling::AttributeValueVariationPoints			
Note	This class represents the ability to express a formula for a numerical time value.			
Base	*ARObject*, *AttributeValueVariationPoint*, *FormulaExpression*, *SwSystemconstDependentFormula*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table 7.19: TimeValueValueVariationPoint

7.6.9 Combining *PreBuild* and *PostBuild* Variation Points

If `vh.latestBindingTime` is set to `PostBuild` ([constr_2517]) for a particular variation point, then this variation point may have a *PostBuild* branch represented by `PostBuildVariantCondition`[10]. However, it may also contain a *PreBuild* branch. This is because the *PreBuild* and *PostBuild* branches of `VariationPoint` are not mutually exclusive.

[TPS_GST_00266] PreBuild Disabling PostBuild support ⌈It is possible to define a variation point as *both PreBuild and PostBuild*. If the PreBuild condition is false, it is not expected that the variant object (including the PostBuild condition) will be implemented.

In other words, a system constant expression in a variation point may be used to disable the *PostBuild* variability even at *PreBuild* time.⌋*()*

Table 7.20 summarizes the options provided by [TPS_GST_00266].

PreBuild	PostBuild	
No Condition at all	No Condition at all	The element to which the VP is attached is always selected
No Condition at all	Unbound Condition	Pure PostBuild Variation Point
No Condition at all	Condition bound to true	Bound, selected PostBuild Variation Point (not visible in XML)
No Condition at all	Condition bound to false	Bound, deselected PostBuild Variation Point (not visible in XML)
Unbound Condition	No Condition at all	Pure PreBuild Variation Point
Unbound Condition	Unbound Condition	PreBuild selectable Postbuild Variation Point

[10] Vice versa, whether that a variation point is *PostBuild* can be recognized from the fact that it contains `PostBuildVariantCondition`.

Unbound Condition	Condition bound to true	PreBuild selectable Postbuild Variation Point (not visible in XML)
Unbound Condition	Condition bound to false	PreBuild selectable Postbuild Variation Point (not visible in XML)
Condition bound to true	No Condition at all	The element to which the VP is attached is always selected
Condition bound to true	Unbound Condition	Pure PostBuild Variation Point
Condition bound to true	Condition bound to true	Bound, selected PostBuild Variation Point (not visible in XML)
Condition bound to true	Condition bound to false	Bound, deselected PostBuild Variation Point (not visible in XML)
Condition bound to false	No Condition at all	Deselected PreBuild Variation Point, no Post-Build Variation Point
Condition bound to false	Unbound Condition	Deselected PreBuild Variation Point, no Post-Build Variation Point
Condition bound to false	Condition bound to true	Deselected PreBuild Variation Point, no Post-Build Variation Point
Condition bound to false	Condition bound to false	Deselected PreBuild Variation Point, no Post-Build Variation Point

Table 7.20: Combining *PreBuild* and *PostBuild* Variation Points

7.6.10 Notes and Restrictions

1. It is not supported ([TPS_GST_00199], [TPS_GST_00200]) to aggregate more than one `VariationPoint` at the same location. It is however possible to define both *PreBuild* and *PostBuild* conditions for a single variation point.

2. **[TPS_GST_00267] Only one BindingTime** ⌈It is not possible to state multiple binding times for a single variation point. The rationale for restriction is that if multiple binding times would really be used at the same location, then their conditions are likely to differ anyway. That is, there would not be a single variation point with multiple binding times, but several variation points instead.⌋*()*.

3. Due to the very nature of dealing with variants, it is possible to have multiple elements with the *same* `ShortName` until all *PreBuild* variants are resolved.

   This also means that the checking the model for consistency might not be fully possible until after all variants are resolved. This is because one purpose of variant handling is to model several incompatible variants and provide means to select one of the and discard the others, but this implies that the model cannot be fully consistent until this selection has been made.

   See also Section 7.6.2 ([constr_2512]) for more details on this issue.

4. **[TPS_GST_00268] Rationale for Different Approach for PreBuild and PostBuild Variation** ⌈The reason for handling *PreBuild* and *PostBuild* variation points differently is that if we would use only a `ConditionByFormula` (i.e., only the *PreBuild* branch) in both cases, then this condition would have to be evaluated

at start-up time. However, this would impose a performance penalty that is generally not acceptable. Hence, we use a simpler approach for *PostBuild* variation points which requires only a comparison.⌋*()*

5. **[TPS_GST_00269] Reference from invariant to variant parts.** ⌈References within an AUTOSAR Model may also be from invariant (respectively *PreBuild* variant) elements to post build variant elements if all variants of the variant element do have a meaning for the invariant elements (see [constr_2503])⌋*()*

   A sample use case is conditionally existing `SwComponentPrototype`, with `LatestBindingTime = postBuild`. The runnable to task mapping of the variant `SwComponentPrototype` is a `preCompileTime` ECUC parameter. But this mapping has a meaning for all variants of the variant `SwComponentPrototype`, because it is resolved before and does not need any additional condition.

6. A `shortLabel` could also be implemented by making a `VariationPoint` an `Identifiable` (see Section 4.3). However, `Identifiable` would be expensive for our purposes:

   - `Identifiable` has a significantly higher XML footprint than the `shortLabel` attribute.
   - A `shortLabel` is always optional while `Identifiable` adds a required `shortName`.

   In addition, `shortLabel` is intended for local identification (within the next enclosing `Identifiable`), while `Identifiable` is intended for reference purposes.

7.6.11 Using Variation Points for Blueprinting

As specified in [2] [TPS_STDT_00028], `VariationPoint` and `AttributeValueVariationPoint` are also used to specify details of deriving objects from blueprints.

[TPS_GST_00270] Variation Point in Blueprints ⌈Variation handling in Blueprints works differently from AUTOSAR variant handling elsewhere:

- Processing Blueprints can be seen as a separate binding time that occurs before `systemDesignTime`.
- The model does not give precise instructions how to handle the variation. Instead, only a textual description of what needs to be done is available.
- Variation points may occur for all elements in the blueprint that are subclasses of `ARElement`; [constr_2537] states that it does not apply for Blueprints.

⌋*()*

7.6.11.1 When is a variation point a Blueprint variation point?

The class VariationPoint (see Table 7.8) aggregates a `blueprintCondition`, or `formalBlueprintGenerator`. This object may only be aggregated in blueprints – that is, it may only be present if the `VariationPoint` lives in an AUTOSAR package of category `BLUEPRINT`. It may not be present In a system configuration.

If such an `DocumentationBlock` is present, then it contains instructions how to further process this variation point. The specific format of these instructions is not prescribed in detail, as the instructions are meant for humans or specialized (probably proprietary) tools.

Similarly, `AttributeValueVariationPoint` has an attribute `blueprintValue` that serves the same purpose as a `DocumentationBlock` in the role `blueprintCondition`.

[TPS_GST_00271] `blueprintCondition` cannot be variant ⌈As consequence of [constr_2559], a `VariationPoint` within a `DocumentationBlock` that is aggregated by `VariationPoint` is not allowed. The rationale for this is that such variations would have to be resolved at `systemDesignTime` or later, which comes *after* the blueprint has been processed.⌋*()*

7.6.11.2 `BlueprintDerivationTime`

[TPS_GST_00272] Semantics of BlueprintDerivationTime ⌈To support blueprints, the tag `vh.latestBindingTime` may have the value `BluePrintDerivationTime`. Such a variation point may only be present in a blueprint and may *not* be copied to a system configuration.

In this case, a variation point cannot have a `swSyscond` nor `postBuildVariantCondition` (as defined by [constr_2558] in [2]) because the information contained in these fields cannot be processed at `BluePrintDerivationTime`.

Similarly, the value an `AttributeValueVariationPoint` has no meaning in this case. Therefore it shall not be evaluated and the value shall be `undefined`.⌋*()* See also Section 7.4.2.

7.6.11.3 Which AUTOSAR model elements can be blueprint variation points?

Non-blueprint variation points – that is, variation points which are resolved at `systemDesignTime` or later – may not be used everywhere in a model. Their applicability is restricted by the meta-model to those locations that carry the stereotype ≪`atpVariation`≫. There are good reasons to do this; for example, the RTE shall be able to cope with the variation point at this location and shall be able to generate appropriate code.

The AUTOSAR elements `PackageableElement` and `ARElement` – which derives from it – form a special case. Since `PackageableElement` is variable with a latest binding time of `systemDesignTime`, *any* `ARElement` could be variable, which is clearly not intended. Hence, [constr_2537] restricts that to certain elements.

In Blueprints, [constr_2537] has been relaxed, and any `ARElement` may be a variation point.

[TPS_GST_00273] Resolve BlueprintVariationPoints on time ⌈When elements are copied from a variation point to a system configuration, then only those variation points may be transferred that are allowed to be variation points at `systemDesignTime` or later.

Hence, any variation of an `ARElement` (or something derived from it) in a blueprint that would be prohibited by [constr_2537] (which limits variation points to certain elements) has an implicit latest binding time of `bluePrintDerivationTime`.⌋*()*

7.7 Evaluated Variants

7.7.1 Motivation

Variant handling does not end with a description of *where* variation occurs (that is, the patterns we described in the previous sections of this chapter). Quite often, this description implies a huge number of variants[11], but only a subset of those is actually used.

This may be because the software is built to support a wider range of options than those of one particular OEM. But since the supplier has several OEM's as customer, he might design the software in such a way that it satisfies all variants. What is shipped to the OEM may only contain artifacts for his particular variants[12], or is at least certified for only those.

The variations are described by sets of system constant values. Hence, there is a need to describe which combinations of system constant values are valid. This provides the basis for OEMs and suppliers to exchange information on this subject in a standardized way.

7.7.2 Example

"APPROVED"	Basic	**Economy**	**Senior**	Sportive	Junior
Turbocharge	0	**0**	**1**	1	0
Automatic Transmission	0	**1**	**0**	1	0
Headlight	0	**1**	**2**	3	0

[11] Just five alternatives with three mutually exclusive options each yield a total of $3^5 = 243$ options.

[12] Especially if the variability is at `systemDesignTime` or `preCompileTime`; although this is generally not possible for *PostBuild* variations.

Sunroof	0	**0**	**0**	1	1

Table 7.21: Evaluated Variant Example, full table

Table 7.21 illustrates a example where we have four system constants (*Turbocharge*, *Automatic Transmission*, *Headlight* and *Sunroof*) which can assume integer values. In this example five variants were evaluated and named *Basic*, *Economy*, *Senior*, *Sportive*, *Junior*.

Basic, *Economy*, *Senior*, *Sportive*, *Junior* are called **predefined variants**. Each predefined variant is a combination of system constant values[13]. In other words, a `PredefinedVariant` is a column in the table above representing all evaluated variants.

The result of the evaluation is stated by the attribute `approvalStatus` in `EvaluatedVariantSet`. Corresponding to the example above the bold columns can be represented by an `EvaluatedVariantSet` with `approvalStatus` set to “APPROVED” as shown in table 7.22.

Furthermore, let us assume that a supplier is able to provide all five combinations (internally), but a fictitious OEM is interested in buying only *Economy* and *Senior*, as indicated by the bold column in Table 7.21.

In this case, the table that is exchanged between OEM and supplier will contain only the two columns (predefined variants) *Economy* and *Senior*.

”APPROVED”	**Economy**	**Senior**
Turbocharge	0	1
Automatic Transmission	1	0
Headlight	1	2
Sunroof	0	0

Table 7.22: Table that is exchanged between supplier and OEM

Table 7.22 shows the table that is exchanged between OEM and supplier. This table is also the basis for the XML example in Section 7.7.5.

7.7.2.1 Beyond the example

There are four more aspects in the concept for *evaluated variants* which are not shown in the above example:

1. An evaluated variant may refer to a specific component (or other element) for which the approval status in table applies.
2. A predefined variant may not only define values for system constants, but also for PostBuild variant criteria.

[13] A predefined variant also includes values for *PostBuild* variation points, which are omitted in Table 7.21 for clarity.

3. Columns in the table – even sub-columns – may be re-used by other tables. This is done by implementing the table through references, not aggregations. Both features are helpful if the data for a table of evaluated variants gets reused, or comes from different sources.
4. An evaluated variant may have an approval status which further details the meaning of the table. An evaluated variant may be “approved’, in which case the table contains predefined variants that are known to work, or it may be “rejected”, in which cases the predefined variants are known *not* to work.
5. A `PredefinedVariant` can include other `PredefinedVariant`s with a `includedVariant` association.

7.7.2.2 Use Cases covered in the example

The example above covers the following use cases:

- An integrator can use a table of evaluated variants to check whether a certain non-variant system (i.e. one where the variants have been resolved) is based on a predefined variant.
- A system designer can import preconfigured settings to build a particular variant.
- A component provider can use this mechanism to deliver a set of valid variants to a user of a component. This does not need to be the whole set of valid variants; for example the user may only be entitled to see (i.e., get information on) a certain subset.

7.7.3 Description

[TPS_GST_00277] Purpose of Evaluated Variants ⌈An `EvaluatedVariantSet` provides a way to describe a particular product variant and its approval status (approved/rejected) with respect to particular set of `CollectableElement`. This allows to state if a particular element is approved/rejected for a given variant.⌋*()*

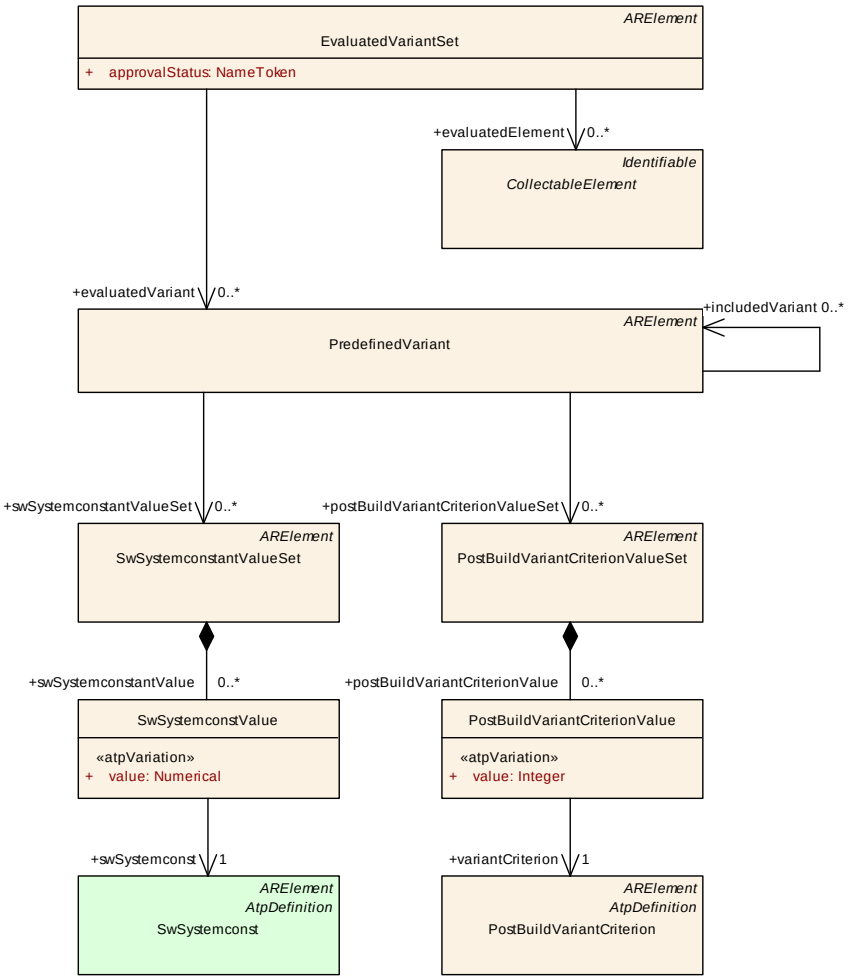


Figure 7.12: `EvaluatedVariantSet`

Tables 7.21 and 7.22 translate to Figure 7.12 as follows:

`EvaluatedVariantSet` The whole table is represented by the class `EvaluatedVariantSet`. `EvaluatedVariantSet` is an `ARElement`, so it has its own `shortName`, which is the name of the table.

[TPS_GST_00278] Establishing Multiple Validities with `EvaluatedVariantSet` for Different Aspects ⌈It is possible to have multiple `EvaluatedVariantSet`s. If there are several such sets, each set establishes a validity for a particular aspect. Individual aspects may be addressed by `shortName`.⌋*()* For example, unit tests may use their own specialized `EvaluatedVariantSet`s. Individual sets may be addressed by `shortName`.

`PredefinedVariant` An `EvaluatedVariantSet` contains a number of `PredefinedVariant`s. Each `PredefinedVariant` plus its included variants are a column in the table. The name of the column is the `shortName` of the `PredefinedVariant`.

[TPS_GST_00279] Definition of a Predefined Variant ⌈A PredefinedVariant represents a particular variant as a given combination of settings of variant selectors represented by SwSystemconstValue respectively PostBuildVariantCriterionValue.⌋*()*

The selection of Predefined Variants with SwSystemconstantValueSets applicable for ECU Configuration is done by the ECUC Variation Resolver (see 3.3.3 Variation Resolver Description in ECU Configuration).

The handling of PredefinedVariants with PostBuildVariantCriterionValueSets is described in [19] (see [SWS_Rte_06638]).

SwSystemconstantValueSet A PredefinedVariant contains a number of SwSystemconstantValueSet objects. In the simplest case, there is only one such object, which represents the entries of the column. More precisely, the PredefinedVariant represents the column including the header, while the SwSystemconstantValueSet is all that is below the header.

[TPS_GST_00280] SwSystemconstantValueSets from different sources ⌈It is also possible to distribute the entries of a column over several SwSystemconstantValueSet objects. The reason behind using several SwSystemconstantValueSets is to allow a predefined variant to be composed of system constant assignments that come from different sources.⌋*()*

To remain with the picture that was drawn in Figure 7.21, each column is then composed of several SwSystemconstantValueSets, whose contents are concatenated.

[constr_2519] PredefinedVariants need to be consistent ⌈If a PredefinedVariant plus its includedVariants references more than one SwSystemconstantValueSet all value attributes in SwSystemconstValues for a particular SwSystemconst shall be identical.⌋*()*

By constraint [constr_2519] contradicting value assignment are positively avoided.

SwSystemconstValue A SwSystemconstantValueSet contains a number of SwSystemconstValue objects, each of which represents a cell in the table, and implements – as the name says – a value for a single system constant.

The value that is stored in the cell is represented by the attribute value, which in turn subject to variation through *attribute value pattern* (see Section 7.4).

[TPS_GST_00281] Indirect value assignment for system constants ⌈The primary motivation for using a variation point here is convenience: the value is determined by an expression, and this is exactly what an attribute value variation point does. In practice, the expression should consist of a single system constant, most of the time.⌋*()*

SwSystemconst Each SwSystemconstValue provides a reference to a SwSystemconst. This is the system constant whose value is defined by SwSystemconstValue.

SwSystemconstantValueSet, SwSystemconstValue and SwSystemconst define *prebuild* variants. There is a second branch for PostBuild variants:

[TPS_GST_00282] Analogy between Predefined Variant for Pre Build and Post Build branch ⌈

PostBuildVariantCriterionValueSet is the *postbuild* analogue for SwSystemconstantValueSet.

PostBuildVariantCriterionValue is the *postbuild* analogue for SwSystemconstValue.

PostBuildVariantCriterion is the *postbuild* analogue for SwSystemconst.

⌋*()*

[TPS_GST_00283] Validity of Post Build combined with Pre Build Variant ⌈When both *prebuild* and *postbuild* variants are defined, then the *postbuild* variants apply to *all prebuild* variants.⌋*()*

Furthermore, PredefinedVariant, SwSystemconstantValueSet and PostBuildVariantCriterionValueSet are referenced, rather than aggregated, to enable reuse of variants. For example, a vendor might have several PredefinedVariant collections – one for each OEM – and reuse them in separate EvaluatedVariantSets (see [TPS_GST_00280]).

7.7.3.1 evaluatedElement

A EvaluatedVariantSet provides a reference to one or more CollectableElements (see chapter 13). This is used to identify the packages and elements that are covered by the PredefinedVariant. Note that EvaluatedVariantSet is also CollectableElement.

[constr_2507] EvaluatedVariantSet shall not refer to itself ⌈An EvaluatedVariantSet shall not refer to itself directly or via other EvaluatedVariantSet.⌋ *()*

For more details refer to Section 7.7.4.

7.7.3.2 approvalStatus

[TPS_GST_00284] Semantics of approvalStatus ⌈The attribute approvalStatus of an EvaluatedVariantSet further details the status of the evaluated variant. The following values are standardized:

1. APPROVED An “approved” variant is known to work.
2. REJECTED An “rejected” variant is known *not* to work.

Note that all other values are use case specific. Hence EvaluatedVariantSet with approvalStatus other than APPROVED or REJECTED is not defined by the standard and therefore shall be ignored when evaluating a system.⌋() See chapter 7.7.4).

7.7.3.3 includedVariant

[TPS_GST_00285] Purpose of includedVariant in PredefinedVariant ⌈The association includedVariant defines that settings of the referenced PredefinedVariants are handled as part of the settings of the referencing PredefinedVariant.⌋()

Suppose a variant rich system is composed out of variant rich sub systems delivered by several parties. In this case the providers of the subsystems need to define appropriate EvaluatedVariantSet for their delivery. Additionally the responsible party for the system needs to specify EvaluatedVariantSet for the entire system.

In order to do this he can use the includedVariants to refer to the definition of the PredefinedVariants of the sub system. Without this he would need to repeat those definitions. This would require a knowledge about the sub system and reduce the maintainablility of the system. By using includedVariants the creator of the system does not need any knowledge about the SwSystemconstantValueSets and PostBuildVariantCriterionValueSet used in the sub system.

7.7.4 Consistency

A particular EvaluatedVariantSet refers to CollectableElements in order to express their approval status. As EvaluatedVariantSet is also a CollectableElement, a hierarchy of EvaluatedVariantSets can occur.

On the other hand the meta-model establishes another hierarchy by aggregation of objects. It is important to clearly distinguish these two hierarchies when considering evalutated variants.

This section defines the details regarding consistency of such hierarchies.

[TPS_GST_00286] REJECTED precedes APPROVED ⌈Generally the status REJECTED takes precedence over the status APPROVED. That is, if e.g. an “approved” package contains a “rejected” software component, the whole package shall be regarded as REJECTED for this EvaluatedVariantSet.⌋()

[TPS_GST_00287] APPROVED for CollectableElement ⌈A CollectableElement is *rejected* for a given variant if

- it is referenced by at least one appropriate EvaluatedVariantSet withapprovalStatus set to *REJECTED*
- or aggregates (possibly over many levels) a CollectableElement which is referenced in an appropriate EvaluatedVariantSet with approvalStatus set to *REJECTED*.

⌋() Details to CollectableElement are described in chapter 3.

[TPS_GST_00288] REJECTED for CollectableElement ⌈A CollectableElement is *approved* for a given variant if

- it is referenced by at least one appropriate EvaluatedVariantSet with approvalStatus set to *APPROVED* or is not a *rejected* EvaluatedVariantSet. This makes sure that EvaluatedVariantSet are considered approved by default.
- and is not a *rejected* CollectableElement.

⌋() Details to CollectableElement are described in chapter 3.

7.7.5 XML Example for EvaluatedVariantSet

The following listing illustrates how an EvaluatedVariantSet is expressed in XML:

Listing 7.4: Example for evaluated variant in ARXML

```
<AR-PACKAGE>
  <SHORT-NAME>Variants</SHORT-NAME>
  <ELEMENTS>
    <PREDEFINED-VARIANT>
      <SHORT-NAME>Basic</SHORT-NAME>
      <SW-SYSTEMCONSTANT-VALUE-SET-REFS>
        <SW-SYSTEMCONSTANT-VALUE-SET-REF DEST="SW-SYSTEMCONSTANT-VALUE-
            SET">/SystemConstantValues/V1</SW-SYSTEMCONSTANT-VALUE-SET-
            REF>
        <SW-SYSTEMCONSTANT-VALUE-SET-REF DEST="SW-SYSTEMCONSTANT-VALUE-
            SET">/SystemConstantValues/V2</SW-SYSTEMCONSTANT-VALUE-SET-
            REF>
      </SW-SYSTEMCONSTANT-VALUE-SET-REFS>
    </PREDEFINED-VARIANT>
    <PREDEFINED-VARIANT>
      <SHORT-NAME>Economy</SHORT-NAME>
      <SW-SYSTEMCONSTANT-VALUE-SET-REFS>
        <SW-SYSTEMCONSTANT-VALUE-SET-REF DEST="SW-SYSTEMCONSTANT-VALUE-
            SET">/SystemConstantValues/V1</SW-SYSTEMCONSTANT-VALUE-SET-
            REF>
        <SW-SYSTEMCONSTANT-VALUE-SET-REF DEST="SW-SYSTEMCONSTANT-VALUE-
            SET">/SystemConstantValues/V3</SW-SYSTEMCONSTANT-VALUE-SET-
            REF>
      </SW-SYSTEMCONSTANT-VALUE-SET-REFS>
    </PREDEFINED-VARIANT>
    <PREDEFINED-VARIANT>
      <SHORT-NAME>Senior</SHORT-NAME>
```

```
        <SW-SYSTEMCONSTANT-VALUE-SET-REFS>
          <SW-SYSTEMCONSTANT-VALUE-SET-REF DEST="SW-SYSTEMCONSTANT-VALUE-
              SET">/SystemConstantValues/V4</SW-SYSTEMCONSTANT-VALUE-SET-
              REF>
        </SW-SYSTEMCONSTANT-VALUE-SET-REFS>
      </PREDEFINED-VARIANT>
      <PREDEFINED-VARIANT>
        <SHORT-NAME>Sportive</SHORT-NAME>
        <SW-SYSTEMCONSTANT-VALUE-SET-REFS>
          <SW-SYSTEMCONSTANT-VALUE-SET-REF DEST="SW-SYSTEMCONSTANT-VALUE-
              SET">/SystemConstantValues/V5</SW-SYSTEMCONSTANT-VALUE-SET-
              REF>
        </SW-SYSTEMCONSTANT-VALUE-SET-REFS>
      </PREDEFINED-VARIANT>
    </ELEMENTS>
  </AR-PACKAGE>

  <!--  now we have the systemconstant value sets -->
  <AR-PACKAGE>
    <SHORT-NAME>SystemConstantValues</SHORT-NAME>
    <ELEMENTS>
      <SW-SYSTEMCONSTANT-VALUE-SET>
        <SHORT-NAME>V1</SHORT-NAME>
        <SW-SYSTEMCONSTANT-VALUES>
          <SW-SYSTEMCONST-VALUE>
            <SW-SYSTEMCONST-REF DEST="SW-SYSTEMCONST">/SwSystemconsts/
                Turbocharge</SW-SYSTEMCONST-REF>
            <VALUE>0</VALUE>
          </SW-SYSTEMCONST-VALUE>
          <SW-SYSTEMCONST-VALUE>
            <SW-SYSTEMCONST-REF DEST="SW-SYSTEMCONST">/SwSystemconsts/
                Sunroof</SW-SYSTEMCONST-REF>
            <VALUE>0</VALUE>
          </SW-SYSTEMCONST-VALUE>
        </SW-SYSTEMCONSTANT-VALUES>
      </SW-SYSTEMCONSTANT-VALUE-SET>
      <SW-SYSTEMCONSTANT-VALUE-SET>
        <SHORT-NAME>V2</SHORT-NAME>
        <SW-SYSTEMCONSTANT-VALUES>
          <SW-SYSTEMCONST-VALUE>
            <SW-SYSTEMCONST-REF DEST="SW-SYSTEMCONST">/SwSystemconsts/
                Automatictransmission</SW-SYSTEMCONST-REF>
            <VALUE>0</VALUE>
          </SW-SYSTEMCONST-VALUE>
          <SW-SYSTEMCONST-VALUE>
            <SW-SYSTEMCONST-REF DEST="SW-SYSTEMCONST">/SwSystemconsts/
                Headlight</SW-SYSTEMCONST-REF>
            <VALUE>0</VALUE>
          </SW-SYSTEMCONST-VALUE>
        </SW-SYSTEMCONSTANT-VALUES>
      </SW-SYSTEMCONSTANT-VALUE-SET>
      <SW-SYSTEMCONSTANT-VALUE-SET>
        <SHORT-NAME>V3</SHORT-NAME>
        <SW-SYSTEMCONSTANT-VALUES>
          <SW-SYSTEMCONST-VALUE>
```

```
        <SW-SYSTEMCONST-REF DEST="SW-SYSTEMCONST">/SwSystemconsts/
            Automatictransmission</SW-SYSTEMCONST-REF>
        <VALUE>1</VALUE>
      </SW-SYSTEMCONST-VALUE>
      <SW-SYSTEMCONST-VALUE>
        <SW-SYSTEMCONST-REF DEST="SW-SYSTEMCONST">/SwSystemconsts/
            Headlight</SW-SYSTEMCONST-REF>
        <VALUE>1</VALUE>
      </SW-SYSTEMCONST-VALUE>
    </SW-SYSTEMCONSTANT-VALUES>
  </SW-SYSTEMCONSTANT-VALUE-SET>
  <!-- note that this set is used in all variants above -->
  <SW-SYSTEMCONSTANT-VALUE-SET>
    <SHORT-NAME>V4</SHORT-NAME>
    <SW-SYSTEMCONSTANT-VALUES>
      <SW-SYSTEMCONST-VALUE>
        <SW-SYSTEMCONST-REF DEST="SW-SYSTEMCONST">/SwSystemconsts/
            Turbocharge</SW-SYSTEMCONST-REF>
        <VALUE>1</VALUE>
      </SW-SYSTEMCONST-VALUE>
      <SW-SYSTEMCONST-VALUE>
        <SW-SYSTEMCONST-REF DEST="SW-SYSTEMCONST">/SwSystemconsts/
            Automatictransmission</SW-SYSTEMCONST-REF>
        <VALUE>0</VALUE>
      </SW-SYSTEMCONST-VALUE>
      <SW-SYSTEMCONST-VALUE>
        <SW-SYSTEMCONST-REF DEST="SW-SYSTEMCONST">/SwSystemconsts/
            Headlight</SW-SYSTEMCONST-REF>
        <VALUE>2</VALUE>
      </SW-SYSTEMCONST-VALUE>
      <SW-SYSTEMCONST-VALUE>
        <SW-SYSTEMCONST-REF DEST="SW-SYSTEMCONST">/SwSystemconsts/
            Sunroof</SW-SYSTEMCONST-REF>
        <VALUE>0</VALUE>
      </SW-SYSTEMCONST-VALUE>
    </SW-SYSTEMCONSTANT-VALUES>
  </SW-SYSTEMCONSTANT-VALUE-SET>
  <SW-SYSTEMCONSTANT-VALUE-SET>
    <SHORT-NAME>V5</SHORT-NAME>
    <SW-SYSTEMCONSTANT-VALUES>
      <SW-SYSTEMCONST-VALUE>
        <SW-SYSTEMCONST-REF DEST="SW-SYSTEMCONST">/SwSystemconsts/
            Turbocharge</SW-SYSTEMCONST-REF>
        <VALUE>1</VALUE>
      </SW-SYSTEMCONST-VALUE>
      <SW-SYSTEMCONST-VALUE>
        <SW-SYSTEMCONST-REF DEST="SW-SYSTEMCONST">/SwSystemconsts/
            Automatictransmission</SW-SYSTEMCONST-REF>
        <VALUE>1</VALUE>
      </SW-SYSTEMCONST-VALUE>
      <SW-SYSTEMCONST-VALUE>
        <SW-SYSTEMCONST-REF DEST="SW-SYSTEMCONST">/SwSystemconsts/
            Headlight</SW-SYSTEMCONST-REF>
        <VALUE>3</VALUE>
      </SW-SYSTEMCONST-VALUE>
      <SW-SYSTEMCONST-VALUE>
```

```
            <SW-SYSTEMCONST-REF DEST="SW-SYSTEMCONST">/SwSystemconsts/
               Sunroof</SW-SYSTEMCONST-REF>
            <VALUE>1</VALUE>
          </SW-SYSTEMCONST-VALUE>
        </SW-SYSTEMCONSTANT-VALUES>
      </SW-SYSTEMCONSTANT-VALUE-SET>
    </ELEMENTS>
  </AR-PACKAGE>

  <!--  here we have the evaluated variants -->
  <AR-PACKAGE>
    <SHORT-NAME>EvaluatedVariants</SHORT-NAME>
    <ELEMENTS>
      <EVALUATED-VARIANT-SET>
        <SHORT-NAME>foobar</SHORT-NAME>
        <EVALUATED-ELEMENT-REFS>
          <EVALUATED-ELEMENT-REF DEST="APPLICATION-SW-COMPONENT-TYPE">/
            Components/foo</EVALUATED-ELEMENT-REF>
          <EVALUATED-ELEMENT-REF DEST="APPLICATION-SW-COMPONENT-TYPE">/
            Components/bar</EVALUATED-ELEMENT-REF>
        </EVALUATED-ELEMENT-REFS>
        <EVALUATED-VARIANT-REFS>
          <EVALUATED-VARIANT-REF DEST="PREDEFINED-VARIANT">/Variants/
            Economy</EVALUATED-VARIANT-REF>
          <EVALUATED-VARIANT-REF DEST="PREDEFINED-VARIANT">/Variants/
            Senior</EVALUATED-VARIANT-REF>
        </EVALUATED-VARIANT-REFS>
      </EVALUATED-VARIANT-SET>
    </ELEMENTS>
  </AR-PACKAGE>


  <!-- here we have the components -->
  <AR-PACKAGE>
    <SHORT-NAME>Components</SHORT-NAME>
    <ELEMENTS>
      <APPLICATION-SW-COMPONENT-TYPE>
        <SHORT-NAME>foo</SHORT-NAME>
      </APPLICATION-SW-COMPONENT-TYPE>
      <APPLICATION-SW-COMPONENT-TYPE>
        <SHORT-NAME>bar</SHORT-NAME>
      </APPLICATION-SW-COMPONENT-TYPE>
    </ELEMENTS>
  </AR-PACKAGE>
 </AR-PACKAGES>
</AUTOSAR>
```

Note: system constants are not only identified by their name (as implied in Table 7.21), but they are in fact identified by the full reference. This avoids name clashes in hierarchies, e.g. if two (sub)components come from different vendors that for some reason have used the same names.

7.7.6 Classtables

Class	**EvaluatedVariantSet**			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling			
Note	This meta class represents the ability to express if a set of ARElements is able to support one or more particular variants. In other words, for a given set of evaluatedElements this meta class represents a table of evaluated variants, where each PredefinedVariant represents one column. In this column each descendant sw SystemconstantValue resp. postbuildVariantCriterionValue represents one entry. In a graphical representation each swSystemconstantValueSet / postBuildVariantCriterionValueSet could be used as an intermediate headline in the table column. If the approvalStatus is "APPROVED" it expresses that the collection of CollectableElements is known be valid for the given evaluatedVariants. Note that the EvaluatedVariantSet is a CollectableElement. This allows to establish a hierarchy of EvaluatedVariantSets. **Tags:**atp.recommendedPackage=EvaluatedVariantSets			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
approvalStatus	NameToken	1	attr	Defines the approval status of a predefined variant. Two values are predefined: "APPROVED" and "REJECTED": • Approved variants are known to work. • Rejected variants are known NOT to work. Further values can be approved on a per-company basis; within AUTOSAR only "APPROVED" and "REJECTED" should be recognized.
evaluated Element	CollectableElement	*	ref	This represents a particular element which is evaluated in context of the EvaluatedVariants. The approvalStatus applies to this element (and all of its descendants). In other words, the referenced elements are those that were considered when the predefined variant was evaluated.
evaluated Variant	PredefinedVariant	*	ref	This metaclass represents one particular variant which was evaluated. LowerMultiplicity is set to 0 to support a stepwise approach.

Table 7.23: EvaluatedVariantSet

Class	**PredefinedVariant**			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling			
Note	This specifies one predefined variant. It is characterized by the union of all system constant values and post-build variant criterion values aggregated within all referenced system constant value sets and post build variant criterion value sets plus the value sets of the included variants. **Tags:**atp.recommendedPackage=PredefinedVariants			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
includedVariant	PredefinedVariant	*	ref	The associated variants are considered part of this PredefinedVariant. This means the settings of the included variants are included in the settings of the referencing PredefinedVariant. Nevertheless the included variants might be included in several predefined variants.

▽

△

Class	**PredefinedVariant**			
postBuildVariant CriterionValue Set	PostBuildVariant CriterionValueSet	*	ref	This is the postBuildVariantCriterionValueSet contributing to the predefinded variant.
sw Systemconstant ValueSet	SwSystemconstant ValueSet	*	ref	This ist the set of Systemconstant Values contributing to the predefined variant.

Table 7.24: PredefinedVariant

Class	**SwSystemconstantValueSet**			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling			
Note	This meta-class represents the ability to specify a set of system constant values. **Tags:**atp.recommendedPackage=SwSystemconstantValueSets			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
sw Systemconstant Value	SwSystemconstValue	*	aggr	This is one particular value of a system constant.

Table 7.25: SwSystemconstantValueSet

see also `SwSystemconstValue` see also `SwSystemconst`

Class	**PostBuildVariantCriterionValueSet**			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling			
Note	This meta-class represents the ability to denote one set of postBuildVariantCriterionValues. **Tags:**atp.recommendedPackage=PostBuildVariantCriterionValueSets			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
postBuildVariant CriterionValue	PostBuildVariant CriterionValue	*	aggr	This is is one particular postbuild variant criterion/value pair being part of the PostBuildVariantSet.

Table 7.26: PostBuildVariantCriterionValueSet

Class	**PostBuildVariantCriterionValue**			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling			
Note	This class specifies the value which shall be assigned to a particular variant criterion in order to bind the variation point. If multiple criterion/value pairs are specified, they all shall match to bind the variation point.			
Base	*ARObject*			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
annotation	Annotation	*	aggr	This provides the ability to add information why the value is set like it is. **Tags:**xml.sequenceOffset=30

▽

△

Class	**PostBuildVariantCriterionValue**			
value	Integer	1	attr	This is the particular value of the post-build variant criterion. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=20
variantCriterion	PostBuildVariant Criterion	1	ref	This association selects the variant criterion whose value is specified. **Tags:**xml.sequenceOffset=10

Table 7.27: PostBuildVariantCriterionValue

see also `PostBuildVariantCriterion`

7.8 Choosing Variants

In this section, we describe how to represent a particular variant of an AUTOSAR system.

7.8.1 What is a Variant?

[TPS_GST_00289] Definition of a Variant ⌈A variant is a collection of bound variation points. In other words, it is represented by a complete assignment of values to system constants respectively post build variant criteria.⌋*()*

More formally, a *variant* is characterized by the following:

1. **Binding Time:** Variants are tied to a particular binding time. This binding time is given in the following model elements:

   - `ConditionByFormula`, which is aggregated by `VariationPoint` in the *aggregation pattern* (Figure 7.2), the *association pattern* (Figure 7.3), and the *property set pattern* (Figures 7.7, 7.6).
   - `AttributeValueVariationPoint` in the *attribute value pattern* (Figure 7.4).

   [constr_2518] Binding time is constrained ⌈Note that this binding time is again constrained by the value of the tag `vh.latestBindingTime`.⌋*()*

   See Section 7.1.6 for details.

2. **Variation Points:** The set of variation points that need to bound at the binding time of the variant. In the model, this affects the following elements:

   - `VariationPoint` is used in the *aggregation pattern*, (Figure 7.2), the *association pattern* (Figure 7.3) and the *property set pattern* (Figures 7.7, 7.6).

- AttributeValueVariationPoint in the *attribute value pattern* (Figure 7.4).

3. **System Constants:** The set of system constants that are referenced from within variation points. System constants are referenced from SwSystemconstDependentFormula and its subclasses (Section 7.6.8).
4. **Value assignments:** The system constants determined in the previous step need to be assigned values.

7.8.2 Valid Variants

[TPS_GST_00290] Defintion of Valid Variants ⌈A *valid variant* is defined as follows:

1. A valid variant $\mathcal{V}$ for a binding time $\mathcal{T}$ is an assignment of values to all system constants that are referenced by all variation points which have the binding time $\mathcal{T}$.
2. There exists no PredefinedVariant that covers the systems in $\mathcal{V}$, and its status is REJECTED.

⌋()

7.9 Examples

In this section, we provide four examples for variant handling, each focusing on a single template. We also provide possible implementations to illustrate how these use cases might be translated into practice. It should however be noted that these are just sample implementations, and are not necessarily the only possible way to realize a particular use case.

7.9.1 Example for *Aggregation Pattern*

Description Software component type with variant ports.

Use Case A body control application uses one of two light controllers: low end systems uses the *low-end-light control*, while a high end systems uses *adaptive-curve light-control*.

Implementation The body control application has a port which is used to communicate with the light controller. There are two variants for the connector of this port: a low end and an adaptive curve variant. These two variants are exclusive.

A system constant CarType is used to switch between the two variants; a value of 1 means “low end” while a value of 2 means “adaptive curve”.

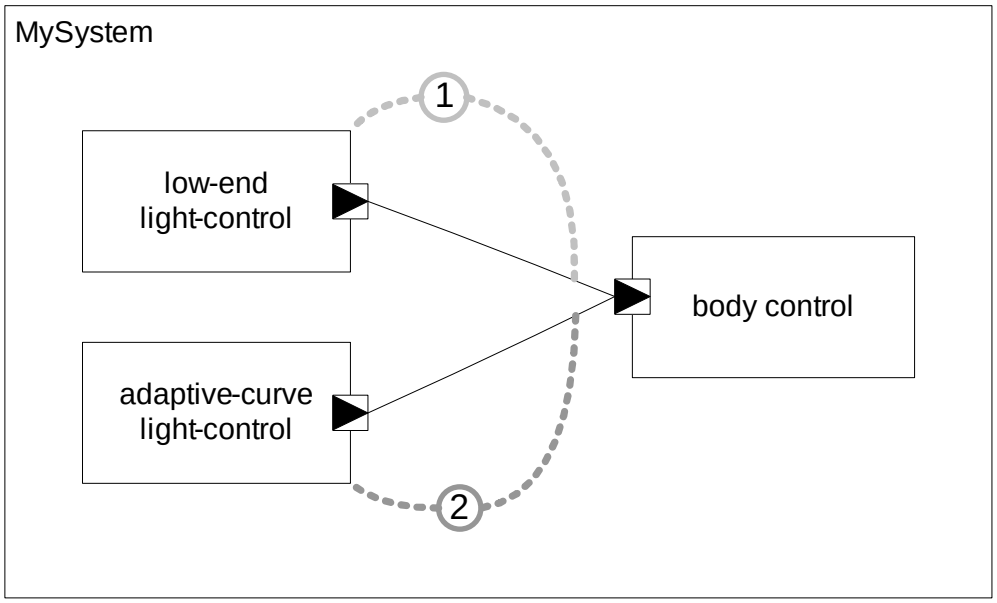


Figure 7.13: Example for *Aggregation Pattern*

Listing 7.5: Example for aggregation pattern in ARXML

```
<AR-PACKAGE>
  <SHORT-NAME>AggregationPattern</SHORT-NAME>
  <INTRODUCTION>
    <P>
      <L-1
        L="EN">Note that the example slightly varies from
        the example in the document: Body control has one port
        which is used for both variants. </L-1>
    </P>
  </INTRODUCTION>
  <REFERENCE-BASES>
    <REFERENCE-BASE>
      <SHORT-LABEL>default</SHORT-LABEL>
      <IS-DEFAULT>true</IS-DEFAULT>
      <IS-GLOBAL>false</IS-GLOBAL>
      <BASE-IS-THIS-PACKAGE>false</BASE-IS-THIS-PACKAGE>
      <PACKAGE-REF
        DEST="AR-PACKAGE">/AggregationPattern</PACKAGE-REF>
    </REFERENCE-BASE>
  </REFERENCE-BASES>
  <ELEMENTS>
    <SW-SYSTEMCONST>
      <SHORT-NAME>CarType</SHORT-NAME>
    </SW-SYSTEMCONST>
    <COMPOSITION-SW-COMPONENT-TYPE>
      <SHORT-NAME>MySystem</SHORT-NAME>
      <COMPONENTS>
        <SW-COMPONENT-PROTOTYPE>
          <SHORT-NAME>LowEndLightControl</SHORT-NAME>
          <TYPE-TREF
            DEST="APPLICATION-SW-COMPONENT-TYPE">LowEndLightControl</
              TYPE-TREF>
          <VARIATION-POINT>
```

```
        <SHORT-LABEL>LowEnd</SHORT-LABEL>
        <SW-SYSCOND
          BINDING-TIME="SYSTEM-DESIGN-TIME">
          <SYSC-REF
          DEST="SW-SYSTEMCONST">CarType</SYSC-REF>
          == 1
        </SW-SYSCOND>
      </VARIATION-POINT>
    </SW-COMPONENT-PROTOTYPE>
    <SW-COMPONENT-PROTOTYPE>
      <SHORT-NAME>AdaptiveCurveLightControl</SHORT-NAME>
      <TYPE-TREF
        DEST="APPLICATION-SW-COMPONENT-TYPE">
          AdaptiveCurveLightControl</TYPE-TREF>
      <VARIATION-POINT>
        <SHORT-LABEL>HighEnd</SHORT-LABEL>
        <SW-SYSCOND
          BINDING-TIME="SYSTEM-DESIGN-TIME">
          <SYSC-REF
          DEST="SW-SYSTEMCONST">CarType</SYSC-REF>
          == 2
        </SW-SYSCOND>
      </VARIATION-POINT>
    </SW-COMPONENT-PROTOTYPE>
    <SW-COMPONENT-PROTOTYPE>
      <SHORT-NAME>BodyControl</SHORT-NAME>
      <TYPE-TREF
        DEST="APPLICATION-SW-COMPONENT-TYPE">BodyControl</TYPE-TREF
          >
    </SW-COMPONENT-PROTOTYPE>
  </COMPONENTS>
  <CONNECTORS>
    <ASSEMBLY-SW-CONNECTOR>
      <SHORT-NAME>LightControl</SHORT-NAME>
      <INTRODUCTION>
        <P>
          <L-1
            L="EN">Note that the shortname in both variants is
            the same.</L-1>
        </P>
      </INTRODUCTION>
      <VARIATION-POINT>
        <SHORT-LABEL>LowEnd</SHORT-LABEL>
        <SW-SYSCOND
          BINDING-TIME="SYSTEM-DESIGN-TIME">
          <SYSC-REF
          DEST="SW-SYSTEMCONST">CarType</SYSC-REF>
          == 1
        </SW-SYSCOND>
      </VARIATION-POINT>
      <PROVIDER-IREF>
        <CONTEXT-COMPONENT-REF
          DEST="SW-COMPONENT-PROTOTYPE">LowEndLightControl</CONTEXT
            -COMPONENT-REF>
        <TARGET-P-PORT-REF
```

```
            DEST="P-PORT-PROTOTYPE">/AggregationPattern/
              LowEndLightControl/ToLight</TARGET-P-PORT-REF>
          </PROVIDER-IREF>
          <REQUESTER-IREF>
            <CONTEXT-COMPONENT-REF
              DEST="SW-COMPONENT-PROTOTYPE">BodyControl</CONTEXT-
                COMPONENT-REF>
            <TARGET-R-PORT-REF
              DEST="R-PORT-PROTOTYPE">/AggregationPattern/BodyControl/
                LightControlInput</TARGET-R-PORT-REF>
          </REQUESTER-IREF>
        </ASSEMBLY-SW-CONNECTOR>
        <ASSEMBLY-SW-CONNECTOR>
          <SHORT-NAME>LightControl</SHORT-NAME>
          <VARIATION-POINT>
            <SHORT-LABEL>HighEnd</SHORT-LABEL>
            <SW-SYSCOND
              BINDING-TIME="SYSTEM-DESIGN-TIME">
              <SYSC-REF
              DEST="SW-SYSTEMCONST">CarType</SYSC-REF>
              == 2
            </SW-SYSCOND>
          </VARIATION-POINT>
          <PROVIDER-IREF>
            <CONTEXT-COMPONENT-REF
              DEST="SW-COMPONENT-PROTOTYPE">AdaptiveCurveLightControl</
                CONTEXT-COMPONENT-REF>
            <TARGET-P-PORT-REF
              DEST="P-PORT-PROTOTYPE">/AggregationPattern/
                AdaptiveCurveLightControl/ToLight</TARGET-P-PORT-REF>
          </PROVIDER-IREF>
          <REQUESTER-IREF>
            <CONTEXT-COMPONENT-REF
              DEST="SW-COMPONENT-PROTOTYPE">BodyControl</CONTEXT-
                COMPONENT-REF>
            <TARGET-R-PORT-REF
              DEST="R-PORT-PROTOTYPE">/AggregationPattern/BodyControl/
                LightControlInput</TARGET-R-PORT-REF>
          </REQUESTER-IREF>
        </ASSEMBLY-SW-CONNECTOR>
      </CONNECTORS>
    </COMPOSITION-SW-COMPONENT-TYPE>
    <APPLICATION-SW-COMPONENT-TYPE>
      <SHORT-NAME>LowEndLightControl</SHORT-NAME>
      <PORTS>
        <P-PORT-PROTOTYPE>
          <SHORT-NAME>ToLight</SHORT-NAME>
        </P-PORT-PROTOTYPE>
      </PORTS>
    </APPLICATION-SW-COMPONENT-TYPE>
    <APPLICATION-SW-COMPONENT-TYPE>
      <SHORT-NAME>AdaptiveCurveLightControl</SHORT-NAME>
      <PORTS>
        <P-PORT-PROTOTYPE>
          <SHORT-NAME>ToLight</SHORT-NAME>
        </P-PORT-PROTOTYPE>
```

```
      </PORTS>
    </APPLICATION-SW-COMPONENT-TYPE>
    <APPLICATION-SW-COMPONENT-TYPE>
      <SHORT-NAME>BodyControl</SHORT-NAME>
      <PORTS>
        <R-PORT-PROTOTYPE>
          <SHORT-NAME>LightControlInput</SHORT-NAME>
        </R-PORT-PROTOTYPE>
      </PORTS>
    </APPLICATION-SW-COMPONENT-TYPE>
  </ELEMENTS>
</AR-PACKAGE>
```

Listing 7.5 contains two pairs of VARIATION-POINT elements: one that switches between the alternative component prototypes, and one that switches between the alternative connectors.

The following excerpt from the above XML shows the code for the three component prototypes *LowEndLightControl* , *AdaptiveCurveLightControl* and *BodyControl*:

Listing 7.6: Variant Component Prototypes

```
<COMPONENTS>
  <SW-COMPONENT-PROTOTYPE>
    <SHORT-NAME>LowEndLightControl</SHORT-NAME>
    <TYPE-TREF
      DEST="APPLICATION-SW-COMPONENT-TYPE">LowEndLightControl</
          TYPE-TREF>
    <VARIATION-POINT>
      <SHORT-LABEL>LowEnd</SHORT-LABEL>
      <SW-SYSCOND
        BINDING-TIME="SYSTEM-DESIGN-TIME">
        <SYSC-REF
        DEST="SW-SYSTEMCONST">CarType</SYSC-REF>
        == 1
      </SW-SYSCOND>
    </VARIATION-POINT>
  </SW-COMPONENT-PROTOTYPE>
  <SW-COMPONENT-PROTOTYPE>
    <SHORT-NAME>AdaptiveCurveLightControl</SHORT-NAME>
    <TYPE-TREF
      DEST="APPLICATION-SW-COMPONENT-TYPE">
          AdaptiveCurveLightControl</TYPE-TREF>
    <VARIATION-POINT>
      <SHORT-LABEL>HighEnd</SHORT-LABEL>
      <SW-SYSCOND
        BINDING-TIME="SYSTEM-DESIGN-TIME">
        <SYSC-REF
        DEST="SW-SYSTEMCONST">CarType</SYSC-REF>
        == 2
      </SW-SYSCOND>
    </VARIATION-POINT>
  </SW-COMPONENT-PROTOTYPE>
  <SW-COMPONENT-PROTOTYPE>
    <SHORT-NAME>BodyControl</SHORT-NAME>
    <TYPE-TREF
```

```
            DEST="APPLICATION-SW-COMPONENT-TYPE">BodyControl</TYPE-TREF
              >
          </SW-COMPONENT-PROTOTYPE>
        </COMPONENTS>
```

Both *LowEndLightControl* and *AdaptiveCurveLightControl* contain a VARIATION-POINT element, which checks the value of the system constant named CarType.

There are similar variation points embedded in the description of the assembly connector prototypes. Note that these variation points have identical SHORT-NAMEs (*LightControl*), but use different SHORT-LABELs (*LowEnd* and *HighEnd*), as described in Section 7.6.2.

7.9.2 Example for *Association Pattern*

Description Software component type with variant diagnostics procedures.

Use Case The light controller from the previous example provides a standard and a high end method for diagnostics.

Implementation There are two diagnostics procedures: standard_light_diagnostics_proc and complex_diagnostics_proc. These procedures realize different algorithms, which require different API function calls. standard_light_diagnostics_proc is always called, complex_diagnostics_proc only in high-end vehicles.

The system constant *CarType* discriminates between the two types of vehicles: a value of 1 means standard car, a value of 2 high end car.

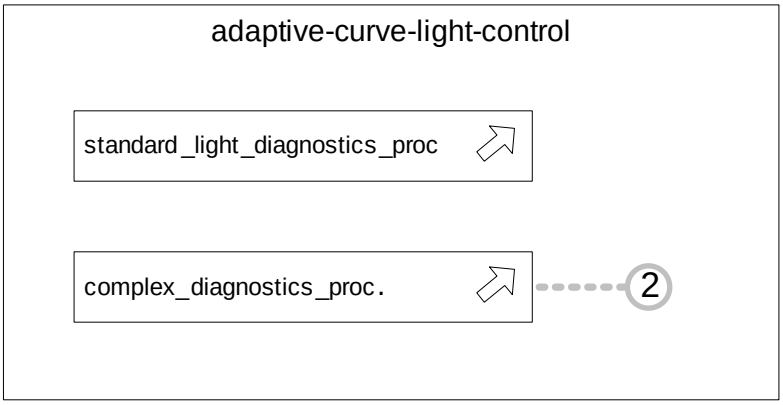


Figure 7.14: Example for *association pattern*

Listing 7.7: Example for association pattern in ARXML

```
<AR-PACKAGE>
  <SHORT-NAME>AssociationPattern</SHORT-NAME>
  <REFERENCE-BASES>
    <REFERENCE-BASE>
      <SHORT-LABEL>default</SHORT-LABEL>
      <IS-DEFAULT>true</IS-DEFAULT>
```

```
      <IS-GLOBAL>false</IS-GLOBAL>
      <BASE-IS-THIS-PACKAGE>false</BASE-IS-THIS-PACKAGE>
      <PACKAGE-REF
        DEST="AR-PACKAGE">/AssociationPattern</PACKAGE-REF>
    </REFERENCE-BASE>
  </REFERENCE-BASES>
  <ELEMENTS>
    <BSW-MODULE-ENTRY>
      <SHORT-NAME>standard_light_diagnostics_proc</SHORT-NAME>
    </BSW-MODULE-ENTRY>
    <BSW-MODULE-ENTRY>
      <SHORT-NAME>complex_diagnostics_proc</SHORT-NAME>
    </BSW-MODULE-ENTRY>
    <BSW-MODULE-DESCRIPTION>
      <SHORT-NAME>MyDiagnosticManager</SHORT-NAME>
      <INTERNAL-BEHAVIORS>
        <BSW-INTERNAL-BEHAVIOR>
          <SHORT-NAME>Behavior</SHORT-NAME>
          <ENTITYS>
            <BSW-SCHEDULABLE-ENTITY>
              <SHORT-NAME>TheMainFunc</SHORT-NAME>
              <INTRODUCTION>
                <P>
                  <L-1
                    L="EN">complex_diagnostics_proc is called
                    in case of Highend. standard_light_diagnostics_proc
                    is always called.</L-1>
                </P>
              </INTRODUCTION>
              <CALLED-ENTRYS>
                <BSW-MODULE-ENTRY-REF-CONDITIONAL>
                  <BSW-MODULE-ENTRY-REF
                    DEST="BSW-MODULE-ENTRY">complex_diagnostics_proc</
                      BSW-MODULE-ENTRY-REF>
                  <VARIATION-POINT>
                    <SW-SYSCOND
                      BINDING-TIME="CODE-GENERATION-TIME">
                      <SYSC-REF
                      DEST="SW-SYSTEMCONST">/AggregationPattern/CarType
                          </SYSC-REF>
                      == 2
                    </SW-SYSCOND>
                  </VARIATION-POINT>
                </BSW-MODULE-ENTRY-REF-CONDITIONAL>
                <BSW-MODULE-ENTRY-REF-CONDITIONAL>
                  <BSW-MODULE-ENTRY-REF
                    DEST="BSW-MODULE-ENTRY">
                      standard_light_diagnostics_proc</BSW-MODULE-
                      ENTRY-REF>
                </BSW-MODULE-ENTRY-REF-CONDITIONAL>
              </CALLED-ENTRYS>
            </BSW-SCHEDULABLE-ENTITY>
          </ENTITYS>
        </BSW-INTERNAL-BEHAVIOR>
      </INTERNAL-BEHAVIORS>
    </BSW-MODULE-DESCRIPTION>
```

```
    </ELEMENTS>
  </AR-PACKAGE>
```

The variation in Listing 7.7 is contained in the part where the BswModuleEntity calls the BswModuleEntry:

Listing 7.8: Example for association pattern in ARXML

```
<CALLED-ENTRYS>
  <BSW-MODULE-ENTRY-REF-CONDITIONAL>
    <BSW-MODULE-ENTRY-REF
      DEST="BSW-MODULE-ENTRY">complex_diagnostics_proc</
          BSW-MODULE-ENTRY-REF>
    <VARIATION-POINT>
      <SW-SYSCOND
        BINDING-TIME="CODE-GENERATION-TIME">
        <SYSC-REF
        DEST="SW-SYSTEMCONST">/AggregationPattern/CarType
            </SYSC-REF>
        == 2
      </SW-SYSCOND>
    </VARIATION-POINT>
  </BSW-MODULE-ENTRY-REF-CONDITIONAL>
  <BSW-MODULE-ENTRY-REF-CONDITIONAL>
    <BSW-MODULE-ENTRY-REF
      DEST="BSW-MODULE-ENTRY">
          standard_light_diagnostics_proc</BSW-MODULE-
          ENTRY-REF>
  </BSW-MODULE-ENTRY-REF-CONDITIONAL>
</CALLED-ENTRYS>
```

BswModuleEntity has a variant reference to BswModuleEntry (under the role name calledEntry). Hence, <BSW-MODULE-ENTITY> contains *two* elements named <BSW-MODULE-ENTRY-REF-CONDITIONAL>, each of which contains a reference to the called entry. The first one includes a <VARIATION-POINT>.

If the reference from BswModuleEntity to BswModuleEntry would not be variable, then there could be only a single element named <BSW-MODULE-ENTRY-REF>, and of course there would be no <VARIATION-POINT>. But otherwise, the code would look the same.

7.9.3 Example for *Attribute Value Pattern*

Description Adaptive algorithm for a variant array size

Use Case An engine control system needs to adapt to a wide range of engines. Especially, the number of cylinders may vary from engine to engine. The implementation needs to store data for each cylinder.

Implementation Data for the cylinders is stored in an array, who's size corresponds to the number of cylinders. Since we are in an embedded system, we cannot use dynamic arrays, but shall use a static array, e.g. CylinderData

cylinder_measures[2*NO_OF_CYLINDERS]. In the meta-model, the size of the array (2*NO_OF_CYLINDERS) is an attribute of a class. This attribute shall be variant.

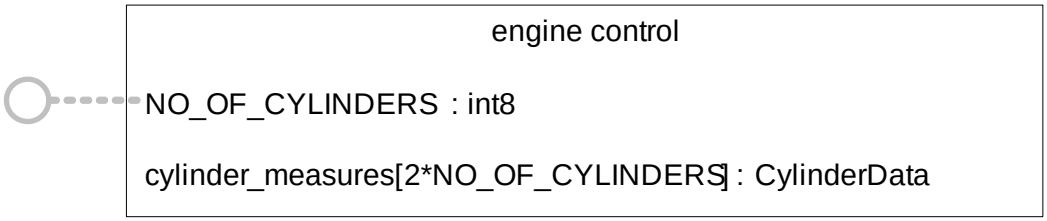


Figure 7.15: Example for *attribute value pattern*

Listing 7.9: Example for attribute value pattern in ARXML

```
<AR-PACKAGE>
  <SHORT-NAME>AttributeValuePattern</SHORT-NAME>
  <REFERENCE-BASES>
    <REFERENCE-BASE>
      <SHORT-LABEL>default</SHORT-LABEL>
      <IS-DEFAULT>true</IS-DEFAULT>
      <IS-GLOBAL>false</IS-GLOBAL>
      <BASE-IS-THIS-PACKAGE>false</BASE-IS-THIS-PACKAGE>
      <PACKAGE-REF
        DEST="AR-PACKAGE">/AttributeValuePattern</PACKAGE-REF>
    </REFERENCE-BASE>
  </REFERENCE-BASES>
  <ELEMENTS>
    <SW-SYSTEMCONST>
      <SHORT-NAME>NO_OF_CYLINDERS</SHORT-NAME>
    </SW-SYSTEMCONST>
    <APPLICATION-RECORD-DATA-TYPE>
      <SHORT-NAME>CylinderMeasure</SHORT-NAME>
    </APPLICATION-RECORD-DATA-TYPE>
    <APPLICATION-ARRAY-DATA-TYPE>
      <SHORT-NAME>Cylinder_measures</SHORT-NAME>
      <ELEMENT>
        <SHORT-NAME>Cylinder_Measure</SHORT-NAME>
        <CATEGORY>VALUE</CATEGORY>
        <TYPE-TREF
          DEST="APPLICATION-DATA-TYPE">CylinderMeasure</TYPE-TREF>
        <MAX-NUMBER-OF-ELEMENTS
          BINDING-TIME="CODE-GENERATION-TIME">
          2 * <SYSC-REF
          DEST="SW-SYSTEMCONST">NO_OF_CYLINDERS</SYSC-REF>
        </MAX-NUMBER-OF-ELEMENTS>
      </ELEMENT>
    </APPLICATION-ARRAY-DATA-TYPE>
  </ELEMENTS>
</AR-PACKAGE>
```

In this example, the variation lies in the size of the array cylinder_measures. Normally, this size would be a value; in this example, it is a reference to a system constant:

```
<MAX-NUMBER-OF-ELEMENTS BINDING-TIME="CODE-GENERATION-TIME">
 2 * <SYSC-REF DEST="SW-SYSTEMCONST">NO_OF_CYLINDERS</SYSC-REF>
```

```
</MAX-NUMBER-OF-ELEMENTS>
```

As one can see, there is no tag VARIATION-POINT. The variation point can be recognized by the attribute BINDING-TIME[14].

7.9.4 Example for *Property Set Pattern*

Description Varying properties of a FlexrayCommunicationController.

Use Case An ECU is connected to a FlexRay bus in two different configurations (car types), requiring different FlexrayCommunicationController settings.

Implementation Several property values of a FlexrayCommunicationController need to be consistently set or modified in one configuration step.

In our example, we have two variants for the attributes microPerCycle, microtickDuration, and samplesPerMicrotick of the FlexrayCommunicationController. Since these three attributes are mutually dependent in both configurations, and their values need to be set together.

The system constant CarType is used to discriminate between the two variants, CarType1 and CarType2, illustrated in Table 7.28.

All other attributes are non-variant in our example. We show only the (alphabetically) first and last attributes of FlexrayCommunicationController, namely acceptedStartupRange and wakeUpPattern in the XML file, omitting the others for brevity.

	CarType 1	CarType 2
microPerCycle	320000	80000
microtickDuration	$50E-9$	$12.5E-9$
samplesPerMicrotick	4	1

Table 7.28: Property Set Pattern: CarType

This example is illustrated in Figure 7.16.

[14] Consequently, a formula without a BINDING-TIME attribute on the enclosing XML element is an error.

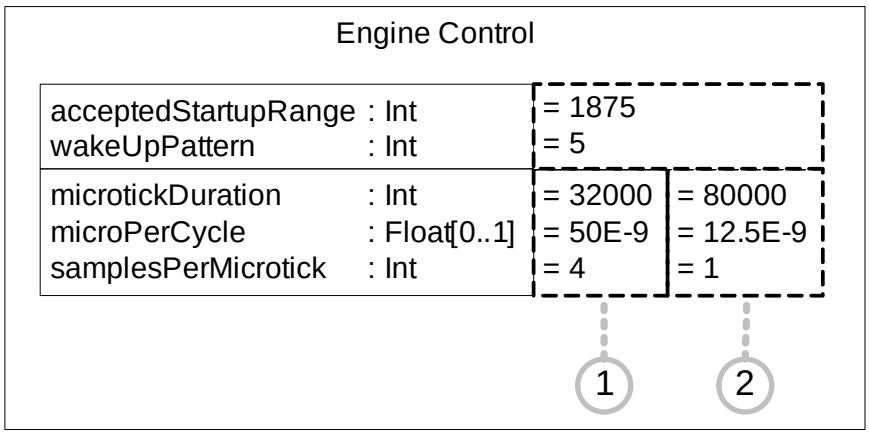


Figure 7.16: Example for *property set pattern*

Listing 7.10 shows the XML code for our example.

Listing 7.10: Example for property set pattern

```
<AR-PACKAGE>
  <SHORT-NAME>PropertySetPattern</SHORT-NAME>
  <ELEMENTS>
    <ECU-INSTANCE>
      <SHORT-NAME>iPhone</SHORT-NAME>
      <COMM-CONTROLLERS>
        <FLEXRAY-COMMUNICATION-CONTROLLER>
          <SHORT-NAME>iFlex</SHORT-NAME>
          <FLEXRAY-COMMUNICATION-CONTROLLER-VARIANTS>
            <FLEXRAY-COMMUNICATION-CONTROLLER-CONDITIONAL>
              <ACCEPTED-STARTUP-RANGE>1875</ACCEPTED-STARTUP-RANGE>
              <WAKE-UP-PATTERN>5</WAKE-UP-PATTERN>
            </FLEXRAY-COMMUNICATION-CONTROLLER-CONDITIONAL>
            <FLEXRAY-COMMUNICATION-CONTROLLER-CONDITIONAL>
              <MICRO-PER-CYCLE>320000</MICRO-PER-CYCLE>
              <MICROTICK-DURATION>50E-9</MICROTICK-DURATION>
              <SAMPLES-PER-MICROTICK>4</SAMPLES-PER-MICROTICK>
              <VARIATION-POINT>
                <SW-SYSCOND
                  BINDING-TIME="CODE-GENERATION-TIME">
                      <SYSC-REF
                  DEST="SW-SYSTEMCONST">/AggregationPattern/CarType</
                      SYSC-REF>
                        == 1
                    </SW-SYSCOND>
              </VARIATION-POINT>
            </FLEXRAY-COMMUNICATION-CONTROLLER-CONDITIONAL>
            <FLEXRAY-COMMUNICATION-CONTROLLER-CONDITIONAL>
              <MICRO-PER-CYCLE>80000</MICRO-PER-CYCLE>
              <MICROTICK-DURATION>12.5E-9</MICROTICK-DURATION>
              <SAMPLES-PER-MICROTICK>1</SAMPLES-PER-MICROTICK>
              <VARIATION-POINT>
                <SW-SYSCOND
                  BINDING-TIME="CODE-GENERATION-TIME">
                      <SYSC-REF
```

```
                    DEST="SW-SYSTEMCONST">/AggregationPattern/CarType</
                        SYSC-REF>
                            == 2
                          </SW-SYSCOND>
                      </VARIATION-POINT>
                    </FLEXRAY-COMMUNICATION-CONTROLLER-CONDITIONAL>
                  </FLEXRAY-COMMUNICATION-CONTROLLER-VARIANTS>
                </FLEXRAY-COMMUNICATION-CONTROLLER>
              </COMM-CONTROLLERS>
            </ECU-INSTANCE>
          </ELEMENTS>
        </AR-PACKAGE>
```

At the heart of example 7.10 is the following construct, which defines three sets of variant attributes for the `FlexrayCommunicationController`.

```
<FLEXRAY-COMMUNICATION-CONTROLLER>
  <SHORT-NAME>iFlex</SHORT-NAME>
  <FLEXRAY-COMMUNICATION-CONTROLLER-VARIANTS>
    <FLEXRAY-COMMUNICATION-CONTROLLER-CONDITIONAL>
      ...
    </FLEXRAY-COMMUNICATION-CONTROLLER-CONDITIONAL>
    <FLEXRAY-COMMUNICATION-CONTROLLER-CONDITIONAL>
      ...
      <VARIATION-POINT>
        <SW-SYSCOND BINDING-TIME="CODE-GENERATION-TIME">
          <SYSC-REF DEST="SW-SYSTEMCONST">/AggregationPattern/CarType</SYSC
              -REF>
            == 1
        </SW-SYSCOND>
      </VARIATION-POINT>
    </FLEXRAY-COMMUNICATION-CONTROLLER-CONDITIONAL>
    <FLEXRAY-COMMUNICATION-CONTROLLER-CONDITIONAL>
      ...
      <VARIATION-POINT>
        <SW-SYSCOND BINDING-TIME="CODE-GENERATION-TIME">
          <SYSC-REF DEST="SW-SYSTEMCONST">/AggregationPattern/CarType</SYSC
              -REF>
            == 2
        </SW-SYSCOND>
      </VARIATION-POINT>
    </FLEXRAY-COMMUNICATION-CONTROLLER-CONDITIONAL>
  </FLEXRAY-COMMUNICATION-CONTROLLER-VARIANTS>
</FLEXRAY-COMMUNICATION-CONTROLLER>
```

As one can see, `FLEXRAY-COMMUNICATION-CONTROLLER` contains a wrapper element `FLEXRAY-COMMUNICATION-CONTROLLER-VARIANTS`, and a `FLEXRAY-COMMUNICATION-CONTROLLER-CONDITIONAL` for each variation:

- The first `CONDITIONAL` holds the non-variant attributes (`acceptedStartupRange` to `wakeUpPattern`). Note that this `CONDITIONAL` does not contain a variation point.
- The second `CONDITIONAL` holds the first set of variant attributes, namely those for *CarType* 1.

- The third `CONDITIONAL` holds the first set of variant attributes, namely those for *CarType* 2.

8 Splitable Elements

8.1 Introduction

From the perspective of AUTOSAR tools working in the context of an entire ECU project, all meta data belonging to this project resides in a single big data model according to the meta-model standardized by AUTOSAR. The natural choice for making this data model persistent on disk would be to store it in a single AUTOSAR XML (arxml) file. While this might still be feasible for tiny or demonstrator projects, those files get prohibitively huge in real-world ECU projects. Additionally, parallel work on different aspects of this data model by different developers leads to substantial concurrency problems. There are even further reasons for splitting the model into several files which are explained below.

Consequently, the ECU project's AUTOSAR data model has to be split into better manageable fragments. There is not one single, self-evident dividing line through this data model, but a whole bunch of different motivations for splitting out model fragments into dedicated files:

- Various technical aspects require dedicated competencies, e.g. interface design vs. documentation
- Some kinds of information are required earlier in the development process than other kinds of information, e.g. basic data types shall be defined before sender/receiver interfaces can be instantiated
- The properties of some more complex model entities are stepwise refined by different developers with different scope; e.g. an application software developer defines the name and size of an NVRAM Manager (NVM) block for persistent storage, an NVM expert assigns the more elusive block properties, and finally a project integrator defines the precise storage location of this NVM block
- There are model items which can be automatically derived from other model items (e.g. configuration IDs); those items are often created by machines instead of humans
- Model fragments with a high update rate shall be separated from model fragments with a low update rate to minimize the effort for software configuration and version management or diff/merge operations
- Work split and IP protection between different organizations e.g. OEM and Tier1
- Documentation-related aspects might be subject to translation but the translation process shall never change the technical properties of the documented entity
- Additional general organizational or location-related constraints may apply (OEM)

This all boils down to a rather simple rule: whenever different developers work on a particular artifact or if different triggers exist for contributing to this artifact, then splitting this artifact into separate files should be considered.

The processes and working strategies in cooperative software development highly depend on the chosen business cases.

The following example describes a collaboration scenario between an OEM and a supplier. Detailed ARXML files for the example can be found in the appendix F.2.

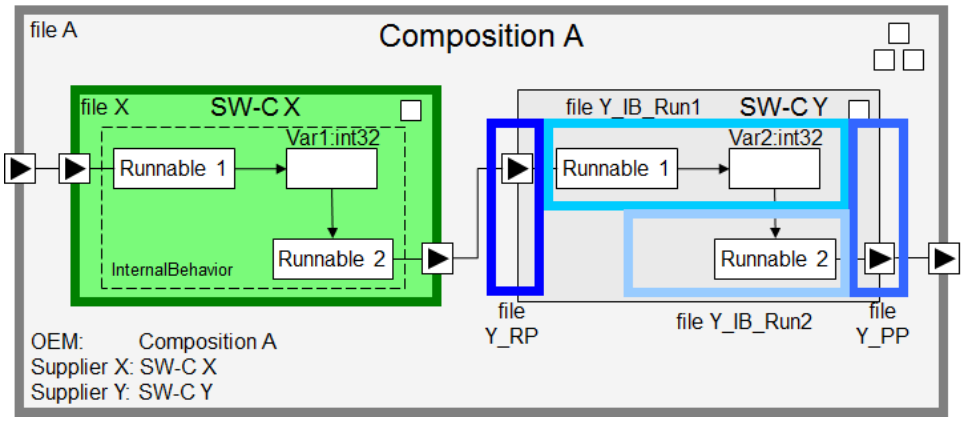


Figure 8.1: Composition assembled from several components using splitable elements

The OEM provides the Composition A including a require port (left side) and a provide port (right side). Initially this Composition A does not contain any Atomic Software Components and will be persisted in one ARXML file, see listing F.5.

Additionally, the OEM requires the integration of a predefined Atomic Software Component (SW-C X) by supplier X e.g. to include a brand-specific functionality. Also the Atomic Software Component (SW-C X) will be persisted in a dedicated ARXML file, see listing F.6.

To complete the functionality of the Composition A, the supplier Y has to develop and integrate the Atomic Software Component SW-C Y, see listing F.7.

The splitting feature allows to adapt the implementation in an optimized way to the existing development process. The ARPackage SW-C Y is split e.g. in separate port definitions. They ensure the interaction with the Composition A and the Atomic Software Component (SW-C X), see listing F.8 and listing F.9.

Another motivation for splitting could be to add further aspects to an element without changing the original artifact. In case the Internal Behavior is developed in two steps, a first step could define e.g. a rough calculation (Runnable 1) and a later step in the development process refines this calculation by a second calculation (Runnable 2), two splitable artifacts are useful (see listing F.10 and listing F.11).

The following chapters provide all relevant information how splitable elements are modelled on M2 level (see chapter 8.2) and how they are applied in different scenarios, M1 level (see chapter 8.3). Appendix F.2 gives detailed information for M1 users (ARXML authors) based on the application example described above.

8.2 Distribution in Multiple Physical Files (M2 level)

AUTOSAR distinguishes between elements in the model that may be split up over several physical files, and elements that need to be defined together and therefore shall be described completely in exactly one physical file. By default, all properties of an element shall be in the same physical file (see [constr_2524]). Properties that have been decorated with the stereotype ≪atpSplitable≫ are called splitable elements. Properties (aggregations, references and primitive attributes) that have not been decorated with the stereotype ≪atpSplitable≫ are called non-splitable elements.

By introducing splitable elements, AUTOSAR supports

- flexible methodology (e.g. optimizing processes by specific distribution of Information to physical artifacts)
- to add further aspects to an element without changing the original artifact

In the following the term partial model is understood as defined in [TPS_GST_00423].

[TPS_GST_00423] AUTOSAR Partial Model ⌈The AUTOSAR XML description can consist of several files. Each individual file represents exactly one AUTOSAR partial model. One partial model is represented in an AUTOSAR XML description by one file. The partial model does not need to fulfill all semantic constraints applicable to an AUTOSAR model, but it shall validate successfully against the AUTOSAR XML schema.⌋*()*

[TPS_GST_00414] Modeling of splitable elements ⌈For each individual property (attribute, aggregation and reference), the meta-model can specify by application of the stereotype ≪atpSplitable≫ that and how a concrete AUTOSAR model can be distributed over separate files.⌋*()*

[constr_2502] Merged model shall be compliant to the meta-model ⌈A model merged from ≪atpSplitable≫ elements shall adhere to the consistency rules of the *meta-model*. Note that the required lower multiplicities depend on the process phase therefore the AUTOSAR schema sets them mainly to 0. This also applies to the bound model.⌋*()*

The semantics of ≪atpSplitable≫ is defined in the following rules:

[constr_2524] Non splitable elements in one file ⌈If the *aggregation*/*attribute* is **not** ≪atpSplitable≫, then all aggregated element(s) shall be described in the same physical file as the aggregating element.⌋*()*

[TPS_GST_00415] Splitkey ⌈Each ≪atpSplitable≫ is characterized by a splitkey. The identity of a split element is given by the splitkey path up to the root.⌋*()*

Note: The splitkey shall follow the [TPS_GST_00021]. That means the splitkey is case sensitive.

[TPS_GST_00418] Definition Splitkey Path ⌈The splitkey path is the combination of the splitkey values defined in [TPS_GST_00047] cross all nesting levels, starting point of the splitkey path is always the root package.⌋*()*

In contrast to the AUTOSAR path the splitkey path identifies elements in the variant-rich model. This applies also to non-referable elements.

An example of an aggregation chain is illustrated in figure 8.2.

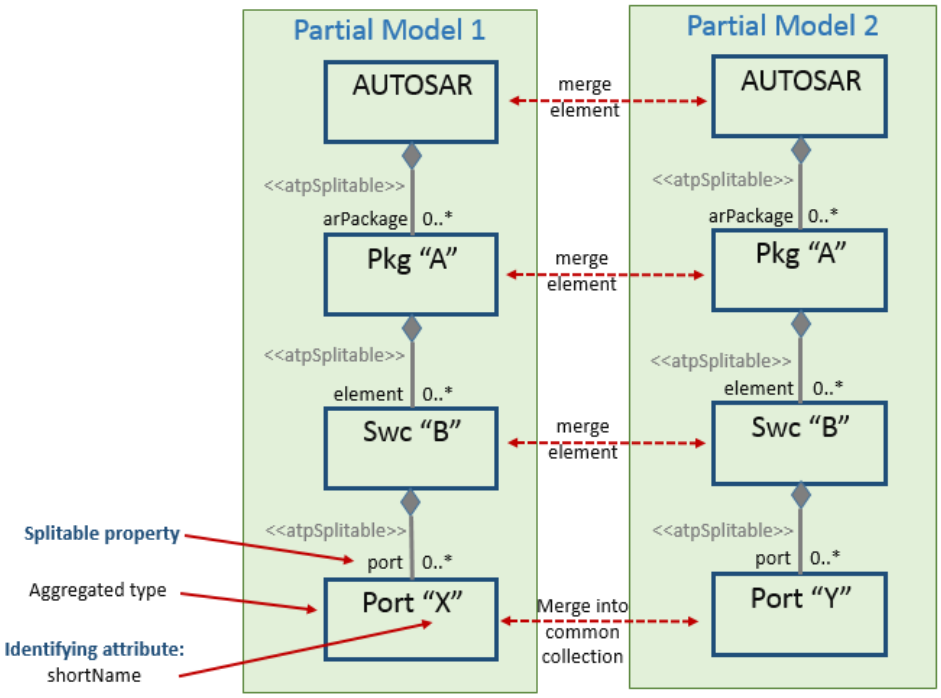


Figure 8.2: Aggregation Chain and Identities of Elements

The simple example in figure 8.2 does not have `VariationPoint`s therefore the `variationPoint.shortLabel` does not appear in the splitkey path. The splitkey paths are ("AUTOSAR", "A", "B", "X") and ("AUTOSAR", "A", "B", "Y"). The last part of the splitkey paths is not equal, therefore this ends up in two elements "X" and "Y" in package /AUTOSAR/A/B.

[TPS_GST_00416] Rules for the definition of a splitkey at M2 level ⌈The splitkey of a splitable property shall consist of the following parts:

- The role name of the splitable property (i.e. an attribute, association or aggregation).
- The role names of the properties of the aggregated type which are stereotyped by ≪`atpIdentityContributor`≫, in particular:
  - `shortName`, if the aggregated type is a `Referrable`,
  - `shortLabel`, `ident.shortName`, or other special attributes which contribute to the identity,
  - role name of the definition ref, if the aggregated type is subject to the definition-ref pattern,

- role name of all sub-references, if the aggregated type is subject to the instance-ref pattern.
- `variationPoint.shortLabel`, if the splitable property is stereotyped with ≪`atpVariation`≫. This applies to aggregations and associations, but not to `AttributeValueVariationPoint`s.
- If the aggregated type is an abstract class, then the role names of the ≪`atpIdentityContributor`≫ properties of the derived subclasses shall also be added to the splitkey.

⌋()

The ≪`atpIdentityContributor`≫ properties of an M2 class (second bullet in [TPS_GST_00416]) have to be decided case by case, depending on the use case of a splitable element.

Note, that most ≪`atpIdentityContributor`≫ properties contribute to the name-identity, while the `definition` reference of an ECU-C value contributes to the type-identity. Both parts of the identity, name and type, are relevant when merging splitable elements, see [constr_2630] and [TPS_GST_00047]. The splitkey is never empty, because it contains the role name of the splitable property, which always exists.

See [TPS_GST_00382] for the interaction of ≪`atpStructuredComment`≫ and ≪`atpSplitable`≫, e.g. the handling of `FileInfoComment`s in partial models.

[TPS_GST_00048] Splitable up to the Root ⌈

- If an element contains properties marked as ≪`atpSplitable`≫ then all aggregations up to the root element shall be marked as ≪`atpSplitable`≫.
- If a type aggregates splitable elements, it shall have attributes which define the identity according to the definition in [TPS_GST_00416]. This is required for unambiguous identification of splitable elements in partial models. Splitable elements on deeper levels will only be merged if the identities match up to the root.

⌋()

[TPS_GST_00046] Splitable collections ⌈If a ≪`atpSplitable`≫ aggregation is of upper multiplicity > 1 it represents a collection of elements. This collection then may be split to different files. The partial models represented by these different files still provide collections contributing to the merged model. Therefore, according to [3], the wrapper xml element exist in each individual file as well.⌋()

[TPS_GST_00383] Ordered collections ⌈Ordered collections (relations with upper multiplicity > 1) are collections whose order of elements is semantically meaningful. They are flagged as *ordered* according to the UML specification [21].⌋()

Note that ≪`atpSplitable`≫ on ordered collections indicates that the entire collection shall be in a partial model.

[constr_2547] Ordered collections cannot be split into different partial models ⌈Ordered collections cannot be split. In other words: Contrary to the semantics of unordered collections - which can be distributed between partial models - ordered collections can only be placed as a whole in one of the partial models. Otherwise the merge approach would influence the semantics of the collections.⌋ *()*

[constr_2525] Non splitable elements shall not be repeated ⌈Properties (namely aggregations, references and primitive attributes) which are not marked as ≪atpSplitable≫ shall be placed in one physical file. They shall not be repeated in the split files unless they are an attribute which is used as a part of the split key. Another special case is handling of ≪atpStructuredComment≫, see [TPS_GST_00382].⌋ *()*

Note, that AUTOSAR tools should resolve conflicts with [constr_2525] by accepting multiple defined ARElements as long as their non-splitable properties are the same.

8.3 Merging of Splitable Elements from Partial Models (M1 level)

For combining several partial models it is necessary to define the criteria for elements to be merged. Different criteria are relevant, depending on the kind of the ≪atpSplitable≫ (aggregation, association, or primitive attribute). [TPS_GST_00047] lists the criteria for the identification of elements in a partial model.

- **[TPS_GST_00047] Identification of M1 elements in partial models** ⌈The identification (name and type) of M1 elements is based on the properties stereotyped with ≪atpIdentityContributor≫, which are collected in the splitkey.
  - If the splitkey contains the shortName, add the value of the shortName attribute to the name-identity.
  - If the splitkey contains a special attribute, e.g. shortLabel or ident.shortName, add the value of this attribute to the name-identity.
  - If the splitkey contains a definition reference, add the value of this attribute to the type-identity.
  - If the splitkey contains the role name of a reference property, add the full AUTOSAR path of the referenced M1 element to the name-identity.
  - If the splitkey contains the role name of an instance reference, add the full AUTOSAR path of all referenced M1 elements to the name-identity.
  - If the splitkey contains variationPoint.shortLabel, add the value of the shortLabel to the name-identity.

  ⌋ *()*

  Note, that there still may be M1 elements without identity, e.g. SwcInternalBehavior.includedDataTypeSet. The handling of elements which do not have

a defined identity depends on the multiplicities defined in the meta-model, see [TPS_GST_00417].

For more details refer to the sub sections F.1.1 (variation), F.2.2 (aggregation), F.2.3 (association) and F.2.4 (ECU parameter).

[constr_2630] M1 elements with same identity but different type are not allowed ⌈Splitable M1 elements with the same identity but different type shall not exist.⌋*()*

[constr_2629] Defined identity up to the root ⌈If an element in the M1 model aggregates splitable elements on deeper levels, it shall have a defined identity, i.e. the identifying attributes (e.g. `shortName` or `shortLabel`); see [TPS_GST_00047]; shall be set in the M1 model.⌋*()*

[TPS_GST_00420] Matching identities up to the root ⌈Merging of splitable elements is only applied if the identities of all parents on the aggregation path match up to the root.⌋*()*

- **[TPS_GST_00417] Merging of splitable elements from partial models** ⌈The following rules apply for the merging of splitable elements from partial models:
  - If the upper multiplicity of a splitable property in the variant-rich model is > 1, then M1 elements with the same identity according to [TPS_GST_00047] are merged. M1 Elements with different identity or without identity are not merged, i.e. they are just copied to the output collection.
  - If the upper multiplicity of a splitable property in the variant-rich model is 1, then M1 elements with different identity lead to a multiplicity conflict. M1 elements with the same identity according to [TPS_GST_00047] or without identity are merged.

  After binding the variation, the multiplicities shall match the multiplicities of the pure meta-model.⌋*()*

The [TPS_GST_00417] demands for an illustration which is given in figure 8.3. The partial models 1 and 2 contain different scenarios. For the illustration, the elements are arranged in a matrix where the rows represent the identity (without variation) and the columns represent the different `VariationPoint`s.

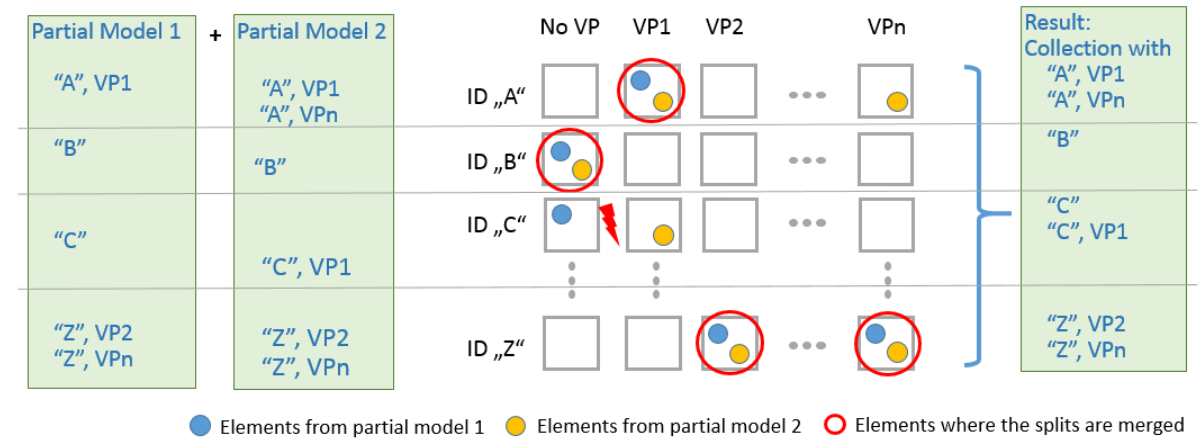


Figure 8.3: Type with identity and variation, aggregated with multiplicity > 1

Only elements with the same identity and `variationPoint.shortLabel` are merged. This is the case with ("A", VP1), ("Z", VP2), and ("Z", VPn). A corresponding example is given in F.1.1. In case of identity "B" the merge bases upon the `shortName`. The case of identity "C" illustrates a clash because a `VariationPoint` shall be defined either for all splitable elements or for none. This is a consequence of [constr_2503]. Otherwise there would be two elements with the same identity after binding the variants, e.g. if the `VariationPoint` VP1 is selected.

9 Documentation Support

9.1 Introduction

The AUTOSAR meta-model and XML schema are based upon ASAM FSX[22]. They provide support for well-structured, content-focused and integrated documentation.

The documentation supported by AUTOSAR is "well structured" in the sense that the building blocks used to build the documentation delimit the beginning and the end of each construct, the constructs are context-neutral and can be cut and pasted wherever the construct is legal[1].

The author of an AUTOSAR documentation does not need to focus on formatting of the text. Instead, he/she can focus on the content, structuring this content in a hierarchy of chapters. He can use figures, lists, tables, and formula to express his though clearly, even making notes[2] to stress some points. However, he does not specify the appearance of these elements. He lets a tool format all of them according to a consistent style (e.g. corporate identity).

One advantages of such a well-structured and content-focused documentation is that it can be processed and transformed easily, thus supporting operation like merging different sources to produce a comprehensive documentation from multiple components' documentation. Different parts of the content can be annotated with their intended audience, so that it is possible to generate audience-specific documents from the same composite source.

The documentation can be integrated within an M1-level artifact (e.g., a SWC-T instance) in the following levels:

- **[TPS_GST_00305] Single Paragraph** ⌈A single paragraph in the role `desc` provides a short description of an identifiable object to help a human being identify the object. It can be inserted in any `Identifiable` element for description of identifiable).⌋*()* See Chapter 9.2.1 and Section 4.3 for more details.
- **[TPS_GST_00306] Documentation Block** ⌈A documentation block is available in any `Identifiable` element as `introduction`. This type of documentation is typically used to capture a short introduction about the role of an element or respectively how it is built.⌋*()* See Chapter 9.2 and Section 4.3 for more details.
- **[TPS_GST_00307] Standalone Documentation** ⌈A standalone documentation structured into multiple chapters is also offered in AUTOSAR. It supports references to particular model elements. This type of documentation is suited to describe the interactions among multiple elements and to provide a complete overview.⌋*()* See Section 9.3 for details.

[1] The construct chapter, for example, can be nested into any other chapter becoming a sub-chapter or a sub-sub-chapter. The advantage becomes clear when one tries to restructure a LaTeX report moving chapters, sections, and subsections at different level in the document hierarchy.

[2] Examples of notes include caution, hints, exercises.

Figure 9.1 illustrates[3] the macroscopic text model of AUTOSAR, in particular the available entry points on which documentation is integrated in particular AUTOSAR meta-model elements. DocumentationContainer is not a real meta-model element. It is used in this diagram to illustrate any AUTOSAR meta-model element which may contain documentation.

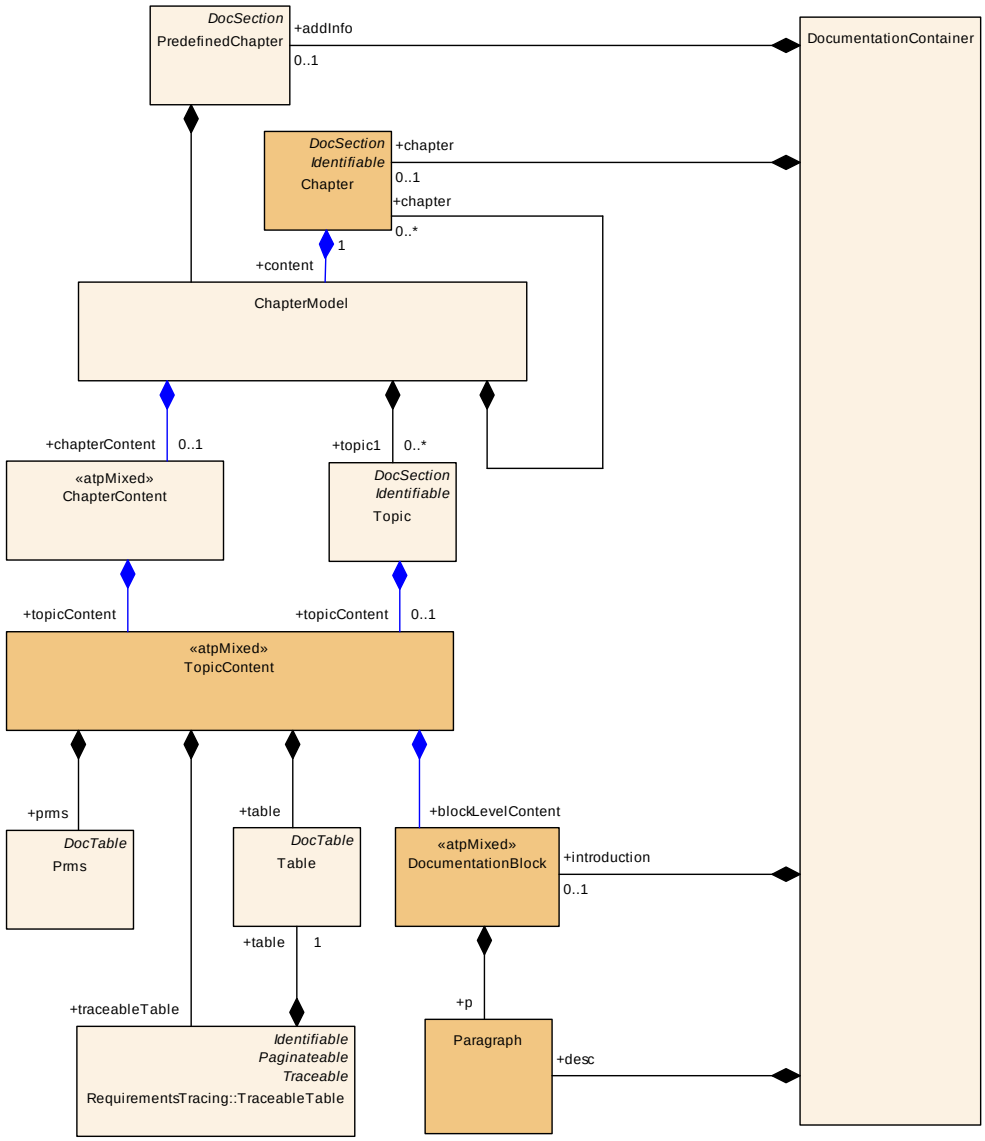


Figure 9.1: Macroscopic text model (simplified)

In figure 9.1, each of the three levels extends the mechanisms offered by the previous one. The documentation block used in the `introduction` contains paragraphs used in `desc`. The standalone documentation are build around chapters containing documentation block as well as generic tables `table`, `traceableTable`s and parameter tables `prms`. There are two types of chapters: the `Chapter`, which have a title and the predefined chapter `PredefinedChapter`, which name is given by their context[4].

[3] note that this is a simplified illustration of the real model

[4] In the software component template, for example, a component has an optional set of predefined documentation aspects (e.g., swFeatureDesc, swTestDesc, swCalibrationNotes)

[TPS_GST_00308] Purpose of `Chapter` ⌈A chapter addresses one or more `topicContent`s. It may be organized into tables and documentation blocks followed by sub chapters or topics.⌋*()*

[TPS_GST_00309] Purpose of `Topic1` ⌈A Topic is a logical unit, which is given explicit boundaries and a label concisely capturing its content. Usually a topic does not get a number nor an entry in table of contents of e.g. printed document.⌋*()*

[TPS_GST_00310] Synopsis of Chapters and Topics ⌈Chapters and topics have basically the same content. However, a chapter is typically formatted differently, may offer additional representation means and will probably be extended independently of the topic in the future. Therefore, `ChapterContent` contains the `TopicContent`.⌋*()*

[TPS_GST_00419] Synopsis of `TopicContent` ⌈The `TopicContent` consists of objects which can be rendered as direct content of a `Chapter` resp. a Topic. In contrast to a DocumentationBlock, `TopicContent` cannot appear in `table`s and `list`s.⌋*()*

Please note that the various names (e.g., `prms`, `desc`, `introduction`) are used to remain consistent with [22].

The remaining sections of this chapter describe the following aspects of AUTOSAR documentation:

- Section 9.2.1 describes the mechanisms used within a single paragraph.
- Section 9.2 describes the documentation features supported by the `introduction` and the classes used to implement them.
- Section 9.3 describes the exhaustive documentation features used to produce a standalone documentation.
- Section 9.6 describes the multi-language support offered by AUTOSAR documentation.

9.2 Documentation Block

[TPS_GST_00311] `DocumentationBlock` fits in a table cell ⌈The DocumentationBlock is limited such that it can be represented in a table cell and therefore can not contain tabular elements. It contains the following basic documentation elements:

- **paragraph**: The construct `p` identifies the text organized as a paragraph.
- **List**: A `list` is a sequence of items. In AUTOSAR, the concept of list implemented using different keywords.
- **Figure**: The `figure` is the construct used to insert a diagram in a document. It is composed of a `figureCaption` and the diagram itself. Depending on the

tool used, different diagram format are supported (example of graphical format include SVG (Scalable Vector Graphics), JPEG (Joint Photographic Experts Group) and GIF(Graphics Interchange Format)).

- **Formula**: The construct `formula` is used to represent a mathematical expression. Different kinds of specification of the formula are possible (for example the TEXmath format).
- **Verbatim**: `verbatim` represents a block in which whitespace (in particular blanks and line feeds) are kept "as they are". This enables basic formatting to be carried out, which can even be displayed on simple devices. Verbatim is often used to represent source code or xml in books.
- **note**: The construct `note` is used to express an annotation. Examples of annotation types include hint, caution, tip, instruction, and exercise.
- **trace**: The construct `trace` is used to manage tracing between items in documents. See [2] for more details.
- **structuredReq**: The construct `structuredReq` this is used for requirements documents. See [2] for more details.

⌋() See figure 9.2 for more details.

Class	<<atpMixed>> **DocumentationBlock**			
Package	M2::MSR::Documentation::BlockElements			
Note	This class represents a documentation block. It is made of basic text structure elements which can be displayed in a table cell.			
Base	*ARObject*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
defList	DefList	0..1	aggr	This represents a definition list in the documentation block. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=postBuild xml.sequenceOffset=40
figure	MIFigure	0..1	aggr	This represents a figure in the documentation block. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=postBuild xml.sequenceOffset=70
formula	MIFormula	0..1	aggr	This is a formula in the definition block. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=postBuild xml.sequenceOffset=60
labeledList	LabeledList	0..1	aggr	This represents a labeled list. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=postBuild xml.sequenceOffset=50

▽

△

Class	<<atpMixed>> **DocumentationBlock**			
list	List	0..1	aggr	This represents numbered or unnumbered list. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=postBuild xml.sequenceOffset=30
msrQueryP2	MsrQueryP2	0..1	aggr	This represents automatically contributed contents provided by an msrquery in the context of Documentation Block.
note	Note	0..1	aggr	This represents a note in the text flow. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=postBuild xml.sequenceOffset=80
p	MultiLanguage Paragraph	0..1	aggr	This is one particular paragraph. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=postBuild xml.sequenceOffset=10
structuredReq	StructuredReq	0..1	aggr	This aggregation supports structured requirements embedded in a documentation block. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=postBuild xml.sequenceOffset=100
trace	TraceableText	0..1	aggr	This represents traceable text in the documentation block. This allows to specify requirements/constraints in any documentation block. The kind of the trace is specified in the category. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=postBuild xml.sequenceOffset=90
verbatim	MultiLanguageVerbatim	0..1	aggr	This represents one particular verbatim text. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=postBuild xml.sequenceOffset=20

Table 9.1: DocumentationBlock

Figure 9.2 illustrates the constructs contained in a documentation block and the constructs that contains a documentation block.

[TPS_GST_00312] Variation in Documentation ⌈The elements in `DocumentationBlock` are aggregated as ≪`atpVariation`≫ with a latest binding time set to `PostBuild`. This indicates that also post build variation can be documented properly.⌋*()* See Chapter 7.6.4 for details.

A documentation part with binding time `PostBuild` can not be bound at startup of an ECU. This part describes functionality which depends on `PostBuild` variability.

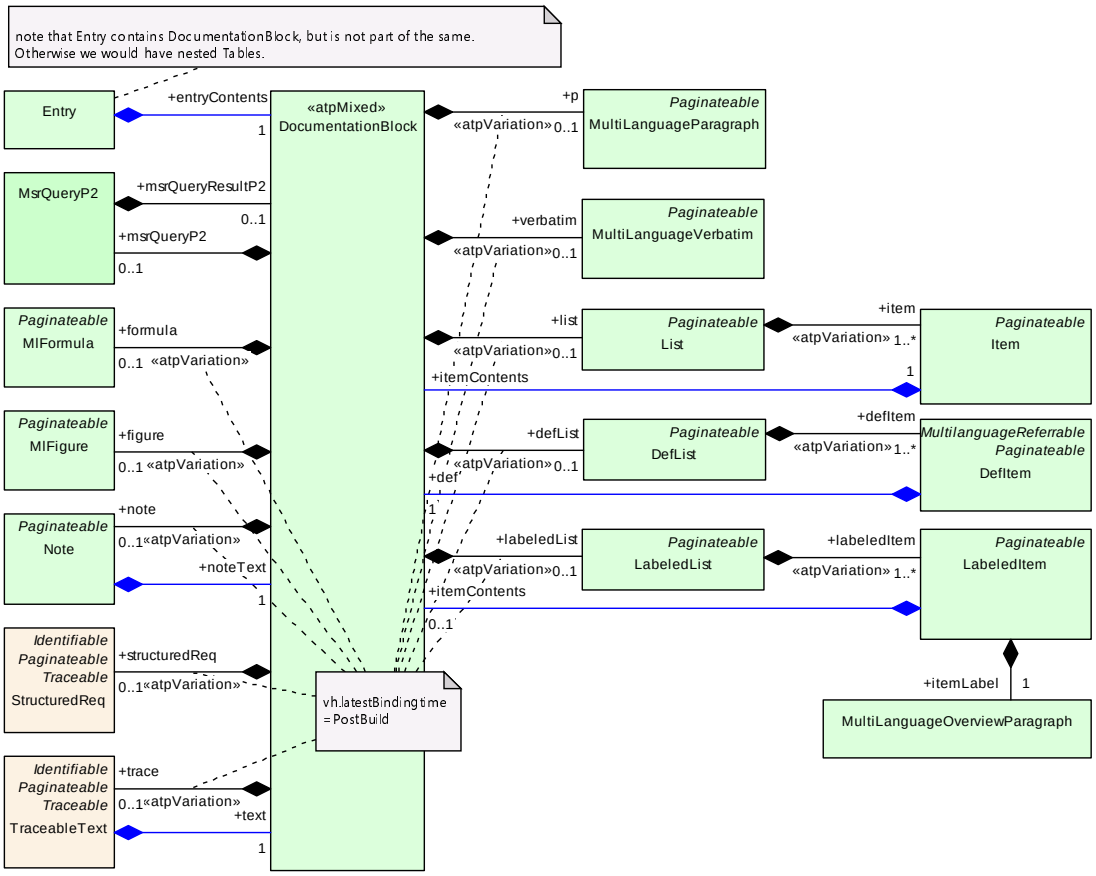


Figure 9.2: documentation block overview

9.2.1 Paragraph

[TPS_GST_00313] Types of Paragraph ⌈There are two different types of paragraphs in AUTOSAR. They differ only in the elements, which they may contain:

- the overview paragraph (`LOverviewParagraph`) is used as a compact descriptive paragraph `desc`. It therefore cannot embed Identifiables which might be rendered as structure (such as `Xdoc` etc.).
- the regular paragraph (`LParagraph`) (used as p) additionally provides elements which might be rendered as structures such as meta data of external files, document and standards.

⌋()

[constr_2595] Footnotes should not be nested ⌈Note that even if supported by the meta-model, footnotes shall not be nested. Nested footnotes might lead to problems with the processing of the footnote link. In other words `LParagraph` shall not be aggregated with role `ft` within a `LParagraph` which already has the role `ft`.⌋()[5]

[5]The L attribute of the ft class was introduced in R4.2.2. This unintentionally led to backwards incompatibility. To correct this the classes `SlParagraph` and `SlOverviewParagraph` are introduced

Note: The value of attribute L within ft should be consistent to the attribute L of the container of ft.

Class	<<atpMixedString>> ***MixedContentForParagraph*** (abstract)			
Package	M2::MSR::Documentation::TextModel::InlineTextModel			
Note	This mainly represents the text model of a full blown paragraph within a documentation.			
Base	*ARObject*			
Subclasses	LParagraph, SlParagraph			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
br	Br	1	aggr	This element is the same as function here as in a HTML document i.e. it forces a line break. **Tags:**xml.sequenceOffset=40
e	EmphasisText	1	aggr	This is emphasized text. **Tags:**xml.sequenceOffset=70
ft	SlParagraph	1	aggr	This is a foot note within a paragraph.
ie	IndexEntry	1	aggr	This is an index entry. **Tags:**xml.sequenceOffset=110
std	Std	1	aggr	This is a refeernce to a standard. **Tags:**xml.sequenceOffset=120
sub	Superscript	1	attr	This is subscript text. **Tags:**xml.sequenceOffset=100
sup	Superscript	1	attr	This is superscript text. **Tags:**xml.sequenceOffset=90
trace	Traceable	1	ref	This allows to place an arbitrary reference to a traceable object in documentation.
tt	Tt	1	aggr	This is a technical term. **Tags:**xml.sequenceOffset=30
xdoc	Xdoc	1	aggr	This is a reference to a printable external document. **Tags:**xml.sequenceOffset=130
xfile	Xfile	1	aggr	This represents a reference to an external file which usually cannot be printed. **Tags:**xml.sequenceOffset=140
xref	Xref	1	aggr	This is a cross reference. **Tags:**xml.sequenceOffset=50
xrefTarget	XrefTarget	1	aggr	This element specifies a reference target which can be scattered throughout the text. **Tags:**xml.sequenceOffset=60

Table 9.2: MixedContentForParagraph

Class	<<atpMixedString>> ***MixedContentForOverviewParagraph*** (abstract)
Package	M2::MSR::Documentation::TextModel::InlineTextModel

▽

as vehicle to overcome the incompatibility. Both classes own a L attribute. This attribute restores the backwards compatibility to R4.1 (where the attribute was not available) while maintaining backwards compatibility to 4.2.2 (where the attribute was required). The value of this attribute shall be ignored in any case.

△

Class	<<atpMixedString>> ***MixedContentForOverviewParagraph*** (abstract)			
Note	This is the text model of a restricted paragraph item within a documentation. Such restricted paragraphs are used mainly for overview items, e.g. desc.			
Base	*ARObject*			
Subclasses	LOverviewParagraph, SlOverviewParagraph			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
br	Br	1	aggr	This element is the same as function here as in a HTML document i.e. it forces a line break.
e	EmphasisText	1	aggr	This is emphasis text. **Tags:**xml.sequenceOffset=60
ft	SlOverviewParagraph	1	aggr	This is a foot note within a paragraph.
ie	IndexEntry	1	aggr	This is an index entry. **Tags:**xml.sequenceOffset=100
sub	Superscript	1	attr	This is superscript text. **Tags:**xml.sequenceOffset=90
sup	Superscript	1	attr	This is subscript text. **Tags:**xml.sequenceOffset=80
trace	Traceable	1	ref	This allows to place an arbitrary reference to a traceable object in documentation.
tt	Tt	1	aggr	This is a technical term. **Tags:**xml.sequenceOffset=30
xref	Xref	1	aggr	This is a cross reference. **Tags:**xml.sequenceOffset=40
xrefTarget	XrefTarget	1	aggr	This element specifies a reference target which can be scattered throughout the text. **Tags:**xml.sequenceOffset=50

Table 9.3: MixedContentForOverviewParagraph

Class	**MultiLanguageParagraph**			
Package	M2::MSR::Documentation::TextModel::MultilanguageData			
Note	This is the content model of a multilingual paragraph in a documentation.			
Base	*ARObject*, *DocumentViewSelectable*, *Paginateable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
helpEntry	String	0..1	attr	This specifies an entry point in an online help system to be linked with the parent class. The syntax shall be defined by the applied help system respectively help system generator. **Tags:**xml.attribute=true
l1	LParagraph	1..*	aggr	This is the paragraph content in one partiucular language. **Tags:** xml.roleElement=true xml.roleWrapperElement=false xml.sequenceOffset=20 xml.typeElement=false xml.typeWrapperElement=false

Table 9.4: MultiLanguageParagraph

see also `MixedContentForParagraph`

9.2.2 Verbatim

[TPS_GST_00314] Purpose of Verbatim ⌈Verbatim (represented by `MultiLanguageVerbatim` respectively `MixedContentForVerbatim`) marks the text as "preformatted" – all the spaces and carriage returns are rendered exactly as they are typed be the user respectively appear in the model. Such content should also be rendered using a mono spaced font.⌋*()*

[TPS_GST_00315] Rendering of inline elements of Verbatim ⌈Even if the content needs to be rendered verbatim, the model still allows inline elements. It is expected the number of characters are not changed when those elements are rendered.⌋*()*

Class	**MultiLanguageVerbatim**			
Package	M2::MSR::Documentation::TextModel::MultilanguageData			
Note	This class represents multilingual Verbatim. Verbatim means, that white-space is maintained. When Verbatim is rendered in PDF or Online media, white-space is obeyed. Blanks are rendered as well as newline characters.			
Base	*ARObject*, *DocumentViewSelectable*, *Paginateable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
allowBreak	NameToken	0..1	attr	This indicates if the verbatim text might be split on multiple pages. Default is "1". **Tags:**xml.attribute=true
float	FloatEnum	0..1	attr	Indicate whether it is allowed to break the element. The following values are allowed: **Tags:**xml.attribute=true
helpEntry	String	0..1	attr	This specifies an entry point in an online help system to be linked with the parent class. The syntax shall be defined by the applied help system respectively help system generator. **Tags:**xml.attribute=true
l5	LVerbatim	1..*	aggr	This the text in one particular language. **Tags:** xml.roleElement=true xml.roleWrapperElement=false xml.sequenceOffset=20 xml.typeElement=false xml.typeWrapperElement=false
pgwide	PgwideEnum	0..1	attr	Used to indicate wether the figure should take the complete page width (value = "pgwide") or not (value = "noPgwide"). **Tags:**xml.attribute=true

Table 9.5: MultiLanguageVerbatim

Class	<<atpMixedString>> ***MixedContentForVerbatim*** (abstract)
Package	M2::MSR::Documentation::TextModel::InlineTextModel

▽

△

Class	<<atpMixedString>> ***MixedContentForVerbatim*** (abstract)			
Note	This is the text model for preformatted (verbatim) text. It mainly consists of attributes which do not change the length on rendering. This class represents multilingual verbatim. Verbatim, sometimes called preformatted text, means that white-space is maintained. When verbatim is rendered in PDF or Online media, it is rendered using a monospaced font while white-space is obeyed. Blanks are rendered as well as newline characters. Even if there are inline elements, the length of the data shall not be influenced by formatting.			
Base	*ARObject*, *WhitespaceControlled*			
Subclasses	LVerbatim			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
br	Br	1	aggr	This element is the same as function here as in a HTML document i.e. it forces a line break. **Tags:**xml.sequenceOffset=50
e	EmphasisText	1	aggr	This is emphsized text. Note that in verbatim, the attribute font should not be considered since verbatim is always rendered as monospace font. **Tags:**xml.sequenceOffset=30
tt	Tt	1	aggr	This represents a technical term in verbatim. Note that it's the responibility of the user not to take a tt that would add additional character to the text (such as SgmlElement).
xref	Xref	1	aggr	This is a crossreference within a verbatim text. The attributes may disturb the arrangement of the text. It is subject to the author to keep this under control. **Tags:**xml.sequenceOffset=40

Table 9.6: MixedContentForVerbatim

Class	***WhitespaceControlled*** (abstract)			
Package	M2::MSR::Documentation::TextModel::LanguageDataModel			
Note	This meta-class represents the ability to control the white-space handling e.g. in xml serialization. This is implemented by adding the attribute "space".			
Base	*ARObject*			
Subclasses	*MixedContentForPlainText*, *MixedContentForVerbatim*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
xmlSpace	XmlSpaceEnum	1	attr	This attribute is used to signal an intention that in that element, white space should be preserved by applications. It is defined according to xml:space as declared by W3C. **Tags:** xml.attribute=true xml.attributeRef=true xml.enforceMinMultiplicity=true xml.name=space xml.nsPrefix=xml

Table 9.7: WhitespaceControlled

9.2.3 Lists in Documentation

In documentation it is often appropriate to present facts in form of lists. AUTOSAR supports the following kinds of lists:

- **[TPS_GST_00316] Plain List** ⌈The **plain list** (List) is composed of one or more list items (item). There are two types of lists, numbered and unnumbered[6] This is controlled by type in List.

  An Item contains a DocumentationBlock. In most cases, it is a simple paragraph, but it often one or more paragraphs followed by a sub-list.⌋*()*

- **[TPS_GST_00317] Labeled List** ⌈The **labeled list** (LabeledList) is a list, where every item has a label (LabeledItem) and a content, which is a DocumentationBlock.

  The policy how to render the labeled list is denoted in itemLabelPos in IndentSample.⌋*()*

- **[TPS_GST_00318] Definition List** ⌈The **definition list** (DefList) is used to introduce terms and their definition. The term is captured in a DefItem and the definition expressed in a DocumentationBlock.

  Note that the DefList maintains the specific semantics of definitions, even if it might be rendered in the same way as LabeledList. For example the defItems in a DefList are Referrable.⌋*()*

[constr_2520] Nesting of lists shall be limited ⌈The nesting of lists shall be limited to a reasonable depth such that it can safely be rendered on A4 pages. A reasonable approach is not to nest more than three levels.⌋*()*

[6] An unnumbered list is also called bullet list.

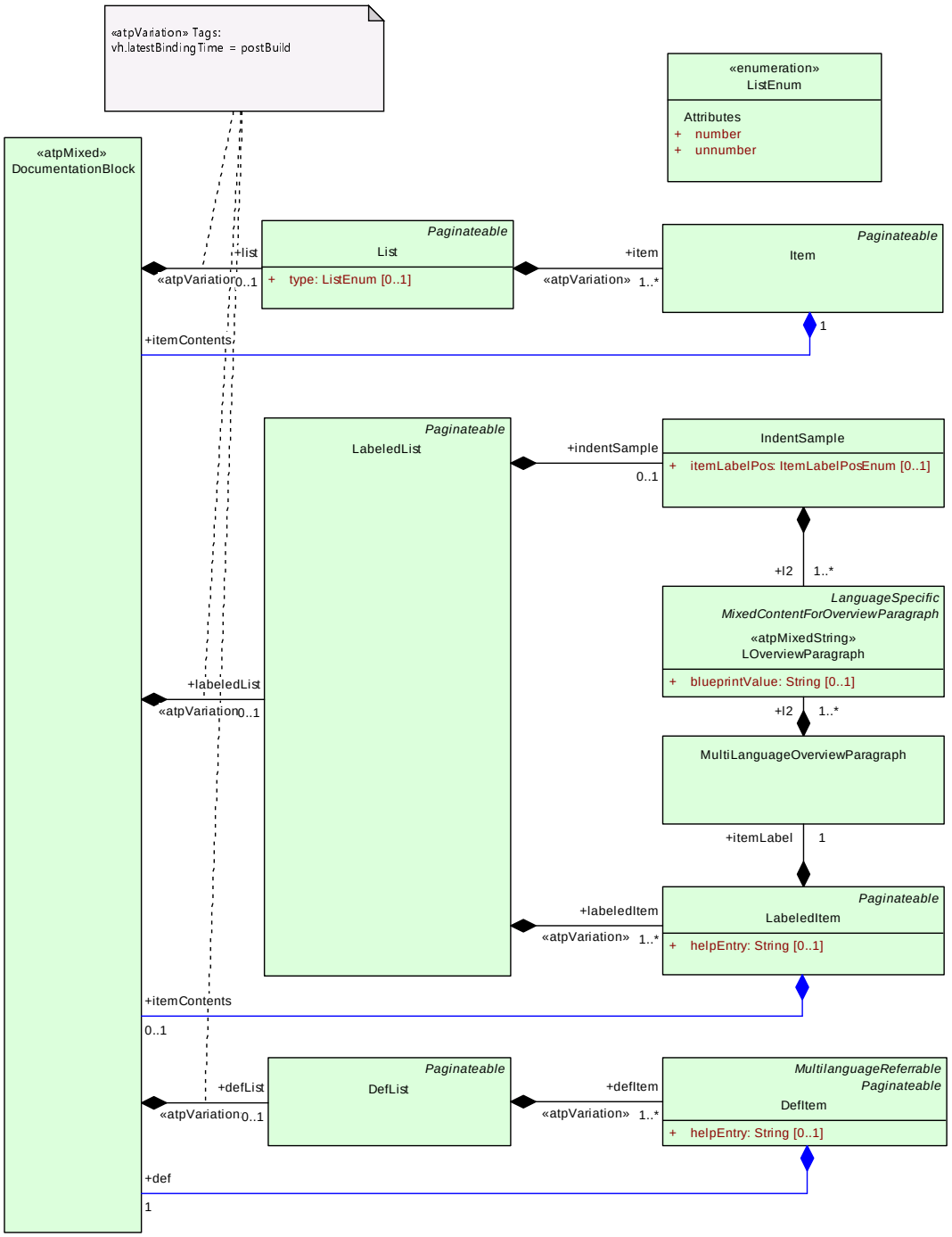


Figure 9.3: List Element Overview

9.2.3.1 Class tables for `List`

Class	**List**			
Package	M2::MSR::Documentation::BlockElements::ListElements			
Note	This meta-class represents the ability to express a list. The kind of list is specified in the attribute.			
Base	*ARObject*, *DocumentViewSelectable*, *Paginateable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
item	Item	1..*	aggr	this represents a particular list item. Note that this is again a documentation block.Therefore lists can be arbitrarily nested. It is discouraged to have a very deep nesting. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=postBuild xml.roleElement=true xml.roleWrapperElement=false xml.sequenceOffset=20 xml.typeElement=false xml.typeWrapperElement=false
type	ListEnum	0..1	attr	The type of the list. Default is "UNNUMBER" **Tags:**xml.attribute=true

Table 9.8: List

Class	**Item**			
Package	M2::MSR::Documentation::BlockElements::ListElements			
Note	This meta-class represents one particular item in a list.			
Base	*ARObject*, *DocumentViewSelectable*, *Paginateable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
itemContents	DocumentationBlock	1	aggr	this represents the actual content of the item. It is composed of a DocumentationBlock. This way it is possible to use simple paragraphs to nested lists, formula, figures or notes. **Tags:** xml.roleElement=false xml.roleWrapperElement=false xml.typeElement=false xml.typeWrapperElement=false

Table 9.9: Item

Enumeration	**ListEnum**
Package	M2::MSR::Documentation::BlockElements::ListElements
Note	This meta-class represents the notation of the various types of lists.
Literal	***Description***
number	This indicates that the list is an numerated list. **Tags:**atp.EnumerationLiteralIndex=0
unnumber	This indicates that it is an enumeration (bulleted list) **Tags:**atp.EnumerationLiteralIndex=1

Table 9.10: ListEnum

9.2.3.2 Class tables for `LabeledList`

Class	**LabeledList**			
Package	M2::MSR::Documentation::BlockElements::ListElements			
Note	This meta-class represents a labeled list, in which items have a label and a content. The policy how to render such items is specified in the labeled list.			
Base	*ARObject*, *DocumentViewSelectable*, *Paginateable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
indentSample	IndentSample	0..1	aggr	This is a sample item. This sample is used by a rendering system to measure out the width of indentation. Since this depends on the particular fontsize etc. the indentation cannot be specified e.g. in mm. **Tags:**xml.sequenceOffset=20
labeledItem	LabeledItem	1..*	aggr	This represents one particular item in the labeled list. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=postBuild xml.roleElement=true xml.roleWrapperElement=false xml.sequenceOffset=30 xml.typeElement=false xml.typeWrapperElement=false

Table 9.11: LabeledList

Class	**LabeledItem**			
Package	M2::MSR::Documentation::BlockElements::ListElements			
Note	this represents an item of a labeled list.			
Base	*ARObject*, *DocumentViewSelectable*, *Paginateable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
helpEntry	String	0..1	attr	This specifies an entry point in an online help system to be linked with the parent class. The syntax shall be defined by the applied help system respectively help system generator. **Tags:**xml.attribute=true
itemContents	DocumentationBlock	0..1	aggr	This represents the actual content of the item. It is composed of a DocumentationBlock. This way it is possible to use simple paragraphs to nested lists, formula, figures or notes. **Tags:** xml.roleElement=false xml.roleWrapperElement=false xml.sequenceOffset=30 xml.typeElement=false xml.typeWrapperElement=false
itemLabel	MultiLanguageOverview Paragraph	1	aggr	This is the label of the item. **Tags:**xml.sequenceOffset=20

Table 9.12: LabeledItem

Class	**IndentSample**			
Package	M2::MSR::Documentation::BlockElements::ListElements			
Note	This represents the ability to specify indentation of a labeled list by providing a sample content. This content can be measured by the rendering system in order to determine the width of indentation.			
Base	*ARObject*			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
itemLabelPos	ItemLabelPosEnum	0..1	attr	The position of the label in case the label is too long. The default is "NO-NEWLINE" **Tags:**xml.attribute=true
l2	LOverviewParagraph	1..*	aggr	This represents the indent sample in one particular language. **Tags:** xml.roleElement=true xml.roleWrapperElement=false xml.sequenceOffset=20 xml.typeElement=false xml.typeWrapperElement=false

Table 9.13: IndentSample

Enumeration	**ItemLabelPosEnum**
Package	M2::MSR::Documentation::BlockElements::ListElements
Note	This enumerator specifies, how the label of a labeled list shall be rendered.
Literal	*Description*
newline	The label is renders in a new line. **Tags:**atp.EnumerationLiteralIndex=0
newlineIfNecessary	The label is rendered in a new line if it is longer than the indentation. **Tags:**atp.EnumerationLiteralIndex=1
noNewline	The label is rendered in one line with the item even if it is longer than the indentation. **Tags:**atp.EnumerationLiteralIndex=2

Table 9.14: ItemLabelPosEnum

9.2.3.3 Class tables for `DefList`

Class	**DefList**			
Package	M2::MSR::Documentation::BlockElements::ListElements			
Note	This meta-class represents the ability to express a list of definitions. Note that a definition list might be rendered similar to a labeled list but has a particular semantics to denote definitions.			
Base	*ARObject*, *DocumentViewSelectable*, *Paginateable*			
Attribute	*Type*	*Mult.*	*Kind*	*Note*

▽

△

Class	DefList			
defItem	DefItem	1..*	aggr	This is one entry in the definition list. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=postBuild xml.roleElement=true xml.roleWrapperElement=false xml.sequenceOffset=20 xml.typeElement=false xml.typeWrapperElement=false

Table 9.15: DefList

Class	DefItem			
Package	M2::MSR::Documentation::BlockElements::ListElements			
Note	This represents an entry in a definition list. The defined item is specified using shortName and longName.			
Base	*ARObject*, *DocumentViewSelectable*, *MultilanguageReferrable*, *Paginateable*, *Referrable*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
def	DocumentationBlock	1	aggr	This represents the definition part of the DefItem. **Tags:**xml.sequenceOffset=20
helpEntry	String	0..1	attr	This specifies an entry point in an online help system to be linked with the parent class. The syntax shall be defined by the applied help system respectively help system generator. **Tags:**xml.attribute=true

Table 9.16: DefItem

9.2.4 Figures in Documentation

[TPS_GST_00319] Figures in Documentation ⌈AUTOSAR supports to include figures in documentation by `MlFigure`. This is composed of a caption (`figureCaption`) and a graphic `lGraphic` with language attribute. The caption gives a title to the figure and also makes the figure referable.⌋*()*

[TPS_GST_00320] Details of Figures in Documentation ⌈`figureCaption` refers to the actual diagram in a standardized diagram format as specified in `GraphicNotationEnum`.

The figure contains also a `map` specifying regions of an image or object and assigning a specific action to each region (e.g., retrieve a document, run a program, etc.) When the region is activated by the user, the action is executed.⌋*()*

`MultiLanguageOverviewParagraph` can be added to further describe the `Caption`

- this allows to provide more elaborate description of the related figure,
- this allows to provide references to other items such as traces, related figures etc..

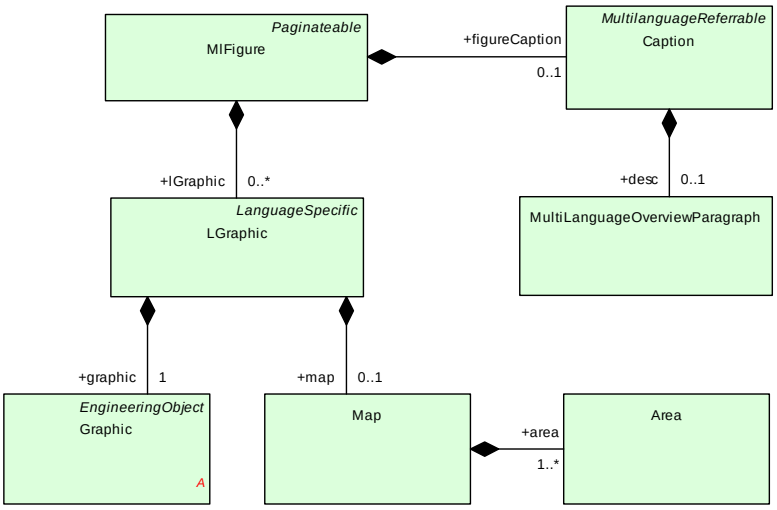


Figure 9.4: Figure Overview

Class	**Area**			
Package	M2::MSR::Documentation::BlockElements::Figure			
Note	This element specifies a region in an image map. Image maps enable authors to specify regions in an object (e.g. a graphic) and to assign a specific activity to each region (e.g. load a document, launch a program etc.). For more details refer to the specification of HTML.			
Base	*ARObject*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
accesskey	String	0..1	attr	This attribute assigns an access key to an element. An access key is an individual character (e.g. "B") within the document character range. If an access key with an element assigned to it is pressed, the element comes into focus. The activity performed when an element comes into focus, is dependent on the element itself **Tags:**xml.attribute=true
alt	String	0..1	attr	This attribute specifies the text to be inserted as an alternative to illustrations, shapes or applets, where these cannot be displayed by user agents. **Tags:**xml.attribute=true
class	String	0..1	attr	Blank separated list of classes **Tags:**xml.attribute=true
coords	String	0..1	attr	This attribute specifies the position and shape on the screen. The number of values and their order depend on the geometrical figure defined. **Tags:**xml.attribute=true
href	String	0..1	attr	This attribute specifies the memory location of a web resource. It is therefore able to specify a link between the current element and the target element. **Tags:**xml.attribute=true
nohref	AreaEnumNohref	0..1	attr	If this attribute is set, the Area has no associated link. **Tags:**xml.attribute=true

△

Class	**Area**			
onblur	String	0..1	attr	The ONBLUR-Event occurs, when focus is switched away from an element. A script can be stored in this attribute to be performed in the Event. **Tags:**xml.attribute=true
onclick	String	0..1	attr	The ONCLICK-Event occurs, if the current element is clicked-on. A script can be stored in this attribute to be performed in the Event. **Tags:**xml.attribute=true
ondblclick	String	0..1	attr	The ONCLICK-Event occurs, if the current element is "double" clicked-on. A script can be stored in this attribute to be performed in the Event. **Tags:**xml.attribute=true
onfocus	String	0..1	attr	The ONFOCUS-Event occurs, if an element comes into focus (e.g., through navigation using the tab button). A script can be stored in this attribute to be performed in the Event. **Tags:**xml.attribute=true
onkeydown	String	0..1	attr	The ONKEYDOWN-Event occurs, if a button on the current element is pressed down. A script can be stored in this attribute to be performed in the event. **Tags:**xml.attribute=true
onkeypress	String	0..1	attr	The ONKEYPRESS-Event occurs, if a button on the current element is pressed down and released. A script can be stored in this attribute to be performed in the Event. **Tags:**xml.attribute=true
onkeyup	String	0..1	attr	The ONKEYUP-Event occurs, if a button on the current element is released. A script can be stored in this attribute to be performed in the Event. **Tags:**xml.attribute=true
onmousedown	String	0..1	attr	The ONMOUSEDOWN-Event occurs, if the mouse button used for clicking is held down on the current element. A script can be stored in this attribute to be performed in the Event. **Tags:**xml.attribute=true
onmousemove	String	0..1	attr	The ONMOUSEMOVE-Event occurs, if the mouse pointer is moved on the current element (i.e. it is located on the current element). A script can be stored in this attribute to be performed in the Event. **Tags:**xml.attribute=true

▽

△

Class	**Area**			
onmouseout	String	0..1	attr	The ONMOUSEOUT-Event occurs, if the mouse pointer is moved from the current element. A script can be stored in this attribute to be performed in the Event. **Tags:**xml.attribute=true
onmouseover	String	0..1	attr	The ONMOUSEOVER-Event occurs, if the mouse pointer is moved to the current element from another location outside it. A script can be stored in this attribute to be performed in the Event. **Tags:**xml.attribute=true
onmouseup	String	0..1	attr	The ONMOUSEUP-Event occurs if the mouse button used for clicking is released on the current element. A script can be stored in this attribute to be performed in the Event. **Tags:**xml.attribute=true
shape	AreaEnumShape	0..1	attr	The shape of the area. Note that in HTML this is defaulted to RECT. **Tags:**xml.attribute=true
style	String	0..1	attr	Information on the associated style **Tags:**xml.attribute=true
tabindex	String	0..1	attr	This attribute specifies the position of the current element in tabbing-order for the corresponding document. The value shall lie between 0 and 32767. The Tabbing Order defines the sequence in which elements are focused on, when the user navigates using the keyboard. **Tags:**xml.attribute=true
title	String	0..1	attr	Title information of the Area element **Tags:**xml.attribute=true

Table 9.17: Area

Enumeration	**AreaEnumNohref**
Package	M2::MSR::Documentation::BlockElements::Figure
Note	This enumerator specifies the fact that the area has no reference.
Literal	***Description***
nohref	This indicates that the area has no active link. **Tags:**atp.EnumerationLiteralIndex=0

Table 9.18: AreaEnumNohref

Enumeration	**AreaEnumShape**
Package	M2::MSR::Documentation::BlockElements::Figure
Note	This enumerator specifies the shape of the area.
Literal	***Description***
circle	The shape is a circle. **Tags:**atp.EnumerationLiteralIndex=0

▽

△

Enumeration	**AreaEnumShape**
default	This specifies the fact that the area covers the rest of the figure. **Tags:**atp.EnumerationLiteralIndex=1
poly	The area is specified as polygon. **Tags:**atp.EnumerationLiteralIndex=2
rect	The shape is specified as rectangle. **Tags:**atp.EnumerationLiteralIndex=3

Table 9.19: AreaEnumShape

Class	**Graphic**			
Package	M2::MSR::Documentation::BlockElements::Figure			
Note	This class represents an artifact containing the image to be inserted in the document			
Base	*ARObject*, *EngineeringObject*			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
editfit	GraphicFitEnum	0..1	attr	Specifies how the graphic shall be displayed in an editor. If the attribute is missing, **Tags:**xml.attribute=true
editHeight	String	0..1	attr	Specifies the height of the graphic when it is displayed in an editor. The unit can be added to the number in the string. Possible units are: cm, mm, px, pt. The default unit is px. **Tags:**xml.attribute=true
editscale	String	0..1	attr	Set the proportional scale when displayed in an editor. **Tags:**xml.attribute=true
editWidth	String	0..1	attr	Specifies the width of the graphic when it is displayed in an editor. The unit can be added to the number in the string. Possible units are: cm, mm, px, pt. The default unit is px. **Tags:**xml.attribute=true
filename	String	0..1	attr	Name of the file that should be displayed. This attribute is supported in ASAM FSX and kept in AUTOSAR in order to support cut and paste. **Tags:**xml.attribute=true
fit	GraphicFitEnum	0..1	attr	It determines the way in which the graphic should be inserted. Enter the attribute value "AS-IS" , to insert a graphic in its original dimensions. The graphic is adapted, if it is too big for the space for which it was intended. Default is "AS-IS" **Tags:**xml.attribute=true
generator	NameToken	0..1	attr	This attribute specifies the generator which is used to generate the image. Use case is that when editing a documentation, a figure (to be delivered by the modeling tool) is inserted by the authoring tool as reference (this is the role of graphic). But the real figure maybe injected during document processing. To be able to recognize this situation, this attribute can be applied. **Tags:**xml.attribute=true

▽

△

Class	**Graphic**			
height	String	0..1	attr	Define the displayed height of the figure. The unit can be added to the number in the string. Possible units are: cm, mm, px, pt. The default unit is px. **Tags:**xml.attribute=true
htmlFit	GraphicFitEnum	0..1	attr	How to fit the graphic in an online media. Default is AS-IS. **Tags:**xml.attribute=true
htmlHeight	String	0..1	attr	Specifies the height of the graphic when it is displayed online. The unit can be added to the number in the string. Possible units are: cm, mm, px, pt. The default unit is px. **Tags:**xml.attribute=true
htmlScale	String	0..1	attr	Set the proportional scale when displayed online. **Tags:**xml.attribute=true
htmlWidth	String	0..1	attr	Specifies the width of the graphic when it is displayed online. The unit can be added to the number in the string. Possible units are: cm, mm, px, pt. The default unit is px. **Tags:**xml.attribute=true
notation	GraphicNotationEnum	0..1	attr	This attribute captures the format used to represent the graphic. **Tags:**xml.attribute=true
scale	String	0..1	attr	In this element the dimensions of the graphic can be altered proportionally. **Tags:**xml.attribute=true
width	String	0..1	attr	Define the displayed width of the figure. The unit can be added to the number in the string. Possible units are: cm, mm, px, pt. The default unit is px. **Tags:**xml.attribute=true

Table 9.20: Graphic

Enumeration	**GraphicFitEnum**
Package	M2::MSR::Documentation::BlockElements::Figure
Note	This enumerator specifies the policy how to place and scale the figure on the page.
Literal	***Description***
AsIs	This indicates that the image shall be incorporated as is without scaling, rotation etc. **Tags:**atp.EnumerationLiteralIndex=0
FitToPage	Fit to the page **Tags:**atp.EnumerationLiteralIndex=1
FitToText	fit to the text containing the graphic. **Tags:**atp.EnumerationLiteralIndex=2
LimitToPage	This indicates that the width of the graphic shall be limited to the **page width**. The image shall not be scaled down but cropped. **Tags:**atp.EnumerationLiteralIndex=3
LimitToText	This indicates that the width of the graphic shall be limited to the width of the current **text flow**. The image shall not be scaled down but cropped. **Tags:**atp.EnumerationLiteralIndex=4
Rotate180	Rotate 180 degree **Tags:**atp.EnumerationLiteralIndex=5

▽

△

Enumeration	**GraphicFitEnum**
Rotate180LimitTo Text	Rotate 180 degree **Tags:**atp.EnumerationLiteralIndex=6
Rotate90ccw	Rotate 90 degree counter clockwise **Tags:**atp.EnumerationLiteralIndex=7
Rotate90CcwFitTo Text	Rotate by 90 degree counter clock wise and then fit to text **Tags:**atp.EnumerationLiteralIndex=8
Rotate90CcwLimit ToText	Rotate by 90 degree counter clock wise and then fit to text **Tags:**atp.EnumerationLiteralIndex=9
Rotate90Cw	Rotate 90 degree clockwise **Tags:**atp.EnumerationLiteralIndex=10
Rotate90CwFitTo Text	Rotate by 90 degree and then fit to text **Tags:**atp.EnumerationLiteralIndex=11
Rotate90CwLimitTo Text	Rotate by 90 degree and then fit to text **Tags:**atp.EnumerationLiteralIndex=12

Table 9.21: GraphicFitEnum

Enumeration	**GraphicNotationEnum**
Package	M2::MSR::Documentation::BlockElements::Figure
Note	This enumerator specifies the various notations (finally file types) used to represent the figure.
Literal	***Description***
bmp	bitmap image **Tags:**atp.EnumerationLiteralIndex=0
eps	Encapsulated Postscript **Tags:**atp.EnumerationLiteralIndex=1
gif	Graphics Interchange Format **Tags:**atp.EnumerationLiteralIndex=2
jpg	"Joint Photographic Experts Group" format **Tags:**atp.EnumerationLiteralIndex=3
pdf	Portable Document Format **Tags:**atp.EnumerationLiteralIndex=4
png	Portable Network Graphics **Tags:**atp.EnumerationLiteralIndex=5
svg	scalable vector graphic **Tags:**atp.EnumerationLiteralIndex=6
tiff	Tagged Image File Format **Tags:**atp.EnumerationLiteralIndex=7

Table 9.22: GraphicNotationEnum

Class	**Map**
Package	M2::MSR::Documentation::BlockElements::Figure

▽

△

Class	**Map**			
Note	Image maps enable authors to specify regions of an image or object and assign a specific action to each region (e.g., retrieve a document, run a program, etc.) When the region is activated by the user, the action is executed. The class follows the html approach and is intended to support interactive documents.			
Base	*ARObject*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
area	Area	1..*	aggr	This element specifies a region in an image map. Image maps enable authors to specify regions in an object (e.g. a graphic) and to assign a specific activity to each region (e.g. load a document, launch a program etc.). **Tags:** xml.roleElement=true xml.roleWrapperElement=false xml.sequenceOffset=20 xml.typeElement=false xml.typeWrapperElement=false
class	String	0..1	attr	This attribute assigns a class name or set of class names to an element. Any number of elements may be assigned the same class name or set of class names. Multiple class names shall be separated by white space characters. Class names are typically used to apply CSS formatting rules to an element. **Tags:**xml.attribute=true
name	NameToken	0..1	attr	This attribute assigns a name to the image map in the MAP element. This name can be used to be referenced in an HTML image through the attribute USEMAP. Although this is not actually necessary in the MSR model, it was inserted in order to support the MAPs which were created for HTML. **Tags:**xml.attribute=true
onclick	String	0..1	attr	The ONCLICK-Event occurs, if the current element is clicked on. A script can be stored in this attribute to be performed in the Event. **Tags:**xml.attribute=true
ondblclick	String	0..1	attr	The ONDBLCLICK-Event occurs, if the current Event is "double" clicked-on. A script can be stored in this attribute to be performed in the Event. **Tags:**xml.attribute=true
onkeydown	String	0..1	attr	The ONKEYDOWN-Event occurs, if a button on the current element is pressed down. A script can be stored in this attribute to be performed in the event. **Tags:**xml.attribute=true
onkeypress	String	0..1	attr	The ONKEYPRESS-Event occurs, if a button on the current element is pressed down and released. A script can be stored in this attribute to be performed in the Event. **Tags:**xml.attribute=true
onkeyup	String	0..1	attr	The ONKEYUP-Event occurs, if a button on the current element is released. A script can be stored in this attribute to be performed in the Event. **Tags:**xml.attribute=true

▽

△

Class	**Map**			
onmousedown	String	0..1	attr	The ONMOUSEDOWN-Event occurs, if the mouse button used for clicking is held down on the current element. A script can be stored in this attribute to be performed in the Event. **Tags:**xml.attribute=true
onmousemove	String	0..1	attr	The ONMOUSEMOVE-Event occurs, if the mouse pointer is moved on the current element (i.e. it is located on the current element). A script can be stored in this attribute to be performed in the Event. **Tags:**xml.attribute=true
onmouseout	String	0..1	attr	The ONMOUSEOUT-Event occurs, if the mouse pointer is moved from the current element. A script can be stored in this attribute to be performed in the Event. **Tags:**xml.attribute=true
onmouseover	String	0..1	attr	The ONMOUSEOVER-Event occurs, if the mouse pointer is moved to the current element from another location outside it. A script can be stored in this attribute to be performed in the Event. **Tags:**xml.attribute=true
onmouseup	String	0..1	attr	The ONMOUSEUP-Event occurs if the mouse button used for clicking is released on the current element. A script can be stored in this attribute to be performed in the Event. **Tags:**xml.attribute=true
title	String	0..1	attr	This attribute offers advisory information. Some Web browsers will display this information as tooltips. Authoring tools may make this information available to users as additional information about the element. **Tags:**xml.attribute=true

Table 9.23: Map

Class	**MIFigure**			
Package	M2::MSR::Documentation::BlockElements::Figure			
Note	This metaclass represents the ability to embed a figure.			
Base	*ARObject*, *DocumentViewSelectable*, *Paginateable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
figureCaption	Caption	0..1	aggr	This element specifies the title of an illustration.

▽

△

Class	**MIFigure**			
frame	FrameEnum	0..1	attr	Used to defined the frame line around a figure. It can assume the following values: • TOP - Border at the top of the figure • BOTTOM - Border at the bottom of the figure • TOPBOT - Borders at the top and bottom of the figure • ALL - Borders all around the figure • SIDES - Borders at the sides of the figure • NONE - No borders around the figure **Tags:**xml.attribute=true
helpEntry	String	0..1	attr	This specifies an entry point in an online help system to be linked with the parent class. The syntax shall be defined by the applied help system respectively help system generator. **Tags:**xml.attribute=true
lGraphic	LGraphic	*	aggr	Container of the graphic (or diagram) and optional map of the figure in a given language. **Tags:** xml.roleWrapperElement=false xml.sequenceOffset=30
pgwide	PgwideEnum	0..1	attr	Used to indicate wether the figure should take the complete page width (value = "pgwide") or not (value = "noPgwide"). **Tags:**xml.attribute=true
verbatim	MultiLanguageVerbatim	0..1	aggr	<verbatim> is a paragraph in which white-space (in particular blanks and line feeds) is obeyed. This enables basic preformatting to be carried out, which can even be displayed on simple devices. Behavior is the same as PRE in HTML . **Tags:**xml.sequenceOffset=50

Table 9.24: MIFigure

Class	**LGraphic**			
Package	M2::MSR::Documentation::BlockElements::Figure			
Note	This meta-class represents the figure in one particular language.			
Base	*ARObject*, *LanguageSpecific*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
graphic	Graphic	1	aggr	Reference to the actual graphic represented in the figure. **Tags:**xml.sequenceOffset=20
map	Map	0..1	aggr	Image maps enable authors to specify regions of an image or object and assign a specific action to each region. **Tags:**xml.sequenceOffset=30

Table 9.25: LGraphic

9.2.5 Formula in Documentation

[TPS_GST_00321] Mathematical Subjects in Documentation ⌈AUTOSAR supports to use formula to document mathematical subjects. `MlFormula` supports an optional caption (`formulaCaption`) to give a title to the formula and also makes the formula referable.⌋*()*

`MultiLanguageOverviewParagraph` can be added to further describe the `Caption`

- this allows to provide more elaborate description of the related formula,
- this allows to provide references to other items such as traces, related formulas etc..

[TPS_GST_00322] Various Formula Representation ⌈The formula itself takes at least one of the following forms:

- a graphic (`LGraphic`), which contains the formula represented as a graphic.
- a math captured in TEXmath mode (`texMath`)
- a generic math (`genericMath`)
- a verbatim (`verbatim`) when the formula is written as simple text, where the spaces are preserved

These forms may exist simultaneously. When a documentation is rendered, the rendering engine should use the representation with best fit to the rendition format.⌋*()*

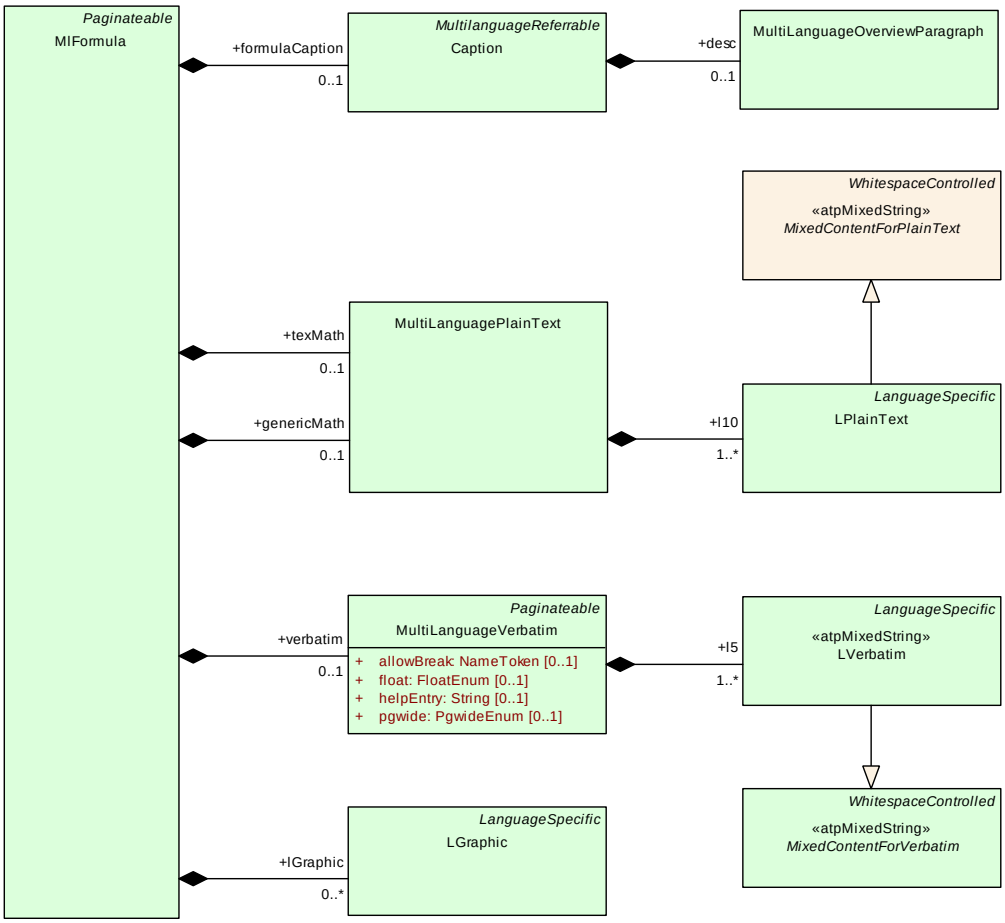


Figure 9.5: Formula Overview

Class	**MIFormula**			
Package	M2::MSR::Documentation::BlockElements::Formula			
Note	This meta-class represents the ability to express a formula in a documentation. The formula can be expressed by various means. If more than one representation is available, they need to be consistent. The rendering system can use the representation which is most appropriate.			
Base	*ARObject*, *DocumentViewSelectable*, *Paginateable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
formulaCaption	Caption	0..1	aggr	This element specifies the identification or heading of a formula. **Tags:**xml.sequenceOffset=20
genericMath	MultiLanguagePlainText	0..1	aggr	this rpresents the semantic and mathematical descriptions which are processed by a math-processor. **Tags:**xml.sequenceOffset=80
lGraphic	LGraphic	*	aggr	This represents a formula as an embedded figure. **Tags:** xml.roleWrapperElement=false xml.sequenceOffset=30
texMath	MultiLanguagePlainText	0..1	aggr	this is the TeX representation of TeX formula. A TeX formula can be processed by a TeX or a LaTeX processor. **Tags:**xml.sequenceOffset=60

△

Class	**MIFormula**			
verbatim	MultiLanguageVerbatim	0..1	aggr	this represents a formula using only text and white-space. It can be used to denote the formula in a kind of pseudo code or whatever appears approprate. **Tags:**xml.sequenceOffset=50

Table 9.26: MIFormula

9.2.6 Notes in Documentation

[TPS_GST_00323] Purpose of `Note` ⌈The meta-class `Note` can be used to place notes with side heads and icons in a document. It is used for example to highlight instructions, exercises, cautions etc. The note itself contains a documentation block. It is composed of an optional label and one or more paragraphs.⌋*()*

[constr_2522] Notes should not be nested ⌈Note even if it is possible to nest notes it is not recommended to do so, since it might lead to problems with the rendering of the note icon.⌋*()*

Class	**Note**			
Package	M2::MSR::Documentation::BlockElements::Note			
Note	This represents a note in a documentation, which may be used to highlight specific issues such as hints or caution notes. N.B., Documentation notes can be nested recursively, even if this is not really intended. In case of nested notes e.g. the note icon of inner notes might be omitted while rendering the note.			
Base	*ARObject*, *DocumentViewSelectable*, *Paginateable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
label	MultilanguageLongName	0..1	aggr	This label can be used to superseed the default label specified by the noteType attribute. It is in particular useful for noteType="other". **Tags:**xml.sequenceOffset=20
noteText	DocumentationBlock	1	aggr	This is the text content of the note. **Tags:** xml.roleElement=false xml.roleWrapperElement=false xml.sequenceOffset=30 xml.typeElement=false xml.typeWrapperElement=false
noteType	NoteTypeEnum	0..1	attr	Type of the Note. Default is "HINT" **Tags:**xml.attribute=true

Table 9.27: Note

Enumeration	**NoteTypeEnum**
Package	M2::MSR::Documentation::BlockElements::Note
Note	This enumerator specifies the type of the note. It can be used to render a note label or even a note icon.

▽

△

Enumeration	**NoteTypeEnum**
Literal	***Description***
caution	This indicates that the note is an alert which shall be considered carefully. **Tags:**atp.EnumerationLiteralIndex=0
example	This indicates that the note represents an example, e.g. a code example etc. **Tags:**atp.EnumerationLiteralIndex=1
exercise	This indicates that the note represents an exercise for the reader. **Tags:**atp.EnumerationLiteralIndex=2
hint	This indicates that the note represents a hint which helps the user for better understanding. **Tags:**atp.EnumerationLiteralIndex=3
instruction	This indicates that the note represents an instruction, e.g. a step by step procedure. **Tags:**atp.EnumerationLiteralIndex=4
other	This indicates that the note is something else. The particular type of the note shall then be specified in the label of the note. **Tags:**atp.EnumerationLiteralIndex=5
tip	This indicates that the note represents which is good to know. It is similar to a hint, but focuses more to good practice than to better understanding. **Tags:**atp.EnumerationLiteralIndex=6

Table 9.28: NoteTypeEnum

9.2.7 Support for Traceability in Documentation

AUTOSAR documentation support includes the ability to perform tracing between text elements. This tracing is primarily intended to be applied as bottom up tracing such as tracing from a specification statement to particular requirements which are fulfilled by the specified item. See [2] for more details.

[TPS_GST_00243] Informal references to traceable text ⌈It is also possible to provide informal references to `Traceable` within a paragraph by `trace`.

This association specifies a kind of citation of the trace. In contrast to `Traceable`. `trace` it is an arbitrary dependency which is for reference purpose. This association is not intended to be counted in requirements tracing analysis.

Note that when generating hyperlinked documents, this association can be represented as [TPS_STDT_0042] without hyperlink if the associated trace is not part of the document.

⌋()

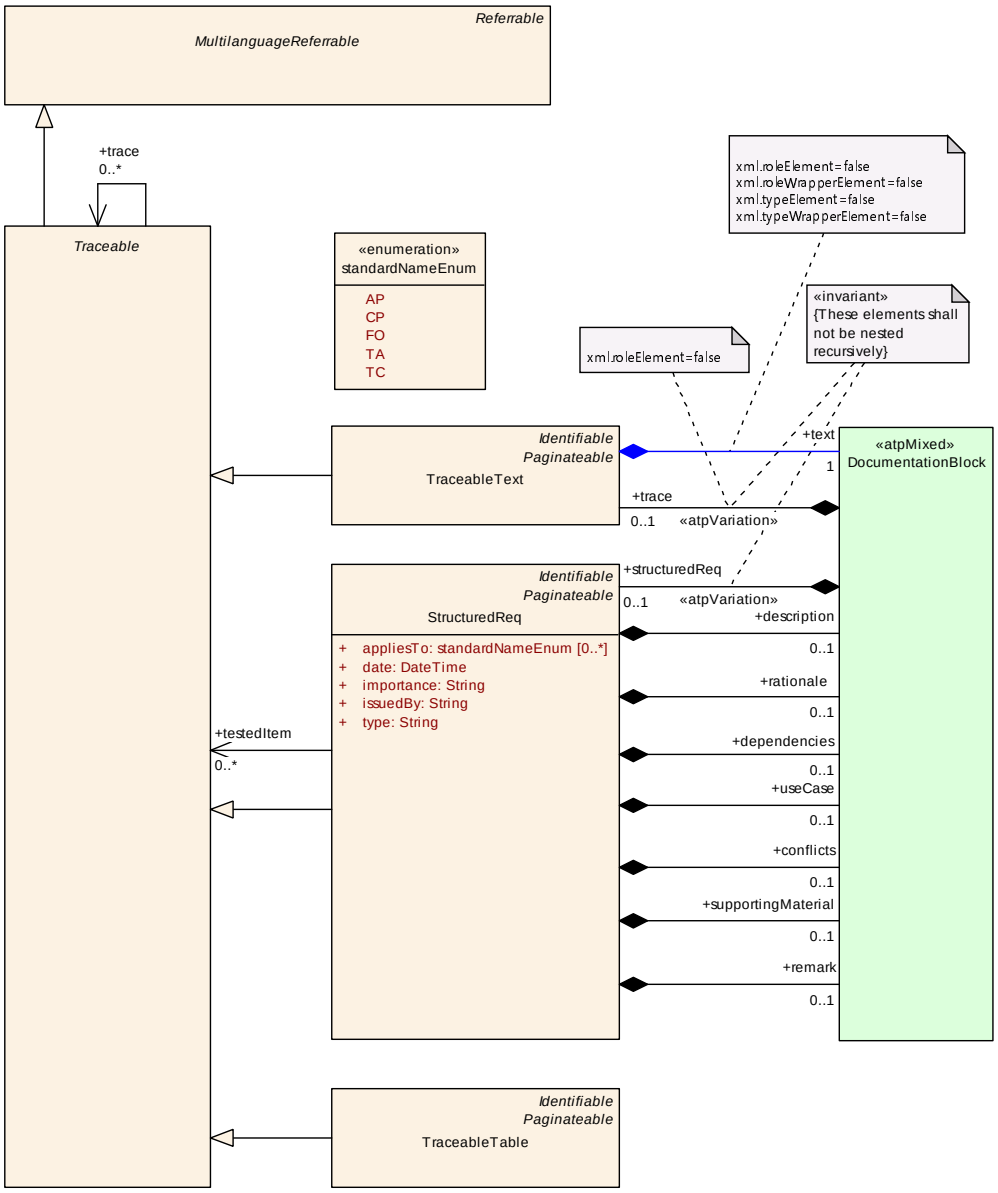


Figure 9.6: Support for Traceability

Class	***Traceable*** (abstract)			
Package	M2::MSR::Documentation::BlockElements::RequirementsTracing			
Note	This meta class represents the ability to be subject to tracing within an AUTOSAR model. Note that it is expected that its subclasses inherit either from MultilanguageReferrable or from Identifiable. Nevertheless it also inherits from MultilanguageReferrable in order to provide a common reference target for all Traceables.			
Base	*ARObject*, *MultilanguageReferrable*, *Referrable*			
Subclasses	StructuredReq, *TimingConstraint*, TraceableTable, TraceableText			
Attribute	***Type***	***Mult.***	***Kind***	***Note***

▽

△

Class	*Traceable* (abstract)			
trace	Traceable	*	ref	This association represents the ability to trace to upstream requirements / constraints. This supports for example the bottom up tracing ProjectObjectives <- MainRequirements <- Features <- RequirementSpecs <- BSW/AI **Tags:**xml.sequenceOffset=20

Table 9.29: Traceable

Class	**TraceableText**			
Package	M2::MSR::Documentation::BlockElements::RequirementsTracing			
Note	This meta-class represents the ability to denote a traceable text item such as requirements etc. The following approach applies: • **shortName** represents the tag for tracing • **longName** represents the head line • **category** represents the kind of the tagged text			
Base	*ARObject*, *DocumentViewSelectable*, *Identifiable*, *MultilanguageReferrable*, *Paginateable*, *Referrable*, *Traceable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
text	DocumentationBlock	1	aggr	This represents the text to which the tag applies. **Tags:** xml.roleElement=false xml.roleWrapperElement=false xml.sequenceOffset=30 xml.typeElement=false xml.typeWrapperElement=false

Table 9.30: TraceableText

Class	**StructuredReq**			
Package	M2::MSR::Documentation::BlockElements::RequirementsTracing			
Note	This represents a structured requirement. This is intended for a case where specific requirements for features are collected. Note that this can be rendered as a labeled list.			
Base	*ARObject*, *DocumentViewSelectable*, *Identifiable*, *MultilanguageReferrable*, *Paginateable*, *Referrable*, *Traceable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
appliesTo	standardNameEnum	*	attr	This attribute represents the platform the requirement is assigned to. **Tags:** xml.namePlural=APPLIES-TO-DEPENDENCIES xml.sequenceOffset=25
conflicts	DocumentationBlock	0..1	aggr	This represents an informal specification of conflicts. **Tags:**xml.sequenceOffset=40
date	DateTime	1	attr	This represents the date when the requirement was initiated. **Tags:**xml.sequenceOffset=5

▽

△

Class	StructuredReq			
dependencies	DocumentationBlock	0..1	aggr	This represents an informal specification of dependencies. Note that upstream tracing should be formalized in the property trace provided by the superclass Traceable. **Tags:**xml.sequenceOffset=30
description	DocumentationBlock	0..1	aggr	This represents the general description of the requirement. **Tags:**xml.sequenceOffset=10
importance	String	1	attr	This allows to represent the importance of the requirement. **Tags:**xml.sequenceOffset=8
issuedBy	String	1	attr	This represents the person, organization or authority which issued the requirement. **Tags:**xml.sequenceOffset=6
rationale	DocumentationBlock	0..1	aggr	This represents the rationale of the requirement. **Tags:**xml.sequenceOffset=20
remark	DocumentationBlock	0..1	aggr	This represents an informal remark. Note that this is not modeled as annotation, since these remark is still essential part of the requirement. **Tags:**xml.sequenceOffset=60
supporting Material	DocumentationBlock	0..1	aggr	This represents an informal specification of the supporting material. **Tags:**xml.sequenceOffset=50
testedItem	Traceable	*	ref	This association represents the ability to trace on the same specification level. This supports for example the of acceptance tests. **Tags:**xml.sequenceOffset=70
type	String	1	attr	This attribute allows to denote the type of requirement to denote for example is it an "enhancement", "new feature" etc. **Tags:**xml.sequenceOffset=7
useCase	DocumentationBlock	0..1	aggr	This describes the relevant use cases. Note that formal references to use cases should be done in the trace relation. **Tags:**xml.sequenceOffset=35

Table 9.31: StructuredReq

9.2.8 Mixed Content and Inline Text Model Element

[TPS_GST_00324] Inline Elements in Documentation ⌈Depending on the context, AUTOSAR supports various inline elements. Inline elements represents specific markup of text within e.g. a paragraph. Example for this is subscript/superscript etc.⌋*()*

In listing 9.1 the `desc` uses the inline elements:

- `Tt` to express specific technical terms by the type "PARAMETER",
- `EmphasisText` to emphasized text by the font type "ITALIC" and

- Br to force a line break.

Listing 9.1: Inline Elements in Documentation

```
<BSW-MODULE-ENTRY>
  <SHORT-NAME>Dcm_ReadMemory</SHORT-NAME>
  <LONG-NAME>
    <L-4 L="EN">Dcm_ReadMemory</L-4>
  </LONG-NAME>
  <DESC>
    <L-2 L="EN">The Dcm_ReadMemory callout is used to request memory data
       identified by the parameter <TT TYPE="PARAMETER">memoryAddress</TT
       > and <TT TYPE="PARAMETER">memorySize</TT> from the UDS request
       message.
       This service is needed for the implementation of <E TYPE="ITALIC">
          UDS services</E>:<BR/>
       - ReadMemoryByAddress<BR/>
       - RequestUpload<BR/>
       - ReadDataByIdentifier (in case of Dynamical DID defined by memory
          address)</L-2>
  </DESC>
</BSW-MODULE-ENTRY>
```

Table 9.32 indicates the availability of inline elements in the various content models.

	Br	EmphasisText	IndexEntry	Std	Superscript	Tt	Traceable	Xdoc	Xfile	Xref	XrefTarget	MixedContentForOverviewParagraph	MixedContentForParagraph
MixedContentForLongName		e	ie		sub / sup		trace						
MixedContentForOverviewParagraph	br	e	ie		sub / sup	tt	trace			xref	xrefTarget	ft	
MixedContentForParagraph	br	e	ie	std	sub / sup	tt	trace	xdoc	xfile	xref	xrefTarget		ft
MixedContentForPlainText													
MixedContentForUnitNames					sub / sup								
MixedContentForVerbatim	br	e				tt				xref			
EmphasisText					sub / sup	tt							

Table 9.32: Inline Text Model

[constr_2596] Used colors of attributes color and bgcolor ⌈The used colors of the attributes color and bgcolor shall base on the 6 digits RGB hex-code following

```
|#([a-fA-F0-9]{6})|
```

.⌋()

Class	**Br**			
Package	M2::MSR::Documentation::TextModel::InlineTextElements			
Note	This element is the same as function here as in a HTML document i.e. it forces a line break.			
Base	*ARObject*			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
–	–	–	–	–

Table 9.33: Br

Class	<<atpMixedString>> **EmphasisText**			
Package	M2::MSR::Documentation::TextModel::InlineTextElements			
Note	This is an emphasized text. As a compromise it contains some rendering oriented attributes such as color and font.			
Base	*ARObject*			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
color	String	0..1	attr	This allows to recommend a color of the emphasis. It is specified bases on 6 digits RGB hex-code. **Tags:**xml.attribute=true
font	EEnumFont	0..1	attr	This specifies the font style in which the emphasized text shall be rendered. **Tags:**xml.attribute=true
sub	Superscript	1	attr	this is subscript text
sup	Superscript	1	attr	This is superscript text
tt	Tt	0..1	aggr	This is a technical term. **Tags:**xml.sequenceOffset=30
type	EEnum	0..1	attr	Indicates how the text may be emphasized. Note that this is only a proposal which can be overridden or ignored by particular formatting engines. Default is BOLD. **Tags:**xml.attribute=true

Table 9.34: EmphasisText

Primitive	**ExtIdClassEnum**
Package	M2::MSR::Documentation::TextModel::InlineAttributeEnums
Note	This is in fact an enumerator. The possible values are all legal XML names of identifiable objects even those of other XML files. If the schemas of all questionable files are generated from a common meta-model, this is something like an IdentifiableSubtypesEnum. Maybe a future version of the Schema generator can generate such an enum. As of now it is specified as string. **Tags:** xml.xsd.customType=EXT-ID-CLASS-ENUM xml.xsd.type=string

Table 9.35: ExtIdClassEnum

Class	<<atpMixedString>> **IndexEntry**
Package	M2::MSR::Documentation::TextModel::InlineTextElements
Note	This class represents an index entry.

▽

△

Class	<<atpMixedString>> **IndexEntry**			
Base	*ARObject*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
sub	Superscript	1	attr	This is subscript text. **Tags:**xml.sequenceOffset=40
sup	Superscript	1	attr	This is superscript text. **Tags:**xml.sequenceOffset=30

Table 9.36: IndexEntry

Class	**Std**			
Package	M2::MSR::Documentation::TextModel::InlineTextElements			
Note	This represents a reference to external standards.			
Base	*ARObject*, *Referrable*, *SingleLanguageReferrable*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
date	DateTime	0..1	attr	This element specifies the release date of the external standard if applicable. **Tags:**xml.sequenceOffset=50
position	String	0..1	attr	This represents the reference to the relevant positions of a standard. Kept as a string. **Tags:**xml.sequenceOffset=70
state	String	0..1	attr	This represents version and state of a standard. Kept as a string. **Tags:**xml.sequenceOffset=40
subtitle	String	0..1	attr	This represents the subtitle of the standard. **Tags:**xml.sequenceOffset=30
url	Url	0..1	aggr	This represents the URL of the standard. **Tags:**xml.sequenceOffset=60

Table 9.37: Std

Primitive	**Superscript**
Package	M2::MSR::Documentation::TextModel::InlineTextElements
Note	This is text which is rendered superscript or subscript depending on the role. **Tags:** xml.xsd.customType=SUPSCRIPT xml.xsd.type=string

Table 9.38: Superscript

Class	**Tt**			
Package	M2::MSR::Documentation::TextModel::InlineTextElements			
Note	This meta-class represents the ability to express specific technical terms. The kind of term is denoted in the attribute "type".			
Base	*ARObject*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**

▽

△

Class	**Tt**			
term	String	1	attr	This is the term itself. **Tags:** xml.roleElement=false xml.roleWrapperElement=false xml.typeElement=false xml.typeWrapperElement=false
texRender	String	0..1	attr	This attribute holds information how the content (represented by attribute "term") of the particular technical term is rendered using LaTeX. This allows to inject specific LaTeX commands such as \sep{}. An example is to render "MyClass" as "My\sep{}Class". Default is the value of the attribute "term". **Tags:**xml.attribute=true
type	NameToken	1	attr	This attribute specifies the type of the technical term. Values are such as "VARIABLE" "CALPRM". It is no longer an enum in order to support process specific extensions. **Tags:**xml.attribute=true

Table 9.39: Tt

Class	**Xdoc**			
Package	M2::MSR::Documentation::TextModel::InlineTextElements			
Note	This meta-class represents the ability to refer to an external document which can be rendered as printed matter.			
Base	*ARObject*, *Referrable*, *SingleLanguageReferrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
date	DateTime	0..1	attr	This element specifies the release date of the external document if applicable. **Tags:**xml.sequenceOffset=50
number	String	0..1	attr	This represents document number of an external document that is referenced. Kept as a string. **Tags:**xml.sequenceOffset=30
position	String	0..1	attr	This represents the reference to the relevant positions of a standard. Kept as a string. **Tags:**xml.sequenceOffset=80
publisher	String	0..1	attr	This represents the publisher of an external document that is being referenced. Kept as a string. **Tags:**xml.sequenceOffset=60
state	String	0..1	attr	This represents version and state of the external document. Kept as a string. **Tags:**xml.sequenceOffset=40
url	Url	0..1	aggr	This specifies the URL of the external document. **Tags:**xml.sequenceOffset=70

Table 9.40: Xdoc

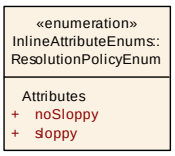
Class	**Xfile**
Package	M2::MSR::Documentation::TextModel::InlineTextElements

▽

△

Class	**Xfile**			
Note	This represents to reference an external file within a documentation.			
Base	*ARObject*, *Referrable*, *SingleLanguageReferrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
tool	String	0..1	attr	This element describes the tool which was used to generate the corresponding Xfile . Kept as a string since no specific syntax can be provided to denote a tool. **Tags:**xml.sequenceOffset=50
toolVersion	String	0..1	attr	This element describes the tool version which was used to generate the corresponding xfile. Kept as a string, since no specific syntax can be specified. **Tags:**xml.sequenceOffset=60
url	Url	0..1	aggr	This represents the URL of the external file. **Tags:**xml.sequenceOffset=30

Table 9.41: Xfile



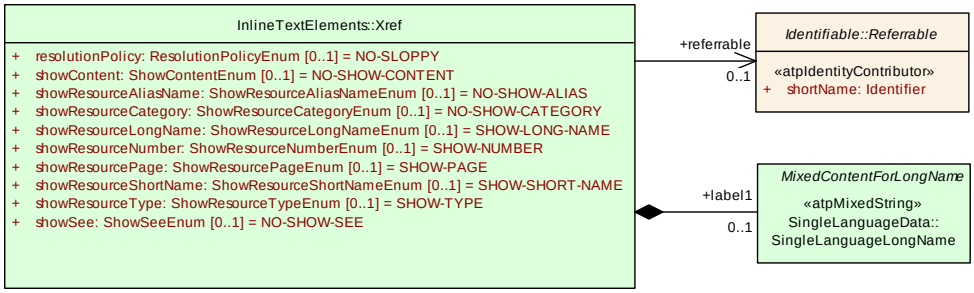


Figure 9.7: Xref overview

Class	**Xref**			
Package	M2::MSR::Documentation::TextModel::InlineTextElements			
Note	This represents a cross-reference within documentation.			
Base	*ARObject*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
label1	SingleLanguageLongName	0..1	aggr	This allows to specify a replacement text which shall be rendered if showContent is selected.
referrable	Referrable	0..1	ref	This establishes the reference in Autosar style
resolutionPolicy	ResolutionPolicyEnum	0..1	attr	Indicates if the content of the xref element follow a dedicated resolution policy. The default is "NO-SLOPPY". **Tags:**xml.attribute=true

▽

△

Class	**Xref**			
showContent	ShowContentEnum	0..1	attr	Indicates if the content of the xref element shall be rendered. The default is "NO-SHOW-CONTENT". **Tags:**xml.attribute=true
showResource AliasName	ShowResourceAlias NameEnum	0..1	attr	This indicates if the alias names of the referenced objects shall be rendered. This means this is some kind of backward searching: look whether there is an alias for the referenced object, if yes, print it. If there is more than one AliasNameSet, Xref might render all of those. If no alias is found and showResourceShortName is set to NoShowShortName, then the shortName of the reference target shall be displayed. By this showResourceAlias Name is similar to showResourceShortName but shows the aliasName instead of the shortName. Default is NO-SHOW-ALIAS-NAME. **Tags:**xml.attribute=true
showResource Category	ShowResource CategoryEnum	0..1	attr	Indicates if the category of the referenced resource shall be rendered. Default is "NO-SHOW-CATEGORY". **Tags:**xml.attribute=true
showResource LongName	ShowResourceLong NameEnum	0..1	attr	Indicates if the longName of the referenced resource shall be rendered. Default is "SHOW-LONG-NAME". **Tags:**xml.attribute=true
showResource Number	ShowResourceNumber Enum	0..1	attr	Indicates if the Number of the referenced resource shall be shown. Default is "SHOW–NUMBER" **Tags:**xml.attribute=true
showResource Page	ShowResourcePage Enum	0..1	attr	Indicates if the page number of the referenced resource shall be shown. Default is "SHOW-PAGE" **Tags:**xml.attribute=true
showResource ShortName	ShowResourceShort NameEnum	0..1	attr	Indicates if the shortJName of the referenced resource shall be shown. Default is "SHOW-SHORT-NAME" **Tags:**xml.attribute=true
showResource Type	ShowResourceType Enum	0..1	attr	Indicates if the type of the referenced Resource shall be shown. Default is "SHOW-TYPE" **Tags:**xml.attribute=true
showSee	ShowSeeEnum	0..1	attr	Indicates if the word "see " shall be shown before the reference. Default is "NO-SHOW-SEE". Note that this is there for compatibility reasons only. **Tags:**xml.attribute=true

Table 9.42: Xref

Class	**XrefTarget**			
Package	M2::MSR::Documentation::TextModel::InlineTextElements			
Note	This element specifies a reference target which can be scattered throughout the text.			
Base	*ARObject*, *Referrable*, *SingleLanguageReferrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table 9.43: XrefTarget

Enumeration	**EEnumFont**
Package	M2::MSR::Documentation::TextModel::InlineAttributeEnums
Note	This specifies the possible kind of fonts to be used for emphasis.
Literal	***Description***
default	The emphasis uses the default font. **Tags:**atp.EnumerationLiteralIndex=0
mono	The emphasis uses a monospaced font. **Tags:**atp.EnumerationLiteralIndex=1

Table 9.44: EEnumFont

Enumeration	**EEnum**
Package	M2::MSR::Documentation::TextModel::InlineAttributeEnums
Note	This specifies the possible kinds of emphasis as proposal how to render it on paper or screen. Note that it would have been better to use plain, weak (italic), strong (bold), veryStrong (bolditalic) ... But users complained about this.
Literal	***Description***
bold	The emphasis is preferably represented in boldface font. **Tags:**atp.EnumerationLiteralIndex=0
bolditalic	The emphasis is preferably represented in boldface plus italic font. **Tags:**atp.EnumerationLiteralIndex=1
italic	The emphasis is preferably represented in italic font. **Tags:**atp.EnumerationLiteralIndex=2
plain	The emphasis has no specific rendering. It is used if e.g. semantic information is applied to the emphasis text. **Tags:**atp.EnumerationLiteralIndex=3

Table 9.45: EEnum

Enumeration	**ResolutionPolicyEnum**
Package	M2::MSR::Documentation::TextModel::InlineAttributeEnums
Note	This specifies if the content of the xref element follow a dedicated resolution policy.
Literal	***Description***
noSloppy	The content of the xref element is **not** linked by a sloppy reference. **Tags:**atp.EnumerationLiteralIndex=0
sloppy	The content of the xref element is linked by a sloppy reference. **Tags:**atp.EnumerationLiteralIndex=1

Table 9.46: ResolutionPolicyEnum

Enumeration	**ShowContentEnum**
Package	M2::MSR::Documentation::TextModel::InlineAttributeEnums
Note	This specifies if the content of the xref element shall be rendered.
Literal	***Description***
noShowContent	The content of the Xref.label is **not** rendered at the place of the reference. **Tags:**atp.EnumerationLiteralIndex=0

△

Enumeration	**ShowContentEnum**
showContent	The content of the element is rendered at the place of the reference. **Tags:**atp.EnumerationLiteralIndex=1

Table 9.47: ShowContentEnum

Enumeration	**ShowResourceAliasNameEnum**
Package	M2::MSR::Documentation::TextModel::InlineAttributeEnums
Note	This enumerator specifies if the alias names of the reference target shall be rendered with the xref.
Literal	***Description***
noShowAliasName	This indicates that alias names of the referenced object shall **not** be rendered at the place of the reference. **Tags:**atp.EnumerationLiteralIndex=0
showAliasName	This indicates that the alias names of the referenced object shall be rendered at the place of the reference. **Tags:**atp.EnumerationLiteralIndex=1

Table 9.48: ShowResourceAliasNameEnum

Enumeration	**ShowResourceCategoryEnum**
Package	M2::MSR::Documentation::TextModel::InlineAttributeEnums
Note	This enumerator specifies if the category of the reference target shall be rendered with the xref.
Literal	***Description***
noShowCategory	The category of the target is **not** rendered at the place of the reference. **Tags:**atp.EnumerationLiteralIndex=0
showCategory	The category of the target is rendered at the place of the reference. **Tags:**atp.EnumerationLiteralIndex=1

Table 9.49: ShowResourceCategoryEnum

Enumeration	**ShowResourceLongNameEnum**
Package	M2::MSR::Documentation::TextModel::InlineAttributeEnums
Note	This enumerator specifies if the long name of the reference target shall be rendered with the xref.
Literal	***Description***
noShowLongName	The long name of the target is **not** rendered at the place of the reference. **Tags:**atp.EnumerationLiteralIndex=0
showLongName	The long name of the target is rendered at the place of the reference. **Tags:**atp.EnumerationLiteralIndex=1

Table 9.50: ShowResourceLongNameEnum

Enumeration	**ShowResourceNumberEnum**
Package	M2::MSR::Documentation::TextModel::InlineAttributeEnums
Note	This enumerator specifies if the number (e.g. chapter number) of the reference target shall be rendered with the xref.
Literal	***Description***

▽

△

Enumeration	**ShowResourceNumberEnum**
noShowNumber	The number of the target is **not** rendered at the place of the reference. **Tags:**atp.EnumerationLiteralIndex=0
showNumber	The number of the target is rendered at the place of the reference. **Tags:**atp.EnumerationLiteralIndex=1

Table 9.51: ShowResourceNumberEnum

Enumeration	**ShowResourcePageEnum**
Package	M2::MSR::Documentation::TextModel::InlineAttributeEnums
Note	This enumerator specifies if the page number of the reference target shall be rendered with the xref.
Literal	***Description***
noShowPage	The page number of the target is **not** rendered at the place of the reference. **Tags:**atp.EnumerationLiteralIndex=0
showPage	The page number of the target is rendered at the place of the reference. **Tags:**atp.EnumerationLiteralIndex=1

Table 9.52: ShowResourcePageEnum

Enumeration	**ShowResourceShortNameEnum**
Package	M2::MSR::Documentation::TextModel::InlineAttributeEnums
Note	This enumerator specifies if the short name of the reference target shall be rendered with the xref.
Literal	***Description***
noShowShortName	The short name of the target is **not** rendered at the place of the reference. **Tags:**atp.EnumerationLiteralIndex=0
showShortName	The short name of the target is rendered at the place of the reference. **Tags:**atp.EnumerationLiteralIndex=1

Table 9.53: ShowResourceShortNameEnum

Enumeration	**ShowResourceTypeEnum**
Package	M2::MSR::Documentation::TextModel::InlineAttributeEnums
Note	This enumerator specifies if the type (e.g. derived from the class) of the reference target shall be rendered with the xref.
Literal	***Description***
noShowType	The type of the target is **not** rendered at the place of the reference. **Tags:**atp.EnumerationLiteralIndex=0
showType	The type of the target is rendered at the place of the reference. **Tags:**atp.EnumerationLiteralIndex=1

Table 9.54: ShowResourceTypeEnum

Enumeration	**ShowSeeEnum**
Package	M2::MSR::Documentation::TextModel::InlineAttributeEnums
Note	This enumerator specifies if the word "see" shall be rendered before the xref.

▽

△

Enumeration	**ShowSeeEnum**
Literal	***Description***
noShowSee	The word "see" is **not** rendered before the reference. **Tags:**atp.EnumerationLiteralIndex=0
showSee	The word "see"is rendered before the reference. **Tags:**atp.EnumerationLiteralIndex=1

Table 9.55: ShowSeeEnum

9.3 Standalone Documentation

[TPS_GST_00325] Standalone Documentation ⌈The standalone documentation provides means to capture documentation independently of the structure of an AUTOSAR system. In order to achieve this, it extends the introduction by adding chapters, topics, visual tables, and generic parameter sets (prms). One could say, it wraps the DocumentationBlock in chapters, topics, tables.

In addition to this, it allows to refer to AUTOSAR-Objects, which are the context of the documentation.

It is also provided as Documentation which is an ARElement of its own rights allowing for a reference to the document's context.⌋*()*

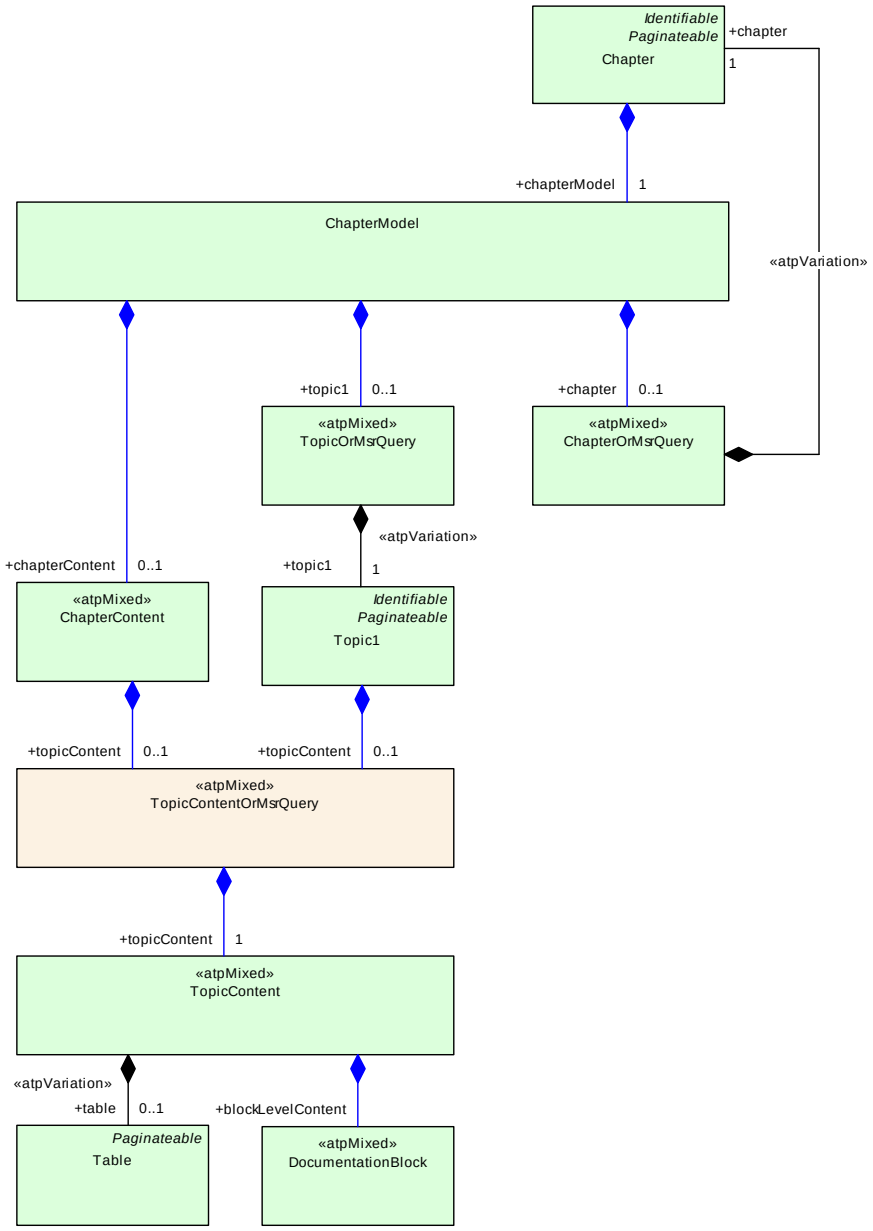


Figure 9.8: Standalone Documentation Overview

9.3.1 Documentation's Context

[TPS_GST_00326] Context of Standalone Documentation ⌈Standalone Documentation can specify a context to which it relates to. This context serves two purposes:

- reference targets to make the documentation self contained,
- support assembly of the complete documentation, e.g. within a project.

⌋()

Figure 9.9 depicts how the documentation context is captured. AUTOSAR provides `Documentation`, which is a `PackageableElement` and can be used to depict documentation in the context of any identifiable element or even M1 instance.

[constr_2533] Documentation context is either a feature or an identifiable ⌈One particular `DocumentationContext` shall be either a feature or an identifiable but not both at the same time. If this is desired, one should create multiple `DocumentationContext`.⌋*()*

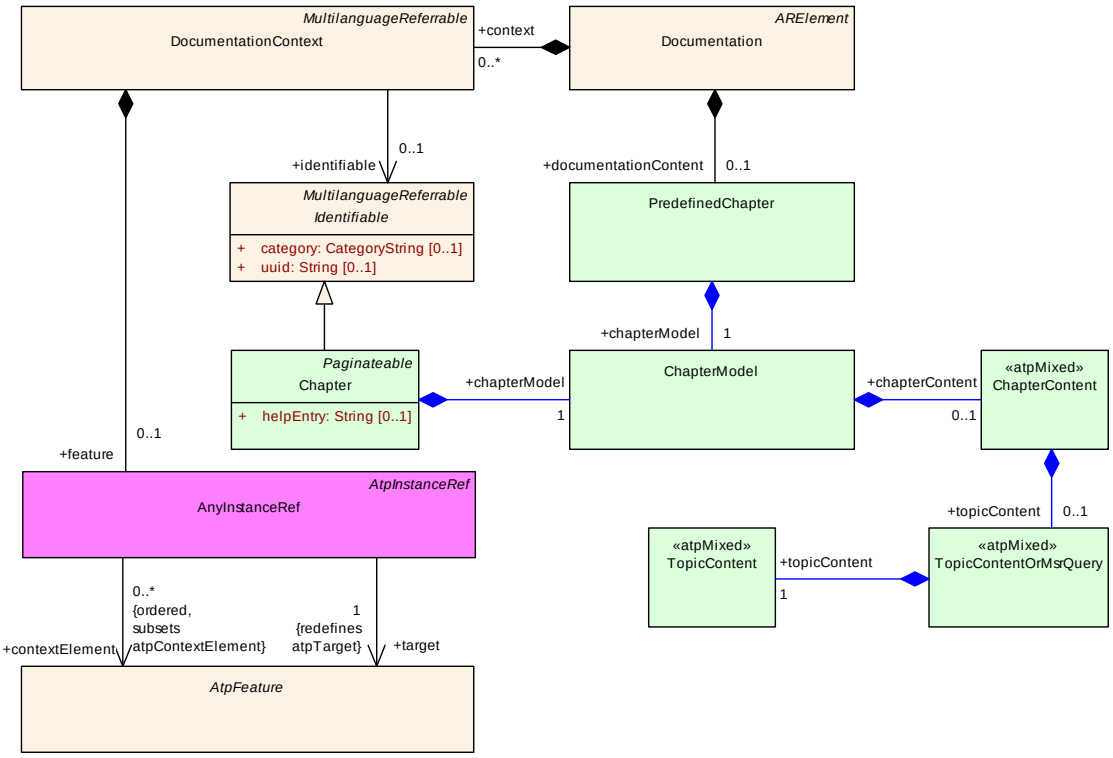


Figure 9.9: Standalone documentation as ARElement

Class	**DocumentationContext**			
Package	M2::AUTOSARTemplates::GenericStructure::DocumentationOnM1			
Note	This class represents the ability to denote a context of a so called standalone documentation. Note that this is an <<atpMixed>>. The contents needs to be considered as ordered.			
Base	*ARObject*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
feature	AtpFeature	0..1	iref	This refers to a particular feature (instance in the M0 model) to which is the context of the documentation. **InstanceRef implemented by:**AnyInstanceRef
identifiable	Identifiable	0..1	ref	This is an identifiable object which is part of the context of the documentation.

Table 9.56: DocumentationContext

Class	**AnyInstanceRef**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::AnyInstanceRef			
Note	Describes a reference to any instance in an AUTOSAR model. This is the most generic form of an instance ref. Refer to the superclass notes for more details.			
Base	*ARObject*, *AtpInstanceRef*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
base	AtpClassifier	1	ref	This is the base from which navigation path begins. **Stereotypes:** atpDerived
contextElement (ordered)	AtpFeature	*	ref	This is one step in the navigation path specified by the instance ref.
target	AtpFeature	1	ref	This is the target of the instance ref.

Table 9.57: AnyInstanceRef

9.3.2 Chapter

[TPS_GST_00327] Chapter ⌈The chapter element is composed of its caption provided by `Identifiable`, its immediate content `chapterContent` and more logical blocks grouped as topics `topic1` and sub chapters `chapter`. The chapter's content is composed of parameters, tables and documentation blocks.⌋*()*

[TPS_GST_00328] Predefined Chapter ⌈The `PredefinedChapter` is a chapter which cannot be nested because it depicts a particular semantics. Anyhow it can have nested chapters inside.⌋*()*

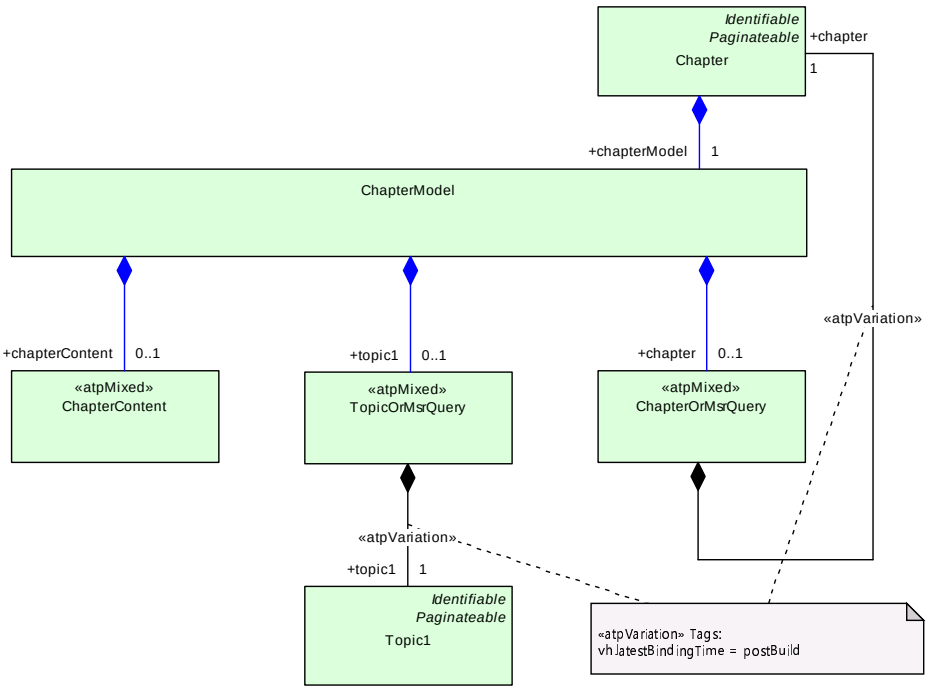


Figure 9.10: Chapter Overview

Class	**Chapter**			
Package	M2::MSR::Documentation::Chapters			
Note	This meta-class represents a chapter of a document. Chapters are the primary structuring element in documentation.			
Base	*ARObject*, *DocumentViewSelectable*, *Identifiable*, *MultilanguageReferrable*, *Paginateable*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
chapterModel	ChapterModel	1	aggr	This represents the overall contents of the chapter. **Tags:** xml.roleElement=false xml.roleWrapperElement=false xml.typeElement=false xml.typeWrapperElement=false
helpEntry	String	0..1	attr	This specifies an entry point in an online help system to be linked with the parent class. The syntax shall be defined by the applied help system respectively help system generator. Maybe it is a concatenated Identifier, but as of now we leave it as an arbitrary string. **Tags:**xml.attribute=true

Table 9.58: Chapter

Class	**ChapterModel**			
Package	M2::MSR::Documentation::Chapters			
Note	This is the basic content model of a chapter except the Chapter title. This can be utilized in general chapters as well as in predefined chapters. A chapter has content on three levels: 1. chapter content 2. topics 3. subchapters			
Base	*ARObject*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
chapter	ChapterOrMsrQuery	0..1	aggr	This is a particular subchapter. **Tags:** xml.roleElement=false xml.roleWrapperElement=false xml.sequenceOffset=200 xml.typeElement=false xml.typeWrapperElement=false
chapterContent	ChapterContent	0..1	aggr	This is the chapter content which is not a topic or a subchapter. It is the content which is directly in the chapter. **Tags:** xml.roleElement=false xml.roleWrapperElement=false xml.sequenceOffset=30 xml.typeElement=false xml.typeWrapperElement=false

△

Class	**ChapterModel**			
topic1	TopicOrMsrQuery	0..1	aggr	This is a topic within the chapter. **Tags:** xml.roleElement=false xml.roleWrapperElement=false xml.sequenceOffset=170 xml.typeElement=false xml.typeWrapperElement=false

Table 9.59: ChapterModel

Class	<<atpMixed>> **ChapterContent**			
Package	M2::MSR::Documentation::Chapters			
Note	This class represents the content which is directly in a chapter. It is basically the same as the one in a Topic but might have additional complex structures (e.g. Synopsis)			
Base	*ARObject*			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
prms	Prms	1	aggr	This is a parameter table within a chapter. **Tags:**xml.sequenceOffset=150
topicContent	TopicContentOrMsr Query	0..1	aggr	This is that part of a chapter content which may appear in a chapter as well as in a topic. **Tags:** xml.roleElement=false xml.roleWrapperElement=false xml.sequenceOffset=40 xml.typeElement=false xml.typeWrapperElement=false

Table 9.60: ChapterContent

Enumeration	**ChapterEnumBreak**
Package	M2::MSR::Documentation::BlockElements::PaginationAndView
Note	This allows to specify the page break policy of a paginatable element.
Literal	*Description*
break	This indicates the a page break shall be applied before the current block. **Tags:**atp.EnumerationLiteralIndex=0
noBreak	This indicates that there is no need to force a page break before this block. **Tags:**atp.EnumerationLiteralIndex=1

Table 9.61: ChapterEnumBreak

Class	**PredefinedChapter**			
Package	M2::MSR::Documentation::Chapters			
Note	This represents a predefined chapter.			
Base	*ARObject*			
Attribute	*Type*	*Mult.*	*Kind*	*Note*

▽

△

Class	**PredefinedChapter**			
chapterModel	ChapterModel	1	aggr	This is the content of the predefined chapter. **Tags:** xml.roleElement=false xml.roleWrapperElement=false xml.typeElement=false xml.typeWrapperElement=false

Table 9.62: PredefinedChapter

9.3.3 Tables in Documentation

[TPS_GST_00329] Tables in Documentation ⌈AUTOSAR supports to use tables in documentation by the meta-class `Table` which is an implementation of the Oasis exchange table model ([23]).⌋*()* The model is depicted in Figure 9.11.

[TPS_GST_00330] Partitions of a Table ⌈A table (`Table`) contains one or more partitions (`Tgroup`). The first partition has column specification (`Colspec`), which specifies the attributes of column within the partition.

Subsequent partitions can define their own column specification or inherit from the last partition that had a specification. A partition is composed of exactly one body (`tbody`) one optional header (`thead`) and one optional footer (`tfoot`).⌋*()*

[TPS_GST_00331] Table Row ⌈The table partition (body, header and footer) are composed of one or more rows (`Row`). On the level of table rows is possible to control the page breaks and view. Each Row is composed of one or more entries (`Entry`), which contains a documentation block.⌋*()*

A table can also have a caption.

`MultiLanguageOverviewParagraph` can be added to further describe the `Caption`

- this allows to provide more elaborate description of the related table,
- this allows to provide references to other items such as traces, related tables etc..

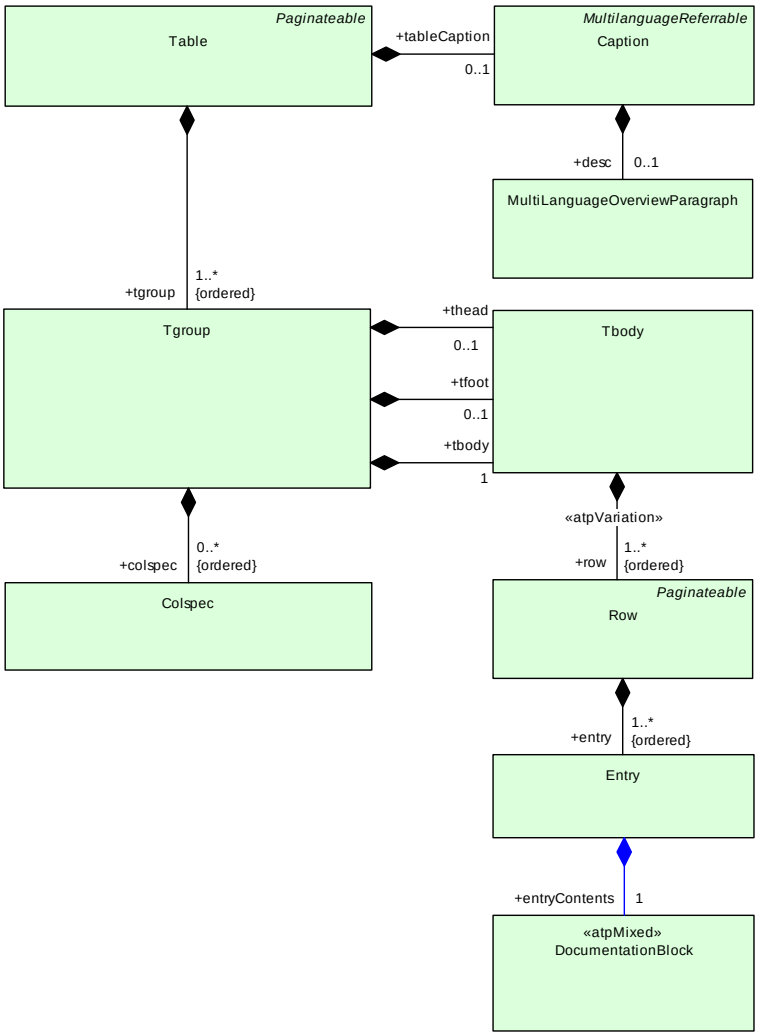


Figure 9.11: Table Model Overview

Class	**Table**			
Package	M2::MSR::Documentation::BlockElements::OasisExchangeTable			
Note	This class implements an exchange table according to OASIS Technical Resolution TR 9503:1995. http://www.oasis-open.org/specs/a503.htm			
Base	*ARObject*, *DocumentViewSelectable*, *Paginateable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
colsep	TableSeparatorString	0..1	attr	Indicates if by default a line should be drawn between the columns of this table. **Tags:**xml.attribute=true
float	FloatEnum	1	attr	Indicate whether it is allowed to break the element. **Tags:**xml.attribute=true
frame	FrameEnum	0..1	attr	Used to defined the frame line around a table. **Tags:**xml.attribute=true

▽

△

Class	Table			
helpEntry	String	0..1	attr	This specifies an entry point in an online help system to be linked with the parent class. The syntax shall be defined by the applied help system respectively help system generator. **Tags:**xml.attribute=true
orient	OrientEnum	0..1	attr	Indicate whether a table should be represented as landscape or portrait. • land : landscape • port : portrait **Tags:**xml.attribute=true
pgwide	NameToken	0..1	attr	Used to indicate whether the figure should take the complete page width (value = "pgwide") or not (value = "noPgwide"). **Tags:**xml.attribute=true
rowsep	TableSeparatorString	0..1	attr	Indicates if by default a line should be drawn at the bottom of table rows. **Tags:**xml.attribute=true
tableCaption	Caption	0..1	aggr	This element specifies the table heading. **Tags:**xml.sequenceOffset=20
tabstyle	NameToken	0..1	attr	Indicates an external table style. **Tags:**xml.attribute=true
tgroup (ordered)	Tgroup	1..*	aggr	A table can be built of individual segments. Such a segment is called tgroup. **Tags:** xml.roleElement=true xml.roleWrapperElement=false xml.sequenceOffset=30 xml.typeElement=false xml.typeWrapperElement=false

Table 9.63: Table

Enumeration	FloatEnum
Package	M2::MSR::Documentation::BlockElements::OasisExchangeTable
Note	This enumerator specifies the policy how an objects floats on a page.
Literal	*Description*
float	This indicates that a page formatter is allowed to float the table to optimize the pagination. This is for example supported by TeX. **Tags:**atp.EnumerationLiteralIndex=0
noFloat	This indicates that a page formatter is not allowed to float the object to optimize the pagination. **Tags:**atp.EnumerationLiteralIndex=1

Table 9.64: FloatEnum

Enumeration	FrameEnum
Package	M2::MSR::Documentation::BlockElements::OasisExchangeTable
Note	This enumerator specifies the policy, where to place a frame border around the table.
Literal	*Description*

▽

△

Enumeration	**FrameEnum**
all	Borders all around the table **Tags:**atp.EnumerationLiteralIndex=0
bottom	Border at the bottom of the table **Tags:**atp.EnumerationLiteralIndex=1
none	No borders around the table **Tags:**atp.EnumerationLiteralIndex=2
sides	Borders at the sides of the table **Tags:**atp.EnumerationLiteralIndex=3
top	Border at the top of the table **Tags:**atp.EnumerationLiteralIndex=4
topbot	Borders at the top and bottom of the table **Tags:**atp.EnumerationLiteralIndex=5

Table 9.65: FrameEnum

Class	**Tgroup**			
Package	M2::MSR::Documentation::BlockElements::OasisExchangeTable			
Note	This meta-class represents the ability to denote a table section.			
Base	*ARObject*			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
align	AlignEnum	0..1	attr	Specifies how the cell entries shall be horizontally aligned within the specified TGROUP. Default is "LEFT" **Tags:**xml.attribute=true
cols	Integer	1	attr	This attribute represents the number of columns in the table. **Tags:**xml.attribute=true
colsep	TableSeparatorString	0..1	attr	Indicates if by default a line shall be drawn between the columns of this table group. **Tags:**xml.attribute=true
colspec (ordered)	Colspec	*	aggr	This specifies one particular column specification in the table. There shall be one entry for each column. **Tags:** xml.roleElement=true xml.roleWrapperElement=false xml.sequenceOffset=20 xml.typeElement=false xml.typeWrapperElement=false
rowsep	TableSeparatorString	0..1	attr	Indicates if by default a line shall be drawn at the bottom of the rows in this table group. **Tags:**xml.attribute=true
tbody	Tbody	1	aggr	This is the main part of the table segment, called the table body. **Tags:**xml.sequenceOffset=60
tfoot	Tbody	0..1	aggr	This represents the footer of the table segment. This segment is printed at the end of the table or before a page break. **Tags:**xml.sequenceOffset=50

▽

△

Class	**Tgroup**			
thead	Tbody	0..1	aggr	This represents the heading of the table section. The heading is usually repeated at the beginning of each new page. **Tags:**xml.sequenceOffset=40

Table 9.66: Tgroup

Enumeration	**AlignEnum**
Package	M2::MSR::Documentation::BlockElements::OasisExchangeTable
Note	This enumerator specifies horizontal alignment.
Literal	*Description*
center	The content of the table is horizontally centered. **Tags:**atp.EnumerationLiteralIndex=0
justify	This indicates that the content of table cell shall be justified (rendered as a block where white-space is expanded such that all lines are filled up). **Tags:**atp.EnumerationLiteralIndex=1
left	This indicates that the content of a table cell is left justified. **Tags:**atp.EnumerationLiteralIndex=2
right	This indicates that the content of a table cell is left justified. **Tags:**atp.EnumerationLiteralIndex=3

Table 9.67: AlignEnum

Class	**Tbody**			
Package	M2::MSR::Documentation::BlockElements::OasisExchangeTable			
Note	This meta-class represents a part within a table group. Such a part can be the table head, the table body or the table foot.			
Base	*ARObject*			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
row (ordered)	Row	1..*	aggr	This is a particular row in a table. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=postBuild xml.roleElement=true xml.roleWrapperElement=false xml.sequenceOffset=20 xml.typeElement=false xml.typeWrapperElement=false
valign	ValignEnum	0..1	attr	Indicates how the cells in the rows shall be aligned. Default is inherited from tbody, otherwise it is "TOP" **Tags:**xml.attribute=true

Table 9.68: Tbody

Enumeration	**ValignEnum**
Package	M2::MSR::Documentation::BlockElements::OasisExchangeTable
Note	This enumerator specifies vertical alignment.

▽

△

Enumeration	**ValignEnum**
Literal	***Description***
bottom	The contents of the table cell is bottom aligned. **Tags:**atp.EnumerationLiteralIndex=0
middle	The contents of the table is vertically centered. **Tags:**atp.EnumerationLiteralIndex=1
top	The contents of the table cell is top aligned. **Tags:**atp.EnumerationLiteralIndex=2

Table 9.69: ValignEnum

Class	**Row**			
Package	M2::MSR::Documentation::BlockElements::OasisExchangeTable			
Note	This meta-class represents the ability to express one row in a table.			
Base	*ARObject*, *DocumentViewSelectable*, *Paginateable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
entry (ordered)	Entry	1..*	aggr	This represents one particular table cell. It is an entry in the table. **Tags:** xml.roleElement=true xml.roleWrapperElement=false xml.sequenceOffset=20 xml.typeElement=false xml.typeWrapperElement=false
rowsep	TableSeparatorString	0..1	attr	Indicates if by default a line should be displayed below the row. **Tags:**xml.attribute=true
valign	ValignEnum	0..1	attr	Indicates how the cells in the rows shall be aligned. Default is inherited from tbody, otherwise it is "TOP" **Tags:**xml.attribute=true

Table 9.70: Row

Class	**Entry**			
Package	M2::MSR::Documentation::BlockElements::OasisExchangeTable			
Note	This represents one particular table cell.			
Base	*ARObject*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
align	AlignEnum	0..1	attr	Specifies how the cell ENTRY shall be horizontally aligned. Default is "LEFT" **Tags:**xml.attribute=true
bgcolor	String	1	attr	This allows to recommend a background color of the entry. It is specified bases on 6 digits RGB hex-code. **Tags:**xml.attribute=true
colname	String	0..1	attr	Indicate the name of the column, where the entry should appear. **Tags:**xml.attribute=true

▽

△

Class	Entry			
colsep	TableSeparatorString	0..1	attr	Indicates whether a line should be displayed end of this entry. **Tags:**xml.attribute=true
entryContents	DocumentationBlock	1	aggr	This is the content of the TableEntry **Tags:** xml.roleElement=false xml.roleWrapperElement=false xml.typeElement=false xml.typeWrapperElement=false
morerows	String	0..1	attr	Number of additional rows. Default is "0" **Tags:**xml.attribute=true
nameend	String	0..1	attr	When an entry spans multiple column this is the name of the last column. **Tags:**xml.attribute=true
namest	String	0..1	attr	When an entry spans multiple column this is the name of the first column. **Tags:**xml.attribute=true
rotate	String	0..1	attr	Indicates if the cellcontent shall be rotated. Default is 0; 1 would rotate the contents 90 degree counterclockwise. This attribute is defined by OASIS. **Tags:**xml.attribute=true
rowsep	TableSeparatorString	0..1	attr	Indicates whether a line should be displayed at the bottom end of the cell. **Tags:**xml.attribute=true
spanname	String	0..1	attr	Capture the name of entry merging multiple columns. **Tags:**xml.attribute=true
valign	ValignEnum	0..1	attr	Indicates how the content of the cell shall be aligned. Default is inherited from row or tbody, otherwise "TOP" **Tags:**xml.attribute=true

Table 9.71: Entry

Primitive	TableSeparatorString
Package	M2::MSR::Documentation::BlockElements::OasisExchangeTable
Note	This represents the ability to denote a separator string within an OASIS exchange table. • **0**: no line is displayed • **1**: line is displayed **Tags:** xml.xsd.customType=TABLE-SEPARATOR-STRING xml.xsd.pattern=[0-1] xml.xsd.type=string

Table 9.72: TableSeparatorString

9.3.4 Topics in Documentation

[TPS_GST_00332] Topics in Documentation ⌈A topic (Topic1)[7] is a logical unit, which subdivides a content of a chapter mainly by providing intermediate head lines. Note that these head lines are not part of generated table of contents.⌋*()*

Class	**Topic1**			
Package	M2::MSR::Documentation::Chapters			
Note	This meta-class represents a topic of a documentation. Topics are similar to chapters but they cannot be nested. They also do not appear in the table of content. Topics can be used to produce intermediate headlines thus structuring a chapter internally.			
Base	*ARObject*, *DocumentViewSelectable*, *Identifiable*, *MultilanguageReferrable*, *Paginateable*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
helpEntry	String	0..1	attr	This specifies an entry point in an online help system to be linked with the parent class. The syntax shall be defined by the applied help system respectively help system generator. **Tags:**xml.attribute=true
topicContent	TopicContentOrMsrQuery	0..1	aggr	This is the content of the topic. **Tags:** xml.roleElement=false xml.roleWrapperElement=false xml.sequenceOffset=20 xml.typeElement=false xml.typeWrapperElement=false

Table 9.73: Topic1

9.3.5 Parameter tables

[TPS_GST_00333] Parameter Tables ⌈Parameter tables can be used to collect numerical or textual parameters in a documentation. Such parameters should not to be confused with parameters in the software of an ECU. Parameter tables are intended to create a kind of data sheets.⌋*()*

Class	**Prms**			
Package	M2::MSR::Documentation::BlockElements::GerneralParameters			
Note	This metaclass represents the ability to specify a parameter table. It can be used e.g. to specify parameter tables in a data sheet.			
Base	*ARObject*, *DocumentViewSelectable*, *Paginateable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
label	MultilanguageLongName	0..1	aggr	This represents the caption of the parameter table. **Tags:**xml.sequenceOffset=20

▽

[7] The name topic1 is given to remain compatible with [22]

△

Class	**Prms**			
prm	GeneralParameter	1..*	aggr	This represents one particular parameter in the table. **Tags:** xml.roleElement=true xml.roleWrapperElement=false xml.sequenceOffset=30 xml.typeElement=false xml.typeWrapperElement=false

Table 9.74: Prms

9.4 Document production

Production of e.g. printed documents is done using document processors. These processors determine the document content and layout. Nevertheless it is necessary to support this document production process by some policies in order to tweak the final output.

[TPS_GST_00334] Support of Pagination of Documents ⌈AUTOSAR provides basic support for a pagination policy (Paginateable) which allows to tweak the page breaks of generated documents.⌋ *()*

Class	***Paginateable*** (abstract)			
Package	M2::MSR::Documentation::BlockElements::PaginationAndView			
Note	This meta-class represents the ability to control the pagination policy when creating documents.			
Base	*ARObject*, *DocumentViewSelectable*			
Subclasses	Chapter, DefItem, DefList, Item, LabeledItem, LabeledList, List, MlFigure, MlFormula, MsrQueryChapter, MsrQueryP1, MsrQueryTopic1, MultiLanguageParagraph, MultiLanguageVerbatim, Note, Prms, Row, StructuredReq, Table, Topic1, TraceableTable, TraceableText			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
break	ChapterEnumBreak	0..1	attr	This attributes allows to specify a forced page break. **Tags:**xml.attribute=true
keepWith Previous	KeepWithPreviousEnum	0..1	attr	This attribute denotes the pagination policy. In particular it defines if the containing text block shall be kept together with the previous block. **Tags:**xml.attribute=true

Table 9.75: Paginateable

see also ChapterEnumBreak

Enumeration	**KeepWithPreviousEnum**
Package	M2::MSR::Documentation::BlockElements::PaginationAndView
Note	This enumerator specifies a page break policy by controlling blocks which shall be kept together.
Literal	***Description***

▽

△

Enumeration	**KeepWithPreviousEnum**
keep	This indicates that the block shall be kept together with the previous block. **Tags:**atp.EnumerationLiteralIndex=0
noKeep	This indicates that there is no need to keep the block with the previous one. This is the same as if the attribute itself is missing. **Tags:**atp.EnumerationLiteralIndex=1

Table 9.76: KeepWithPreviousEnum

Class	***DocumentViewSelectable*** (abstract)			
Package	M2::MSR::Documentation::BlockElements::PaginationAndView			
Note	This meta-class represents the ability to be dedicated to a particular audience or document view.			
Base	*ARObject*			
Subclasses	*Paginateable*			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
si	NameTokens	1	attr	This attribute allows to denote a semantic information which is used to identify documentation objects to be selected in customizable document views. It shall be defined in agreement between the involved parties. **Tags:**xml.attribute=true
view	ViewTokens	0..1	attr	This attribute lists the document views in which the object shall appear. If it is missing, the object appears in all document views. **Tags:**xml.attribute=true

Table 9.77: DocumentViewSelectable

Primitive	**ViewTokens**
Package	M2::MSR::Documentation::BlockElements::PaginationAndView
Note	This primitive specifies the tokens to specify a documentation view. **Tags:** xml.xsd.customType=VIEW-TOKENS xml.xsd.pattern=(-?[a-zA-Z_]+)(( )+-?[a-zA-Z_]+)* xml.xsd.type=string

Table 9.78: ViewTokens

9.5 Including generated documentation parts

[TPS_GST_00336] Including generated Documentation Parts ⌈AUTOSAR supports an approach where parts of the documentation are automatically generated and included at a particular location within the documentation.

This support is provided by the so called MSR query mechanism (`MsrQueryP1`, `MsrQueryP2`, `MsrQueryTopic1` and `MsrQueryChapter`). These classes allow to represent the properties of the inclusion as well as the result of the inclusion. Thereby the intermediate results can be visualized and exchanged after the generated parts

were included. Hence it is not necessary that all parties involved in the project are able to perform the inclusion process.

Details are subject to mutual agreement.⌋()

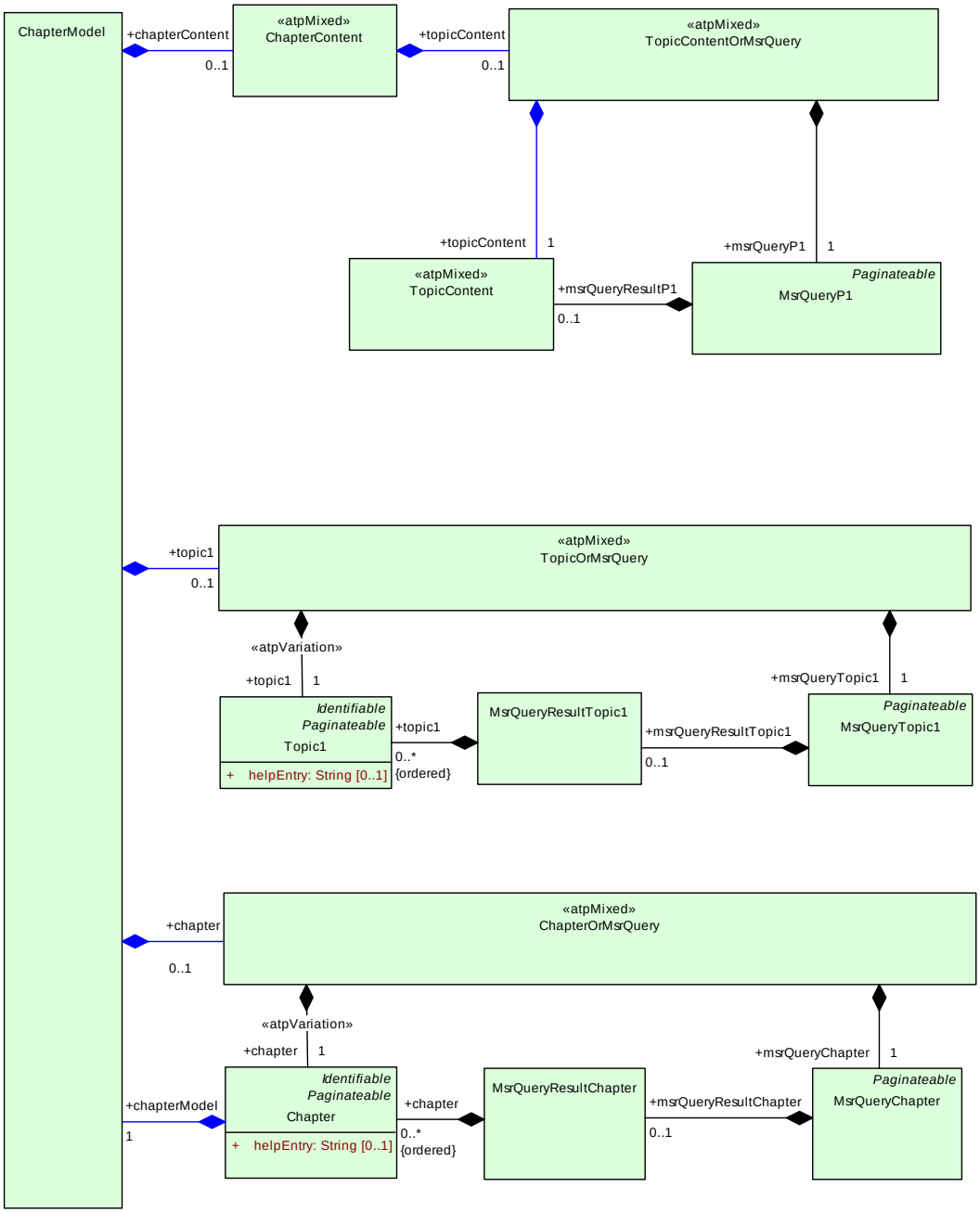


Figure 9.12: Including generated documentation parts by MsrQuery

The following meta-classes represent the alternative of manually edited documentation and included generated parts.

Class	<<atpMixed>> **TopicContentOrMsrQuery**			
Package	M2::MSR::Documentation::Chapters			
Note	This meta-class represents a topic or a topic content which is generated using queries.			
Base	*ARObject*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
msrQueryP1	MsrQueryP1	1	aggr	This represents automatically contributed contents provided by an msrquery.
topicContent	TopicContent	1	aggr	This is the content of a topic. **Tags:**xml.roleElement=false

Table 9.79: TopicContentOrMsrQuery

Class	<<atpMixed>> **TopicOrMsrQuery**			
Package	M2::MSR::Documentation::Chapters			
Note	This class provides the alternative of a Topic with an MsrQuery which delivers a topic.			
Base	*ARObject*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
msrQuery Topic1	MsrQueryTopic1	1	aggr	This represents automatically contributed topics provided by an msrquery. **Tags:**xml.sequenceOffset=190
topic1	Topic1	1	aggr	This is used to create particular topics within a chapter. A topic is similar to a subchapter, but cannot be nested and will not appear in the table of contents of the document. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=postBuild xml.sequenceOffset=180

Table 9.80: TopicOrMsrQuery

Class	<<atpMixed>> **ChapterOrMsrQuery**			
Package	M2::MSR::Documentation::Chapters			
Note	This meta-class represents the ability to denote a particular chapter or a query returning a chapter.			
Base	*ARObject*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
chapter	Chapter	1	aggr	This establishes a subschapter. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=postBuild xml.sequenceOffset=210
msrQuery Chapter	MsrQueryChapter	1	aggr	This represents automatically contributed chapters provided by an msrquery. **Tags:**xml.sequenceOffset=220

Table 9.81: ChapterOrMsrQuery

The following meta-classes represent the included generated parts.

Class	MsrQueryP1			
Package	M2::MSR::Documentation::MsrQuery			
Note	This meta-class represents the ability to express a query which yields the content of a topic as a result.			
Base	*ARObject*, *DocumentViewSelectable*, *Paginateable*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
msrQueryProps	MsrQueryProps	1	aggr	This is argument and properties of the paragraph query. **Tags:**xml.sequenceOffset=20
msrQueryResultP1	TopicContent	0..1	aggr	This represents the result of the query. **Tags:**xml.sequenceOffset=30

Table 9.82: MsrQueryP1

Class	MsrQueryTopic1			
Package	M2::MSR::Documentation::MsrQuery			
Note	This meta-class represents the ability to specify a query which yields a set of topics as a result.			
Base	*ARObject*, *DocumentViewSelectable*, *Paginateable*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
msrQueryProps	MsrQueryProps	1	aggr	This is argument and properties of the topic query. **Tags:**xml.sequenceOffset=20
msrQueryResultTopic1	MsrQueryResultTopic1	0..1	aggr	This represents the result of the query. **Tags:**xml.sequenceOffset=30

Table 9.83: MsrQueryTopic1

Class	MsrQueryChapter			
Package	M2::MSR::Documentation::MsrQuery			
Note	This meta-class represents the ability to express a query which yields a set of chapters as a result.			
Base	*ARObject*, *DocumentViewSelectable*, *Paginateable*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
msrQueryProps	MsrQueryProps	1	aggr	This is argument and properties of the chapter query. **Tags:**xml.sequenceOffset=20
msrQueryResultChapter	MsrQueryResultChapter	0..1	aggr	This represents the result of the query. **Tags:**xml.sequenceOffset=30

Table 9.84: MsrQueryChapter

The following meta-classes control the inclusion process.

Class	MsrQueryProps			
Package	M2::MSR::Documentation::MsrQuery			
Note	This metaclass represents the ability to specificy a query which yields some documentation text. The qualities of the result are determined by the context in which the query is used.			
Base	*ARObject*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
comment	String	0..1	attr	This element contains a commentary in text form. **Tags:**xml.sequenceOffset=40

▽

△

Class	MsrQueryProps			
msrQueryArg	MsrQueryArg	*	aggr	This element specifies an argument within an MsrQuery. **Tags:** xml.roleElement=true xml.roleWrapperElement=false xml.sequenceOffset=30 xml.typeElement=false xml.typeWrapperElement=false
msrQueryName	String	1	attr	This element specifies the name of the MSR-QUERY triggered. **Tags:**xml.sequenceOffset=20

Table 9.85: MsrQueryProps

Class	MsrQueryArg			
Package	M2::MSR::Documentation::MsrQuery			
Note	This represents an argument to the query. Note that the arguments are not standardized and therefore subject to mutual agreement.			
Base	*ARObject*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
arg	String	1	attr	This is the value of the argument. **Tags:** xml.roleElement=false xml.roleWrapperElement=false xml.typeElement=false xml.typeWrapperElement=false
si	NameToken	1	attr	This denotes the name of the query argument (semantic information) **Tags:**xml.attribute=true

Table 9.86: MsrQueryArg

The following meta-classes represent the included results.

Class	MsrQueryResultChapter			
Package	M2::MSR::Documentation::MsrQuery			
Note	This metaclass represents the result of an msrquery which is a set of chapters.			
Base	*ARObject*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
chapter (ordered)	Chapter	*	aggr	This is one particular chapter in the query result. **Tags:** xml.roleElement=true xml.roleWrapperElement=false xml.sequenceOffset=20 xml.typeElement=false xml.typeWrapperElement=false

Table 9.87: MsrQueryResultChapter

Class	**MsrQueryResultTopic1**			
Package	M2::MSR::Documentation::MsrQuery			
Note	This metaclass represents the ability to express the result of a query which is a set of topics.			
Base	*ARObject*			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
topic1 (ordered)	Topic1	*	aggr	This represents one particular topic in the query result. **Tags:** xml.roleElement=true xml.roleWrapperElement=false xml.sequenceOffset=20 xml.typeElement=false xml.typeWrapperElement=false

Table 9.88: MsrQueryResultTopic1

9.6 Handling Multiple Languages in an AUTOSAR Artifact

[TPS_GST_00337] Multiple Languages ⌈AUTOSAR supports a multi-language documentation[8], where each construct of the document can have the optional attribute `l` in `LanguageSpecific` with a value that denotes the language in which it is written. The languages available in a document are defined in `adminData`.⌋*()*

See Chapter 3 for details.

The following block level elements and their sub-constructs support multilingual text/-graphics:

- paragraph
- formula
- figure
- verbatim

Figure 9.13 illustrates the approach for multi-language support using LongName as an example. It shows how the single language and the multi-language version are derived from a common model.

[8]based on [22]

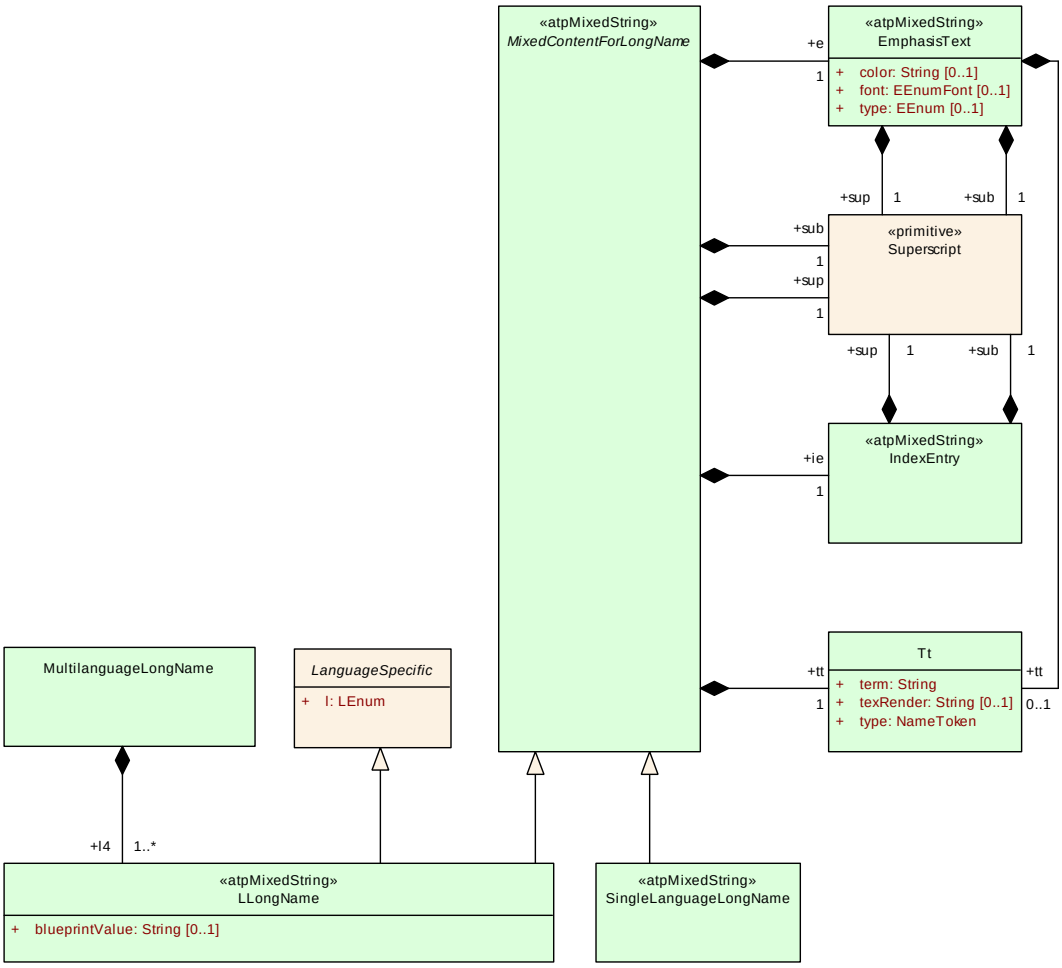


Figure 9.13: The `MultilanguageLongname`

The annotation of each construct with the language it which it is written allows the author of a documentation block to write interlinear in two or more languages weaved together in a single source document, then to generate the documentation block in each of these languages.

[constr_2523] Used languages need to be consistent ⌈The used languages of an AUTOSAR file are specified in the top level `adminData`. All other elements shall be provided in the languages specified for the document.⌋*()* See Chapter 3 for more details.

This approach supports a better maintainability. In a documentation block written in English, French and German, for example, it is easier to maintain the three versions consistent when each paragraph is immediately followed by its translation in the other languages as when they are found at completely different locations or worse separated documents.

The following example illustrate this approach:

Listing 9.2: Example of Excerpt from a Multilingual Documentation Block in ARXML

```
<AR-PACKAGE>
  <SHORT-NAME>Documentation</SHORT-NAME>
```

```
    <INTRODUCTION>
      <P>
        <L-1 L="EN">
          In a documentation block written  in English,
          French and German, for example, it is easier
          to maintain the three versions consistent
          when each paragraph is immediately followed
          by its translation in the other languages as
          when they are found at completely different
          locations or worse separated documents.
          </L-1>
        <L-1 L="FR">
          Dans un bloque documentaire écrit en anglais,
          français et allemand, par exemple, il est plus
          facile de maintenir les trois versions
          consistantes entre elles, quand chaque paragraphe
          est suivi immédiatement de sa traduction dans les
          autres langues que si ces dernières ce trouvaient
          dans des endroits différents ou même pire dans des
          documents séparés.
          </L-1>
      </P>
    </INTRODUCTION>
  </AR-PACKAGE>
```

see also MultiLanguageVerbatim

Class	<<atpMixedString>> **LVerbatim**			
Package	M2::MSR::Documentation::TextModel::LanguageDataModel			
Note	MixedContentForVerbatim in one particular language. The language is denoted in the attribute l.			
Base	*ARObject*, *LanguageSpecific*, *MixedContentForVerbatim*, *WhitespaceControlled*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table 9.89: LVerbatim

Class	**MultiLanguageOverviewParagraph**			
Package	M2::MSR::Documentation::TextModel::MultilanguageData			
Note	This is the content of a multilingual paragraph in an overview item.			
Base	*ARObject*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
l2	LOverviewParagraph	1..*	aggr	This represents the text in one particular language. **Tags:** xml.roleElement=true xml.roleWrapperElement=false xml.sequenceOffset=20 xml.typeElement=false xml.typeWrapperElement=false

Table 9.90: MultiLanguageOverviewParagraph

Class	<<atpMixedString>> **LOverviewParagraph**			
Package	M2::MSR::Documentation::TextModel::LanguageDataModel			
Note	MixedContentForOverviewParagraph in one particular language. The language is denoted in the attribute l.			
Base	*ARObject*, *LanguageSpecific*, *MixedContentForOverviewParagraph*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
blueprintValue	String	0..1	attr	This represents a description that documents how the value shall be defined when deriving objects from the blueprint. **Tags:** atp.Status=draft xml.attribute=true

Table 9.91: LOverviewParagraph

see also `MultiLanguageParagraph`

Class	<<atpMixedString>> **LParagraph**			
Package	M2::MSR::Documentation::TextModel::LanguageDataModel			
Note	This is the text for a paragraph in one particular language. The language is denoted in the attribute l.			
Base	*ARObject*, *LanguageSpecific*, *MixedContentForParagraph*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table 9.92: LParagraph

Enumeration	**PgwideEnum**
Package	M2::MSR::Documentation::BlockElements::OasisExchangeTable
Note	This enumerator specifies, if the table shall be rendered across the entire page, even if it is placed in side-head layouts.
Literal	***Description***
noPgwide	This indicates that the table shall be fit in the current text flow. **Tags:**atp.EnumerationLiteralIndex=0
pgwide	This indicates that the table may use the entire page width. This is in particular important in case of so called "side-head layouts" but also if the table is in a list or in a note. **Tags:**atp.EnumerationLiteralIndex=1

Table 9.93: PgwideEnum

Class	<<atpMixedString>> ***MixedContentForPlainText*** (abstract)			
Package	M2::MSR::Documentation::TextModel::InlineTextModel			
Note	This represents a plain text which conceptually is handled as mixed contents. It is modeled as such for symmetry reasons.			
Base	*ARObject*, *WhitespaceControlled*			
Subclasses	LPlainText			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table 9.94: MixedContentForPlainText

Class	**MultiLanguagePlainText**			
Package	M2::MSR::Documentation::TextModel::MultilanguageData			
Note	This is a multilingual plaint Text.It is intended to be rendered as a paragraph.			
Base	*ARObject*			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
l10	LPlainText	1..*	aggr	This is the plain text in one particular language. **Tags:** xml.roleElement=true xml.roleWrapperElement=false xml.sequenceOffset=20 xml.typeElement=false xml.typeWrapperElement=false

Table 9.95: MultiLanguagePlainText

Class	<<atpMixedString>> **LPlainText**			
Package	M2::MSR::Documentation::TextModel::LanguageDataModel			
Note	This represents plain string in one particular language. The language is denoted in the attribute l.			
Base	*ARObject*, *LanguageSpecific*, *MixedContentForPlainText*, *WhitespaceControlled*			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
–	–	–	–	–

Table 9.96: LPlainText

Class	***LanguageSpecific*** (abstract)			
Package	M2::MSR::Documentation::TextModel::LanguageDataModel			
Note	This meta-class represents the ability to denote a particular language for which an object is applicable.			
Base	*ARObject*			
Subclasses	LGraphic, LLongName, LOverviewParagraph, LParagraph, LPlainText, LVerbatim			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
l	LEnum	1	attr	'This attribute denotes the language in which the language specific document entity is given. Note that "FOR-ALL" means, that the entity is applicable to all languages. It is language neutral. It follows ISO 639-1:2002 and is specified in upper case. **Tags:** xml.attribute=true xml.enforceMinMultiplicity=true

Table 9.97: LanguageSpecific

Enumeration	**LEnum**
Package	M2::MSR::Documentation::TextModel::LanguageDataModel
Note	This denotes the possible language designators according to the two letter code of ISO 693.
Literal	*Description*
aa	Afar **Tags:**atp.EnumerationLiteralIndex=0

▽

△

Enumeration	**LEnum**
ab	Abkhazian **Tags:**atp.EnumerationLiteralIndex=1
af	Afrikaans **Tags:**atp.EnumerationLiteralIndex=2
am	Amharic **Tags:**atp.EnumerationLiteralIndex=3
ar	Arabic **Tags:**atp.EnumerationLiteralIndex=4
as	Assamese **Tags:**atp.EnumerationLiteralIndex=5
ay	Aymara **Tags:**atp.EnumerationLiteralIndex=6
az	Azerbaijani **Tags:**atp.EnumerationLiteralIndex=7
ba	Bashkir **Tags:**atp.EnumerationLiteralIndex=8
be	Byelorussian **Tags:**atp.EnumerationLiteralIndex=9
bg	Bulgarian **Tags:**atp.EnumerationLiteralIndex=10
bh	Bihari **Tags:**atp.EnumerationLiteralIndex=11
bi	Bislama **Tags:**atp.EnumerationLiteralIndex=12
bn	Bengali **Tags:**atp.EnumerationLiteralIndex=13
bo	Tibetian **Tags:**atp.EnumerationLiteralIndex=14
br	Breton **Tags:**atp.EnumerationLiteralIndex=15
ca	Catalan **Tags:**atp.EnumerationLiteralIndex=16
co	Corsican **Tags:**atp.EnumerationLiteralIndex=17
cs	Czech **Tags:**atp.EnumerationLiteralIndex=18
cy	Welsh **Tags:**atp.EnumerationLiteralIndex=19
da	Danish **Tags:**atp.EnumerationLiteralIndex=20
de	German **Tags:**atp.EnumerationLiteralIndex=21
dz	Bhutani **Tags:**atp.EnumerationLiteralIndex=22

▽

△

Enumeration	**LEnum**
el	Greek **Tags:**atp.EnumerationLiteralIndex=23
en	English **Tags:**atp.EnumerationLiteralIndex=24
eo	Esperanto **Tags:**atp.EnumerationLiteralIndex=25
es	Spanish **Tags:**atp.EnumerationLiteralIndex=26
et	Estonian **Tags:**atp.EnumerationLiteralIndex=27
eu	Basque **Tags:**atp.EnumerationLiteralIndex=28
fa	Persian **Tags:**atp.EnumerationLiteralIndex=29
fi	Finnish **Tags:**atp.EnumerationLiteralIndex=30
fj	Fiji **Tags:**atp.EnumerationLiteralIndex=31
fo	Faeroese **Tags:**atp.EnumerationLiteralIndex=32
forAll	The content applies to all languages **Tags:**atp.EnumerationLiteralIndex=33
fr	French **Tags:**atp.EnumerationLiteralIndex=34
fy	Frisian **Tags:**atp.EnumerationLiteralIndex=35
ga	Irish **Tags:**atp.EnumerationLiteralIndex=36
gd	Scots Gaelic **Tags:**atp.EnumerationLiteralIndex=37
gl	Galician **Tags:**atp.EnumerationLiteralIndex=38
gn	Guarani **Tags:**atp.EnumerationLiteralIndex=39
gu	Gjarati **Tags:**atp.EnumerationLiteralIndex=40
ha	Hausa **Tags:**atp.EnumerationLiteralIndex=41
hi	Hindi **Tags:**atp.EnumerationLiteralIndex=42
hr	Croatian **Tags:**atp.EnumerationLiteralIndex=43
hu	Hungarian **Tags:**atp.EnumerationLiteralIndex=44

▽

△

Enumeration	LEnum
hy	Armenian **Tags:**atp.EnumerationLiteralIndex=45
ia	Interlingua **Tags:**atp.EnumerationLiteralIndex=46
ie	Interlingue **Tags:**atp.EnumerationLiteralIndex=47
ik	Inupiak **Tags:**atp.EnumerationLiteralIndex=48
in	Indonesian **Tags:**atp.EnumerationLiteralIndex=49
is	Icelandic **Tags:**atp.EnumerationLiteralIndex=50
it	Italian **Tags:**atp.EnumerationLiteralIndex=51
iw	Hebrew **Tags:**atp.EnumerationLiteralIndex=52
ja	Japanese **Tags:**atp.EnumerationLiteralIndex=53
ji	Yiddish **Tags:**atp.EnumerationLiteralIndex=54
jw	Javanese **Tags:**atp.EnumerationLiteralIndex=55
ka	Georgian **Tags:**atp.EnumerationLiteralIndex=56
kk	Kazakh **Tags:**atp.EnumerationLiteralIndex=57
kl	Greenlandic **Tags:**atp.EnumerationLiteralIndex=58
km	Cambodian **Tags:**atp.EnumerationLiteralIndex=59
kn	Kannada **Tags:**atp.EnumerationLiteralIndex=60
ko	Korean **Tags:**atp.EnumerationLiteralIndex=61
ks	Kashmiri **Tags:**atp.EnumerationLiteralIndex=62
ku	Kurdish **Tags:**atp.EnumerationLiteralIndex=63
ky	Kirghiz **Tags:**atp.EnumerationLiteralIndex=64
la	Latin **Tags:**atp.EnumerationLiteralIndex=65
ln	Lingala **Tags:**atp.EnumerationLiteralIndex=66

▽

△

Enumeration	LEnum
lo	Laothian **Tags:**atp.EnumerationLiteralIndex=67
lt	Lithuanian **Tags:**atp.EnumerationLiteralIndex=68
lv	Lavian, Lettish **Tags:**atp.EnumerationLiteralIndex=69
mg	Malagasy **Tags:**atp.EnumerationLiteralIndex=70
mi	Maori **Tags:**atp.EnumerationLiteralIndex=71
mk	Macedonian **Tags:**atp.EnumerationLiteralIndex=72
ml	Malayalam **Tags:**atp.EnumerationLiteralIndex=73
mn	Mongolian **Tags:**atp.EnumerationLiteralIndex=74
mo	Moldavian **Tags:**atp.EnumerationLiteralIndex=75
mr	Marathi **Tags:**atp.EnumerationLiteralIndex=76
ms	Malay **Tags:**atp.EnumerationLiteralIndex=77
mt	Maltese **Tags:**atp.EnumerationLiteralIndex=78
my	Burmese **Tags:**atp.EnumerationLiteralIndex=79
na	Nauru **Tags:**atp.EnumerationLiteralIndex=80
ne	Nepali **Tags:**atp.EnumerationLiteralIndex=81
nl	Dutch **Tags:**atp.EnumerationLiteralIndex=82
no	Norwegian **Tags:**atp.EnumerationLiteralIndex=83
oc	Occitan **Tags:**atp.EnumerationLiteralIndex=84
om	(Afan) Oromo **Tags:**atp.EnumerationLiteralIndex=85
or	Oriya **Tags:**atp.EnumerationLiteralIndex=86
pa	Punjabi **Tags:**atp.EnumerationLiteralIndex=87
pl	Polish **Tags:**atp.EnumerationLiteralIndex=88

▽

△

Enumeration	**LEnum**
ps	Pashto, Pushto **Tags:**atp.EnumerationLiteralIndex=89
pt	Portuguese **Tags:**atp.EnumerationLiteralIndex=90
qu	Quechua **Tags:**atp.EnumerationLiteralIndex=91
rm	Rhaeto-Romance **Tags:**atp.EnumerationLiteralIndex=92
rn	Kirundi **Tags:**atp.EnumerationLiteralIndex=93
ro	Romanian **Tags:**atp.EnumerationLiteralIndex=94
ru	Russian **Tags:**atp.EnumerationLiteralIndex=95
rw	Kinyarwanda **Tags:**atp.EnumerationLiteralIndex=96
sa	Sanskrit **Tags:**atp.EnumerationLiteralIndex=97
sd	Sindhi **Tags:**atp.EnumerationLiteralIndex=98
sg	Sangro **Tags:**atp.EnumerationLiteralIndex=99
sh	Serbo-Croatian **Tags:**atp.EnumerationLiteralIndex=100
si	Singhalese **Tags:**atp.EnumerationLiteralIndex=101
sk	Slovak **Tags:**atp.EnumerationLiteralIndex=102
sl	Slovenian **Tags:**atp.EnumerationLiteralIndex=103
sm	Samoan **Tags:**atp.EnumerationLiteralIndex=104
sn	Shona **Tags:**atp.EnumerationLiteralIndex=105
so	Somali **Tags:**atp.EnumerationLiteralIndex=106
sq	Albanian **Tags:**atp.EnumerationLiteralIndex=107
sr	Serbian **Tags:**atp.EnumerationLiteralIndex=108
ss	Siswati **Tags:**atp.EnumerationLiteralIndex=109
st	Sesotho **Tags:**atp.EnumerationLiteralIndex=110

▽

△

Enumeration	LEnum
su	Sundanese **Tags:**atp.EnumerationLiteralIndex=111
sv	Swedish **Tags:**atp.EnumerationLiteralIndex=112
sw	Swahili **Tags:**atp.EnumerationLiteralIndex=113
ta	Tamil **Tags:**atp.EnumerationLiteralIndex=114
te	Tegulu **Tags:**atp.EnumerationLiteralIndex=115
tg	Tajik **Tags:**atp.EnumerationLiteralIndex=116
th	Thai **Tags:**atp.EnumerationLiteralIndex=117
ti	Tigrinya **Tags:**atp.EnumerationLiteralIndex=118
tk	Turkmen **Tags:**atp.EnumerationLiteralIndex=119
tl	Tagalog **Tags:**atp.EnumerationLiteralIndex=120
tn	Setswana **Tags:**atp.EnumerationLiteralIndex=121
to	Tonga **Tags:**atp.EnumerationLiteralIndex=122
tr	Turkish **Tags:**atp.EnumerationLiteralIndex=123
ts	Tsonga **Tags:**atp.EnumerationLiteralIndex=124
tt	Tatar **Tags:**atp.EnumerationLiteralIndex=125
tw	Twi **Tags:**atp.EnumerationLiteralIndex=126
uk	Ukrainian **Tags:**atp.EnumerationLiteralIndex=127
ur	Urdu **Tags:**atp.EnumerationLiteralIndex=128
uz	Uzbek **Tags:**atp.EnumerationLiteralIndex=129
vi	Vietnamese **Tags:**atp.EnumerationLiteralIndex=130
vo	Volapuk **Tags:**atp.EnumerationLiteralIndex=131
wo	Wolof **Tags:**atp.EnumerationLiteralIndex=132

▽

△

Enumeration	**LEnum**
xh	Xhosa **Tags:**atp.EnumerationLiteralIndex=133
yo	Yoruba **Tags:**atp.EnumerationLiteralIndex=134
zh	Chinese **Tags:**atp.EnumerationLiteralIndex=135
zu	Zulu **Tags:**atp.EnumerationLiteralIndex=136

Table 9.98: LEnum

9.7 Document Views

[TPS_GST_00335] View Approach ⌈AUTOSAR provides support of multiple document views. They are supported by `DocumentViewSelectable`.⌋*()*

In the following the illustrated example uses:

- for `LanguageSpecific` the LEnum "en",
- for the stereotype ≪`atpVariation`≫ the `SwSystemconst` "Country" with the `SwSystemconstValue`s "1" and "2" and
- for the `DocumentViewSelectable` the ViewTokens "INTERNAL" and "DETAILED".

Based on these settings the document representation is defined inside the blue marked slice in Figure 9.14.

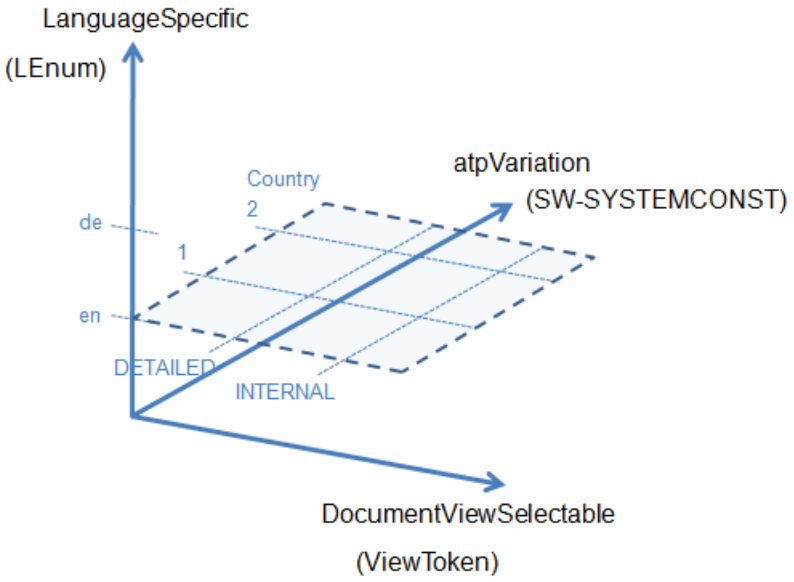


Figure 9.14: View Approach

Beside the `LanguageSpecific` aspect of a document and the system inherent configuration by the stereotype ≪`atpVariation`≫ the `DocumentViewSelectable` opens a third dimension to generate documentation to a particular audience or document view.

In a first step the `Documentation` content is enhanced with document views in which the object shall appear. At a later development step a dedicated view is selected which ensures that the intended audience is met (see Figure 9.15).

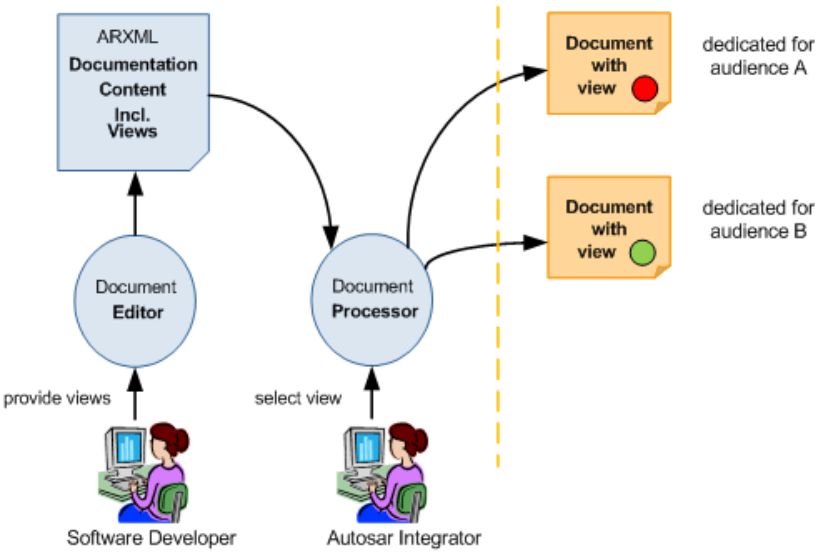


Figure 9.15: Application of document views

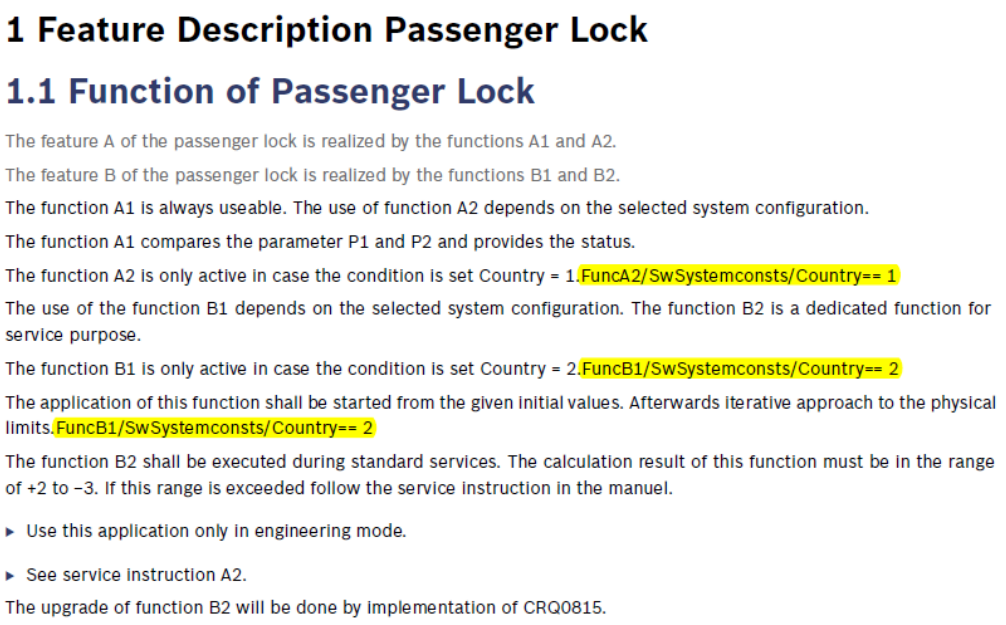
Several use cases are possible:

- no view specified [TPS_GST_00366],
- dedicated view specified [TPS_GST_00367],
- multiple views specified [TPS_GST_00368],
- exclude content [TPS_GST_00369],

[TPS_GST_00366] No View Specified ⌈A document without views specified shall be shown completely to all audiences.⌋*()*

LEnum==en		SW-SYSTEMCONST	
		Country==1	Country==2
ViewToken	DETAILED	-	-
	INTERNAL	-	-
	-	-	-

Table 9.99: Use Case - No selection



1 Feature Description Passenger Lock

1.1 Function of Passenger Lock

The feature A of the passenger lock is realized by the functions A1 and A2.

The feature B of the passenger lock is realized by the functions B1 and B2.

The function A1 is always useable. The use of function A2 depends on the selected system configuration.

The function A1 compares the parameter P1 and P2 and provides the status.

The function A2 is only active in case the condition is set Country = 1.FuncA2/SwSystemconsts/Country== 1

The use of the function B1 depends on the selected system configuration. The function B2 is a dedicated function for service purpose.

The function B1 is only active in case the condition is set Country = 2.FuncB1/SwSystemconsts/Country== 2

The application of this function shall be started from the given initial values. Afterwards iterative approach to the physical limits.FuncB1/SwSystemconsts/Country== 2

The function B2 shall be executed during standard services. The calculation result of this function must be in the range of +2 to -3. If this range is exceeded follow the service instruction in the manuel.

- Use this application only in engineering mode.
- See service instruction A2.

The upgrade of function B2 will be done by implementation of CRQ0815.

Figure 9.16: Use Case - No selection

The yellow marked text in Figure 9.16 indicates that these paragraphs contain a `VariationPoint` with the `SwSystemconst` "Country".

Listing 9.3: Paragraph contains a `VariationPoint` with the `SwSystemconst` "Country==1"

```
<P>
  <L-1 L="EN">The function A2 is only active in case the condition is
     set Country = 1.</L-1>
  <L-1 L="DE">Die Funktion A2 wird nur aktiv, wenn die Bedingung
     Country = 1 gesetzt ist.</L-1>
```

```
    <VARIATION-POINT>
      <SHORT-LABEL>FuncA2</SHORT-LABEL>
      <SW-SYSCOND BINDING-TIME="SYSTEM-DESIGN-TIME"><SYSC-REF DEST="SW-
          SYSTEMCONST">/RB/SwSystemconsts/Country</SYSC-REF>== 1</SW-
          SYSCOND>
    </VARIATION-POINT>
  </P>
```

[TPS_GST_00367] Dedicated View Specified ⌈Specifying a view for a dedicated audience at a part of a document means "show it to this audience only" or in other words "do not show parts marked for other audiences".⌋*()*

Specifying a view in a document is an explicit inclusion of the affected parts of a document. To ensure that an element is shown to all intended audiences, it shall have all accordingly views explicitly set ([TPS_GST_00368]).

$$include = selectedViews \in declaredViews$$

LEnum==en		SW-SYSTEMCONST	
		Country==1	Country==2
ViewToken	DETAILED	-	-
	INTERNAL	X	-
	-	-	-

Table 9.100: Use Case - Dedicated View specified and selected (INTERNAL)

The DocumentViewSelectable with the ViewToken "INTERNAL" and the SwSystemconst "Country" with the SwSystemconstValue "1" are selected. Content specified with the ViewTokens "DETAILED" is not selected.

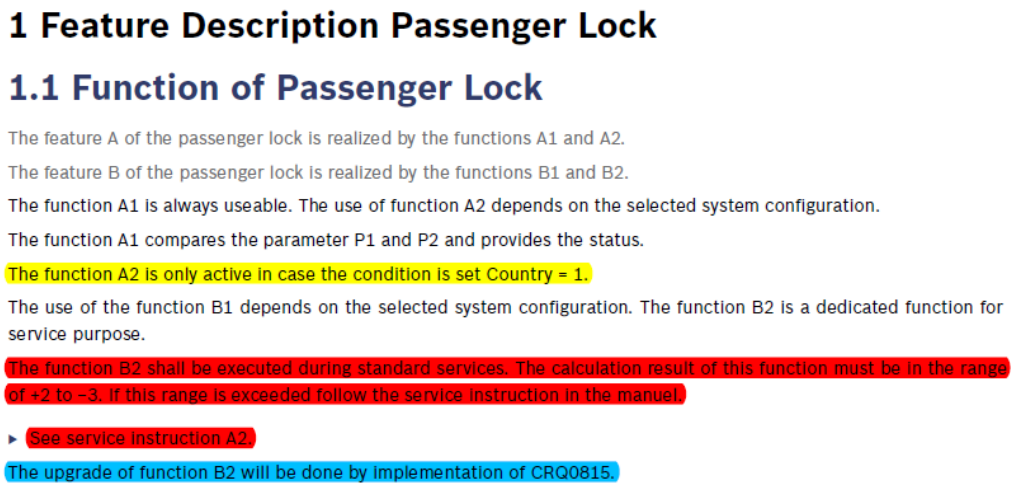


Figure 9.17: Use Case - Dedicated View specified and selected (INTERNAL)

This use case brings out two essential items:

- The red marked text in Figure 9.17 indicate that these paragraphs contain DocumentViewSelectable with the ViewToken "INTERNAL".

  Listing 9.4: Paragraphs contain DocumentViewSelectable with the ViewToken "INTERNAL"

```
<P VIEW="INTERNAL">
  <L-1 L="EN">The function B2 shall be executed during standard
     services. The calculation result of this function must be
     in the range of +2 to -3. If this range is exceeded follow
     the service instruction in the manuel.</L-1>
  <L-1 L="DE">Die Funktion B2 wird im Standardservice ausgefÃ¼hrt
     . Die Berechnungsergebnisse dieser Funktion muss in Bereich
      von +2 bis -3 liegen. Wird dieser Bereich Ã¼berschritten,
     ist der Serviceanweiseung im Handbuch zu folgen.</L-1>
</P>
<LIST TYPE="UNNUMBER" VIEW="INTERNAL">
  <ITEM VIEW="DETAILED">
    <P>
      <L-1 L="EN">Use this application only in engineering mode.<
         /L-1>
      <L-1 L="DE">Diese Anwendung ist nur im Engineering Mode zu
         verwenden.</L-1>
    </P>
  </ITEM>
  <ITEM>
    <P>
      <L-1 L="EN">See service instruction A2.</L-1>
      <L-1 L="DE">Siehe Serviceanweisung A2.</L-1>
    </P>
  </ITEM>
</LIST>
```

  The first item of the list is assigned to the ViewToken "DETAILED" and therefore it does not appear in the generated document.

- The blue marked text in Figure 9.17 indicates that this paragraph contains DocumentViewSelectable with the ViewTokens "INTERNAL" and "DETAILED" (see listing 9.5).

  Listing 9.5: Paragraphs contain DocumentViewSelectable with the ViewTokens "INTERNAL" and "DETAILED".

```
<P VIEW="INTERNAL␣DETAILED">
  <L-1 L="EN">The upgrade of function B2 will be done by
     implementation of CRQ0815.</L-1>
  <L-1 L="DE">Die Erweiterung der Funktion B2 wird durch die
     Implementierung von CRQ0815 erfolgen.</L-1>
</P>
```

Selecting the ViewToken "DETAILED" the blue marked text also appears because of its declaration as can be seen in Figure 9.18.

[TPS_GST_00368] Multiple Views Specified ⌈It is possible to an entity to be visible in a list of views. This shall be interpreted as "show the information if currently selected view is denoted in the list".⌋*()*

LEnum==en		SW-SYSTEMCONST	
		Country==1	Country==2
ViewToken	DETAILED	X	-
	INTERNAL	-	-
	-	-	-

Table 9.101: Use Case - Dedicated View specified and selected (DETAILED)

The generated output will look like in Figure 9.18.

1 Feature Description Passenger Lock

1.1 Function of Passenger Lock

The feature A of the passenger lock is realized by the functions A1 and A2.

The feature B of the passenger lock is realized by the functions B1 and B2.

The function A1 is always useable. The use of function A2 depends on the selected system configuration.

The function A1 compares the parameter P1 and P2 and provides the status.

The function A2 is only active in case the condition is set Country = 1.

The use of the function B1 depends on the selected system configuration. The function B2 is a dedicated function for service purpose.

The upgrade of function B2 will be done by implementation of CRQ0815.

Figure 9.18: Use Case - Dedicated View specified and selected (DETAILED)

[TPS_GST_00369] Exclude content ⌈Excluding content means "do not present it for any audience". This can be achieved by specifying an arbitrary named view (e.g. DONOTPRESENT), which will never be selected for production.⌋*()*

In case the ViewToken "DETAILED" is selected (Table 9.101) only text specified with this dedicated view and text without any view specified will appear. The text specified with the ViewToken "INTERNAL" will be excluded form the content (see Figure 9.18).

10 The Build Action Manifest

10.1 Introduction

Any exchange of BSW modules also includes the exchange of ECU configuration code generators (a.k.a. ECUC processors, Module Generators).

- Seamless integration of these Module Generators into SW build frameworks requires precise knowledge on input/output data.
- With this knowledge Integration of foreign stacks is simplified and less error prone
- Without a formal specification of this knowledge, the integration of "foreign" Module Generators is inefficient and error-prone.

With such formal specifications being available, the following benefits exist for the BSW module integrators:

- Significant reduction of integration effort and ambiguity.
- As code generation is not tied to specialized editor applications anymore, a single configuration editor can be used for the entire project.

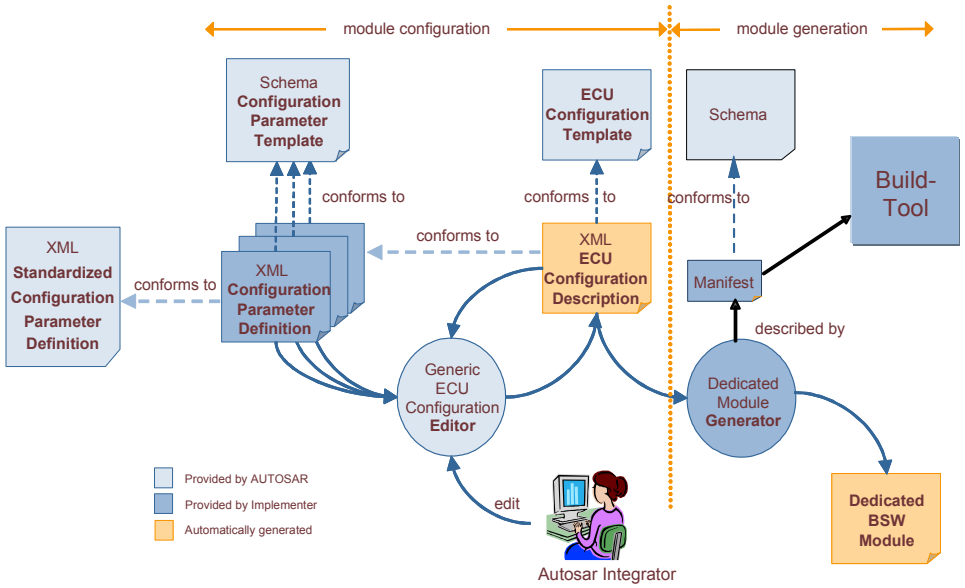


Figure 10.1: Generic Configuration Editor

Mainly the following use cases are addressed:

- A tier 1 supplier has to integrate BSW modules of various BSW vendors.
- A BSW vendor can market and sell not only complete BSW solutions, but also single BSW modules.

`BuildActionManifest` can be denoted in the `Implementation` of the module [TPS_BSWMDT_04085].

A general sequence is:

1. Initialize
2. Perform build
3. Tear down build

The following diagram illustrates the sequence of applied processors.

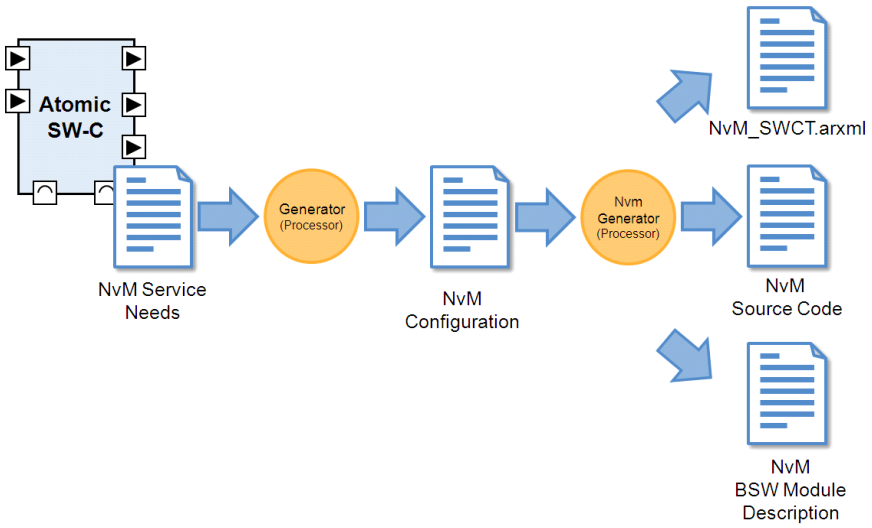


Figure 10.2: Applied Processors

10.2 `BuildActionManifest` Overview

The following diagram illustrates the basic approach:

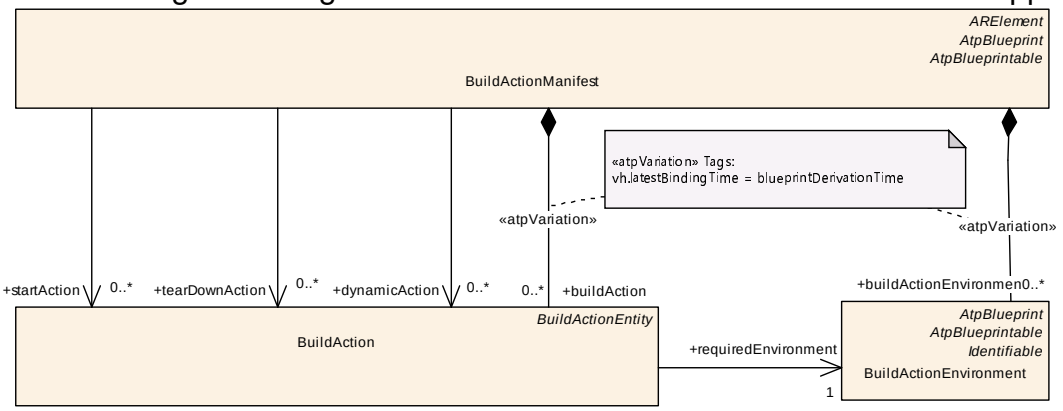


Figure 10.3: Build Action Manifest Overview

Class	**BuildActionManifest**			
Package	M2::AUTOSARTemplates::GenericStructure::BuildActionManifest			
Note	This meta-class represents the ability to specify a manifest for processing artifacts. An example use case is the processing of ECUC parameter values. **Tags:** atp.recommendedPackage=BuildActionManifests xml.globalElement=false			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
buildAction	BuildAction	*	aggr	This represents a particular action in the build chain. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=blueprintDerivationTime
buildAction Environment	BuildActionEnvironment	*	aggr	This represents a build action environment. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=blueprintDerivationTime
dynamicAction	BuildAction	*	ref	This denotes an Action which is to be executed as part of the dynamic action set.
startAction	BuildAction	*	ref	This specifies the list of actions to be performed at the beginning of the process. **Tags:**xml.sequenceOffset=-90
tearDownAction	BuildAction	*	ref	This specifies the set of action which shall be performed after all other actions in the manifest were performed. **Tags:**xml.sequenceOffset=-80

Table 10.1: BuildActionManifest

[TPS_GST_00294] Build Action Manifest Overview ⌈The `BuildActionManifest` is an `ARElement` providing the ability to describe particular steps to be performed in a given environment which need to be performed e.g. when an AUTOSAR executable. It mainly consists of the two entities:

- `BuildAction` is one particular action to be performed in order to contribute to the generation of e.g. the ECU executable.
- `BuildActionEnvironment` provides information about the environment, which is used to perform the BuildActions.

⌋()

Thereby the manifest defines three groups of actions:

- `startAction`: shall be executed in the given order before any other action is performed.
- `tearDownAction`: shall be executed in the given order when all other actions have been completed.
- `dynamicAction`: is part of the essential build process. These actions shall be executed after the completion of the `startAction`s. The sequence of these is determined by the input/output relationship respectively the `followUpAction` / `predecessorAction`. If order cannot be determined or is not specified, then

the results [of every `dynamicAction`] shall be independent of the execution order.

10.2.1 BuildAction

This represents one particular action which contributes to the activities generating ECU executable.

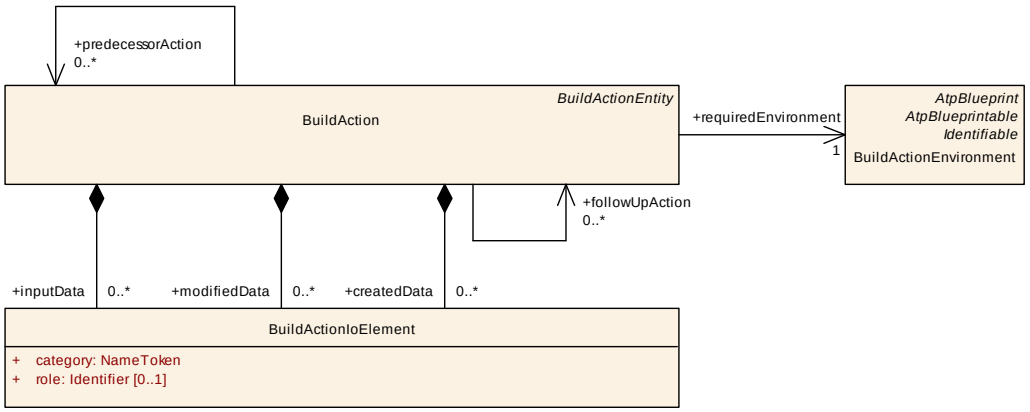


Figure 10.4: Build Action Details

Class	**BuildAction**			
Package	M2::AUTOSARTemplates::GenericStructure::BuildActionManifest			
Note	This meta-class represents the ability to specify a build action.			
Base	*ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *BuildActionEntity*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
createdData	BuildActionIoElement	*	aggr	This represents the artifacts which are created by the processor.
followUpAction	BuildAction	*	ref	This association specifies a set of follow up actions. **Tags:**xml.sequenceOffset=-80
inputData	BuildActionIoElement	*	aggr	This represents the artifacts which are read by the processor.
modifiedData	BuildActionIoElement	*	aggr	This denotes the data which are modified by the action.
predecessor Action	BuildAction	*	ref	This association specifies a set of predecessors. These actions shall be finished before but necessarily immediately after the given action.. These actions need to be performed in the specified order. **Tags:**xml.sequenceOffset=-90
required Environment	BuildActionEnvironment	1	ref	This represents the environment which is required to use the specified Processor.

Table 10.2: BuildAction

[TPS_GST_00338] Purpose of `BuildActionEnvironment` ⌈A build action refers to the environment in which it shall be executed. Note that the build action does not provide specific attributes for error handling and abort conditions. This is subject to the environment performing the build actions.⌋ *()*

[TPS_GST_00339] Data involved in Build Actions ⌈A build action specifies the involved data in terms of aggregated BuildActionIoElements. The role of the aggregation of BuildActionIoElement in BuildAction determines if data is only read, modified, or created the first time.⌋*()*

[TPS_GST_00340] Sequence of Build Actions ⌈The data involved in BuildActions can be used to determine an appropriate execution order. The following hints apply:

- Actions consuming data shall be performed after actions producing or modifying the same data.
- An action consuming and producing the same data shall use modifiedData. Otherwise this action would be its own predecessor / successor.
- modifiedData in particular supports the case that an action manipulates or verifies existing data. These action shall be performed before any consuming actions.
- Implicit interdependencies between BuildActions can be specified using predecessorAction respectively followUpAction.

⌋*()*

Details of the implementation / invocation are specified using the properties of the superclass BuildActionEntity.

10.2.2 BuildActionIoElement

This represents data which are involved in a BuildAction.

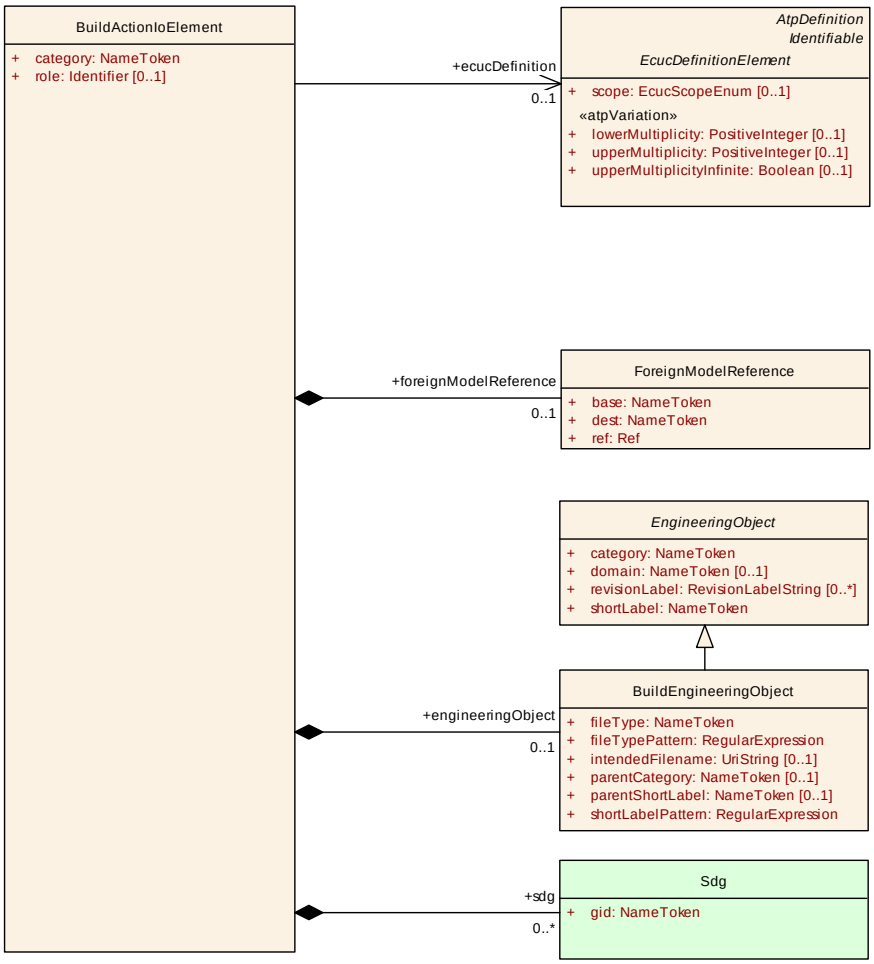


Figure 10.5: Build Action Manifest Input/Output

Class	**BuildActionIoElement**			
Package	M2::AUTOSARTemplates::GenericStructure::BuildActionManifest			
Note	This meta-class represents the ability to specify the input/output entities of a BuildAction.			
Base	*ARObject*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
category	NameToken	1	attr	This element assigns a category to the parent element. It is intended to specialize the usage and/or the content of the object. Such a specialization may also impose particular semantic constraints on the entire substructure. See also Identifiable. **Tags:**xml.sequenceOffset=-100
ecucDefinition	EcucDefinitionElement	0..1	ref	This association denotes an ECUC parameter definition. The such referenced parameters are subject of the build action input/output. Note that the reference to the definition denotes the right for a build action to read and/or write values for the given definition and all contained definitions.
engineering Object	BuildEngineeringObject	0..1	aggr	This represents an artifact applicable to the build action.

△

Class	**BuildActionIoElement**			
foreignModel Reference	ForeignModelReference	0..1	aggr	This is a reference to a foreign model element. Note that it is not modeled as an association because it should also be able to refer also to non AUTOSAR models.
role	Identifier	0..1	attr	This attribute allows to denote a particular role of the collection. Note that the applicable semantics shall be mutually agreed between the two parties. **Tags:**xml.sequenceOffset=30
sdg	Sdg	*	aggr	This special data group allows to denote specific data. The structure is subject of mutual agreement. **Tags:**xml.sequenceOffset=-90

Table 10.3: BuildActionIoElement

[TPS_GST_00341] Input Data for Build Actions ⌈The model covers three kinds of data:

1. Artifacts - denoted as `BuildEngineeringObject`s
2. ECUC-parameters - denoted as references to `EcucDefinitionElement`s
3. reference to any model element - denoted as `GenericModelReference`

⌋*()*

[TPS_GST_00342] ECUC-Parameters in Build Actions ⌈Note that if containers are referenced, the action may also touch the parameter / sub-containers in this container.⌋ *()*

[TPS_GST_00343] ECUC-Containers in Build Actions ⌈Note further, that if a container is referenced in a `BuildActionIoElement` playing the role `createdData`, then a new container is created.⌋*()*

[TPS_GST_00344] General Model Elements in Build Actions ⌈For General Model Elements it is assumed that model elements can be referenced similar to the references within an AUTOSAR model. This also implies that the referenced model element is in a proprietary model.⌋*()*

An example for [TPS_GST_00344] is the setting of status information in a work flow model.

[TPS_GST_00345] Special Data in `BuildActionIoElement` ⌈`sdg` can be used to attach environment specific properties to the `BuildActionIoElement`.⌋*()*

10.2.3 BuildActionEnvironment

This entity specifies the build environment. Main purpose is to provide the possibility that a build action denotes the required environment. The model supports that arbitrary properties are specified which can be used to support the installation and test of the environment.

Class	**BuildActionEnvironment**			
Package	M2::AUTOSARTemplates::GenericStructure::BuildActionManifest			
Note	This meta-class represents the ability to specify a build action environment.			
Base	*ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
sdg	Sdg	*	aggr	This represents a general data structure intended to denote parameters for the BuildActionEnvironment.

Table 10.4: BuildActionEnvironment

10.2.4 BuildActionEntity

BuildActionEntity specifies necessary implementation / invocation properties.

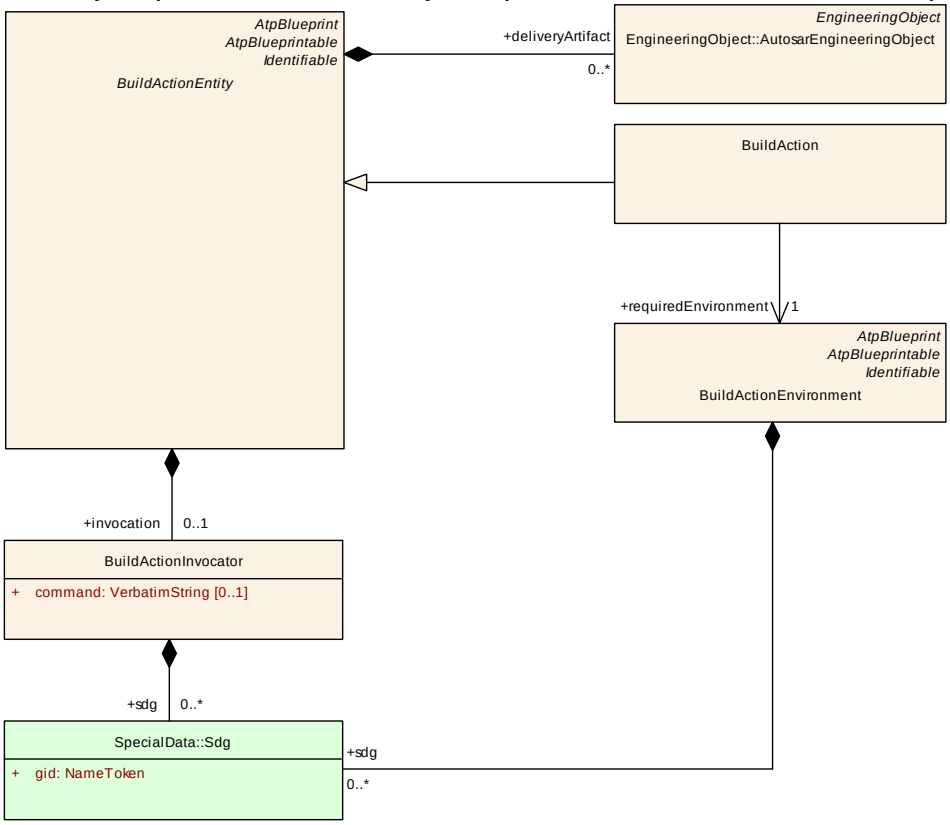


Figure 10.6: Build Action Entity

Class	***BuildActionEntity*** (abstract)
Package	M2::AUTOSARTemplates::GenericStructure::BuildActionManifest
Note	This meta-class represents the ability to describe a build action entity which might be specialized to environments as well as to individual build actions.
Base	*ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *Identifiable*, *MultilanguageReferrable*, *Referrable*
Subclasses	BuildAction

▽

△

Class	***BuildActionEntity*** (abstract)			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
deliveryArtifact	AutosarEngineering Object	*	aggr	This denotes the delivery artifacts for the entity for reference purposes.
invocation	BuildActionInvocator	0..1	aggr	This specifies how to invoke a build action in the given environment.

Table 10.5: BuildActionEntity

10.2.5 Usage of Special Data

[TPS_GST_00357] Usage of Special Data ⌈The Usage of Special Data (Sdg) within the BuildActionManifest are specific to the particular application and therefore not standardized.⌋*()* Listing 10.1 shows an example how to denote a more specific role of an artifact in a BuildAction.

Listing 10.1: Example of Special Data in Build Action Manifest

```
<INPUT-DATAS>
  <BUILD-ACTION-IO-ELEMENT>
    <CATEGORY>ARTIFACT</CATEGORY>
    <SDGS>
      <SDG>
        <SD GID="USED_ACTION_ROLE">PROCESSOR</SD>
      </SDG>
    </SDGS>
    <ENGINEERING-OBJECT>
      <SHORT-LABEL>CanIf_Generator</SHORT-LABEL>
      <CATEGORY>CONFWF</CATEGORY>
      <DOMAIN>ARDEMO</DOMAIN>
      <FILE-TYPE>oaw</FILE-TYPE>
    </ENGINEERING-OBJECT>
  </BUILD-ACTION-IO-ELEMENT>
</INPUT-DATAS>
```

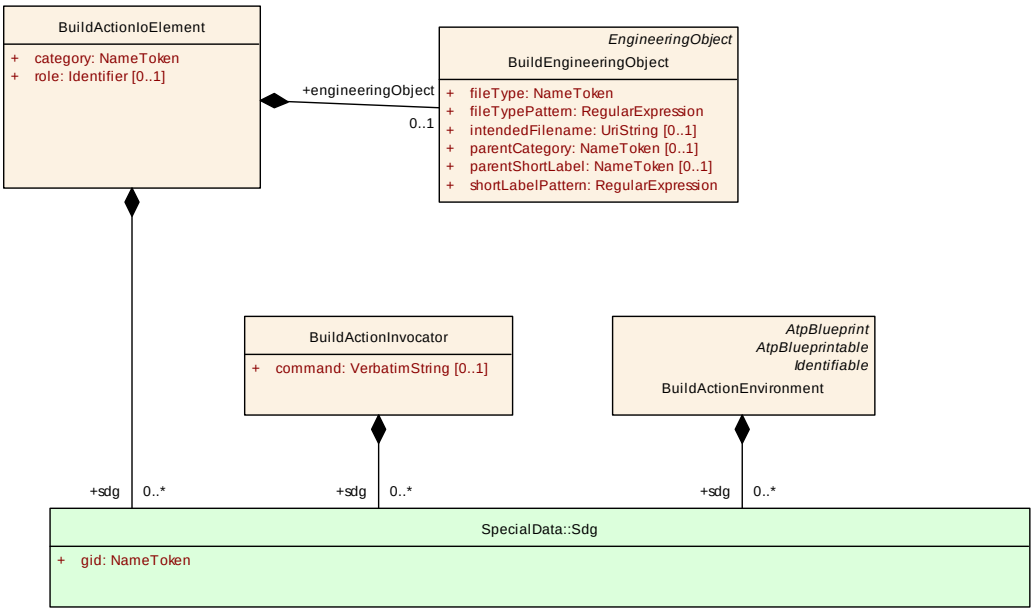


Figure 10.7: BuildAction Usage of Special Data

Class	**BuildActionInvocator**			
Package	M2::AUTOSARTemplates::GenericStructure::BuildActionManifest			
Note	This meta-class represents the ability to specify the invocation of a task in a build action.			
Base	*ARObject*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
command	VerbatimString	0..1	attr	This represents the command to invocate the processor. Note that this is a generic string which can be interpreted properly in the processor environment. Note that it is optional due to the fact that some actions are hardwired in the environment and do not need an explicit command. On the other hand the properties of an invocator can be complex and not standardized.
sdg	Sdg	*	aggr	This represents a general data structure intended to denote parameters for the BuildAction.

Table 10.6: BuildActionInvocator

Class	**BuildEngineeringObject**			
Package	M2::AUTOSARTemplates::GenericStructure::BuildActionManifest			
Note	This meta-class represents the ability to denote an artifact which is processed within a particular build action.			
Base	*ARObject*, *EngineeringObject*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
fileType	NameToken	1	attr	This attribute indicates the file type which shall used for the engineering object. Note that an engineering object may deliver multiple representations of the same artifact. This attribute can select one of the provided representations.

△

Class	**BuildEngineeringObject**			
fileTypePattern	RegularExpression	1	attr	This attribute allows to define a set of engineering objects as pattern based search applied to the filetype of the individual Engineering objects. **Tags:**xml.sequenceOffset=90
intended Filename	UriString	0..1	attr	This attribute represents the name of the file if it is created newly. Note that engineering object resolves category + ShortLabel indicate mainly to refer to an existing file. If the file is created newly, the filename can either be determined by built in policy or predefined here. Note that extensions shall part of file name even if it could be derived from fileType.
parentCategory	NameToken	0..1	attr	This represents the category of the parent object.
parentShort Label	NameToken	0..1	attr	This represents the shortLabel of the parent object. This allows to specify the output position in a hierarchically organized system.
shortLabel Pattern	RegularExpression	1	attr	This attribute allows to define a set of engineering objects as pattern based search applied to the shortLabel of the individual Engineering objects. **Tags:**xml.sequenceOffset=80

Table 10.7: BuildEngineeringObject

10.2.6 Example

This example shall illustrate the representation in the arxml file.

Listing 10.2: Example of Build Action Manifest

```
<?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http:
    //autosar.org/schema/r4.0" xsi:schemaLocation="http://autosar.org/schema
    /r4.0␣AUTOSAR_4-1-3.xsd">
  <ADMIN-DATA>
    <USED-LANGUAGES />
  </ADMIN-DATA>
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>AUTOSAR</SHORT-NAME>
      <AR-PACKAGES>
        <AR-PACKAGE>
          <SHORT-NAME>GenericStructureTemplate</SHORT-NAME>
          <CATEGORY>EXAMPLE</CATEGORY>
          <AR-PACKAGES>
            <AR-PACKAGE>
              <SHORT-NAME>BuildActionManifests</SHORT-NAME>
              <ELEMENTS>
                <BUILD-ACTION-MANIFEST>
                  <SHORT-NAME>CanIf</SHORT-NAME>
                  <BUILD-ACTIONS>
                    <BUILD-ACTION>
                      <SHORT-NAME>CanIf_Generate</SHORT-NAME>
                      <CATEGORY>GENERATOR</CATEGORY>
```

```
<INVOCATION />
<CREATED-DATAS>
  <BUILD-ACTION-IO-ELEMENT>
    <CATEGORY>ARTIFACT</CATEGORY>
    <ENGINEERING-OBJECT>
      <SHORT-LABEL>CanIf_Cfg</SHORT-LABEL>
      <CATEGORY>SWSRC</CATEGORY>
      <DOMAIN>ARDEMO</DOMAIN>
      <FILE-TYPE>c</FILE-TYPE>
      <INTENDED-FILENAME>CanIf_Cfg.c</INTENDED-
          FILENAME>
    </ENGINEERING-OBJECT>
  </BUILD-ACTION-IO-ELEMENT>
  <BUILD-ACTION-IO-ELEMENT>
    <CATEGORY>ARTIFACT</CATEGORY>
    <ENGINEERING-OBJECT>
      <SHORT-LABEL>CanIf_Cfg</SHORT-LABEL>
      <CATEGORY>SWHDR</CATEGORY>
      <DOMAIN>ARDEMO</DOMAIN>
      <FILE-TYPE>h</FILE-TYPE>
      <INTENDED-FILENAME>CanIf_Cfg.h</INTENDED-
          FILENAME>
    </ENGINEERING-OBJECT>
  </BUILD-ACTION-IO-ELEMENT>
  <BUILD-ACTION-IO-ELEMENT>
    <CATEGORY>ARTIFACT</CATEGORY>
    <ENGINEERING-OBJECT>
      <SHORT-LABEL>CanIf_Cfg_Doc</SHORT-LABEL>
      <CATEGORY>Doc</CATEGORY>
      <DOMAIN>ARDEMO</DOMAIN>
      <FILE-TYPE>arxml</FILE-TYPE>
      <INTENDED-FILENAME>CanIf_Cfg_Doc.arxml</
          INTENDED-FILENAME>
    </ENGINEERING-OBJECT>
  </BUILD-ACTION-IO-ELEMENT>
</CREATED-DATAS>
<INPUT-DATAS>
  <BUILD-ACTION-IO-ELEMENT>
    <CATEGORY>ARTIFACT</CATEGORY>
    <SDGS>
      <SDG>
        <SD GID="USED_ACTION_ROLE">PROCESSOR</SD>
      </SDG>
    </SDGS>
    <ENGINEERING-OBJECT>
      <SHORT-LABEL>CanIf_Generator</SHORT-LABEL>
      <CATEGORY>CONFWF</CATEGORY>
      <DOMAIN>ARDEMO</DOMAIN>
      <FILE-TYPE>oaw</FILE-TYPE>
    </ENGINEERING-OBJECT>
  </BUILD-ACTION-IO-ELEMENT>
</INPUT-DATAS>
<REQUIRED-ENVIRONMENT-REF DEST="BUILD-ACTION-
    ENVIRONMENT">/AUTOSAR/GenericStructureTemplate/
    BuildActionManifests/CanIf/ARDEMO_ENV_CODE</
    REQUIRED-ENVIRONMENT-REF>
```

```
    </BUILD-ACTION>
    <BUILD-ACTION>
      <SHORT-NAME>Compile_Doc</SHORT-NAME>
      <CATEGORY>GENERATOR</CATEGORY>
      <INVOCATION>
        <COMMAND>DOC_FORMATTER {CanIf_Static_Doc} {
            CanIf_Doc}</COMMAND>
      </INVOCATION>
      <CREATED-DATAS>
        <BUILD-ACTION-IO-ELEMENT>
          <CATEGORY>ARTIFACT</CATEGORY>
          <ENGINEERING-OBJECT>
            <SHORT-LABEL>CanIf_Doc</SHORT-LABEL>
            <CATEGORY>Doc</CATEGORY>
            <DOMAIN>ARDEMO</DOMAIN>
            <FILE-TYPE>pdf</FILE-TYPE>
            <INTENDED-FILENAME>CanIf_Doc.pdf</INTENDED-
                FILENAME>
            <PARENT-CATEGORY>PRJ</PARENT-CATEGORY>
            <PARENT-SHORT-LABEL>CanIf</PARENT-SHORT-LABEL>
          </ENGINEERING-OBJECT>
        </BUILD-ACTION-IO-ELEMENT>
      </CREATED-DATAS>
      <INPUT-DATAS>
        <BUILD-ACTION-IO-ELEMENT>
          <CATEGORY>ARTIFACT</CATEGORY>
          <ENGINEERING-OBJECT>
            <SHORT-LABEL>CanIf_Cfg_Doc</SHORT-LABEL>
            <CATEGORY>Doc</CATEGORY>
            <DOMAIN>ARDEMO</DOMAIN>
            <FILE-TYPE>arxml</FILE-TYPE>
          </ENGINEERING-OBJECT>
        </BUILD-ACTION-IO-ELEMENT>
        <BUILD-ACTION-IO-ELEMENT>
          <CATEGORY>ARTIFACT</CATEGORY>
          <ENGINEERING-OBJECT>
            <SHORT-LABEL>CanIf_Static_Doc</SHORT-LABEL>
            <CATEGORY>Doc</CATEGORY>
            <DOMAIN>ARDEMO</DOMAIN>
            <FILE-TYPE>arxml</FILE-TYPE>
          </ENGINEERING-OBJECT>
        </BUILD-ACTION-IO-ELEMENT>
      </INPUT-DATAS>
      <REQUIRED-ENVIRONMENT-REF DEST="BUILD-ACTION-
          ENVIRONMENT">/AUTOSAR/GenericStructureTemplate/
          BuildActionManifests/CanIf/ARDEMO_ENV_DOC</
          REQUIRED-ENVIRONMENT-REF>
    </BUILD-ACTION>
  </BUILD-ACTIONS>
  <BUILD-ACTION-ENVIRONMENTS>
    <BUILD-ACTION-ENVIRONMENT>
      <SHORT-NAME>ARDEMO_ENV_CODE</SHORT-NAME>
    </BUILD-ACTION-ENVIRONMENT>
    <BUILD-ACTION-ENVIRONMENT>
      <SHORT-NAME>ARDEMO_ENV_DOC</SHORT-NAME>
    </BUILD-ACTION-ENVIRONMENT>
```

```
                    </BUILD-ACTION-ENVIRONMENTS>
                  </BUILD-ACTION-MANIFEST>
                </ELEMENTS>
              </AR-PACKAGE>
            </AR-PACKAGES>
          </AR-PACKAGE>
        </AR-PACKAGES>
      </AR-PACKAGE>
    </AR-PACKAGES>
</AUTOSAR>
```

10.3 Constraints and assumptions

In case of software exchange, it is necessary also to supply contributions to the build environment of the recipient. It is also assumed that mutual agreements are taken for the full details. The manifest helps to formalize these agreements such that - once the environment is established - a software exchange can be performed on a regular basis. The case that software is exchanged which relies on a certain processor is supported too.

11 Roles and Rights

In order to manage a cooperative engineering approach, AUTOSAR supports the specification of access rights to objects for involved parties. The approach applies to use cases such as

- Software Sharing artifacts are delivered to cooperation partners. Depending on the chosen business case and cooperation model by the partners the delivered files shall be used in read-only mode or specified changes/adds on specified attributes are allowed or even expected to be made by the integrator/supplier.
- In cooperative application software development, AUTOSAR descriptions can be split to multiple files. As AUTOSAR does not specify a particular process, there should be a generic way which allows to specify roles in the development process. Based on such roles, an initial split of AUTOSAR XML descriptions can be performed. It also would allow deriving access control based on meta-classes.
- Some ECU parameter values may fall under different category of responsibility. Therefore based on the ECU parameter definition this responsibility can be specified as access control. In addition the access control can be specified in terms of Artifacts. This finally even allows to define a split/merge policy.

[TPS_GST_00226] Access Control Relation ⌈In general, roles and rights is conceptualized as a relation

$$Permission = f(role, operation, object, context)$$

⌋*()*

Figure 11.1 depicts the meta-classes implementing the access control relation according to [TPS_GST_00226]:

- **[TPS_GST_00227]** **`AclPermission`** ⌈This represents the relation mentioned in [TPS_GST_00226]⌋*()*
- **[TPS_GST_00228]** **`AclRole`** ⌈This represents the possible roles within the definition of Roles and Rights. This role might have a relationship to the roles defined in the methodology. On the other hand, the technical definition of these roles highly depend on the local environment. On the other hand LDAP-Groups are very common, so `ldapUrl` allows to refer to a previously agreed LDAP group.⌋*()*
- **[TPS_GST_00229]** **`AclOperation`** ⌈This is used to specify the possible activities which are performed by a role on a set of objects. This activity might have a relationship to the activities in the methodology.

  Relationships between operations are expressed such as implies / blocks. For example write-access implies read access

  An Operation also specifies it scope by `aclScope`⌋*()*

- **[TPS_GST_00230]** **`AclObjectSet`** ⌈This is used to specify the objects to which the permission applies. This definition can be based on meta level M1 as well as based on M2.⌋*()*
- **[TPS_GST_00231] Context of `AclPermission`** ⌈The context under which an `AclPermission` can be applied is specified in the attribute `aclContext`. More complex contexts may be implemented using `sdg` or via `Collection`. The possible values are subject to mutual agreement between the stakeholders.⌋*()*

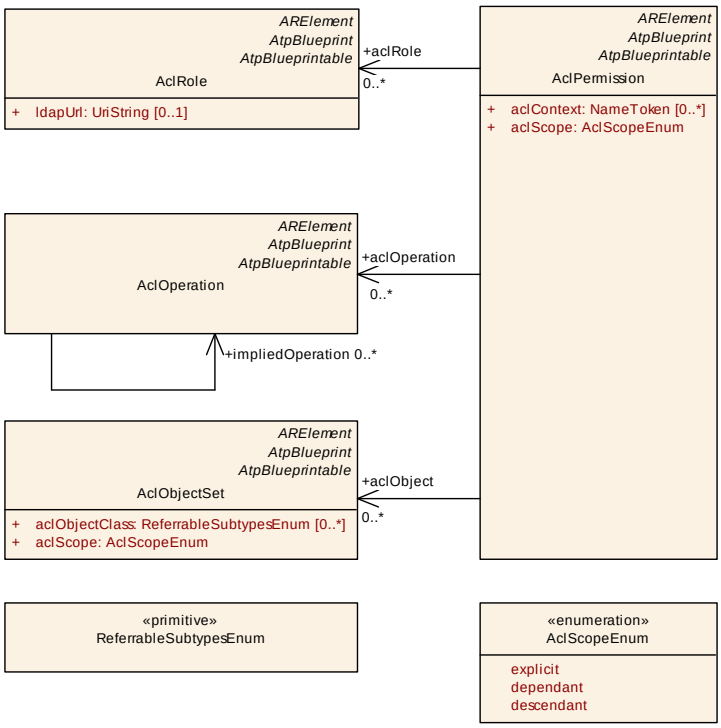


Figure 11.1: Roles and Rights Overview

Figure 11.2 illustrates, how the set of objects is specified. This can be done based on meta level M1 as well as on M2. The following methods are supported:

- **[TPS_GST_00232]** **`aclObjectClass`** ⌈This represents the case that the access controlled objects are defined as a reference to a meta-class resp. to an attribute of a meta-class (M2). The such defined permissions apply to all model elements (instances) of that meta-class.⌋*()*
- **[TPS_GST_00233]** **`collection`** ⌈This represents the case that the access controlled objects are defined by a `Collection`. The objects referenced by the referenced `Collection` belong to the access controlled object set.⌋*()*
- **[TPS_GST_00234]** **`object`** ⌈This represents this case where the M1 objects are explicitly referenced to. Note that in addition to the `Collection` this can also refer to `Referrable`s.⌋*()*
- **[TPS_GST_00235]** **`objectDefinition`** ⌈The AUTOSAR meta-model has the pattern that on M1 we have the pair of Definition/Description. In this case all descriptions for a referenced definition are part of the object set.⌋*()*

Examples for [TPS_GST_00235] are given as

  - `EcucDefinitionElement` - ECUC parameter Value
  - `SwSystemconst` - `SwSystemconstValue`
  - `PostBuildVariantCriterion` - `PostBuildVariantCriterionValue`

- **[TPS_GST_00236]** **`derivedFromBlueprint`** ⌈In this case all objects derived from a referenced blueprint are belong to the object set.

  Note that instanceRefs are not supported, since there was no use case. It could be supported using a `FlatMap`.⌋*()*

- **[TPS_GST_00237]** **`engineeringObject`** ⌈This indicates an engineering object. The specified permissions apply to all objects in the partial model which is stored in the denoted `AutosarEngineeringObject`.⌋*()*

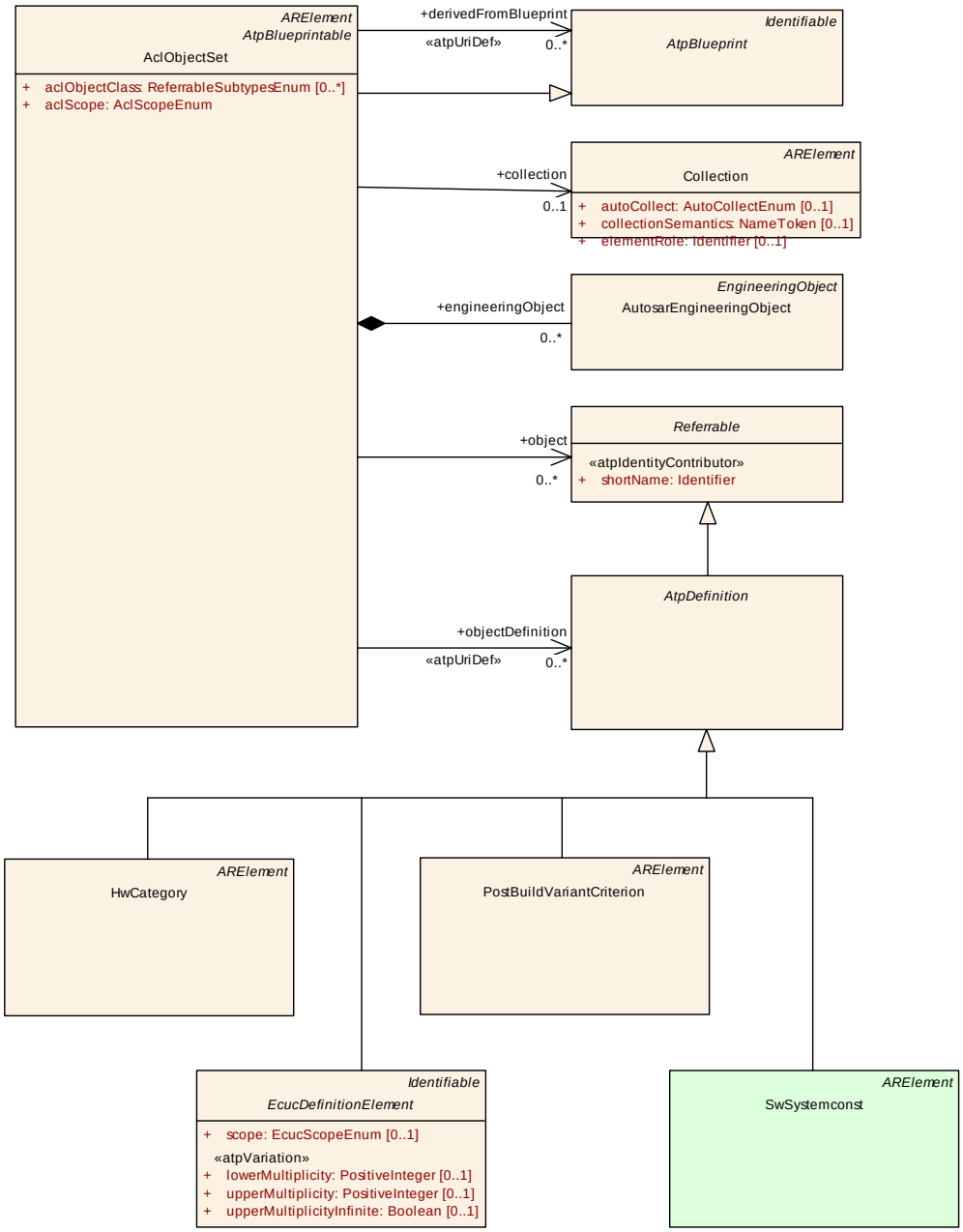


Figure 11.2: Roles and Rights Object Set

Listing 11.1 illustrates how the roles and rights approach can be applied to control the access on objects in an AUTOSAR description. It defines

- a set of permissions named "Integrator" representing the Relation in [TPS_GST_00226] based on the definitions below
- a role named "ECU_Integrator"
- an object set named "MemoryStackConfiguration"
- two operations

Listing 11.1: Example for Access Control

```
<?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http:
    //autosar.org/schema/r4.0" xsi:schemaLocation="http://autosar.org/schema
    /r4.0␣AUTOSAR_4-2-2.xsd">
  <ADMIN-DATA>
    <LANGUAGE>EN</LANGUAGE>
    <USED-LANGUAGES>
      <L-10 L="EN" xml:space="default">English</L-10>
    </USED-LANGUAGES>
  </ADMIN-DATA>
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>AUTOSAR</SHORT-NAME>
      <AR-PACKAGES>
        <AR-PACKAGE>
          <SHORT-NAME>GenericStructureTemplate</SHORT-NAME>
          <CATEGORY>EXAMPLE</CATEGORY>
          <AR-PACKAGES>
            <AR-PACKAGE>
              <SHORT-NAME>AclPermissions</SHORT-NAME>
              <ELEMENTS>
                <ACL-PERMISSION>
                  <SHORT-NAME>Integrator</SHORT-NAME>
                  <ACL-OBJECT-REFS>
                    <ACL-OBJECT-REF DEST="ACL-OBJECT-SET">/AUTOSAR/
                        GenericStructureTemplate/AccessObjectSets/
                        MemoryStackConfiguration</ACL-OBJECT-REF>
                  </ACL-OBJECT-REFS>
                  <ACL-OPERATION-REFS>
                    <ACL-OPERATION-REF DEST="ACL-OPERATION">/AUTOSAR/
                        GenericStructureTemplate/AclOperations/AssignValue</
                        ACL-OPERATION-REF>
                    <ACL-OPERATION-REF DEST="ACL-OPERATION">/AUTOSAR/
                        GenericStructureTemplate/AclOperations/ReassignValue
                        </ACL-OPERATION-REF>
                  </ACL-OPERATION-REFS>
                  <ACL-ROLE-REFS>
                    <ACL-ROLE-REF DEST="ACL-ROLE">/AUTOSAR/
                        GenericStructureTemplate/AclRoles/ECU_Integrator</
                        ACL-ROLE-REF>
                  </ACL-ROLE-REFS>
                </ACL-PERMISSION>
              </ELEMENTS>
            </AR-PACKAGE>
            <AR-PACKAGE>
              <SHORT-NAME>AccessObjectSets</SHORT-NAME>
              <ELEMENTS>
                <ACL-OBJECT-SET>
                  <SHORT-NAME>MemoryStackConfiguration</SHORT-NAME>
                  <ACL-SCOPE>DESCENDANT</ACL-SCOPE>
                  <OBJECT-DEFINITION-REFS>
                    <OBJECT-DEFINITION-REF DEST="ECUC-MODULE-DEF">/AUTOSAR/
                        EcucDefs/MemIf</OBJECT-DEFINITION-REF>
                    <OBJECT-DEFINITION-REF DEST="ECUC-MODULE-DEF">/AUTOSAR/
                        EcucDefs/MemMap</OBJECT-DEFINITION-REF>
```

```
                    <OBJECT-DEFINITION-REF DEST="ECUC-MODULE-DEF">/AUTOSAR/
                        EcucDefs/NvM</OBJECT-DEFINITION-REF>
                    </OBJECT-DEFINITION-REFS>
                  </ACL-OBJECT-SET>
                </ELEMENTS>
              </AR-PACKAGE>
              <AR-PACKAGE>
                <SHORT-NAME>AclOperations</SHORT-NAME>
                <REFERENCE-BASES>
                  <REFERENCE-BASE>
                    <SHORT-LABEL>op</SHORT-LABEL>
                    <BASE-IS-THIS-PACKAGE>true</BASE-IS-THIS-PACKAGE>
                  </REFERENCE-BASE>
                </REFERENCE-BASES>
                <ELEMENTS>
                  <ACL-OPERATION>
                    <SHORT-NAME>AssignValue</SHORT-NAME>
                  </ACL-OPERATION>
                  <ACL-OPERATION>
                    <SHORT-NAME>ReassignValue</SHORT-NAME>
                    <IMPLIED-OPERATION-REFS>
                      <IMPLIED-OPERATION-REF DEST="ACL-OPERATION" BASE="op">
                        AssignValue</IMPLIED-OPERATION-REF>
                    </IMPLIED-OPERATION-REFS>
                  </ACL-OPERATION>
                </ELEMENTS>
              </AR-PACKAGE>
              <AR-PACKAGE>
                <SHORT-NAME>AclRoles</SHORT-NAME>
                <ELEMENTS>
                  <ACL-ROLE>
                    <SHORT-NAME>ECU_Integrator</SHORT-NAME>
                    <LONG-NAME>
                      <L-4 L="EN">See <TT>ECU Integrator</TT></L-4>
                    </LONG-NAME>
                  </ACL-ROLE>
                </ELEMENTS>
              </AR-PACKAGE>
            </AR-PACKAGES>
          </AR-PACKAGE>
        </AR-PACKAGES>
      </AR-PACKAGE>
    </AR-PACKAGES>
</AUTOSAR>
```

Class	**AclPermission**
Package	M2::AUTOSARTemplates::GenericStructure::RolesAndRights
Note	This meta class represents the ability to represent permissions granted on objects in an AUTOSAR model. **Tags:**atp.recommendedPackage=AclPermissions
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*

▽

△

Class	**AclPermission**			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
aclContext	NameToken	*	attr	This attribute is intended to specify the context under which the AclPemission is applicable. The values are subject to mutual agreement between the involved stakeholders. For examples the values can be the names of binding times.
aclObject	AclObjectSet	*	ref	This denotes an object to which the AclPermission applies.
aclOperation	AclOperation	*	ref	This denotes an operation which is granted by the given AclPermission.
aclRole	AclRole	*	ref	This denotes the role (individual or even organization) for which the AclPermission. is granted.
aclScope	AclScopeEnum	1	attr	This indicates the scope of applied permissions: explicit, descendant, dependent;

Table 11.1: AclPermission

Class	**AclObjectSet**			
Package	M2::AUTOSARTemplates::GenericStructure::RolesAndRights			
Note	This meta class represents the ability to denote a set of objects for which roles and rights (access control lists) shall be defined. It basically can define the objects based on • the nature of objects • the involved blueprints • the artifact in which the objects are serialized • the definition of the object (in a definition - value pattern) • individual reference objects **Tags:**atp.recommendedPackage=AclObjectSets			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
aclObjectClass	ReferrableSubtypesEnum	*	attr	This specifies that the considered objects as instances of the denoted meta class.
aclScope	AclScopeEnum	1	attr	this indicates the scope of the referenced objects.
collection	Collection	0..1	ref	This indicates that the relevant objects are specified via a collection.
derivedFromBlueprint	AtpBlueprint	*	ref	This association indicates that the considered objects are the ones being derived from the associated blueprint. **Stereotypes:** atpUriDef
engineeringObject	AutosarEngineeringObject	*	aggr	This indicates an engineering object. The AclPermission relates to all objects in this partial model. This also implies that the other objects in this set shall be placed in the specified engineering object. Note that semantic constraints apply with respect to <<atpSplitable>>
object	Referrable	*	ref	This association applies a particular (usually small) set of objects (e.g. a singular package). Main usage is, if one does not want to create a collection specifically for access control.

▽

△

Class	AclObjectSet			
objectDefinition	AtpDefinition	*	ref	This denotes an object by its definition. For example the right to manipulate the value of a particular ecuc parameter is denoted by reference to the definition of the parameter. Note that this can also be a reference to a Standard Module Definition. Therefore it is stereotyped by atpUriDef. **Stereotypes:** atpUriDef

Table 11.2: AclObjectSet

Class	*AtpDefinition* (abstract)			
Package	M2::AUTOSARTemplates::GenericStructure::RolesAndRights			
Note	This abstract meta class represents "definition"-elements which identify the respective values. For example the value of a particular system constant is identified by the definition of this system constant.			
Base	*ARObject*, *Referrable*			
Subclasses	*EcucDefinitionElement*, HwCategory, PostBuildVariantCriterion, SwSystemconst			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
–	–	–	–	–

Table 11.3: AtpDefinition

Class	AclOperation			
Package	M2::AUTOSARTemplates::GenericStructure::RolesAndRights			
Note	This meta class represents the ability to denote a particular operation which may be performed on objects in an AUTOSAR model. **Tags:**atp.recommendedPackage=AclOperations			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *CollectableElement*, *Identifiable*, *Multilanguage Referrable*, *PackageableElement*, *Referrable*			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
implied Operation	AclOperation	*	ref	This indicates that the related operations are also implied. Therefore the permission is also granted for this operation.

Table 11.4: AclOperation

Class	AclRole			
Package	M2::AUTOSARTemplates::GenericStructure::RolesAndRights			
Note	This meta class represents the ability to specify a particular role which is used to grant access rights to AUTOSAR model. The purpose of this meta-class is to support the mutual agreements between the involved parties. **Tags:**atp.recommendedPackage=AclRoles			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *CollectableElement*, *Identifiable*, *Multilanguage Referrable*, *PackageableElement*, *Referrable*			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
ldapUrl	UriString	0..1	attr	This is an URL which allows to represent users or organizations taking the particular role.

Table 11.5: AclRole

Enumeration	**AclScopeEnum**
Package	M2::AUTOSARTemplates::GenericStructure::RolesAndRights
Note	This enumerator represents the scope of a definition in context of access control.
Literal	***Description***
dependant	This specifies that the AclPermission applies to dependant (in particular referenced) operations / objects as well. Note that this includes the descendant ones. **Tags:**atp.EnumerationLiteralIndex=0
descendant	This specifies that the AclPermission applies to descendant operations / objects as well. **Tags:**atp.EnumerationLiteralIndex=1
explicit	This is indicates that the AclPermission applies to explicit objects / operations only. **Tags:**atp.EnumerationLiteralIndex=2

Table 11.6: AclScopeEnum

12 Life Cycle Support

12.1 Introduction

In order to support evolution and backward compatibility of the AUTOSAR model elements like port prototype blueprints, port interfaces, keyword abbreviations, example SW-Cs (in ASW) or of the API of a BSW module etc. AUTOSAR supports life cycles.

The provided approach can also be applied to standardization (See [TPS_STDT_00038]) as well as for vendor specific development. A life cycle presents information like "outdated/deprecated/legacy", "invalid/not part of standard anymore", "obsolete/will be outdated in near future". It is not identical to the version status information like "in work", "released" etc.

The life cycle approach might also be used for intermediate development between two releases, e.g. by attaching a life cycle like "experimental" to it. For the final release however, elements still having such a life cycle would not be part of the release.

Due to backward compatibility requirements it might not be adequate to just delete a model element and add a new one or - even worse - to change the model element without notification. A life cycle state such as "obsolete" can be used to handle such a situation.

It cannot be expected that all existing elements and all new elements are equally "good". So it is supported to give hints to the user which elements are preferable to be used even if some are not forbidden to be used and still part of the standard.

Note that life cycle states will not allow to have two elements with the same `shortName` within the same `ARPackage`. In particular the name space concept of the meta-model (see Chapter 4.3.1) is not affected.

[TPS_GST_00239] Definition "Life Cycle" ⌈Life Cycle is the course of development/evolutionary stages of a model element during its life time.

A life cycle consists of a set of life cycle states. A life cycle state can be attached to an element in parallel to its version information.⌋*()*

A typical life cycle is {`valid`, `obsolete`} and means that a valid element is up to date when first introduced but is substituted later by a new one and therefore gets the life cycle state "obsolete".

According to Figure 12.1, life cycle support provides two basic means:

- Definition of a life cycle (`LifeCycleStateDefinitionGroup`) see chapter 12.2
- Application of a life cycle (`LifeCycleInfoSet`) see chapter 12.3

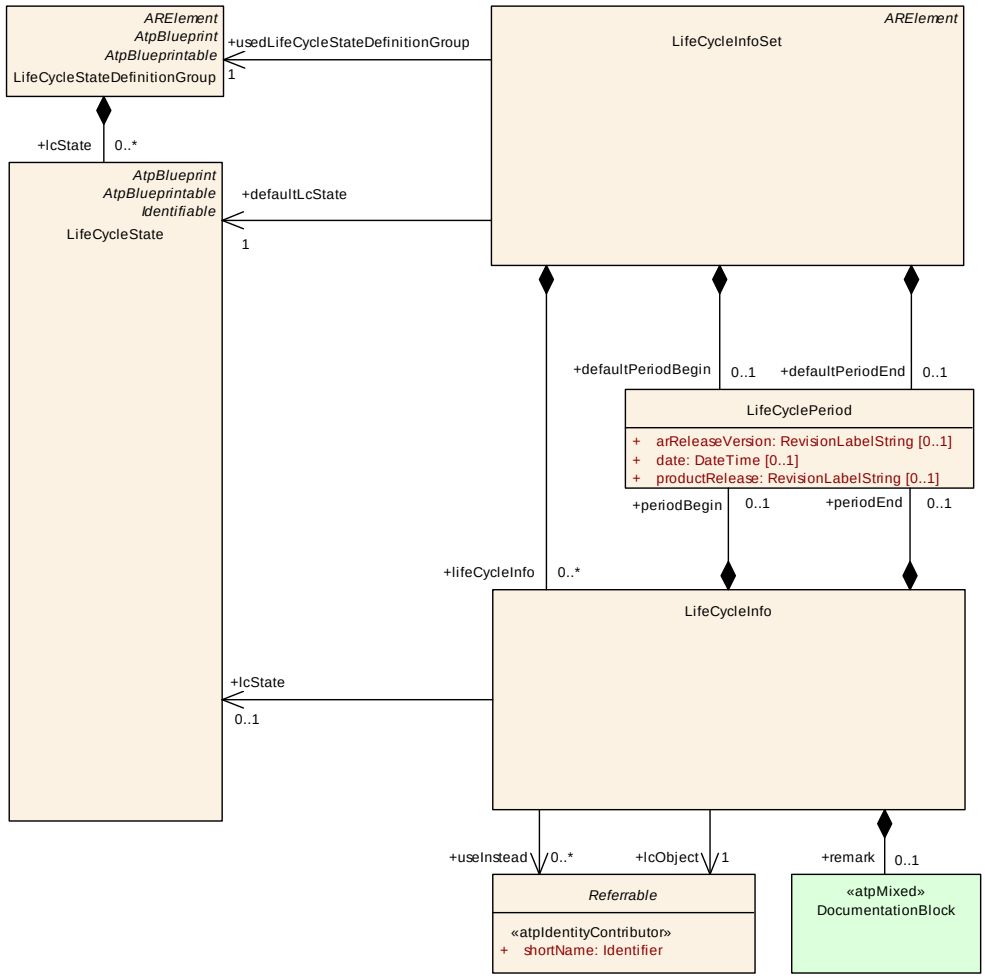


Figure 12.1: Definition of Life Cycles

12.2 Definition of a life cycle

[TPS_GST_00240] **LifeCycleStateDefinitionGroup** ⌈This is used to define a particular life cycle by specifying a group of LifeCycleStates. The purpose of the life cycle is expressed using desc of LifeCycleStateDefinitionGroup. Particular overview details may be expressed in introduction of LifeCycleStateDefinitionGroup.

Information about the purpose of a particular LifeCycleState is specified in the introduction of LifeCycleState.⌋*()*

The following hints apply:

- One model can simultaneously use multiple life cycles in order to support different aspects.
- It is recommended to keep the shortName unique across all LifeCycleStateDefinitionGroups.

Listing 12.1 illustrates the ARXML representation of the life cycle definition according to [TPS_GST_00051].

Listing 12.1: AUTOSAR Standard LifeCycleDefintion

```
<!-- LifeCycleStateDefinitionGroup: AutosarLifeCycleStates -->
<LIFE-CYCLE-STATE-DEFINITION-GROUP>
  <SHORT-NAME>AutosarLifeCycleStates</SHORT-NAME>
  <LONG-NAME>
    <L-4 L="EN">Life Cycle Definitions used in AUTOSAR Standards</L-4>
  </LONG-NAME>
  <DESC>
    <L-2 L="EN">This set represents the life cycle definitions used by
       AUTOSAR on M1 and M2 level. See also [TPS_GST_00051] respectively
       [TPS_GST_00064].</L-2>
  </DESC>
  <LC-STATES>
    <!-- LifeCycleState: valid -->
    <LIFE-CYCLE-STATE>
      <SHORT-NAME>valid</SHORT-NAME>
      <LONG-NAME>
        <L-4 L="EN">VALID</L-4>
      </LONG-NAME>
      <DESC>
        <L-2 L="EN">This indicates that the related entity is a valid part
            of the document. This is the default.</L-2>
      </DESC>
    </LIFE-CYCLE-STATE>
    <!-- LifeCycleState: draft -->
    <LIFE-CYCLE-STATE>
      <SHORT-NAME>draft</SHORT-NAME>
      <LONG-NAME>
        <L-4 L="EN">DRAFT</L-4>
      </LONG-NAME>
      <DESC>
        <L-2 L="EN">This indicates that the related entity is introduced
           newly in the (meta) model but still experimental. This
           information is published but is subject to be changed without
           backward compatibility management.</L-2>
      </DESC>
    </LIFE-CYCLE-STATE>
    <!-- LifeCycleState: candidate -->
    <LIFE-CYCLE-STATE>
      <SHORT-NAME>candidate</SHORT-NAME>
      <LONG-NAME>
        <L-4 L="EN">CANDIDATE</L-4>
      </LONG-NAME>
      <DESC>
        <L-2 L="EN">This indicates that the entity was introduced by a
           concept result and is still not validated.</L-2>
      </DESC>
    </LIFE-CYCLE-STATE>
    <!-- LifeCycleState: obsolete -->
    <LIFE-CYCLE-STATE>
      <SHORT-NAME>obsolete</SHORT-NAME>
      <LONG-NAME>
```

```
      <L-4 L="EN">OBSOLETE</L-4>
    </LONG-NAME>
    <DESC>
      <L-2 L="EN">This indicates that the related entity is obsolete and
          kept in the (meta) model for compatibility reasons. </L-2>
    </DESC>
    <INTRODUCTION>
      <P>
        <L-1 L="EN">If this life cycle state is set, the
        <TT TYPE="ARMetaClassRole">LifeCycleInfo.remark</TT> shall
           express the recommended alternative solution.</L-1>
      </P>
    </INTRODUCTION>
  </LIFE-CYCLE-STATE>
  <!-- LifeCycleState: removed -->
  <LIFE-CYCLE-STATE>
    <SHORT-NAME>removed</SHORT-NAME>
    <LONG-NAME>
      <L-4 L="EN">REMOVED</L-4>
    </LONG-NAME>
    <DESC>
      <L-2 L="EN">This indicates that the related entity is still in the
          (meta) model for whatever reason. It shall not be used and
         should not even appear in documents. </L-2>
    </DESC>
    <INTRODUCTION>
      <P>
```

Class	**LifeCycleStateDefinitionGroup**			
Package	M2::AUTOSARTemplates::GenericStructure::LifeCycles			
Note	This meta class represents the ability to define the states and properties of one particular life cycle. **Tags:**atp.recommendedPackage=LifeCycleStateDefintionGroups			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
lcState	LifeCycleState	*	aggr	Describes a single life cycle state of this life cycle state definition group.

Table 12.1: LifeCycleStateDefinitionGroup

Class	**LifeCycleState**			
Package	M2::AUTOSARTemplates::GenericStructure::LifeCycles			
Note	This meta class represents one particular state in the LifeCycle.			
Base	*ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table 12.2: LifeCycleState

12.3 Application of a life cycle

Figure 12.1 also shows the assignment of life cycle states to particular objects. This assignment is done by the meta-class LifeCycleInfoSet.

12.3.1 LifeCycleInfoSet

[TPS_GST_00241] LifeCycleInfoSet ⌈This denotes the actual assignment of life cycle states to particular objects. In addition to this it defines the applicable defaults for the life cycle state and the period of viability.⌋*()*

[constr_2581] Default life cycle state shall be defined properly ⌈defaultLcState in LifeCycleInfoSet shall reference to a lcState defined in the LifeCycleStateDefinitionGroup referenced by usedLifeCycleStateDefinitionGroup.⌋*()*

12.3.2 LifeCycleInfo

[TPS_GST_00242] LifeCycleInfo ⌈This meta-class specifies the life cycle state of a particular object denoted by lcObject. If no life cycle state is provided, then the defaultLcState applies.⌋*()*

[constr_2585] LifeCycleInfo shall be unambiguous ⌈Within one particular LifeCycleInfoSet lifeCycleInfo.lcObject shall be unique. This ensures that the association of a LifeCycleState to a Referrable is unambiguous.

This contraint applies for a particular point in time under consideration of the period of viability according to [TPS_GST_00244].⌋*()*

[constr_2583] Used life cycle state shall be defined properly ⌈defaultLcState in LifeCycleInfo shall reference to a lcState defined in the LifeCycleStateDefinitionGroup referenced by usedLifeCycleStateDefinitionGroup of the containing LifeCycleInfoSet.⌋*()*

[TPS_GST_00244] Viability Period of Life Cycle Info ⌈The viability of LifeCycleInfo can be restricted for a specific period of time by periodBegin respectively periodEnd. The following items apply:

- If no periodBegin is provided then defaultPeriodBegin applies.
- If no periodEnd is provided then defaultPeriodEnd applies.
- If also no defaultPeriodBegin respectively defaultPeriodEnd is specified then the period is unlimited (e.g. from 2013-01-01 until forever).
- Begin and End are considered to be included in the period. Thus at a given point in time t the definitions of LifeCycleInfo is viable if

$$periodBegin <= t <= periodEnd$$

⌋()

[constr_2586] Constraints on `LifeCyclePeriod` ⌈The attributes `date`, `arReleaseVersion`, `productRelease` in `LifeCyclePeriod` are mutually exclusive.⌋()

[TPS_GST_00238] Specifying replacement approach in `LifeCycleInfo` ⌈If the life cycle state indicates that the object is not the recommended one to use any more, `LifeCycleInfo.useInstead` refers to objects which should be used instead of the one denoted by `lcObject`. In this case `remark` may be used to document the details of the replacement.⌋()

Listing 12.2 shows the assignment of life cycle states to particular objects.

Listing 12.2: Example for a life cycle assignment

```
<LIFE-CYCLE-INFO-SET>
  <SHORT-NAME>AutosarObsoleteKeywords</SHORT-NAME>
  <LIFE-CYCLE-INFOS>
    <LIFE-CYCLE-INFO>
      <LC-OBJECT-REF DEST="KEYWORD">/AUTOSAR/AISpecification/KeywordSets/
          KeywordListLc/Abs</LC-OBJECT-REF>
      <LC-STATE-REF DEST="LIFE-CYCLE-STATE">/AUTOSAR/GeneralDefinitions/
          LifeCycleStateDefintionGroups/AutosarLifeCycleStates/obsolete</LC-
          STATE-REF>
      <PERIOD-BEGIN>
        <AR-RELEASE-VERSION>3.1.1</AR-RELEASE-VERSION>
      </PERIOD-BEGIN>
      <REMARK>
        <P>
          <L-1 L="EN">use Abs3 instead, obsolete since R3</L-1>
        </P>
      </REMARK>
      <USE-INSTEAD-REFS>
        <USE-INSTEAD-REF DEST="KEYWORD">/AUTOSAR/AISpecification/
            KeywordSets/KeywordListLc/Abs3</USE-INSTEAD-REF>
      </USE-INSTEAD-REFS>
    </LIFE-CYCLE-INFO>
  </LIFE-CYCLE-INFOS>
```

12.3.3 Propagation of LifeCycleState

[TPS_GST_00361] Propagation of `LifeCycleState` ⌈The `LifeCycleState` implicitly propagates to all contained children. `LifeCycleState` do not propagate to referenced objects (outgoing references). `LifeCycleState` do not propagate to referencing objects (incoming references).⌋()

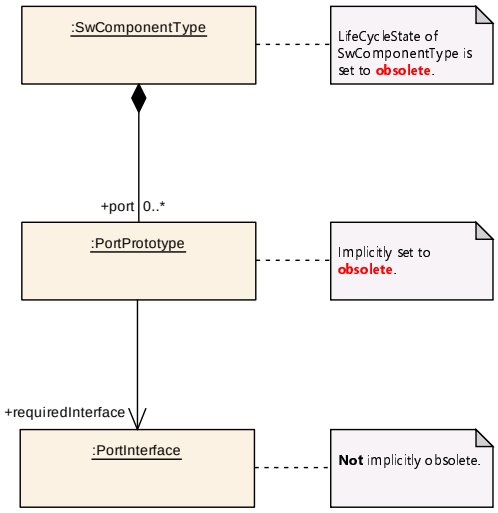


Figure 12.2: Propagation of LifeCycleState

Figure 12.2 illustrates the propagation of `LifeCycleState` in case `SwComponentType` is set to obsolete then:

1. all contained ports are implicitly set to obsolete as well,
2. all referenced objects (outgoing references) `PortInterface`s are not implicitly obsolete,
3. and all referencing objects (incoming references) as `SwComponentPrototype`s that are type by the `SwComponentType` are not implicitly obsolete.

For 1) and 3) a dedicated decision needs to be taken by the software component developer.

Class	**LifeCycleInfoSet**			
Package	M2::AUTOSARTemplates::GenericStructure::LifeCycles			
Note	This meta class represents the ability to attach a life cycle information to a particular set of elements. The information can be defined for a particular period. This supports the definition of transition plans. If no period is specified, the life cycle state applies forever. **Tags:**atp.recommendedPackage=LifeCycleInfoSets			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
defaultLcState	LifeCycleState	1	ref	This denotes the default life cycle state. To be used in all LifeCycleInfo elements within the LifeCycleInfoSet if no life cycle state is stated there explicitly. I.e. the defaultLc State can be overwritten in LifeCycleInfo elements.
defaultPeriod Begin	LifeCyclePeriod	0..1	aggr	Default starting point of period in which all the specified lifeCycleInfo apply. Note that the default period can be overridden for each lifeCycleInfo individually.
defaultPeriod End	LifeCyclePeriod	0..1	aggr	Default expiry date, i.e. default end point of period for which all specified lifeCycleInfo apply. Note that the default period can be overridden for each lifeCycleInfo individually.

△

Class	**LifeCycleInfoSet**			
lifeCycleInfo	LifeCycleInfo	*	aggr	This represents one particular life cycle information.
usedLifeCycle StateDefinition Group	LifeCycleStateDefinition Group	1	ref	This denotes the life cycle states applicable to the current life cycle info set.

Table 12.3: LifeCycleInfoSet

Class	**LifeCyclePeriod**			
Package	M2::AUTOSARTemplates::GenericStructure::LifeCycles			
Note	This meta class represents the ability to specify a point of time within a specified period, e.g. the starting or end point, in which a specific life cycle state is valid/applies to.			
Base	*ARObject*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
arRelease Version	RevisionLabelString	0..1	attr	Version of the AUTOSAR Release the element referred to is part of. The numbering contains three levels (major, minor, revision) which are defined by AUTOSAR. **Tags:**xml.sequenceOffset=20
date	DateTime	0..1	attr	Date within period. **Tags:**xml.sequenceOffset=10
productRelease	RevisionLabelString	0..1	attr	Version of the product within the period. **Tags:**xml.sequenceOffset=30

Table 12.4: LifeCyclePeriod

Class	**LifeCycleInfo**			
Package	M2::AUTOSARTemplates::GenericStructure::LifeCycles			
Note	LifeCycleInfo describes the life cycle state of an element together with additional information like what to use instead			
Base	*ARObject*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
lcObject	Referrable	1	ref	Element(s) have the life cycle as described in lcState.
lcState	LifeCycleState	0..1	ref	This denotes the particular state assigned to the object. If no lcState is given then the default life cycle state of Life CycleInfoSet is assumed.
periodBegin	LifeCyclePeriod	0..1	aggr	Starting point of period in which the element has the denoted life cycle state lcState. If no periodBegin is given then the default period begin of LifeCycleInfoSet is assumed.
periodEnd	LifeCyclePeriod	0..1	aggr	Expiry date, i.e. end point of period the element does not have the denoted life cycle state lcState any more. If no periodEnd is given then the default period begin of Life CycleInfoSet is assumed.
remark	DocumentationBlock	0..1	aggr	Remark describing for example • why the element was given the specified life cycle • the semantics of useInstead

▽

△

Class	**LifeCycleInfo**			
useInstead	Referrable	*	ref	Element(s) that should be used instead of the one denoted in referrable. Only relevant in case of life cycle states lcState unlike "valid". In case there are multiple references the exact semantics shall be individually described in the remark.

Table 12.5: LifeCycleInfo

13 Collections and Collectable Elements

[TPS_GST_00093] Collections ⌈For some use cases it is necessary to establish a collection of elements. Such collections are orthogonal to packages. Therefore a collection resides in a package but is established by associations to the collected elements.

A collection consists of an arbitrary number of flat references to `Referrable`s and/or instance references to `AtpFeature`s in different roles depending on the `category` and `collectionSemantics`. See [TPS_GST_00432], [TPS_GST_00434], and [TPS_GST_00347] for details.⌋*()*

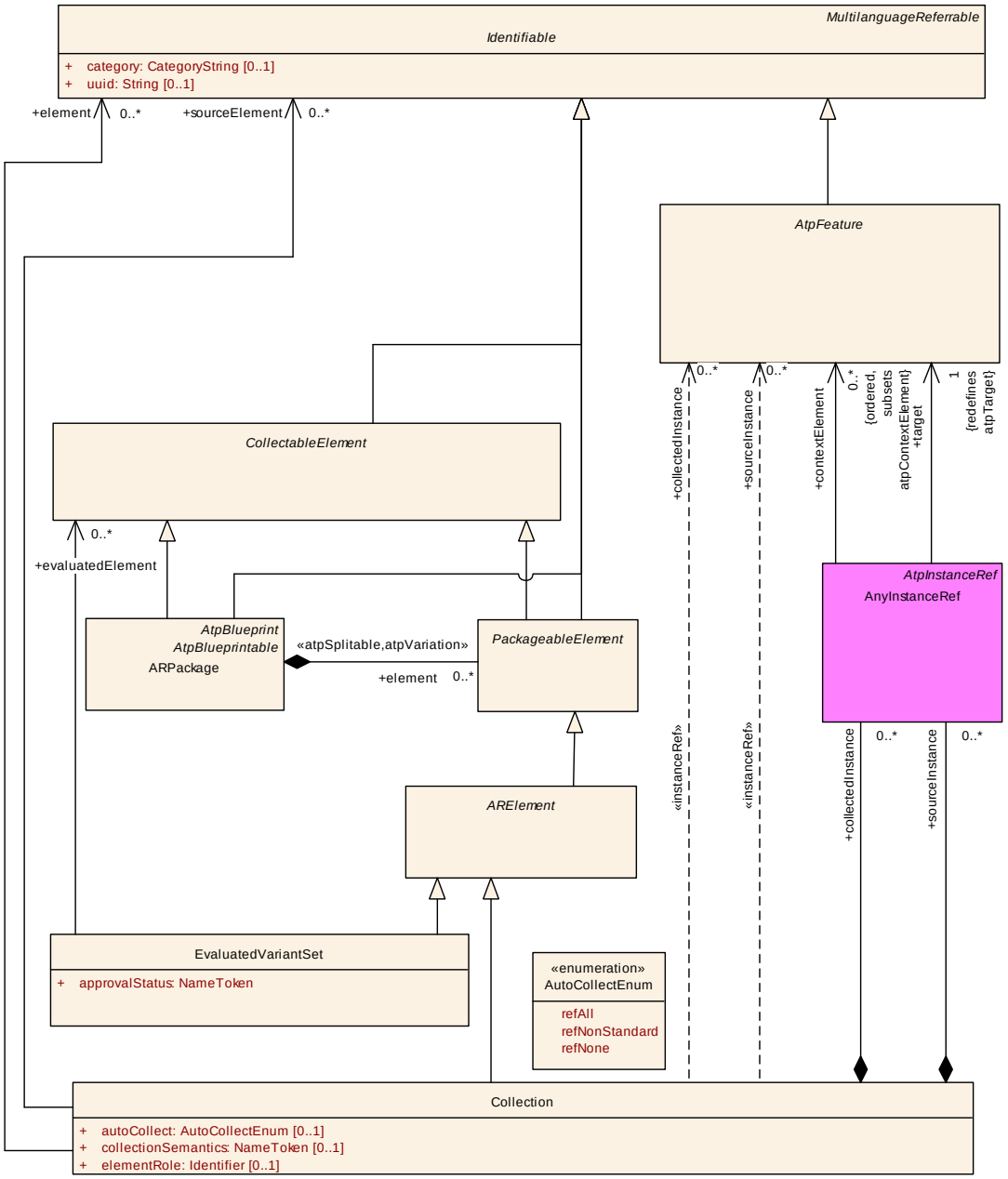


Figure 13.1: Collection

Example use cases for Collection are:

- evaluated variants (see Chapter 7.7)
- define a set of model elements with a specific semantic [TPS_GST_00434]
- establish relationships between objects [TPS_GST_00347]
- define subsets of models (VIEWs on model) [TPS_GST_00348]

[TPS_GST_00346] Automatic Collections ⌈autoCollect denotes if targets of references in the collected elements and collectedInstances are also considered part of the collection. See AutoCollectEnum for particular details.⌋()

[TPS_GST_00434] Expressing a set of elements by collections ⌈Collections can be used to establish a set of elements. In this case

- category of Collection is SET
- element represents flat references of elements belonging to the collection
- collectedInstance represents instance references of elements belonging to the collection
- elementRole is optional and specifies the role of the collected elements and/or collectedInstances. See [TPS_GST_00349].
- collectionSemantics is optional and specifies the semantics of the Collection in the intended use case. See [TPS_GST_00432].

⌋()

[TPS_GST_00347] Expressing Relationships by collections ⌈Collections can also be used to establish relationships between objects. In this case

- category of Collection is RELATION
- Either sourceElement or sourceInstance is mandatory and represents the source end of the relationship
- element and/or collectedInstance (the collected elements) represents the target end of the relationship
- elementRole is mandatory if autoCollect is applied and specifies the role of the collected elements and/or collectedInstances. See [TPS_GST_00349].
- collectionSemantics is optional and specifies the semantics of the Collection in the intended use case. See [TPS_GST_00432].

⌋()

[TPS_GST_00348] Standardized category of Collection ⌈The following values are standardized for category of Collection:

RELATION this indicates that the collection is used to express a relation (see [TPS_GST_00347]).

SET this indicates that the collection is simply a set of elements. This is the default if no category is specified.

⌋()

[constr_2635] No custom values for `Collection.category` ⌈It is not allowed to define any custom or project-specific value of the attribute `Collection.category`.⌋ ()

[TPS_GST_00432] Semantics of a `Collection` ⌈The attribute `collectionSemantics` may be used to express the semantics of a `Collection` depending on the intended use case. The `collectionSemantics` must be agreed by all stakeholders.⌋ ()

A typical example for `collectionSemantics` is the definition of a ServiceInterface for the classic platform using a `Collection` with category `SET` and `collectionSemantics`=`SO_SERVICE_INTERFACE` as described in [TPS_SYST_02283].

[TPS_GST_00349] Standardized `elementRole` of `Collection` ⌈The following values are standardized for `elementRole` of `Collection`:

AUTO_COLLECTED_FROM this is applied if the `Collection` represents the relationship between two equivalent collections of which one is using automatic inclusion of referenced elements and the other one is the equivalent resolved collection. The `category` of the `Collection` is `RELATION`. [TPS_GST_00347]).

PART_OF_SUBSET this indicates that the `element`s respectively the `collectedInstance` in the `Collection` are part of a particular subset of a model. The `category` of the `Collection` is `SET`.

⌋()

[constr_2636] No custom values for `Collection.elementRole` ⌈It is not allowed to define any custom or project-specific value of the attribute `Collection.elementRole`.⌋()

Figure 13.2 illustrates an example for a relation according to [TPS_GST_00349][1]. In this case we have two collections, both representing the same view (even if `elementRole` is not shown in the diagram). One representation uses "auto collect" while the other one is the resolved representation.

[1] Don't be confused by the fact that the collection establishes a relationship between collections.

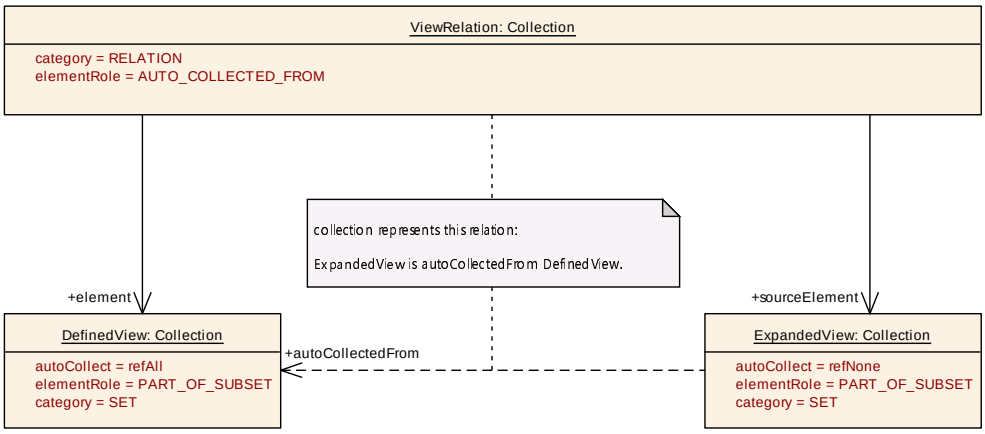


Figure 13.2: Example for Collection as Relation

Listing 13.1 shows the representation in ARXML.

Listing 13.1: Example for Collection as Relation (1)

```
<COLLECTION>
  <SHORT-NAME>DefinedView</SHORT-NAME>
  <CATEGORY>SET</CATEGORY>
  <AUTO-COLLECT>REF-ALL</AUTO-COLLECT>
  <ELEMENT-ROLE>PART_OF_SUBSET</ELEMENT-ROLE>
  <ELEMENT-REFS>
    <ELEMENT-REF BASE="OPEN" DEST="PORT-PROTOTYPE-BLUEPRINT">EngN</ELEMENT
        -REF>
  </ELEMENT-REFS>
</COLLECTION>
<COLLECTION>
  <SHORT-NAME>ExpandedView</SHORT-NAME>
  <CATEGORY>SET</CATEGORY>
  <AUTO-COLLECT>REF-NONE</AUTO-COLLECT>
  <ELEMENT-ROLE>PART_OF_SUBSET</ELEMENT-ROLE>
  <ELEMENT-REFS>
    <ELEMENT-REF BASE="OPEN" DEST="PORT-PROTOTYPE-BLUEPRINT">EngN</ELEMENT
        -REF>
    <ELEMENT-REF BASE="OPEN" DEST="PORT-INTERFACE">EngN1</ELEMENT-REF>
    <ELEMENT-REF BASE="OPEN" DEST="APPLICATION-PRIMITIVE-DATA-TYPE">N1</
        ELEMENT-REF>
    <!-- futher elements are not shown in this example -->
  </ELEMENT-REFS>
</COLLECTION>
<COLLECTION>
  <SHORT-NAME>ViewRelation</SHORT-NAME>
  <CATEGORY>RELATION</CATEGORY>
  <ELEMENT-ROLE>AUTO_COLLECTED_FROM</ELEMENT-ROLE>
  <ELEMENT-REFS>
    <ELEMENT-REF BASE="Coll" DEST="COLLECTION">ExpandedView</ELEMENT-REF>
  </ELEMENT-REFS>
  <SOURCE-ELEMENT-REFS>
    <SOURCE-ELEMENT-REF BASE="Coll" DEST="COLLECTION">DefinedView</SOURCE-
        ELEMENT-REF>
  </SOURCE-ELEMENT-REFS>
</COLLECTION>
```

Listing 13.2 illustrates another example for a relation according to [TPS_GST_00349]. In this case it shows the "stem"-relationship of two keywords.

Listing 13.2: Example for Collection as Relation (2)

```
<COLLECTION>
  <SHORT-NAME>Drv_declinations</SHORT-NAME>
  <CATEGORY>RELATION</CATEGORY>
  <COLLECTION-SEMANTICS>DECLINATION_OF</COLLECTION-SEMANTICS>
  <ELEMENT-REFS>
    <ELEMENT-REF BASE="KW" DEST="KEYWORD">KeywordList/Drvr</ELEMENT-REF>
    <ELEMENT-REF BASE="KW" DEST="KEYWORD">KeywordList/Drvg</ELEMENT-REF>
  </ELEMENT-REFS>
  <SOURCE-ELEMENT-REFS>
    <SOURCE-ELEMENT-REF BASE="KW" DEST="KEYWORD">KeywordList/Drv</SOURCE-
      ELEMENT-REF>
  </SOURCE-ELEMENT-REFS>
</COLLECTION>
```

Class	**Collection**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::ElementCollection			
Note	This meta-class specifies a collection of elements. A collection can be utilized to express additional aspects for a set of elements. Note that Collection is an ARElement. Therefore it is applicable e.g. for EvaluatedVariant, even if this is not obvious. Usually the category of a Collection is "SET". On the other hand, a Collection can also express an arbitrary relationship between elements. This is denoted by the category "RELATION" (see also [TPS_GST_00347]). In this case the collection represents an association from "sourceElement" to "targetElement" in the role "role". **Tags:**atp.recommendedPackage=Collections			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
autoCollect	AutoCollectEnum	0..1	attr	This attribute reflects how far the referenced objects are part of the collection. **Tags:**xml.sequenceOffset=20
collected Instance	AtpFeature	*	iref	This instance ref supports the use case that a particular instance is part of the collection. **Tags:**xml.sequenceOffset=60 **InstanceRef implemented by:**AnyInstanceRef
collection Semantics	NameToken	0..1	attr	Provides the ability to express the semantics of a Collection depending on the intended use case. The collectionSemantics is specified as a NameToken which must be agreed by all stakeholders. **Tags:**xml.sequenceOffset=25
element	Identifiable	*	ref	This is an element in the collection. Note that Collection itself is collectable. Therefore collections can be nested. In case of category="RELATION" this represents the target end of the relation. **Tags:**xml.sequenceOffset=40

▽

△

Class	**Collection**			
elementRole	Identifier	0..1	attr	This attribute allows to denote a particular role of the collection. Note that the applicable semantics shall be mutually agreed between the two parties. In particular it denotes the role of element in the context of sourceElement. **Tags:**xml.sequenceOffset=30
sourceElement	Identifiable	*	ref	Only if Category = "RELATION". This represents the source of a relation. **Tags:**xml.sequenceOffset=50
sourceInstance	AtpFeature	*	iref	Only if Category = "RELATION". This represents the source instance of a relation. **Tags:**xml.sequenceOffset=70 **InstanceRef implemented by:**AnyInstanceRef

Table 13.1: Collection

Enumeration	**AutoCollectEnum**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::ElementCollection
Note	This enumerator defines the possible approaches to determine the final set of elements in a collection.
Literal	*Description*
refAll	All objects being referenced (recursively) from the objects mentioned directly in the collection are also considered as part of the collection. **Tags:**atp.EnumerationLiteralIndex=0
refNone	This indicates that only those objects mentioned directly in the collection are part of the collection. No other objects are considered further. **Tags:**atp.EnumerationLiteralIndex=1
refNonStandard	This indicates that non standard objects ([TPS_GST_00088]) referenced (recursively) by the objects mentioned directly in the collection are also considered to be part of the collection. **Tags:**atp.EnumerationLiteralIndex=2

Table 13.2: AutoCollectEnum

Class	***CollectableElement*** (abstract)			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::ElementCollection			
Note	This meta-class specifies the ability to be part of a specific AUTOSAR collection of ARPackages or ARElements. The scope of collection has been extended beyond CollectableElement with Revision 4.0.3. For compatibility reasons the name of this meta Class was not changed.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	ARPackage, *PackageableElement*			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
–	–	–	–	–

Table 13.3: CollectableElement

14 Mapping Views

Along the development of an AUTOSAR System, various transformations may take place within the model. This leads to the fact that the model represents different views on the same system. These different views can be mapped to each other with the help of ViewMap.

[TPS_GST_00350] Mapping Model Elements of different Views ⌈ViewMap represents an non directed relationship between two model elements. The general semantics is that the mapped elements are mainly the same entity but represent a different view. A refined semantics of the mapping can be specified in role.⌋*()*

Note that even if it is called "view", view in DocumentViewSelectable represents another concept and should not be confused with ViewMap.

Note further that ViewMap represents a specific relationship in contrast to Collection according to [TPS_GST_00347] which can be used to represent arbitrary but **directed** relationships.

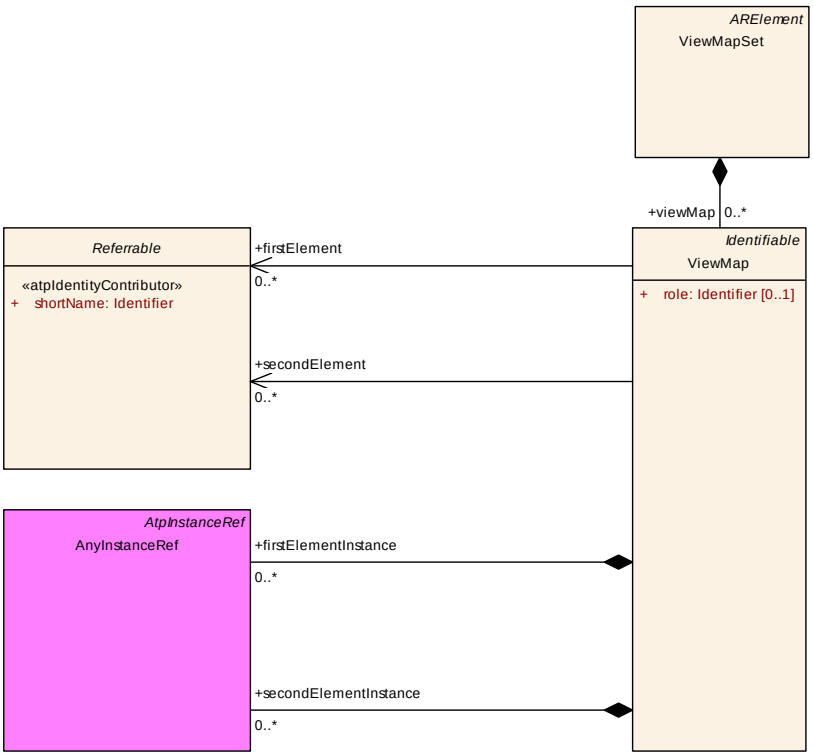


Figure 14.1: View Map

Example use cases for ViewMap are (see [TPS_SYST_01136]):

- Mapping between Abstract System Description to a System Description
- Mapping between System Description and System Extract

Class	**ViewMapSet**			
Package	M2::AUTOSARTemplates::GenericStructure::ViewMapSet			
Note	Collection of ViewMaps that are used to establish relationships between different AUTOSAR artifacts. **Tags:**atp.recommendedPackage=ViewMapSets			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
viewMap	ViewMap	*	aggr	ViewMaps that are collected by the ViewMapSet.

Table 14.1: ViewMapSet

Class	**ViewMap**			
Package	M2::AUTOSARTemplates::GenericStructure::ViewMapSet			
Note	The ViewMap allows to relate any number of elements on the "first" side to any number of elements on the "second" side. Since the ViewMap does not address a specific mapping use-case the roles "first" and "second" shall imply this generality. This mapping allows to trace transformations of artifacts within the AUTOSAR environment. The references to the mapped elements can be plain references and/or InstanceRefs.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
firstElement	Referrable	*	ref	Reference to identifiable elements on the first "side". **Tags:**xml.sequenceOffset=20
firstElement Instance	AtpFeature	*	iref	InstanceRefs to elements on the first "side". **Tags:**xml.sequenceOffset=50 **InstanceRef implemented by:**AnyInstanceRef
role	Identifier	0..1	attr	This attribute is used to describe specific mapping scenarios, e.g. the mappings: • AR_AbstractSystemDescription_System Description • AR_SystemDescription_SystemExtract **Tags:**xml.sequenceOffset=10
secondElement	Referrable	*	ref	Reference to identifiable elements on the second "side". **Tags:**xml.sequenceOffset=30
secondElement Instance	AtpFeature	*	iref	InstanceRefs to elements on the second "side". **Tags:**xml.sequenceOffset=60 **InstanceRef implemented by:**AnyInstanceRef

Table 14.2: ViewMap

A Glossary

Artifact This is a Work Product Definition that provides a description and definition for tangible work product types. Artifacts may be composed of other artifacts ([24]).

At a high level, an artifact is represented as a single conceptual file.

AUTOSAR Tool This is a software tool which supports one or more tasks defined as AUTOSAR tasks in the methodology. Depending on the supported tasks, an AUTOSAR tool can act as an authoring tool, a converter tool, a processor tool or as a combination of those (see separate definitions).

AUTOSAR Authoring Tool An AUTOSAR Tool used to create and modify AUTOSAR XML Descriptions. Example: System Description Editor.

AUTOSAR Converter Tool An AUTOSAR Tool used to create AUTOSAR XML files by converting information from other AUTOSAR XML files. Example: ECU Flattener

AUTOSAR Definition This is the definition of parameters which can have values. One could say that the parameter values are Instances of the definitions. But in the meta model hierarchy of AUTOSAR, definitions are also instances of the meta model and therefore considered as a description. Examples for AUTOSAR definitions are: `EcucParameterDef`, `PostBuildVariantCriterion`, `SwSystemconst`.

AUTOSAR XML Description In AUTOSAR this means "filled Template". In fact an AUTOSAR XML description is the XML representation of an AUTOSAR model.

The AUTOSAR XML description can consist of several files. Each individual file represents an AUTOSAR partial model and shall validate successfully against the AUTOSAR XML schema.

AUTOSAR Meta-Model This is an UML2.0 model that defines the language for describing AUTOSAR systems. The AUTOSAR meta-model is an UML representation of the AUTOSAR templates. UML2.0 class diagrams are used to describe the attributes and their interrelationships. Stereotypes, UML tags and OCL expressions (object constraint language) are used for defining specific semantics and constraints.

AUTOSAR Meta-Model Tool The AUTOSAR Meta-Model Tool is the tool that generates different views (class tables, list of constraints, diagrams, XML Schema etc.) on the AUTOSAR meta-model.

AUTOSAR Model This is a representation of an AUTOSAR product. The AUTOSAR model represents aspects suitable to the intended use according to the AUTOSAR methodology.

Strictly speaking, this is an instance of the AUTOSAR meta-model. The information contained in the AUTOSAR model can be anything that is representable according to the AUTOSAR meta-model.

AUTOSAR Partial Model In AUTOSAR, the possible partitioning of models is marked in the meta-model by ≪atpSplitable≫. One partial model is represented in an AUTOSAR XML description by one file. The partial model does not need to fulfill all semantic constraints applicable to an AUTOSAR model.

AUTOSAR Processor Tool An AUTOSAR Tool used to create non-AUTOSAR files by processing information from AUTOSAR XML files. Example: RTE Generator

AUTOSAR Specification Element An AUTOSAR Specification Element is a named element that is part of an AUTOSAR specification. Examples: requirement, constraint, specification item, class or attribute in the meta model, methodology, deliverable, methodology activity, model element, bsw module etc.

AUTOSAR Template The term "Template" is used in AUTOSAR to describe the format different kinds of descriptions. The term template comes from the idea, that AUTOSAR defines a kind of form which shall be filled out in order to describe a model. The filled form is then called the description.

In fact the AUTOSAR templates are now defined as a meta-model.

AUTOSAR Validation Tool A specialized AUTOSAR Tool which is able to check an AUTOSAR model against the rules defined by a profile.

AUTOSAR XML Schema This is a W3C XML schema that defines the language for exchanging AUTOSAR models. This Schema is derived from the AUTOSAR meta-model. The AUTOSAR XML Schema defines the AUTOSAR data exchange format.

Blueprint This is a model from which other models can be derived by copy and refinement. Note that in contrast to meta model resp. types, this process is *not* an instantiation.

Instance Generally this is a particular exemplar of a model or of a type.

Life Cycle Life Cycle is the course of development/evolutionary stages of a model element during its life time.

Meta-Model This defines the building blocks of a model. In that sense, a Meta-Model represents the language for building models.

Meta-Data This includes pertinent information about data, including information about the authorship, versioning, access-rights, timestamps etc.

Model A Model is an simplified representation of reality. The model represents the aspects suitable for an intended purpose.

Partial Model This is a part of a model which is intended to be persisted in one particular artifact.

Pattern in GST This is an approach to simplify the definition of the meta model by applying a model transformation. This transformation creates an enhanced model out of an annotated model.

Profile Authoring Support Data Data that is used for efficient authoring of a profile. E.g. list of referable constraints, meta-classes, meta-attributes or other reusable model assets (blueprints)

Profile Authoring Tool A specialized `AUTOSAR Tool` which focuses on the authoring of profiles for data exchange points. It e.g. provides support for the creation of profiles from scratch, modification of existing profiles or composition of existing profiles.

Profile Compatibility Checker Tool A specialized `AUTOSAR Tool` which focuses on checking the compatibility of profiles for data exchange. Note that this compatibility check includes manual compatibility checks by engineers and automated assistance using more formal algorithms.

Profile Consistency Checker Tool A specialized `AUTOSAR Tool` which focuses on checking the consistency of profiles.

Property A property is a structural feature of an object. As an example a “connector” has the properties “receive port” and “send port”

Properties are made variant by the ≪`atpVariation`≫.

Prototype This is the implementation of a role of a type within the definition of another type. In other words a type may contain Prototypes that in turn are typed by "Types". Each one of these prototypes becomes an instance when this type is instantiated.

Type A type provides features that can appear in various roles of this type.

Value This is a particular value assigned to a “Definition”.

Variability Variability of a system is its quality to describe a set of variants. These variants are characterized by variant specific property settings and / or selections. As an example, such a system property selection manifests itself in a particular “receive port” for a connection.

This is implemented using the ≪`atpVariation`≫.

Variant A system variant is a concrete realization of a system, so that all its properties have been set respectively selected. The software system has no variability anymore with respect to the binding time.

This is implemented using `EvaluatedVariantSet`.

Variation Binding A variant is the result of a variation binding process that resolves the variability of the system by assigning particular values/selections to all the system’s properties.

This is implemented by `VariationPoint`.

Variation Binding Time The variation binding time determines the step in the methodology at which the variability given by a set of variable properties is resolved.

This is implemented by `vh.LatestBindingtime` at the related properties.

Variation Definition Time The variation definition time determines the step in the methodology at which the variation points are defined.

Variation Point A variation point indicates that a property is subject to variation. Furthermore, it is associated with a condition and a binding time which define the system context for the selection / setting of a concrete variant.

This is implemented by `VariationPoint`.

B Constraint History

B.1 Constraint History R4.0.1

B.1.1 Added Constraints

Number	Heading
[constr_2501]	Blueprint of blueprints are not supported
[constr_2502]	Merged model shall be compliant to the meta-model.
[constr_2503]	Bound model shall be compliant to the meta-model
[constr_2504]	Constraint to latest binding time
[constr_2505]	Multiplicity after binding
[constr_2506]	Attributes in property set pattern
[constr_2507]	`EvaluatedVariantSet`
[constr_2508]	`shortName`
[constr_2509]	`ReferenceBase`
[constr_2510]	only one default `ReferenceBase`
[constr_2511]	Named reference bases shall be available
[constr_2512]	`shortName` uniqueness constraint for variants
[constr_2513]	splitted variants shall have a `shortLabel`
[constr_2514]	`shortLabel` in `VariationPoint` shall be unique
[constr_2515]	Avoid conflicting package categories
[constr_2516]	Return type of Formula
[constr_2517]	`postbuildVariantCondition` only for PostBuild
[constr_2518]	Binding time is constrained
[constr_2519]	`PredefinedVariants` need to be consistent
[constr_2520]	Nesting of lists shall be limited
[constr_2521]	The `shortLabel` in `VariationPoint` shall be unique
[constr_2522]	Notes should not be nested
[constr_2523]	Used languages need to be consistent
[constr_2524]	Non splitable elements in one file
[constr_2525]	Non splitable elements shall not be repeated
[constr_2530]	`InstanceRefs` shall be consistent
[constr_2531]	`AtpInstanceRef` shall be close to the base
[constr_2533]	Documentation context is either a feature or an identifiable
[constr_2534]	Limits of unlimited Integer

Table B.1: Added Constraints in 4.0.1

B.2 Constraint History R4.0.2

B.2.1 Added Constraints

Number	Heading
[constr_2537]	Variation of packagable element is limited to components resp. modules
[constr_2538]	Global reference is limited to certain elements

Table B.2: Added Constraints in 4.0.2

B.2.2 Changed Constraints

Number	Heading
[constr_2511]	Named reference bases shall be available
[constr_2519]	`PredefinedVariants` need to be consistent

Table B.3: Changed Constraints in 4.0.2

B.3 Constraint History R4.0.3

B.3.1 Added Constraints

Number	Heading
[constr_2547]	ordered collections cannot be split into partial models
[constr_2557]	no `VariationPoint`s with `latestBindingTime` set to `BlueprintDerivation` in system configurations
[constr_2558]	Only `blueprintCondition/blueprintValue` if `vh.latestBindingTime` is `BlueprintDerivationTime`
[constr_2559]	no nested `VariationPoint`
[constr_4055]	ICS may not contain blueprints

Table B.4: Added Constraints in 4.0.3

B.3.2 Changed Constraints

Number	Heading
[constr_2530]	`InstanceRefs` shall be consistent
[constr_2508]	Name space of `shortName`

Table B.5: Changed Constraints in 4.0.3

B.4 Constraint History R4.1.1

B.4.1 Added Constraints

Number	Heading

Table B.6: Added Constraints in 4.1.1

B.4.2 Changed Constraints

Number	Heading

Table B.7: Changed Constraints in 4.1.1

B.4.3 Deleted Constraints

Number	Heading

Table B.8: Deleted Constraints in 4.1.1

B.4.4 Added Specification Items

Number	Heading
[TPS_GST_00045]	Inherited properties in mixed content
[TPS_GST_00046]	Splitable collections
[TPS_GST_00047]	Identification of Partial Models
[TPS_GST_00048]	Splitable up to the Root
[TPS_GST_00049]	`atp.recommendedPackage`
[TPS_GST_00050]	`atp.Splitkey`
[TPS_GST_00051]	`atp.Status`
[TPS_GST_00052]	`vh.latestBindingTime`
[TPS_GST_00053]	`xml.xsd.*` etc.
[TPS_GST_00054]	`xml.xsd.customType`
[TPS_GST_00055]	`xml.attribute`
[TPS_GST_00056]	`xml.attributeRef`
[TPS_GST_00057]	`xml.enforceMinMultiplicity`
[TPS_GST_00058]	`xml.enforceMaxMultiplicity`
[TPS_GST_00059]	`xml.globalElement`
[TPS_GST_00060]	`xml.mds.type`
[TPS_GST_00061]	`xml.name`
[TPS_GST_00062]	`xml.nsPrefix`
[TPS_GST_00063]	`xml.nsUri`
[TPS_GST_00064]	`xml.roleElement`, `xml.roleWrapperElement`, `xml.typeElement`, `xml.typeWrapperElement`
[TPS_GST_00065]	`xml.sequenceOffset`
[TPS_GST_00066]	`xml.systemIdentifier`
[TPS_GST_00067]	`admin.documentClassification`
[TPS_GST_00068]	`admin.documentIdentificationNo`
[TPS_GST_00069]	`admin.documentOwner`
[TPS_GST_00070]	`admin.documentResponsibility`
[TPS_GST_00071]	`admin.documentStatus`
[TPS_GST_00072]	`admin.documentTitle`
[TPS_GST_00073]	`admin.documentVersion`
[TPS_GST_00074]	`admin.partOfRelease`
[TPS_GST_00075]	`admin.releaseDate`
[TPS_GST_00076]	`admin.revision`
[TPS_GST_00077]	Top-Level Structure of an AUTOSAR Model
[TPS_GST_00078]	AUTOSAR top level AdminData
[TPS_GST_00079]	Language Status of an Artifact
[TPS_GST_00080]	Package Structure for AUTOSAR delivered Models
[TPS_GST_00081]	Pattern for AUTOSAR delivered Models
[TPS_GST_00082]	Package Structure for ECUC parameter definitions
[TPS_GST_00083]	Pattern for AUTOSAR defined Model Elements
[TPS_GST_00084]	Pattern for AUTOSAR defined Model Elements
[TPS_GST_00085]	Pattern for AUTOSAR defined Model Elements
[TPS_GST_00086]	Category of `ARPackage`
[TPS_GST_00087]	`BLUEPRINT`
[TPS_GST_00088]	`STANDARD`

[TPS_GST_00089]	EXAMPLE
[TPS_GST_00090]	Non Standardized Category of ARPackage
[TPS_GST_00091]	ARObject
[TPS_GST_00092]	The purpose of a ARPackage
[TPS_GST_00093]	Collections
[TPS_GST_00095]	Main Purpose of Identifiable
[TPS_GST_00096]	Main Purpose of Referrable
[TPS_GST_00097]	Purpose of shortName
[TPS_GST_00098]	Recommendation to Choose Human Readable shortNames
[TPS_GST_00099]	Purpose of longName
[TPS_GST_00100]	Purpose of desc
[TPS_GST_00101]	Purpose of adminData
[TPS_GST_00102]	Purpose of category
[TPS_GST_00103]	Purpose of introduction
[TPS_GST_00104]	Purpose of annotation
[TPS_GST_00105]	Control of the Document Language by AdminData
[TPS_GST_00106]	Version Management
[TPS_GST_00107]	Merge Operations in Version Management
[TPS_GST_00108]	Special Information in Version Management
[TPS_GST_00109]	Abstraction of Artifacts from Physical File Syetms
[TPS_GST_00110]	EngineeringObject can be resolved via a container catalog as defined in [16] in order to find the physical File.
[TPS_GST_00111]	Negation Operator
[TPS_GST_00112]	Exponentiation Operator
[TPS_GST_00113]	Multiplicative Operator / division
[TPS_GST_00114]	Additive Operator
[TPS_GST_00115]	Shift Operator
[TPS_GST_00116]	Ranking Operator
[TPS_GST_00117]	Comparison: equality
[TPS_GST_00118]	Bit-wise AND
[TPS_GST_00119]	Bit-wise XOR
[TPS_GST_00120]	Bit-wise OR
[TPS_GST_00121]	Boolean AND
[TPS_GST_00122]	Boolean XOR
[TPS_GST_00123]	Boolean OR
[TPS_GST_00124]	Round Function
[TPS_GST_00125]	Round Up Function
[TPS_GST_00126]	Round Down Function
[TPS_GST_00127]	Absolute Value
[TPS_GST_00128]	Natural Logarithm
[TPS_GST_00129]	Decimal Logarithm
[TPS_GST_00130]	Square Root
[TPS_GST_00131]	Sinus
[TPS_GST_00132]	Arcus Sinus
[TPS_GST_00133]	Cosinus
[TPS_GST_00134]	Arcus Cosiuns
[TPS_GST_00135]	Sinus Hyperbolicus
[TPS_GST_00136]	Cosinus Hyperbolicus
[TPS_GST_00137]	Tangens
[TPS_GST_00138]	Arcus Tangens
[TPS_GST_00139]	Tangens Hyperbolicus
[TPS_GST_00140]	Exponential
[TPS_GST_00141]	Is Defined

[TPS_GST_00142]	Signum
[TPS_GST_00143]	Maximum Value
[TPS_GST_00144]	Minium Value
[TPS_GST_00145]	Power Function
[TPS_GST_00146]	Case Sensitive String Compare
[TPS_GST_00147]	Non Case Insensitive String Compare
[TPS_GST_00148]	Annotation
[TPS_GST_00149]	Usage of MultiDimensionalTime
[TPS_GST_00150]	Derived Attributes Do not Appear in the XML Schema
[TPS_GST_00151]	Specializations of Derived Relations
[TPS_GST_00152]	Derived Union
[TPS_GST_00153]	Applying Abstract Structures
[TPS_GST_00154]	Specialization of Relations
[TPS_GST_00155]	Representation of Classifier and Feature
[TPS_GST_00156]	Purpose of `AtpClassifier`
[TPS_GST_00157]	Purpose of `AtpPrototype`
[TPS_GST_00158]	Purpose of `AtpStructureElement`
[TPS_GST_00159]	Deriving features in abstract structures
[TPS_GST_00160]	Instance Reference
[TPS_GST_00161]	Definition of an instance ref
[TPS_GST_00162]	Context path in instance ref
[TPS_GST_00163]	Annotated meta-model
[TPS_GST_00164]	Extended meta-model
[TPS_GST_00165]	specification of a transformation pattern
[TPS_GST_00166]	Model Transformation for Primitives
[TPS_GST_00167]	Case Sensitivity of References
[TPS_GST_00168]	Representation of Type Reference
[TPS_GST_00169]	Absolute `shortName`-Path
[TPS_GST_00170]	Relative `ShortName`-path
[TPS_GST_00171]	Identifying the `ReferenceBase` of a Relative Reference
[TPS_GST_00172]	`ReferenceBase` in Partial Models
[TPS_GST_00173]	Destination Type
[TPS_GST_00174]	Variant Handling Terminology
[TPS_GST_00175]	Variant-rich M1 model
[TPS_GST_00176]	Bound M1 model
[TPS_GST_00177]	Remove Deselected Objects
[TPS_GST_00178]	Remove Binding Function upon Binding
[TPS_GST_00179]	Scope of Variant Handling Specification
[TPS_GST_00180]	Resolving Variation Points along the Development Steps
[TPS_GST_00181]	Annotated meta-model for Variant Handling
[TPS_GST_00182]	Notation of Latest Binding time on M2
[TPS_GST_00183]	Representation of Binding Time
[TPS_GST_00185]	Transformation on meta-model
[TPS_GST_00186]	Description of Variation on M1
[TPS_GST_00187]	Choosing a Particular Variant
[TPS_GST_00188]	Resolving Variation Points
[TPS_GST_00189]	Variation is Restricted to Specific Elements.
[TPS_GST_00190]	Semantic of `bindingTime`
[TPS_GST_00191]	Variant Handling Patterns can be Mixed
[TPS_GST_00192]	Variant Handling Extends Upper Multiplicity
[TPS_GST_00193]	Order of Pattern Resolution in Variant Handling
[TPS_GST_00194]	Variation Points are Optional
[TPS_GST_00195]	Annotated meta-model Defines Applicable Variation Points

[TPS_GST_00196]	ICS
[TPS_GST_00197]	Pure meta-model
[TPS_GST_00198]	Attributes for all meta-classes
[TPS_GST_00199]	Transformation defined by Aggregation Pattern
[TPS_GST_00200]	Schema Generator avoids duplicate VariationPoints
[TPS_GST_00201]	Aggregation Pattern on Primitives
[TPS_GST_00202]	Limitation of non post build
[TPS_GST_00203]	Transformation defined by Association Pattern
[TPS_GST_00204]	Handling of non variant associations
[TPS_GST_00205]	Transformation defined by Attribute Value Pattern
[TPS_GST_00206]	Special meta-classes for AttriuteValueVariationPoint
[TPS_GST_00207]	No Binding time required for Constants
[TPS_GST_00209]	No postbuild variation for attribute values
[TPS_GST_00210]	Multiplicity of AttributeValueVariationPoint
[TPS_GST_00211]	AttributeValueVariationPoint does not support PostBuild Variation
[TPS_GST_00212]	Existence of Attribute cannot be subject to Variation
[TPS_GST_00213]	Arrays should have the same Binding Time
[TPS_GST_00214]	Extending the Application of Attribute Value Pattern
[TPS_GST_00215]	Rationale for BindingTime being optional in AttributeValueVariation-Point
[TPS_GST_00216]	Approach on Property Set Pattern
[TPS_GST_00217]	Transformation defined by Property Set Pattern
[TPS_GST_00218]	Property Set pattern and Inheritance
[TPS_GST_00219]	Binding Time for Property Set Pattern
[TPS_GST_00220]	Attachment of Binding Time
[TPS_GST_00221]	Attachment of Latest Binding Time
[TPS_GST_00222]	Multiplicity in Property Set Pattern
[TPS_GST_00223]	Use Cases for Sdg
[TPS_GST_00224]	Applicable modeling support in Special Data
[TPS_GST_00225]	Specifiation of roles in Special Data
[TPS_GST_00226]	Access Control Relation
[TPS_GST_00227]	AclPermission
[TPS_GST_00228]	AclRole
[TPS_GST_00229]	AclOperation
[TPS_GST_00230]	AclObjectSet
[TPS_GST_00231]	Context of AclPermission
[TPS_GST_00232]	aclObjectClass
[TPS_GST_00233]	collection
[TPS_GST_00234]	object
[TPS_GST_00235]	objectDefinition
[TPS_GST_00236]	derivedFromBlueprint
[TPS_GST_00237]	engineeringObject
[TPS_GST_00238]	Specifying replacement approach in LifeCycleInfo
[TPS_GST_00239]	Definition "Life Cycle"
[TPS_GST_00240]	LifeCycleStateDefinitionGroup
[TPS_GST_00241]	LifeCycleInfoSet
[TPS_GST_00242]	LifeCycleInfo
[TPS_GST_00243]	Informal references to traceable text
[TPS_GST_00244]	Viability Period of Life Cycle Info
[TPS_GST_00245]	PreBuild variation point
[TPS_GST_00246]	PostBuild Variation Point
[TPS_GST_00247]	BlueprintDerivation Variation Point
[TPS_GST_00248]	Combined PreBuild and PostBuild Variation Point

[TPS_GST_00249]	Variation Point without Conditions
[TPS_GST_00250]	Multiplicity of `VariationPoint`
[TPS_GST_00251]	Variant-Rich Model Violates [constr_2508]
[TPS_GST_00252]	Split/Merge of Variant-Rich Model
[TPS_GST_00253]	Distinguish `codeGenerationTime` Variation Points in RTE
[TPS_GST_00254]	Referring to Variation Points from Outside
[TPS_GST_00255]	Definition of *PreBuild* Variation Point
[TPS_GST_00256]	Definition of *PostBuild* Variation Point
[TPS_GST_00257]	BindingTime constrained by vh.latestBindingTime
[TPS_GST_00258]	Binding `VariationPoint`s early
[TPS_GST_00259]	Evaluating `PostBuildVariantCondition`
[TPS_GST_00260]	PreBuild configuration of PostBuild criteria
[TPS_GST_00261]	Possible Values for `PostBuildVariantCriterion`
[TPS_GST_00262]	Representation of `SwSystemconst`
[TPS_GST_00263]	Assigning values to `SwSystemconst`
[TPS_GST_00264]	Purpose of `SwSystemconstDependentFormula`
[TPS_GST_00265]	System Constants in Formula
[TPS_GST_00266]	PreBuild Disabling PostBuild support
[TPS_GST_00267]	Only one BindingTime
[TPS_GST_00268]	Rationale for Different Approach for PreBuild and PostBuild Variation
[TPS_GST_00269]	Reference from invariant to variant parts.
[TPS_GST_00270]	Variation Point in Blueprints
[TPS_GST_00271]	`blueprintCondition` cannot be variant
[TPS_GST_00272]	Semantics of BlueprintDerivationTime
[TPS_GST_00273]	Resolve BlueprintVariationPoints on time
[TPS_GST_00274]	`atp.StatusComment`
[TPS_GST_00275]	Float Literals `INF`, `NaN`
[TPS_GST_00276]	Power of Null
[TPS_GST_00277]	Purpose of Evaluated Variants
[TPS_GST_00278]	Establishing Multiple Validities with `EvaluatedVariantSet` for Different Aspects
[TPS_GST_00279]	Definition of a Predefined Variant
[TPS_GST_00280]	`SwSystemconstantValueSet`s from different sources
[TPS_GST_00281]	Indirect value assignment for system constants
[TPS_GST_00282]	Analogy between Predefined Variant for Pre Build and Post Build branch
[TPS_GST_00283]	Validity of Post Build combined with Pre Build Variant
[TPS_GST_00284]	Semantics of `approvalStatus`
[TPS_GST_00285]	Purpose of `includedVariant` in `PredefinedVariant`
[TPS_GST_00286]	`REJECTED` precedes `APPROVED`
[TPS_GST_00287]	`APPROVED` for `CollectableElement`
[TPS_GST_00288]	`REJECTED` for `CollectableElement`
[TPS_GST_00289]	Definition of a Variant
[TPS_GST_00290]	Defintion of Valid Variants
[TPS_GST_00291]	UML-tags for Configuration of XML schema production
[TPS_GST_00292]	Adminstrative UML Tags
[TPS_GST_00293]	Use Case Specific Extension of Formula Language
[TPS_GST_00294]	Build Action Manifest Overview
[TPS_GST_00295]	`atp.StatusRevisionBegin`
[TPS_GST_00296]	`atp.StatusRevisionEnd`
[TPS_GST_00297]	Tags to denote life cycle information
[TPS_GST_00298]	Tags to denote Variant Handling Properties
[TPS_GST_00299]	Tags to specify Upstream Mapping
[TPS_GST_00305]	Single Paragraph

[TPS_GST_00306]	Documentation Block
[TPS_GST_00307]	Standalone Documentation
[TPS_GST_00308]	Purpose of `Chapter`
[TPS_GST_00309]	Purpose of `Topic1`
[TPS_GST_00310]	Synopsis of Chapters and Topics
[TPS_GST_00311]	`DocumentationBlock` fits in a table cell
[TPS_GST_00312]	Variation in Documentation
[TPS_GST_00313]	Types of Paragraph
[TPS_GST_00314]	Purpose of Verbatim
[TPS_GST_00315]	Rendering of inline elements of Verbatim
[TPS_GST_00316]	Plain List
[TPS_GST_00317]	Labeled List
[TPS_GST_00318]	Definition List
[TPS_GST_00319]	Figures in Documentation
[TPS_GST_00320]	Details of Figures in Documentation
[TPS_GST_00321]	Mathematical Subjects in Documentation
[TPS_GST_00322]	Various Formula Representation
[TPS_GST_00323]	Purpose of `Note`
[TPS_GST_00324]	Inline Elements in Documentation
[TPS_GST_00325]	Standalone Documentation
[TPS_GST_00326]	Context of Standalone Documentation
[TPS_GST_00327]	Chapter
[TPS_GST_00328]	Predefined Chapter
[TPS_GST_00329]	Tables in Documentation
[TPS_GST_00330]	Partitions of a Table
[TPS_GST_00331]	Table Row
[TPS_GST_00332]	Topics in Documentation
[TPS_GST_00333]	Parameter Tables
[TPS_GST_00334]	Support of Pagination of Documents
[TPS_GST_00335]	View Approach
[TPS_GST_00336]	Including generated Documentation Parts
[TPS_GST_00337]	Multiple Languages
[TPS_GST_00338]	Purpose of `BuildActionEnvironment`
[TPS_GST_00339]	Data involved in Build Actions
[TPS_GST_00340]	Sequence of Build Actions
[TPS_GST_00341]	Input Data for Build Actions
[TPS_GST_00342]	ECUC-Parameters in Build Actions
[TPS_GST_00343]	ECUC-Containers in Build Actions
[TPS_GST_00344]	General Model Elements in Build Actions
[TPS_GST_00345]	Special Data in `BuildActionIoElement`
[TPS_GST_00346]	Automatic Collections
[TPS_GST_00347]	Expressing Relationships by collections
[TPS_GST_00348]	Standardized `category` of `Collection`
[TPS_GST_00349]	Standardized `elementRole` of `Collection`
[TPS_GST_00351]	Model Transformation on Assosications
[TPS_GST_00352]	Associations in Splitkeys
[TPS_GST_00353]	`mmt.templateTable`
[TPS_GST_00354]	Semantics of `CseCodeType`
[TPS_GST_00355]	Specialization of `FormulaExpression`
[TPS_GST_00356]	Application of `Sdg`
[TPS_GST_00357]	Usage of Special Data
[TPS_GST_00358]	TagWithOptionalValue
[TPS_GST_02501]	Compatibility of Numerical Values

Table B.9: Added Specification Items in 4.1.1

B.4.5 Changed Specification Items

Number	Heading
[TPS_GST_00007]	Shift operation
[TPS_GST_00008]	Types in Formula Expressions
[TPS_GST_00009]	Keyword 'epsilon'
[TPS_GST_00014]	Error handling in Formula Evaluator
[TPS_GST_00017]	{module} denotes a Module Designator
[TPS_GST_00020]	Establishing References
[TPS_GST_00023]	≪atpDerived≫ applicable to relations (associations, aggregations)

Table B.10: Changed Specification Items in 4.1.1

B.4.6 Deleted Specification Items

Number	Heading

Table B.11: Deleted Specification Items in 4.1.1

B.5 Constraint History R4.1.2

B.5.1 Added Traceables from 4.1.1 to 4.1.2

Id	Heading
[TPS_GST_00012]	AUTOSAR Formula language
[TPS_GST_00094]	Return values of the BluePrintFormular.ecuc query
[TPS_GST_00359]	Handling of the Sign

Table B.12: Added Traceables from 4.1.1 to 4.1.2

B.5.2 Changed Traceables from 4.1.1 to 4.1.2

Id	Heading
[TPS_GST_00008]	Types in Formula Expressions
[TPS_GST_00017]	{module} denotes a Module Designator
[TPS_GST_00047]	Identification of Partial Models
[TPS_GST_00052]	vh.latestBindingTime
[TPS_GST_00182]	Notation of Latest Binding time on M2
[TPS_GST_00183]	Representation of Binding Time
[TPS_GST_00202]	Limitation of non post build
[TPS_GST_00209]	No postbuild variation for attribute values
[TPS_GST_00221]	Attachment of Latest Binding Time
[TPS_GST_00257]	BindingTime constrained by vh.latestBindingTime
[TPS_GST_00272]	Semantics of BlueprintDerivationTime
[TPS_GST_00298]	Tags to denote Variant Handling Properties

Table B.13: Changed Traceables from 4.1.1 to 4.1.2

B.5.3 Deleted Traceables from 4.1.1 to 4.1.2

none

B.5.4 Added Constraints from 4.1.1 to 4.1.2

none

B.5.5 Changed Constraints from 4.1.1 to 4.1.2

Id	Heading
[constr_2504]	Constraint to `bindingTime`
[constr_2514]	`shortLabel` in `VariationPoint` must be unique
[constr_2517]	`postBuildVariantCondition` only for PostBuild
[constr_2518]	Binding time is constrained
[constr_2557]	No `VariationPoint`s where `vh.latestBindingTime` set to `BlueprintDerivationTime` in system configurations
[constr_2558]	If `vh.latestBindingTime` is `BlueprintDerivationTime` then there shall only be `blueprintCondition`/`blueprintValue`
[constr_2577]	Binding Time in Aggregation Pattern
[constr_2578]	Binding Time in Association Pattern
[constr_2579]	Binding Time in Attribute Value Pattern
[constr_2580]	Binding Time in Property Set Pattern

Table B.14: Changed Constraints from 4.1.1 to 4.1.2

B.5.6 Deleted Constraints from 4.1.1 to 4.1.2

Id	Heading
[constr_2513]	splitted variants must have a `shortLabel`

Table B.15: Deleted Constraints from 4.1.1 to 4.1.2

B.6 Constraint History R4.1.3

B.6.1 Added Traceables in 4.1.3

Id	Heading
[TPS_GST_00208]	Representation of return type in float

Table B.16: Added Traceables in 4.1.3

B.6.2 Changed Traceables in 4.1.3

Id	Heading
[TPS_GST_00003]	true and false
[TPS_GST_00051]	`atp.Status`
[TPS_GST_00063]	`xml.nsUri`
[TPS_GST_00066]	`xml.systemIdentifier`
[TPS_GST_00094]	Return values of the `BlueprintFormula`.ecuc query
[TPS_GST_00209]	No postbuild variation for attribute values
[TPS_GST_00211]	`AttributeValueVariationPoint` does not support PostBuild Variation
[TPS_GST_00253]	Distinguish `codeGenerationTime` Variation Points in RTE
[TPS_GST_00255]	Definition of *PreBuild* Variation Point
[TPS_GST_00258]	Binding `VariationPoint`s early
[TPS_GST_00260]	PreBuild configuration of PostBuild criteria
[TPS_GST_00270]	Variation Point in Blueprints
[TPS_GST_00271]	`blueprintCondition` cannot be variant
[TPS_GST_00273]	Resolve BlueprintVariationPoints on time
[TPS_GST_00354]	Semantics of `CseCodeType`

Table B.17: Changed Traceables in 4.1.3

B.6.3 Deleted Traceables in 4.1.3

none

B.6.4 Added Constraints in 4.1.3

none

B.6.5 Changed Constraints in 4.1.3

Id	Heading
[constr_2502]	Merged model must be compliant to the meta-model.

Table B.18: Changed Constraints in 4.1.3

B.6.6 Deleted Constraints in 4.1.3

none

B.7 Constraint History R4.2.1

B.7.1 Added Traceables in 4.2.1

Id	Heading

[TPS_GST_00360]	Definition of *PreBuild* Variation Point with Blueprint conditions
[TPS_GST_00361]	Propagation of `LifeCycleState`
[TPS_GST_00362]	`map.Status`
[TPS_GST_00363]	map.Id

Table B.19: Added Traceables in 4.2.1

B.7.2 Changed Traceables in 4.2.1

Id	Heading
[TPS_GST_00094]	Return values of the `BlueprintFormula`.ecuc query
[TPS_GST_00182]	Notation of Latest Binding time on M2
[TPS_GST_00206]	Special Meta Classes for AttributeValueVariationPoint
[TPS_GST_00257]	BindingTime constrained by vh.latestBindingTime
[TPS_GST_00259]	Evaluating `PostBuildVariantCondition`
[TPS_GST_00297]	Tags to denote life cycle information
[TPS_GST_00322]	Various Formula Representation
[TPS_GST_00354]	Semantics of `CseCodeType`

Table B.20: Changed Traceables in 4.2.1

B.7.3 Deleted Traceables in 4.2.1

none

B.7.4 Added Constraints in 4.2.1

none

B.7.5 Changed Constraints in 4.2.1

Id	Heading
[constr_2502]	Merged model shall be compliant to the meta-model
[constr_2575]	`blueprintValue` in blueprints only
[constr_2578]	Binding Time in Association Pattern

Table B.21: Changed Constraints in 4.2.1

B.7.6 Deleted Constraints in 4.2.1

none

B.8 Constraint History R4.2.2

B.8.1 Added Traceables in 4.2.2

Id	Heading
[TPS_GST_00364]	UML tags are attached to the target end of relations if suitable
[TPS_GST_00365]	Purpose of `uuid`
[TPS_GST_00366]	No View Specified
[TPS_GST_00367]	Dedicated View Specified
[TPS_GST_00368]	Multiple Views Specified
[TPS_GST_00369]	Exclude content

Table B.22: Added Traceables in 4.2.2

B.8.2 Changed Traceables in 4.2.2

Id	Heading
[TPS_GST_00331]	Table Row
[TPS_GST_00335]	View Approach
[TPS_GST_00336]	Including generated Documentation Parts

Table B.23: Changed Traceables in 4.2.2

B.8.3 Deleted Traceables in 4.2.2

none

B.8.4 Added Constraints in 4.2.2

Id	Heading
[constr_2594]	Cyclic value assignments to `SwSystemconst` is not allowed
[constr_2595]	Footnotes should not be nested
[constr_2596]	Used colors of attributes color and bgcolor

Table B.24: Added Constraints in 4.2.2

B.8.5 Changed Constraints in 4.2.2

Id	Heading
[constr_2505]	Multiplicity after binding
[constr_2577]	Binding Time in Aggregation Pattern

Table B.25: Changed Constraints in 4.2.2

B.8.6 Deleted Constraints in 4.2.2

none

B.9 Constraint History R4.3.0

B.9.1 Added Traceables in 4.3.0

Id	Heading
[TPS_GST_00370]	`atp.EnumerationValue`
[TPS_GST_00371]	Tag to control the production of specification documents
[TPS_GST_00372]	`mmt.RestrictToStandards`
[TPS_GST_00373]	Default `EnumerationMappingTable`
[TPS_GST_00374]	Purpose of `SdgDef`
[TPS_GST_00375]	Purpose of `SdgClass`
[TPS_GST_00376]	Purpose of Model Restriction Types
[TPS_GST_00377]	Purpose of `AbstractValueRestriction`
[TPS_GST_00378]	Purpose of `AbstractMultiplicityRestriction`
[TPS_GST_00379]	Purpose of `AbstractVariationRestriction`
[TPS_GST_00380]	Countably infinite number of elements

Table B.26: Added Traceables in 4.3.0

B.9.2 Changed Traceables in 4.3.0

Id	Heading
[TPS_GST_00001]	Connection between Formula and Model Elements
[TPS_GST_00002]	aborting logical expressions
[TPS_GST_00012]	AUTOSAR Formula language
[TPS_GST_00013]	Function `defined`
[TPS_GST_00020]	Establishing References
[TPS_GST_00023]	≪`atpDerived`≫ applicable to relations (associations, aggregations)
[TPS_GST_00046]	Splitable collections
[TPS_GST_00049]	`atp.recommendedPackage`
[TPS_GST_00050]	`atp.Splitkey`
[TPS_GST_00051]	`atp.Status`
[TPS_GST_00052]	`vh.latestBindingTime`
[TPS_GST_00053]	`xml.xsd.*` etc.
[TPS_GST_00054]	`xml.xsd.customType`
[TPS_GST_00055]	`xml.attribute`
[TPS_GST_00056]	`xml.attributeRef`
[TPS_GST_00057]	`xml.enforceMinMultiplicity`
[TPS_GST_00058]	`xml.enforceMaxMultiplicity`
[TPS_GST_00059]	`xml.globalElement`
[TPS_GST_00060]	`xml.mds.type`
[TPS_GST_00061]	`xml.name`
[TPS_GST_00062]	`xml.nsPrefix`
[TPS_GST_00063]	`xml.nsUri`
[TPS_GST_00064]	`xml.roleElement`, `xml.roleWrapperElement`, `xml.typeElement`, `xml.typeWrapperElement`
[TPS_GST_00065]	`xml.sequenceOffset`
[TPS_GST_00066]	`xml.systemIdentifier`
[TPS_GST_00067]	`admin.documentClassification`
[TPS_GST_00068]	`admin.documentIdentificationNo`
[TPS_GST_00069]	`admin.documentOwner`
[TPS_GST_00070]	`admin.documentResponsibility`

[TPS_GST_00071]	admin.documentStatus
[TPS_GST_00072]	admin.documentTitle
[TPS_GST_00073]	admin.documentVersion
[TPS_GST_00074]	admin.partOfRelease
[TPS_GST_00075]	admin.releaseDate
[TPS_GST_00076]	admin.revision
[TPS_GST_00087]	BLUEPRINT
[TPS_GST_00088]	STANDARD
[TPS_GST_00089]	EXAMPLE
[TPS_GST_00094]	Return values of the BlueprintFormula.ecuc query
[TPS_GST_00169]	Absolute shortName-Path
[TPS_GST_00170]	Relative shortName-path
[TPS_GST_00171]	Identifying the ReferenceBase of a Relative Reference
[TPS_GST_00196]	ICS
[TPS_GST_00201]	Aggregation Pattern on Primitives
[TPS_GST_00202]	Limitation of non post build
[TPS_GST_00205]	Transformation defined by Attribute Value Pattern
[TPS_GST_00206]	Special Meta Classes for AttributeValueVariationPoint
[TPS_GST_00229]	AclOperation
[TPS_GST_00231]	Context of AclPermission
[TPS_GST_00242]	LifeCycleInfo
[TPS_GST_00251]	Variant Rich Model Violates [constr_2508]
[TPS_GST_00291]	UML-tags for Configuration of XML schema production
[TPS_GST_00295]	atp.StatusRevisionBegin
[TPS_GST_00296]	atp.StatusRevisionEnd
[TPS_GST_00313]	Types of Paragraph
[TPS_GST_00318]	Definition List
[TPS_GST_00327]	Chapter
[TPS_GST_00353]	mmt.templateTable
[TPS_GST_00354]	Semantics of CseCodeType
[TPS_GST_00364]	UML tags are attached to the target end of relations if suitable
[TPS_GST_02501]	Compatibility of Numerical Values

Table B.27: Changed Traceables in 4.3.0

B.9.3 Deleted Traceables in 4.3.0

none

B.9.4 Added Constraints in 4.3.0

Id	Heading
[constr_2599]	Maximum one VariationPoints in ≪atpMixed≫
[constr_2601]	Value of AbstractEnumerationValueVariationPoint
[constr_2602]	Completeness of AnyInstanceRef referencing ImplementationDataType-Element
[constr_2605]	If a SdgClass is referenced then it shall have a caption
[constr_2607]	Existence of upperMultiplicityInfinite and upperMultiplicity of AbstractMultiplicityRestriction is mutually exclusive
[constr_2608]	lowerMultiplicity of AbstractMultiplicityRestriction shall be smaller or equal to upperMultiplicity

Table B.28: Added Constraints in 4.3.0

B.9.5 Changed Constraints in 4.3.0

Id	Heading
[constr_2503]	Bound model must be compliant to the pure meta model
[constr_2505]	Multiplicity after binding
[constr_2511]	Named reference bases shall be available
[constr_2574]	`globalInPackage` for global elements only
[constr_2577]	Binding Time in Aggregation Pattern
[constr_2578]	Binding Time in Association Pattern
[constr_2579]	Binding Time in Attribute Value Pattern
[constr_2580]	Binding Time in Property Set Pattern
[constr_2587]	No `System` in `AnyInstanceRef`
[constr_2595]	Footnotes should not be nested

Table B.29: Changed Constraints in 4.3.0

B.9.6 Deleted Constraints in 4.3.0

none

B.10 Constraint History R4.3.1

B.10.1 Added Traceables in 4.3.1

Number	Heading
[TPS_GST_00381]	≪`atpStructuredComment`≫
[TPS_GST_00382]	Interaction of ≪`atpStructuredComment`≫ and ≪`atpSplitable`≫
[TPS_GST_00383]	Ordered collections
[TPS_GST_00384]	Naming conventions in variant handling patterns

Table B.30: Added Traceables in 4.3.1

B.10.2 Changed Traceables in 4.3.1

Number	Heading
[TPS_GST_00045]	Inherited properties in mixed content
[TPS_GST_00086]	Category of `ARPackage`

▽

△

Number	Heading
[TPS_GST_00151]	Specializations of Derived Relations

Table B.31: Changed Traceables in 4.3.1

B.10.3 Deleted Traceables in 4.3.1

none

B.10.4 Added Constraints in 4.3.1

Number	Heading
[constr_2606]	Existence of `upperMultiplicityInfinite` and `upperMultiplicity` of `AbstractMultiplicityRestriction` is mutually exclusive

Table B.32: Added Constraints in 4.3.1

B.10.5 Changed Constraints in 4.3.1

Number	Heading
[constr_2515]	Categories of packages shall not conflict
[constr_2525]	Non splitable elements shall not be repeated
[constr_2547]	Ordered collections cannot be split into partial models

Table B.33: Changed Constraints in 4.3.1

B.10.6 Deleted Constraints in 4.3.1

Number	Heading
[constr_2608]	`lowerMultiplicity` of `AbstractMultiplicityRestriction` shall be smaller or equal to `upperMultiplicity`

Table B.34: Deleted Constraints in 4.3.1

B.11 Constraint History R4.4.0

B.11.1 Added Traceables in 4.4.0

Number	Heading
[TPS_GST_00184]	Grammar of ARMQL
[TPS_GST_00385]	`atp.ManifestKind`
[TPS_GST_00386]	`atpContextElement`s of `InstanceRefs` shall be consistent
[TPS_GST_00387]	`AtpInstanceRef` shall be close to the base
[TPS_GST_00388]	Types and Values in ARMQL
[TPS_GST_00389]	Type Coercion Rules
[TPS_GST_00390]	Operators and Expressions
[TPS_GST_00391]	Lambda Abstraction, Function and Pseudo-Method Invocation
[TPS_GST_00392]	The Integer Range Expression
[TPS_GST_00393]	Assigning Names to Values
[TPS_GST_00394]	Definition of functions
[TPS_GST_00395]	Single Line FOR Blocks
[TPS_GST_00396]	Multi Line FOR Blocks
[TPS_GST_00397]	Semantics of WHERE Blocks
[TPS_GST_00398]	Type predicates
[TPS_GST_00399]	Checking if an expression is defined
[TPS_GST_00400]	General Functions
[TPS_GST_00401]	String predicates
[TPS_GST_00402]	List processing functions
[TPS_GST_00403]	Type coercion functions
[TPS_GST_00404]	Acccess to ECUC Values
[TPS_GST_00405]	DefinitionRefPath
[TPS_GST_00406]	Acccess to Model Values
[TPS_GST_00407]	Non-Strict evaluation
[TPS_GST_00408]	General scoping rules
[TPS_GST_00409]	Hierarchy levels within one EXPRESSION tag
[TPS_GST_00410]	Hierarchy levels between EXPRESSION tags
[TPS_GST_00411]	Multiplicity of Derived Elements
[TPS_GST_00412]	Variable Interpolation
[TPS_GST_00413]	Default value of `atp.Status`
[TPS_GST_00414]	Modeling of splitable elements

Table B.35: Added Traceables in 4.4.0

B.11.2 Changed Traceables in 4.4.0

Number	Heading
[TPS_GST_00047]	Identification of Partial Models
[TPS_GST_00051]	`atp.Status`

Table B.36: Changed Traceables in 4.4.0

B.11.3 Deleted Traceables in 4.4.0

Number	Heading
[TPS_GST_00094]	Return values of the `BlueprintFormula`.ecuc query

Table B.37: Deleted Traceables in 4.4.0

B.11.4 Added Constraints in 4.4.0

Number	Heading
[constr_2626]	`atpTarget` of InstanceRefs shall be consistent
[constr_2627]	No reassigning of the same name within one LET Block

Table B.38: Added Constraints in 4.4.0

B.11.5 Changed Constraints in 4.4.0

Number	Heading
[constr_2521]	The `shortLabel` in `AttributeValueVariationPoint` shall be unique
[constr_2587]	No `System` in `AnyInstanceRef`

Table B.39: Changed Constraints in 4.4.0

B.11.6 Deleted Constraints in 4.4.0

Number	Heading
[constr_2530]	`InstanceRefs` must be consistent
[constr_2531]	`AtpInstanceRef` shall be close to the base

Table B.40: Deleted Constraints in 4.4.0

B.12 Constraint History R19-11

B.12.1 Added Traceables in 19-11

Number	Heading
[TPS_GST_00415]	Splitkey
[TPS_GST_00416]	Rules for the definition of a splitkey at M2 level
[TPS_GST_00417]	Merging of splitable elements from partial models
[TPS_GST_00418]	Definition Splitkey Path
[TPS_GST_00419]	Synopsis of `TopicContent`
[TPS_GST_00420]	Matching identities up to the root
[TPS_GST_00421]	Categories of `SdgPrimitiveAttribute` and `SdgPrimitiveAttributeWithVariation`
[TPS_GST_00422]	Specification of the value of `SdgClass.extendsMetaClass` and `SdgAbstractForeignReference.destMetaClass`

Table B.41: Added Traceables in 19-11

B.12.2 Changed Traceables in 19-11

Number	Heading
[TPS_GST_00050]	`atp.Splitkey`
[TPS_GST_00205]	Transformation defined by Attribute Value Pattern
[TPS_GST_00206]	Special Meta Classes for `AttributeValueVariationPoint`
[TPS_GST_00217]	Transformation defined by Property Set Pattern
[TPS_GST_00247]	BlueprintDerivation Variation Point
[TPS_GST_00257]	BindingTime constrained by vh.latestBindingTime
[TPS_GST_00314]	Purpose of Verbatim
[TPS_GST_00360]	Definition of *PreBuild* Variation Point with Blueprint conditions
[TPS_GST_00370]	`atp.EnumerationLiteralIndex`
[TPS_GST_00385]	`atp.ManifestKind`

Table B.42: Changed Traceables in 19-11

B.12.3 Deleted Traceables in 19-11

none

B.12.4 Added Constraints in 19-11

Number	Heading
[constr_2628]	Representation of xml.xsd.type=double data types
[constr_2629]	Defined identity up to the root
[constr_2630]	M1 elements with same identity but different type are not allowed

Table B.43: Added Constraints in 19-11

B.12.5 Changed Constraints in 19-11

Number	Heading
[constr_2521]	The `shortLabel` in `AttributeValueVariationPoint` shall be unique
[constr_2558]	If `vh.latestBindingTime` is `BlueprintDerivationTime` then there shall only be `blueprintCondition` or `formalBlueprintGenerator` respectively `blueprintValue`

Table B.44: Changed Constraints in 19-11

B.12.6 Deleted Constraints in 19-11

none

B.13 Constraint History R20-11

B.13.1 Added Traceables in R20-11

Number	Heading
[TPS_GST_00423]	AUTOSAR Partial Model
[TPS_GST_00424]	Variant aggregation of Abstract and Concrete subclasses
[TPS_GST_00425]	Set `SdgClass.caption` = true if the `Sdg` on the value side shall be referrable
[TPS_GST_00426]	≪`instanceRef`≫ in combination with ≪`atpUriDef`≫
[TPS_GST_00427]	≪`atpIdentityContributor`≫ applicable to aggregations, associations and primitive attributes

Table B.45: Added Traceables in R20-11

B.13.2 Changed Traceables in R20-11

Number	Heading
[TPS_GST_00047]	Identification of M1 elements in partial models
[TPS_GST_00074]	admin.partOfRelease
[TPS_GST_00076]	admin.revision
[TPS_GST_00095]	Main Purpose of Identifiable
[TPS_GST_00206]	Special meta-classes for AttributeValueVariationPoint
[TPS_GST_00211]	AttributeValueVariationPoint does not support PostBuild Variation
[TPS_GST_00308]	Purpose of Chapter
[TPS_GST_00320]	Details of Figures in Documentation
[TPS_GST_00416]	Rules for the definition of a splitkey at M2 level

Table B.46: Changed Traceables in R20-11

B.13.3 Deleted Traceables in R20-11

Number	Heading
[TPS_GST_00098]	Recommendation to Choose Human Readable shortNames
[TPS_GST_00209]	No postbuild variation for attribute values
[TPS_GST_00352]	Associations in Splitkeys
[TPS_GST_00385]	atp.ManifestKind

Table B.47: Deleted Traceables in R20-11

B.13.4 Added Constraints in R20-11

Number	Heading
[constr_2631]	Usage of value ANY for AnyServiceInstanceId
[constr_2632]	No postbuild variation for attribute values

Table B.48: Added Constraints in R20-11

B.13.5 Changed Constraints in R20-11

Number	Heading
[constr_2509]	Uniqueness of ReferenceBase.shortLabel in the scope of an ARPackage

Table B.49: Changed Constraints in R20-11

B.13.6 Deleted Constraints in R20-11

Number	Heading
[constr_2605]	If a SdgClass is referenced then it shall have a caption

Table B.50: Deleted Constraints in R20-11

B.14 Constraint History R21-11

B.14.1 Added Traceables in R21-11

Number	Heading
[TPS_GST_00428]	Standards restriction on document level
[TPS_GST_00429]	Application of a standards restriction for meta-model diagrams
[TPS_GST_00430]	Application of a standards restriction for meta-model description fields
[TPS_GST_00431]	Implications of a standards restriction on modelling level
[TPS_GST_00432]	Semantics of a `Collection`
[TPS_GST_00433]	Individual attributes in conditionals
[TPS_GST_00434]	Expressing a set of elements by collections

Table B.51: Added Traceables in R21-11

B.14.2 Changed Traceables in R21-11

Number	Heading
[TPS_GST_00051]	`atp.Status`
[TPS_GST_00093]	Collections
[TPS_GST_00167]	Case Sensitivity of References
[TPS_GST_00346]	Automatic Collections
[TPS_GST_00347]	Expressing Relationships by collections
[TPS_GST_00348]	Standardized `category` of `Collection`
[TPS_GST_00349]	Standardized `elementRole` of `Collection`
[TPS_GST_00416]	Rules for the definition of a splitkey at M2 level

Table B.52: Changed Traceables in R21-11

B.14.3 Deleted Traceables in R21-11

none

B.14.4 Added Constraints in R21-11

Number	Heading
[constr_2633]	Existence of reference decorated with stereotype ≪isOfType≫
[constr_2634]	Conditionals with ordered collections
[constr_2635]	No custom values for Collection.category
[constr_2636]	No custom values for Collection.elementRole

Table B.53: Added Constraints in R21-11

B.14.5 Changed Constraints in R21-11

Number	Heading
[constr_2508]	The shortName shall be unique in its name space
[constr_2626]	atpTarget of InstanceRefs shall be consistent

Table B.54: Changed Constraints in R21-11

B.14.6 Deleted Constraints in R21-11

Number	Heading
[constr_2506]	Attributes in property set pattern

Table B.55: Deleted Constraints in R21-11

C All Variation Points in meta-model

Variation Point	*Latest Binding Time*
AbstractCanCluster	postBuild
AbstractCanCommunicationController	postBuild
AbstractServiceInstance.capabilityRecord	postBuild
AbstractServiceInstance.methodActivationRoutingGroup	postBuild
AccessCount.value	preCompileTime
AccessCountSet.accessCount	preCompileTime
AliasNameSet.aliasName	preCompileTime
ApplicationArrayElement.maxNumberOfElements	preCompileTime
ApplicationRecordDataType.element	preCompileTime
ARPackage.arPackage	blueprintDerivationTime
ARPackage.element	systemDesignTime
ArrayValueSpecification.element	preCompileTime
AtomicSwComponentType.internalBehavior	preCompileTime
AUTOSAR.arPackage	blueprintDerivationTime
BlueprintPolicyList.maxNumberOfElements	blueprintDerivationTime
BlueprintPolicyList.minNumberOfElements	blueprintDerivationTime
BswInternalBehavior.arTypedPerInstanceMemory	preCompileTime
BswInternalBehavior.bswPerInstanceMemoryPolicy	preCompileTime
BswInternalBehavior.clientPolicy	preCompileTime
BswInternalBehavior.distinguishedPartition	preCompileTime
BswInternalBehavior.entity	preCompileTime
BswInternalBehavior.event	preCompileTime
BswInternalBehavior.exclusiveAreaPolicy	preCompileTime
BswInternalBehavior.internalTriggeringPoint	preCompileTime
BswInternalBehavior.internalTriggeringPointPolicy	preCompileTime
BswInternalBehavior.modeReceiverPolicy	preCompileTime
BswInternalBehavior.modeSenderPolicy	preCompileTime
BswInternalBehavior.parameterPolicy	preCompileTime
BswInternalBehavior.perInstanceParameter	preCompileTime
BswInternalBehavior.receptionPolicy	preCompileTime
BswInternalBehavior.releasedTriggerPolicy	preCompileTime
BswInternalBehavior.schedulerNamePrefix	preCompileTime
BswInternalBehavior.sendPolicy	preCompileTime
BswInternalBehavior.serviceDependency	preCompileTime
BswInternalBehavior.triggerDirectImplementation	preCompileTime
BswModuleDependency.targetModuleRef	preCompileTime
BswModuleDescription.bswModuleDependency	preCompileTime
BswModuleDescription.bswModuleDocumentation	preCompileTime
BswModuleDescription.expectedEntry	preCompileTime
BswModuleDescription.implementedEntry	preCompileTime
BswModuleDescription.providedClientServerEntry	preCompileTime
BswModuleDescription.providedData	preCompileTime

▽

△

Variation Point	*Latest Binding Time*
BswModuleDescription.providedModeGroup	preCompileTime
BswModuleDescription.releasedTrigger	preCompileTime
BswModuleDescription.requiredClientServerEntry	preCompileTime
BswModuleDescription.requiredData	preCompileTime
BswModuleDescription.requiredModeGroup	preCompileTime
BswModuleDescription.requiredTrigger	preCompileTime
BswModuleEntity.accessedModeGroup	preCompileTime
BswModuleEntity.activationPoint	preCompileTime
BswModuleEntity.callPoint	preCompileTime
BswModuleEntity.dataReceivePoint	preCompileTime
BswModuleEntity.dataSendPoint	preCompileTime
BswModuleEntity.issuedTrigger	preCompileTime
BswModuleEntity.managedModeGroup	preCompileTime
BswModuleEntry.argument	blueprintDerivationTime
BswServiceDependency.assignedData	preCompileTime
BswServiceDependency.assignedEntryRole	preCompileTime
BuildActionManifest.buildAction	blueprintDerivationTime
BuildActionManifest.buildActionEnvironment	blueprintDerivationTime
BulkNvDataDescriptor.nvBlockDataMapping	preCompileTime
BusMirrorChannel.channel	systemDesignTime
BusMirrorChannelMapping.targetPduTriggering	postBuild
CalibrationParameterValueSet.calibrationParameterValue	preCompileTime
CanCluster	postBuild
CanCommunicationController	postBuild
CanTpConfig.tpAddress	postBuild
CanTpConfig.tpChannel	postBuild
CanTpConfig.tpConnection	postBuild
CanTpConfig.tpEcu	postBuild
CanTpConfig.tpNode	postBuild
ChapterOrMsrQuery.chapter	postBuild
ClientIdDefinitionSet.clientIdDefinition	postBuild
ClientIdRange.lowerLimit	preCompileTime
ClientIdRange.upperLimit	preCompileTime
ClientServerInterface.operation	blueprintDerivationTime
ClientServerInterfaceToBswModuleEntryBlueprintMapping.operationMapping	preCompileTime
ClientServerInterfaceToBswModuleEntryBlueprintMapping.portDefinedArgument Blueprint	preCompileTime
ClientServerOperation.argument	blueprintDerivationTime
CommunicationCluster	postBuild
CommunicationCluster.physicalChannel	systemDesignTime
CommunicationConnector.ecuCommPortInstance	postBuild
CommunicationController	postBuild
CompositionSwComponentType.component	postBuild
CompositionSwComponentType.connector	postBuild
CompositionSwComponentType.instantiationRTEEventProps	codeGenerationTime

▽

△

Variation Point	*Latest Binding Time*
CompuConstFormulaContent.vf	codeGenerationTime
CompuNominatorDenominator.v	preCompileTime
CompuScale.lowerLimit	preCompileTime
CompuScale.upperLimit	preCompileTime
CompuScales.compuScale	blueprintDerivationTime
ConsistencyNeeds.dpgDoesNotRequireCoherency	preCompileTime
ConsistencyNeeds.dpgRequiresCoherency	preCompileTime
ConsistencyNeeds.regDoesNotRequireStability	preCompileTime
ConsistencyNeeds.regRequiresStability	preCompileTime
ConsistencyNeedsBlueprintSet.consistencyNeeds	preCompileTime
ConsumedEventGroup.eventMulticastAddress	postBuild
ConsumedEventGroup.sdClientTimerConfig	postBuild
ConsumedProvidedServiceInstanceGroup.consumedServiceInstance	postBuild
ConsumedProvidedServiceInstanceGroup.providedServiceInstance	postBuild
ConsumedServiceInstance.consumedEventGroup	postBuild
ConsumedServiceInstance.eventMulticastSubscriptionAddress	postBuild
ConsumedServiceInstance.localUnicastAddress	postBuild
ConsumedServiceInstance.remoteUnicastAddress	postBuild
ConsumedServiceInstance.sdClientTimerConfig	postBuild
CouplingElement.couplingPort	postBuild
CouplingPortConnection.nodePort	postBuild
CpSoftwareCluster.swComponentAssignment	postBuild
CpSoftwareCluster.swComposition	systemDesignTime
CpSoftwareClusterMappingSet.portElementToComResourceMapping	postBuild
CpSoftwareClusterMappingSet.resourceToApplicationPartitionMapping	systemDesignTime
CpSoftwareClusterMappingSet.softwareClusterToResourceMapping	preCompileTime
CpSoftwareClusterMappingSet.swcToApplicationPartitionMapping	postBuild
DataPrototypeGroup.dataPrototypeGroup	preCompileTime
DataPrototypeGroup.implicitDataAccess	preCompileTime
DataTransformationSet.dataTransformation	codeGenerationTime
DataTransformationSet.transformationTechnology	codeGenerationTime
DefList.defItem	postBuild
DiagEventDebounceCounterBased.counterDecrementStepSize	preCompileTime
DiagEventDebounceCounterBased.counterFailedThreshold	preCompileTime
DiagEventDebounceCounterBased.counterIncrementStepSize	preCompileTime
DiagEventDebounceCounterBased.counterJumpDown	preCompileTime
DiagEventDebounceCounterBased.counterJumpDownValue	preCompileTime
DiagEventDebounceCounterBased.counterJumpUp	preCompileTime
DiagEventDebounceCounterBased.counterJumpUpValue	preCompileTime
DiagEventDebounceCounterBased.counterPassedThreshold	preCompileTime
DiagEventDebounceTimeBased.timeBasedFdcThresholdStorageValue	preCompileTime
DiagEventDebounceTimeBased.timeFailedThreshold	preCompileTime
DiagEventDebounceTimeBased.timePassedThreshold	preCompileTime
DiagnosticAbstractDataIdentifier.id	preCompileTime
DiagnosticAging.agingCycle	preCompileTime

▽

△

Variation Point	*Latest Binding Time*
DiagnosticAging.threshold	preCompileTime
DiagnosticCommonProps	codeGenerationTime
DiagnosticConnectedIndicator.healingCycleCounterThreshold	preCompileTime
DiagnosticContributionSet.element	postBuild
DiagnosticContributionSet.serviceTable	postBuild
DiagnosticDataIdentifier.dataElement	postBuild
DiagnosticDebounceAlgorithmProps.debounceBehavior	preCompileTime
DiagnosticEnableConditionGroup.enableCondition	postBuild
DiagnosticEvent.confirmationThreshold	preCompileTime
DiagnosticEvent.connectedIndicator	postBuild
DiagnosticFreezeFrame.recordNumber	preCompileTime
DiagnosticIndicator.type	preCompileTime
DiagnosticIumprGroup.iumprGroupIdentifier	postBuild
DiagnosticMeasurementIdentifier.obdMid	preCompileTime
DiagnosticParameter.dataElement	postBuild
DiagnosticParameterIdentifier.dataElement	postBuild
DiagnosticProtocol.diagnosticConnection	postBuild
DiagnosticProtocol.priority	preCompileTime
DiagnosticProtocol.sendRespPendOnTransToBoot	preCompileTime
DiagnosticProtocol.serviceTable	postBuild
DiagnosticRoutine.id	preCompileTime
DiagnosticServiceTable.diagnosticConnection	postBuild
DiagnosticStorageConditionGroup.storageCondition	postBuild
DiagnosticTestIdentifier.id	preCompileTime
DiagnosticTestIdentifier.uasId	preCompileTime
DiagnosticTestResult.diagnosticEvent	preCompileTime
DiagnosticTestResult.updateKind	preCompileTime
DiagnosticTroubleCodeGroup.dtc	postBuild
DiagnosticTroubleCodeGroup.groupNumber	preCompileTime
DiagnosticTroubleCodeObd.considerPtoStatus	preCompileTime
DiagnosticTroubleCodeObd.eventReadinessGroup	postBuild
DiagnosticTroubleCodeObd.obdDTCValue	preCompileTime
DiagnosticTroubleCodeObd.obdDTCValue3Byte	preCompileTime
DiagnosticTroubleCodeProps.extendedDataRecord	preCompileTime
DiagnosticTroubleCodeProps.freezeFrame	preCompileTime
DiagnosticTroubleCodeProps.legislatedFreezeFrameContentUdsObd	preCompileTime
DiagnosticTroubleCodeProps.priority	preCompileTime
DiagnosticTroubleCodeProps.snapshotRecordContent	preCompileTime
DiagnosticTroubleCodeUds.udsDtcValue	preCompileTime
DiagnosticTroubleCodeUds.wwhObdDtcClass	preCompileTime
DltApplication.context	systemDesignTime
DltContext.dltMessage	systemDesignTime
DltEcu.application	systemDesignTime
DocumentationBlock.defList	postBuild

▽

△

Variation Point	*Latest Binding Time*
DocumentationBlock.figure	postBuild
DocumentationBlock.formula	postBuild
DocumentationBlock.labeledList	postBuild
DocumentationBlock.list	postBuild
DocumentationBlock.note	postBuild
DocumentationBlock.p	postBuild
DocumentationBlock.structuredReq	postBuild
DocumentationBlock.trace	postBuild
DocumentationBlock.verbatim	postBuild
EcucAbstractStringParamDef	codeGenerationTime
EcucBooleanParamDef.defaultValue	codeGenerationTime
EcucContainerValue.parameterValue	postBuild
EcucContainerValue.referenceValue	postBuild
EcucContainerValue.subContainer	postBuild
EcucDefinitionElement.lowerMultiplicity	codeGenerationTime
EcucDefinitionElement.upperMultiplicity	codeGenerationTime
EcucDefinitionElement.upperMultiplicityInfinite	codeGenerationTime
EcucFloatParamDef.defaultValue	codeGenerationTime
EcucFloatParamDef.max	codeGenerationTime
EcucFloatParamDef.min	codeGenerationTime
EcucFunctionNameDef	codeGenerationTime
EcucIntegerParamDef.defaultValue	codeGenerationTime
EcucIntegerParamDef.max	codeGenerationTime
EcucIntegerParamDef.min	codeGenerationTime
EcucLinkerSymbolDef	codeGenerationTime
EcucModuleConfigurationValues.container	postBuild
EcucMultilineStringParamDef	codeGenerationTime
EcucNumericalParamValue.value	preCompileTime
EcucStringParamDef	codeGenerationTime
EcucValueCollection.ecucValue	preCompileTime
EcuInstance.associatedConsumedProvidedServiceInstanceGroup	postBuild
EcuInstance.commController	postBuild
EcuInstance.connector	postBuild
EndToEndProtection.endToEndProtectionISignalIPdu	preCompileTime
EndToEndProtection.endToEndProtectionVariablePrototype	preCompileTime
EndToEndProtectionSet.endToEndProtection	preCompileTime
EndToEndTransformationISignalProps	postBuild
ErrorTracerNeeds.tracedFailure	preCompileTime
EthernetCluster	postBuild
EthernetCluster.couplingPortConnection	postBuild
EthernetCommunicationController	postBuild
EventHandler.eventMulticastAddress	postBuild
EventHandler.sdServerEgTimingConfig	postBuild
ExecutableEntity.canEnter	preCompileTime
ExecutableEntity.runsInside	preCompileTime

▽

△

Variation Point	*Latest Binding Time*
FlatMap.instance	postBuild
FlexrayArTpConfig.tpAddress	postBuild
FlexrayArTpConfig.tpChannel	postBuild
FlexrayArTpConfig.tpNode	postBuild
FlexrayCluster	postBuild
FlexrayCommunicationController	postBuild
FlexrayTpConfig.pduPool	postBuild
FlexrayTpConfig.tpAddress	postBuild
FlexrayTpConfig.tpConnection	postBuild
FlexrayTpConfig.tpConnectionControl	postBuild
FlexrayTpConfig.tpEcu	postBuild
FlexrayTpConfig.tpNode	postBuild
Frame.pduToFrameMapping	postBuild
FrameTriggering.pduTriggering	postBuild
Gateway.frameMapping	postBuild
Gateway.iPduMapping	postBuild
Gateway.signalMapping	postBuild
GlobalTimeDomain.gateway	postBuild
GlobalTimeDomain.globalTimeDomainProperty	postBuild
GlobalTimeDomain.globalTimeMaster	postBuild
GlobalTimeDomain.globalTimeSubDomain	postBuild
GlobalTimeDomain.pduTriggering	postBuild
GlobalTimeDomain.slave	postBuild
HwAttributeValue.v	systemDesignTime
HwDescriptionEntity.hwAttributeValue	systemDesignTime
HwElement.hwElementConnection	systemDesignTime
HwElement.hwPinGroup	systemDesignTime
HwElement.nestedElement	systemDesignTime
HwElementConnector.hwPinConnection	systemDesignTime
HwElementConnector.hwPinGroupConnection	systemDesignTime
HwPinGroupConnector.hwPinConnection	systemDesignTime
HwPinGroupContent.hwPin	systemDesignTime
HwPinGroupContent.hwPinGroup	systemDesignTime
IdsDesign.element	systemDesignTime
IdsmInstance.ecuInstance	systemDesignTime
IdsmInstance.rateLimitationFilter	preCompileTime
IdsmInstance.trafficLimitationFilter	preCompileTime
Implementation.buildActionManifest	codeGenerationTime
Implementation.generatedArtifact	preCompileTime
Implementation.requiredArtifact	preCompileTime
Implementation.requiredGeneratorTool	preCompileTime
ImplementationDataType.subElement	preCompileTime
ImplementationDataTypeElement.arraySize	preCompileTime
ImplementationDataTypeElement.subElement	preCompileTime
InternalBehavior.constantMemory	preCompileTime

▽

△

Variation Point	*Latest Binding Time*
InternalBehavior.exclusiveArea	preCompileTime
InternalBehavior.exclusiveAreaNestingOrder	preCompileTime
InternalBehavior.staticMemory	preCompileTime
InternalConstrs.lowerLimit	preCompileTime
InternalConstrs.upperLimit	preCompileTime
ISignal.dataTransformation	codeGenerationTime
ISignalGroup.comBasedSignalGroupTransformation	codeGenerationTime
ISignalIPdu.iPduTimingSpecification	postBuild
ISignalIPdu.iSignalToPduMapping	postBuild
ISignalIPduGroup.iSignalIPdu	postBuild
ISignalIPduGroup.nmPdu	postBuild
J1939Cluster	postBuild
J1939TpConfig.tpAddress	postBuild
J1939TpConfig.tpConnection	postBuild
J1939TpConfig.tpNode	postBuild
LabeledList.labeledItem	postBuild
LinCluster	postBuild
LinCommunicationController	postBuild
LinMaster	postBuild
LinPhysicalChannel.scheduleTable	postBuild
LinSlave	postBuild
LinTpConfig.tpAddress	postBuild
LinTpConfig.tpConnection	postBuild
LinTpConfig.tpNode	postBuild
List.item	postBuild
LogAndTraceMessageCollectionSet.dltMessage	systemDesignTime
McDataInstance.subElement	preCompileTime
McFunctionDataRefSet	preCompileTime
McGroupDataRefSet	preCompileTime
McSupportData.emulationSupport	preCompileTime
McSupportData.mcParameterInstance	postBuild
McSupportData.mcVariableInstance	postBuild
ModeDeclarationGroup.modeDeclaration	blueprintDerivationTime
MultiplexedIPdu.dynamicPart	postBuild
MultiplexedIPdu.staticPart	postBuild
NmCluster.nmNode	postBuild
NmConfig.nmCluster	postBuild
NmConfig.nmClusterCoupling	postBuild
NmConfig.nmIfEcu	preCompileTime
NumericalOrText.vf	preCompileTime
NumericalValueSpecification.value	preCompileTime
NvBlockDescriptor.clientServerPort	preCompileTime
NvBlockDescriptor.instantiationDataDefProps	preCompileTime
NvBlockDescriptor.modeSwitchEventTriggeredActivity	preCompileTime
NvBlockDescriptor.nvBlockDataMapping	preCompileTime

▽

△

Variation Point	*Latest Binding Time*
NvBlockSwComponentType.bulkNvDataDescriptor	preCompileTime
NvBlockSwComponentType.nvBlockDescriptor	preCompileTime
ParameterSwComponentType.instantiationDataDefProps	preCompileTime
PdurIPduGroup.iPdu	postBuild
PduTriggering.iSignalTriggering	postBuild
PerInstanceMemorySize.size	preCompileTime
PhysConstrs.lowerLimit	preCompileTime
PhysConstrs.upperLimit	preCompileTime
PhysicalChannel.commConnector	postBuild
PhysicalChannel.frameTriggering	postBuild
PhysicalChannel.iSignalTriggering	postBuild
PhysicalChannel.pduTriggering	postBuild
PncMapping.pncConsumedProvidedServiceInstanceGroup	postBuild
PortGroup.outerPort	preCompileTime
PortInterfaceMappingSet.portInterfaceMapping	blueprintDerivationTime
PostBuildVariantCondition.value	preCompileTime
PostBuildVariantCriterionValue.value	preCompileTime
ProvidedServiceInstance.eventHandler	postBuild
ProvidedServiceInstance.localUnicastAddress	postBuild
ProvidedServiceInstance.remoteMulticastSubscriptionAddress	postBuild
ProvidedServiceInstance.remoteUnicastAddress	postBuild
ProvidedServiceInstance.sdServerTimerConfig	postBuild
RapidPrototypingScenario.rptContainer	preCompileTime
ReceiverComSpec.maxDeltaCounterInit	preCompileTime
ReceiverComSpec.usesEndToEndProtection	preCompileTime
RecordValueSpecification.field	preCompileTime
ResourceConsumption.accessCountSet	preCompileTime
ResourceConsumption.executionTime	preCompileTime
ResourceConsumption.heapUsage	preCompileTime
ResourceConsumption.memorySection	preCompileTime
ResourceConsumption.sectionNamePrefix	preCompileTime
ResourceConsumption.stackUsage	preCompileTime
RptComponent.rptExecutableEntity	preCompileTime
RptContainer.byPassPoint	preCompileTime
RptContainer.rptContainer	preCompileTime
RptContainer.rptHook	preCompileTime
RptExecutableEntity.rptExecutableEntityEvent	preCompileTime
RptExecutableEntity.rptRead	preCompileTime
RptExecutableEntity.rptWrite	preCompileTime
RptSupportData.rptComponent	preCompileTime
RptSupportData.rptServicePoint	preCompileTime
RuleArguments.vf	preCompileTime
RuleArguments.vtf	preCompileTime
RuleBasedValueSpecification.arguments	preCompileTime
RunnableEntity.asynchronousServerCallResultPoint	preCompileTime

▽

△

Variation Point	*Latest Binding Time*
RunnableEntity.dataReadAccess	preCompileTime
RunnableEntity.dataReceivePointByArgument	preCompileTime
RunnableEntity.dataReceivePointByValue	preCompileTime
RunnableEntity.dataSendPoint	preCompileTime
RunnableEntity.dataWriteAccess	preCompileTime
RunnableEntity.externalTriggeringPoint	preCompileTime
RunnableEntity.internalTriggeringPoint	preCompileTime
RunnableEntity.modeAccessPoint	preCompileTime
RunnableEntity.modeSwitchPoint	preCompileTime
RunnableEntity.parameterAccess	preCompileTime
RunnableEntity.readLocalVariable	preCompileTime
RunnableEntity.serverCallPoint	preCompileTime
RunnableEntity.writtenLocalVariable	preCompileTime
RunnableEntityGroup.runnableEntity	preCompileTime
RunnableEntityGroup.runnableEntityGroup	preCompileTime
ScaleConstr.lowerLimit	preCompileTime
ScaleConstr.upperLimit	preCompileTime
Sdf.value	preCompileTime
SdgContents.sdg	postBuild
SdgContents.sdxf	postBuild
SecurityEventContextMapping.filterChain	preCompileTime
SecurityEventContextMapping.idsmInstance	systemDesignTime
SecurityEventContextMapping.mappedSecurityEvent	preCompileTime
SecurityEventContextMappingCommConnector.commConnector	preCompileTime
SecurityEventContextProps.contextData	systemDesignTime
SecurityEventContextProps.securityEvent	systemDesignTime
SenderComSpec.usesEndToEndProtection	preCompileTime
ServiceDependency.assignedDataType	preCompileTime
ServiceInstanceCollectionSet.serviceInstance	postBuild
SoAdConfig.connection	postBuild
SoAdConfig.connectionBundle	postBuild
SoAdConfig.socketAddress	postBuild
SocketAddress.staticSocketConnection	postBuild
SOMEIPTransformationISignalProps	postBuild
StaticSocketConnection.iPduIdentifier	postBuild
StaticSocketConnection.remoteAddress	postBuild
SubElementMapping.firstElement	preCompileTime
SubElementMapping.secondElement	preCompileTime
SupervisedEntityNeeds.checkpoints	preCompileTime
SwAxisIndividual.swMaxAxisPoints	preCompileTime
SwAxisIndividual.swMinAxisPoints	preCompileTime
SwcBswMapping.runnableMapping	preCompileTime
SwcBswMapping.synchronizedModeGroup	preCompileTime
SwcBswMapping.synchronizedTrigger	preCompileTime
SwcImplementation.perInstanceMemorySize	preCompileTime

▽

△

Variation Point	*Latest Binding Time*
SwcInternalBehavior.arTypedPerInstanceMemory	preCompileTime
SwcInternalBehavior.event	preCompileTime
SwcInternalBehavior.exclusiveAreaPolicy	preCompileTime
SwcInternalBehavior.explicitInterRunnableVariable	preCompileTime
SwcInternalBehavior.implicitInterRunnableVariable	preCompileTime
SwcInternalBehavior.instantiationDataDefProps	preCompileTime
SwcInternalBehavior.perInstanceMemory	preCompileTime
SwcInternalBehavior.perInstanceParameter	preCompileTime
SwcInternalBehavior.portAPIOption	preCompileTime
SwcInternalBehavior.runnable	preCompileTime
SwcInternalBehavior.serviceDependency	preCompileTime
SwcInternalBehavior.sharedParameter	preCompileTime
SwComponentDocumentation.chapter	postBuild
SwComponentType.consistencyNeeds	preCompileTime
SwComponentType.port	preCompileTime
SwComponentType.portGroup	preCompileTime
SwComponentType.swComponentDocumentation	preCompileTime
SwcServiceDependency.assignedData	preCompileTime
SwcServiceDependency.assignedPort	preCompileTime
SwDataDefProps	codeGenerationTime
SwDataDefProps.swValueBlockSize	preCompileTime
SwDataDefProps.swValueBlockSizeMult	preCompileTime
SwGenericAxisParam.vf	preCompileTime
SwSystemconstValue.value	preCompileTime
SwTextProps.swMaxTextSize	preCompileTime
SwValues.vf	preCompileTime
SwValues.vtf	preCompileTime
System.fibexElement	postBuild
System.j1939SharedAddressCluster	postBuild
System.mapping	postBuild
System.rootSoftwareComposition	systemDesignTime
System.swCluster	systemDesignTime
System.systemDocumentation	systemDesignTime
SystemMapping.applicationPartitionToEcuPartitionMapping	postBuild
SystemMapping.comManagementMapping	systemDesignTime
SystemMapping.cryptoServiceMapping	postBuild
SystemMapping.dataMapping	postBuild
SystemMapping.ecuResourceMapping	systemDesignTime
SystemMapping.mappingConstraint	systemDesignTime
SystemMapping.pncMapping	systemDesignTime
SystemMapping.portElementToComResourceMapping	postBuild
SystemMapping.resourceEstimation	systemDesignTime
SystemMapping.resourceToApplicationPartitionMapping	systemDesignTime
SystemMapping.signalPathConstraint	systemDesignTime

▽

△

Variation Point	*Latest Binding Time*
SystemMapping.softwareClusterToResourceMapping	preCompileTime
SystemMapping.swClusterMapping	systemDesignTime
SystemMapping.swcToApplicationPartitionMapping	postBuild
SystemMapping.swImplMapping	preCompileTime
SystemMapping.swMapping	preCompileTime
Tbody.row	postBuild
TDCpSoftwareClusterMappingSet.tdCpSoftwareClusterResourceToTdMapping	postBuild
TDCpSoftwareClusterMappingSet.tdCpSoftwareClusterToTdMapping	postBuild
TextTableMapping.bitfieldTextTableMaskFirst	preCompileTime
TextTableMapping.bitfieldTextTableMaskSecond	preCompileTime
TextTableValuePair.firstValue	preCompileTime
TextTableValuePair.secondValue	preCompileTime
TimingExtension.timingCondition	postBuild
TimingExtension.timingDescription	postBuild
TimingExtension.timingGuarantee	postBuild
TimingExtension.timingRequirement	postBuild
TimingExtensionResource.timingArgument	postBuild
TimingExtensionResource.timingMode	postBuild
TimingExtensionResource.timingVariable	postBuild
TopicContent.table	postBuild
TopicOrMsrQuery.topic1	postBuild
TransformationISignalProps	postBuild
TransformationTechnology.transformationDescription	postBuild
TtcanCluster	postBuild
TtcanCommunicationController	postBuild
UserDefinedCluster	postBuild
UserDefinedCommunicationController	postBuild
UserDefinedTransformationISignalProps	postBuild
ValueList.vf	preCompileTime

Table C.1: Usage of variation points

D Splitable Elements in this Template

Name of splitable element	*Splitkey*
AdminData.sdg	sdg, sdg.variationPoint.shortLabel
ARPackage.arPackage	arPackage.shortName, arPackage.variation Point.shortLabel
ARPackage.element	element.shortName, element.definition, element.variationPoint.shortLabel
ARPackage.referenceBase	referenceBase.shortLabel
AUTOSAR.arPackage	arPackage.shortName, arPackage.variation Point.shortLabel

▽

△

Name of splitable element	*Splitkey*
Describable.adminData	adminData
Identifiable.adminData	adminData

Table D.1: Usage of splitable elements

E Mentioned Class Tables

For the sake of completeness, this chapter contains a set of class tables representing meta-classes mentioned in the context of this document but which are not contained directly in the scope of describing specific meta-model semantics.

Class	**AUTOSAR**			
Package	M2::AUTOSARTemplates::AutosarTopLevelStructure			
Note	Root element of an AUTOSAR description, also the root element in corresponding XML documents. **Tags:**xml.globalElement=true			
Base	*ARObject*			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
adminData	AdminData	0..1	aggr	This represents the administrative data of an Autosar file. **Tags:**xml.sequenceOffset=10
arPackage	ARPackage	*	aggr	This is the top level package in an AUTOSAR model. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=arPackage.shortName, arPackage.variationPoint.shortLabel vh.latestBindingTime=blueprintDerivationTime xml.sequenceOffset=30
fileInfoComment	FileInfoComment	0..1	aggr	This represents a possibility to provide a structured comment in an AUTOSAR file. **Stereotypes:** atpStructuredComment **Tags:** xml.roleElement=true xml.sequenceOffset=-10 xml.typeElement=false
introduction	DocumentationBlock	0..1	aggr	This represents an introduction on the Autosar file. It is intended for example to represent disclaimers and legal notes. **Tags:**xml.sequenceOffset=20

Table E.1: AUTOSAR

Class	<<atpMixedString>> ***AbstractEnumerationValueVariationPoint*** (abstract)			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling::AttributeValueVariationPoints			
Note	This is an abstract EnumerationValueVariationPoint. It is introduced to support the case that additional attributes are required for particular purposes.			
Base	*ARObject*, *AttributeValueVariationPoint*, *FormulaExpression*, *SwSystemconstDependentFormula*			
Subclasses				
Attribute	*Type*	*Mult.*	*Kind*	*Note*

▽

△

Class	<<atpMixedString>> ***AbstractEnumerationValueVariationPoint*** (abstract)			
base	Identifier	0..1	attr	This attribute reflects the base to be used in context of EnumerationMappingTable for this reference. **Tags:**xml.attribute=true
enumTable	Ref	0..1	attr	This represents the assigned enumeration table. **Tags:**xml.attribute=true

Table E.2: AbstractEnumerationValueVariationPoint

Class	***AbstractMultiplicityRestriction*** (abstract)			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::ModelRestrictionTypes			
Note	Restriction that specifies the valid number of occurrences of an element in the current context.			
Base	*ARObject*			
Subclasses	*AttributeCondition*, MultiplicityRestrictionWithSeverity, *SdgAttribute*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
lowerMultiplicity	PositiveInteger	0..1	attr	Specifies the minimal number of times an object shall occur. If this primitive attribute is not set, then the object is optional.
upperMultiplicity	PositiveInteger	0..1	attr	Specifies the maximum number of times an object may occur. If this primitive attribute is not set, then there is no limit with respect to the maximum occurrence.
upperMultiplicity Infinite	Boolean	0..1	attr	This explicitly specifies, that the upper multiplicity is NOT restricted. Note: The use of 'upperMultiplicityInfinite' and 'upperMultiplicity' is mutual exclusive.

Table E.3: AbstractMultiplicityRestriction

Class	***AbstractRequiredPortPrototype*** (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	This abstract class provides the ability to become a required PortPrototype.			
Base	*ARObject*, *AtpBlueprintable*, *AtpFeature*, *AtpPrototype*, *Identifiable*, *MultilanguageReferrable*, *PortPrototype*, *Referrable*			
Subclasses	PRPortPrototype, RPortPrototype			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
requiredComSpec	RPortComSpec	*	aggr	Required communication attributes, one for each interface element.

Table E.4: AbstractRequiredPortPrototype

Enumeration	**AdditionalBindingTimeEnum**
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling
Note	This enumeration specifies the additional binding times applicable for vh.latestBindingTime of variation points.
Literal	***Description***
blueprintDerivationTime	The point in time when an object is created from a blueprint. **Tags:**atp.EnumerationLiteralIndex=0

▽

△

Enumeration	**AdditionalBindingTimeEnum**
postBuild	After the executable has been built. **Tags:**atp.EnumerationLiteralIndex=1

Table E.5: AdditionalBindingTimeEnum

Primitive	**AnyServiceInstanceId**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	This is a positive integer or the literal ALL (the value ANY is technically supported but deprecated) which can be denoted in decimal, octal and hexadecimal. The value is between 0 and 65535. **Tags:** xml.xsd.customType=ANY-SERVICE-INSTANCE-ID xml.xsd.pattern=[1-9][0-9]*|0[xX][0-9a-fA-F]+|0[0-7]*|0[bB][0-1]+|ANY|ALL xml.xsd.type=string

Table E.6: AnyServiceInstanceId

Class	**ApplicationSwComponentType**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	The ApplicationSwComponentType is used to represent the application software. **Tags:**atp.recommendedPackage=SwComponentTypes			
Base	*ARElement*, *ARObject*, *AtomicSwComponentType*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *SwComponentType*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table E.7: ApplicationSwComponentType

Class	**AssemblySwConnector**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Composition			
Note	AssemblySwConnectors are exclusively used to connect SwComponentPrototypes in the context of a CompositionSwComponentType.			
Base	*ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *SwConnector*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
provider	AbstractProvidedPort Prototype	0..1	iref	Instance of providing port. **InstanceRef implemented by:**PPortInComposition InstanceRef
requester	AbstractRequiredPort Prototype	0..1	iref	Instance of requiring port. **InstanceRef implemented by:**RPortInComposition InstanceRef

Table E.8: AssemblySwConnector

Class	***AtpBlueprint*** (abstract)
Package	M2::AUTOSARTemplates::CommonStructure::StandardizationTemplate::AbstractBlueprintStructure
Note	This meta-class represents the ability to act as a Blueprint. As this class is an abstract one, particular blueprint meta-classes inherit from this one.

▽

△

Class	***AtpBlueprint*** (abstract)			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	ARPackage, *AbstractImplementationDataType*, AclObjectSet, AclOperation, AclPermission, AclRole, AliasNameSet, *ApplicationDataType*, BswEntryRelationshipSet, BswModuleDescription, BswModuleEntry, *BuildActionEntity*, BuildActionEnvironment, BuildActionManifest, ClientServerInterfaceToBswModuleEntryBlueprintMapping, CompuMethod, ConsistencyNeeds, DataConstr, DataTypeMappingSet, EcucDefinitionCollection, EcucDestinationUriDefSet, EcucModuleDef, FlatMap, KeywordSet, LifeCycleState, LifeCycleStateDefinitionGroup, ModeDeclarationGroup, *PortInterface*, *PortInterfaceMapping*, PortInterfaceMappingSet, PortPrototypeBlueprint, SwAddrMethod, SwBaseType, *SwComponentType*, VfbTiming			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
blueprintPolicy	BlueprintPolicy	*	aggr	This role indicates whether the blueprintable element will be modifiable or not modifiable.

Table E.9: AtpBlueprint

Class	***AtpBlueprintable*** (abstract)			
Package	M2::AUTOSARTemplates::CommonStructure::StandardizationTemplate::AbstractBlueprintStructure			
Note	This meta-class represents the ability to be derived from a Blueprint. As this class is an abstract one, particular blueprintable meta-classes inherit from this one.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	ARPackage, *AbstractImplementationDataType*, AclObjectSet, AclOperation, AclPermission, AclRole, AliasNameSet, *ApplicationDataType*, BswEntryRelationshipSet, BswModuleDescription, BswModuleEntry, *BuildActionEntity*, BuildActionEnvironment, BuildActionManifest, CompuMethod, ConsistencyNeeds, DataConstr, DataTypeMappingSet, EcucDefinitionCollection, EcucDestinationUriDefSet, EcucModuleDef, FlatMap, KeywordSet, LifeCycleState, LifeCycleStateDefinitionGroup, ModeDeclarationGroup, *PortInterface*, *PortInterfaceMapping*, PortInterfaceMappingSet, *PortPrototype*, SwAddrMethod, SwBaseType, *SwComponentType*, VfbTiming			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table E.10: AtpBlueprintable

Class	**BlueprintGenerator**			
Package	M2::AUTOSARTemplates::CommonStructure::StandardizationTemplate::BlueprintGenerator			
Note	This class express the Extended Language to generate blueprint derivates in complex descriptions. **Tags:**atp.Status=valid			
Base	*ARObject*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
expression	VerbatimString	0..1	attr	This represents a formal term in the expression based on the extended language. **Tags:** atp.Status=valid xml.sequenceOffset=20
introduction	DocumentationBlock	0..1	aggr	This represents a description that documents how the blueprint generator shall be resolved when deriving objects from blueprints. **Tags:** atp.Status=valid xml.sequenceOffset=10

Table E.11: BlueprintGenerator

Primitive	**Boolean**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	A Boolean value denotes a logical condition that is either 'true' or 'false'. It can be one of "0", "1", "true", "false" **Tags:** xml.xsd.customType=BOOLEAN xml.xsd.pattern=0|1|true|false xml.xsd.type=string

Table E.12: Boolean

Class	**BswImplementation**			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswImplementation			
Note	Contains the implementation specific information in addition to the generic specification (BswModule Description and BswBehavior). It is possible to have several different BswImplementations referring to the same BswBehavior. **Tags:**atp.recommendedPackage=BswImplementations			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *Implementation*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
arRelease Version	RevisionLabelString	1	attr	Version of the AUTOSAR Release on which this implementation is based. The numbering contains three levels (major, minor, revision) which are defined by AUTOSAR.
behavior	BswInternalBehavior	1	ref	The behavior of this implementation. This relation is made as an association because • it follows the pattern of the SWCT • since ARElement cannot be splitted, but we want supply the implementation later, the Bsw Implementation is not aggregated in BswBehavior
preconfigured Configuration	EcucModule ConfigurationValues	*	ref	Reference to the set of preconfigured (i.e. fixed) configuration values for this BswImplementation. If the BswImplementation represents a cluster of several modules, more than one EcucModuleConfigurationValues element can be referred (at most one per module), otherwise at most one such element can be referred. **Tags:**xml.roleWrapperElement=true
recommended Configuration	EcucModule ConfigurationValues	*	ref	Reference to one or more sets of recommended configuration values for this module or module cluster.
vendorApiInfix	Identifier	0..1	attr	In driver modules which can be instantiated several times on a single ECU, SRS_BSW_00347 requires that the names of files, APIs, published parameters and memory allocation keywords are extended by the vendorId and a vendor specific name. This parameter is used to specify the vendor specific name. In total, the implementation specific API name is generated as follows: <Module Name>_<vendorId>_ <vendorApiInfix>_<API name from SWS>. E.g. assuming that the vendorId of the implementer is 123 and the implementer chose a vendorApiInfix of "v11r456" an API name Can_Write defined in the SWS will translate to Can_123_v11r456_Write.

△

Class	BswImplementation			
				△ This attribute is mandatory for all modules with upper multiplicity > 1. It shall not be used for modules with upper multiplicity =1. See also SWS_BSW_00102.
vendorSpecific ModuleDef	EcucModuleDef	*	ref	Reference to • the vendor specific EcucModuleDef used in this BswImplementation if it represents a single module • several EcucModuleDefs used in this Bsw Implementation if it represents a cluster of modules • one or no EcucModuleDefs used in this Bsw Implementation if it represents a library **Tags:**xml.roleWrapperElement=true

Table E.13: BswImplementation

Class	BswModuleDescription			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswOverview			
Note	Root element for the description of a single BSW module or BSW cluster. In case it describes a BSW module, the short name of this element equals the name of the BSW module. **Tags:**atp.recommendedPackage=BswModuleDescriptions			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
bswModule Dependency	BswModuleDependency	*	aggr	Describes the dependency to another BSW module. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=bswModuleDependency.shortName, bswModuleDependency.variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=20
bswModule Documentation	SwComponent Documentation	0..1	aggr	This adds a documentation to the BSW module. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=bswModuleDocumentation, bswModuleDocumentation.variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=6
expectedEntry	BswModuleEntry	*	ref	Indicates an entry which is required by this module. Replacement of outgoingCallback / requiredEntry. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=expectedEntry.bswModuleEntry, expectedEntry.variationPoint.shortLabel vh.latestBindingTime=preCompileTime

▽

△

Class	**BswModuleDescription**			
implemented Entry	BswModuleEntry	*	ref	Specifies an entry provided by this module which can be called by other modules. This includes "main" functions, interrupt routines, and callbacks. Replacement of providedEntry / expectedCallback. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=implementedEntry.bswModuleEntry, implementedEntry.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
internalBehavior	BswInternalBehavior	*	aggr	The various BswInternalBehaviors associated with a Bsw ModuleDescription can be distributed over several physical files. Therefore the aggregation is <<atp Splitable>>. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=internalBehavior.shortName xml.sequenceOffset=65
moduleId	PositiveInteger	0..1	attr	Refers to the BSW Module Identifier defined by the AUTOSAR standard. For non-standardized modules, a proprietary identifier can be optionally chosen. **Tags:**xml.sequenceOffset=5
providedClient ServerEntry	BswModuleClientServer Entry	*	aggr	Specifies that this module provides a client server entry which can be called from another partition or core.This entry is declared locally to this context and will be connected to the requiredClientServerEntry of another or the same module via the configuration of the BSW Scheduler. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=providedClientServerEntry.shortName, providedClientServerEntry.variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=45
providedData	VariableDataPrototype	*	aggr	Specifies a data prototype provided by this module in order to be read from another partition or core.The providedData is declared locally to this context and will be connected to the requiredData of another or the same module via the configuration of the BSW Scheduler. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=providedData.shortName, provided Data.variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=55
providedMode Group	ModeDeclarationGroup Prototype	*	aggr	A set of modes which is owned and provided by this module or cluster. It can be connected to the required ModeGroups of other modules or clusters via the configuration of the BswScheduler. It can also be synchronized with modes provided via ports by an associated ServiceSwComponentType, EcuAbstraction SwComponentType or ComplexDeviceDriverSw ComponentType. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=providedModeGroup.shortName, provided ModeGroup.variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=25

▽

△

Class	**BswModuleDescription**			
releasedTrigger	Trigger	*	aggr	A Trigger released by this module or cluster. It can be connected to the requiredTriggers of other modules or clusters via the configuration of the BswScheduler. It can also be synchronized with Triggers provided via ports by an associated ServiceSwComponentType, Ecu AbstractionSwComponentType or ComplexDeviceDriver SwComponentType. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=releasedTrigger.shortName, released Trigger.variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=35
requiredClient ServerEntry	BswModuleClientServer Entry	*	aggr	Specifies that this module requires a client server entry which can be implemented on another partition or core.This entry is declared locally to this context and will be connected to the providedClientServerEntry of another or the same module via the configuration of the BSW Scheduler. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=requiredClientServerEntry.shortName, requiredClientServerEntry.variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=50
requiredData	VariableDataPrototype	*	aggr	Specifies a data prototype required by this module in oder to be provided from another partition or core.The required Data is declared locally to this context and will be connected to the providedData of another or the same module via the configuration of the BswScheduler. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=requiredData.shortName, required Data.variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=60
requiredMode Group	ModeDeclarationGroup Prototype	*	aggr	Specifies that this module or cluster depends on a certain mode group. The requiredModeGroup is local to this context and will be connected to the providedModeGroup of another module or cluster via the configuration of the BswScheduler. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=requiredModeGroup.shortName, required ModeGroup.variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=30
requiredTrigger	Trigger	*	aggr	Specifies that this module or cluster reacts upon an external trigger.This requiredTrigger is declared locally to this context and will be connected to the providedTrigger of another module or cluster via the configuration of the BswScheduler. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=requiredTrigger.shortName, required Trigger.variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=40

Table E.14: BswModuleDescription

Class	***BswModuleEntity*** (abstract)			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	Specifies the smallest code fragment which can be described for a BSW module or cluster within AUTOSAR.			
Base	*ARObject*, *ExecutableEntity*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	BswCalledEntity, BswInterruptEntity, BswSchedulableEntity			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
accessedMode Group	ModeDeclarationGroup Prototype	*	ref	A mode group which is accessed via API call by this entity. It shall be a ModeDeclarationGroupPrototype required by this module or cluster. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime
activationPoint	BswInternalTriggering Point	*	ref	Activation point used by the module entity to activate one or more internal triggers. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime
callPoint	BswModuleCallPoint	*	aggr	A call point used in the code of this entity. The variability of this association is especially targeted at debug scenarios: It is possible to have one variant calling into the AUTOSAR debug module and another one which doesn't. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime
dataReceive Point	BswVariableAccess	*	aggr	The data is received via the BSW Scheduler. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime
dataSendPoint	BswVariableAccess	*	aggr	The data is sent via the BSW Scheduler. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime
implemented Entry	BswModuleEntry	1	ref	The entry which is implemented by this module entity.
issuedTrigger	Trigger	*	ref	A trigger issued by this entity via BSW Scheduler API call. It shall be a BswTrigger released (i.e. owned) by this module or cluster. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime
managedMode Group	ModeDeclarationGroup Prototype	*	ref	A mode group which is managed by this entity. It shall be a ModeDeclarationGroupPrototype provided by this module or cluster. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime
schedulerName Prefix	BswSchedulerName Prefix	0..1	ref	A prefix to be used in generated names for the Bsw ModuleScheduler in the context of this BswModuleEntity, for example entry point prototypes, macros for dealing with exclusive areas, header file names. Details are defined in the SWS RTE. The prefix supersedes default rules for the prefix of those names.

Table E.15: BswModuleEntity

Class	**BswModuleEntry**			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswInterfaces			
Note	This class represents a single API entry (C-function prototype) into the BSW module or cluster. The name of the C-function is equal to the short name of this element with one exception: In case of multiple instances of a module on the same CPU, special rules for "infixes" apply, see description of class BswImplementation. **Tags:**atp.recommendedPackage=BswModuleEntrys			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
argument (ordered)	SwServiceArg	*	aggr	An argument belonging to this BswModuleEntry. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=blueprintDerivationTime xml.sequenceOffset=45
bswEntryKind	BswEntryKindEnum	0..1	attr	This describes whether the entry is concrete or abstract. If the attribute is missing the entry is considered as concrete. **Tags:**xml.sequenceOffset=40
callType	BswCallType	1	attr	The type of call associated with this service. **Tags:**xml.sequenceOffset=25
executionContext	BswExecutionContext	1	attr	Specifies the execution context which is required (in case of entries into this module) or guaranteed (in case of entries called from this module) for this service. **Tags:**xml.sequenceOffset=30
functionPrototypeEmitter	NameToken	0..1	attr	This attribute is used to control the generation of function prototypes. If set to "RTE", the RTE generates the function prototypes in the Module Interlink Header File.
isReentrant	Boolean	1	attr	Reentrancy from the viewpoint of function callers: • True: Enables the service to be invoked again, before the service has finished. • False: It is prohibited to invoke the service again before is has finished. **Tags:**xml.sequenceOffset=15
isSynchronous	Boolean	1	attr	Synchronicity from the viewpoint of function callers: • True: This calls a synchronous service, i.e. the service is completed when the call returns. • False: The service (on semantical level) may not be complete when the call returns. **Tags:**xml.sequenceOffset=20
returnType	SwServiceArg	0..1	aggr	The return type belonging to this bswModuleEntry. **Tags:**xml.sequenceOffset=40
role	Identifier	0..1	attr	Specifies the role of the entry in the given context. It shall be equal to the standardized name of the service call, especially in cases where no ServiceIdentifier is specified, e.g. for callbacks. Note that the ShortName is not always sufficient because it maybe vendor specific (e.g. for callbacks which can have more than one instance). **Tags:**xml.sequenceOffset=10

▽

△

Class	**BswModuleEntry**			
serviceId	PositiveInteger	0..1	attr	Refers to the service identifier of the Standardized Interfaces of AUTOSAR basic software. For non-standardized interfaces, it can optionally be used for proprietary identification. **Tags:**xml.sequenceOffset=5
swServiceImpl Policy	SwServiceImplPolicy Enum	1	attr	Denotes the implementation policy as a standard function call, inline function or macro. This has to be specified on interface level because it determines the signature of the call. **Tags:**xml.sequenceOffset=35

Table E.16: BswModuleEntry

Enumeration	**ByteOrderEnum**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	When more than one byte is stored in the memory the order of those bytes may differ depending on the architecture of the processing unit. If the least significant byte is stored at the lowest address, this architecture is called little endian and otherwise it is called big endian. ByteOrder is very important in case of communication between different PUs or ECUs.
Literal	*Description*
mostSignificantByte First	Most significant byte shall come at the lowest address (also known as BigEndian or as Motorola-Format) **Tags:**atp.EnumerationLiteralIndex=0
mostSignificantByte Last	Most significant byte shall come highest address (also known as LittleEndian or as Intel-Format) **Tags:**atp.EnumerationLiteralIndex=1
opaque	For opaque data endianness conversion has to be configured to Opaque. See AUTOSAR COM Specification for more details. **Tags:**atp.EnumerationLiteralIndex=2

Table E.17: ByteOrderEnum

Class	**Caption**			
Package	M2::MSR::Documentation::BlockElements			
Note	This meta-class represents the ability to express a caption which is a title, and a shortName.			
Base	*ARObject*, *MultilanguageReferrable*, *Referrable*			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
desc	MultiLanguageOverview Paragraph	0..1	aggr	This represents a general but brief (one paragraph) description what the object in question is about. It is only one paragraph! This property helps a human reader to identify the object in question. **Tags:**xml.sequenceOffset=10

Table E.18: Caption

Class	**ClientServerInterface**
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface
Note	A client/server interface declares a number of operations that can be invoked on a server by a client. **Tags:**atp.recommendedPackage=PortInterfaces

▽

△

Class	**ClientServerInterface**			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
operation	ClientServerOperation	*	aggr	ClientServerOperation(s) of this ClientServerInterface. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=blueprintDerivationTime
possibleError	ApplicationError	*	aggr	Application errors that are defined as part of this interface.

Table E.19: ClientServerInterface

Class	**ClientServerOperation**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	An operation declared within the scope of a client/server interface.			
Base	*ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
argument (ordered)	ArgumentDataPrototype	*	aggr	An argument of this ClientServerOperation **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=blueprintDerivationTime
diagArgIntegrity	Boolean	0..1	attr	This attribute shall only be used in the implementation of diagnostic routines to support the case where input and output arguments are allocated in a shared buffer and might unintentionally overwrite input arguments by tentative write operations to output arguments. This situation can happen during sliced execution or while output parameters are arrays (call by reference). The value true means that the ClientServerOperation is aware of the usage of a shared buffer and takes precautions to avoid unintentional overwrite of input arguments. If the attribute does not exist or is set to false the Client ServerOperation does not have to consider the usage of a shared buffer.
possibleError	ApplicationError	*	ref	Possible errors that may by raised by the referring operation.

Table E.20: ClientServerOperation

Class	**Colspec**			
Package	M2::MSR::Documentation::BlockElements::OasisExchangeTable			
Note	This meta-class represents the ability to specify the properties of a column in a table.			
Base	*ARObject*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
align	AlignEnum	0..1	attr	Specifies how the cell entries shall be horizontally aligned within the specified column. Default is "LEFT" **Tags:**xml.attribute=true
colname	String	0..1	attr	Specifies the name of the column. **Tags:**xml.attribute=true
colnum	String	0..1	attr	column number (allows to sort the columns). **Tags:**xml.attribute=true

▽

△

Class	**Colspec**			
colsep	TableSeparatorString	0..1	attr	Indicates whether a line should be displayed right of this column in the column specification. **Tags:**xml.attribute=true
colwidth	String	0..1	attr	Width of the column. You can enter absolute values such as 4 cm, or relative values marked with * (e.g., 2* for column widths double those of other columns with 1*). The unit can be added to the number in the string. Possible units are: cm, mm, px, pt. **Tags:**xml.attribute=true
rowsep	TableSeparatorString	0..1	attr	Indicates whether a line should be displayed at the bottom end of the cells of the column defined in the Colspec. **Tags:**xml.attribute=true

Table E.21: Colspec

Class	**Compiler**			
Package	M2::AUTOSARTemplates::CommonStructure::Implementation			
Note	Specifies the compiler attributes. In case of source code this specifies requirements how the compiler shall be invoked. In case of object code this documents the used compiler settings.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
name	String	0..1	attr	Compiler name (like gcc).
options	String	0..1	attr	Specifies the compiler options.
vendor	String	0..1	attr	Vendor of compiler.
version	String	0..1	attr	Exact version of compiler executable.

Table E.22: Compiler

Class	**CompositionSwComponentType**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Composition			
Note	A CompositionSwComponentType aggregates SwComponentPrototypes (that in turn are typed by Sw ComponentTypes) as well as SwConnectors for primarily connecting SwComponentPrototypes among each others and towards the surface of the CompositionSwComponentType. By this means, hierarchical structures of software-components can be created. **Tags:**atp.recommendedPackage=SwComponentTypes			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *SwComponentType*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
component	SwComponent Prototype	*	aggr	The instantiated components that are part of this composition. The aggregation of SwComponentPrototype is subject to variability with the purpose to support the conditional existence of a SwComponentPrototype. Please be aware: if the conditional existence of Sw ComponentPrototypes is resolved post-build the deselected SwComponentPrototypes are still contained in the ECUs build but the instances are inactive in that they are not scheduled by the RTE. The aggregation is marked as atpSplitable in order to allow the addition of service components to the ECU extract during the ECU integration. ▽

▽

△

Class	**CompositionSwComponentType**			
				△ The use case for having 0 components owned by the CompositionSwComponentType could be to deliver an empty CompositionSwComponentType to e.g. a supplier for filling the internal structure. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=component.shortName, component.variation Point.shortLabel vh.latestBindingTime=postBuild
connector	SwConnector	*	aggr	SwConnectors have the principal ability to establish a connection among PortPrototypes. They can have many roles in the context of a CompositionSwComponentType. Details are refined by subclasses. The aggregation of SwConnectors is subject to variability with the purpose to support variant data flow. The aggregation is marked as atpSplitable in order to allow the extension of the ECU extract with AssemblySw Connectors between ApplicationSwComponentTypes and ServiceSwComponentTypes during the ECU integration. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=connector.shortName, connector.variation Point.shortLabel vh.latestBindingTime=postBuild
constantValue Mapping	ConstantSpecification MappingSet	*	ref	Reference to the ConstantSpecificationMapping to be applied for initValues of PPortComSpecs and RPortCom Spec. **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=constantValueMapping
dataType Mapping	DataTypeMappingSet	*	ref	Reference to the DataTypeMapping to be applied for the used ApplicationDataTypes in PortInterfaces. Background: when developing subsystems it may happen that ApplicationDataTypes are used on the surface of CompositionSwComponentTypes. In this case it would be reasonable to be able to also provide the intended mapping to the ImplementationDataTypes. However, this mapping shall be informal and not technically binding for the implementors mainly because the RTE generator is not concerned about the CompositionSwComponent Types. Rationale: if the mapping of ApplicationDataTypes on the delegated and inner PortPrototype matches then the mapping to ImplementationDataTypes is not impacting compatibility. **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=dataTypeMapping
instantiation RTEEventProps	InstantiationRTEEvent Props	*	aggr	This allows to define instantiation specific properties for RTE Events, in particular for instance specific scheduling. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=instantiationRTEEventProps.shortLabel, instantiationRTEEventProps.variationPoint.shortLabel vh.latestBindingTime=codeGenerationTime

Table E.23: CompositionSwComponentType

Class	**CompuMethod**			
Package	M2::MSR::AsamHdo::ComputationMethod			
Note	This meta-class represents the ability to express the relationship between a physical value and the mathematical representation. Note that this is still independent of the technical implementation in data types. It only specifies the formula how the internal value corresponds to its physical pendant. **Tags:**atp.recommendedPackage=CompuMethods			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
compuInternalToPhys	Compu	0..1	aggr	This specifies the computation from internal values to physical values. **Tags:**xml.sequenceOffset=80
compuPhysToInternal	Compu	0..1	aggr	This represents the computation from physical values to the internal values. **Tags:**xml.sequenceOffset=90
displayFormat	DisplayFormatString	0..1	attr	This property specifies, how the physical value shall be displayed e.g. in documents or measurement and calibration tools. **Tags:**xml.sequenceOffset=20
unit	Unit	0..1	ref	This is the physical unit of the Physical values for which the CompuMethod applies. **Tags:**xml.sequenceOffset=30

Table E.24: CompuMethod

Class	***Describable*** (abstract)			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Identifiable			
Note	This meta-class represents the ability to add a descriptive documentation to non identifiable elements.			
Base	*ARObject*			
Subclasses	CyclicTiming, EventControlledTiming, HwElementConnector, HwPinConnector, HwPinGroupConnector, IPduTiming, Ipv4DhcpServerConfiguration, Ipv6DhcpServerConfiguration, PncMapping, SocketConnection, *TransformationComSpecProps*, *TransformationDescription*, *TransformationISignalProps*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
adminData	AdminData	0..1	aggr	This represents the administrative data for the describable object. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=adminData xml.sequenceOffset=-20
category	CategoryString	0..1	attr	The category is a keyword that specializes the semantics of the Describable. It affects the expected existence of attributes and the applicability of constraints. **Tags:**xml.sequenceOffset=-50
desc	MultiLanguageOverviewParagraph	0..1	aggr	This represents a general but brief (one paragraph) description what the object in question is about. It is only one paragraph! Desc is intended to be collected into overview tables. This property helps a human reader to identify the object in question. More elaborate documentation, (in particular how the object is built or used) should go to "introduction". **Tags:**xml.sequenceOffset=-60

△

Class	***Describable*** (abstract)			
introduction	DocumentationBlock	0..1	aggr	This represents more information about how the object in question is built or is used. Therefore it is a DocumentationBlock. **Tags:**xml.sequenceOffset=-30

Table E.25: Describable

Class	**Documentation**			
Package	M2::AUTOSARTemplates::GenericStructure::DocumentationOnM1			
Note	This meta-class represents the ability to handle a so called standalone documentation. Standalone means, that such a documentation is not embedded in another ARElement or identifiable object. The standalone documentation is an entity of its own which denotes its context by reference to other objects and instances. **Tags:**atp.recommendedPackage=Documentations			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
context	DocumentationContext	*	aggr	This is the context of the particular documentation.
documentation Content	PredefinedChapter	0..1	aggr	This is the content of the documentation related to the specified contexts. **Tags:**xml.sequenceOffset=200

Table E.26: Documentation

Class	**EcucContainerValue**			
Package	M2::AUTOSARTemplates::ECUCDescriptionTemplate			
Note	Represents a Container definition in the ECU Configuration Description.			
Base	*ARObject*, *EcucIndexableValue*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
definition	EcucContainerDef	0..1	ref	Reference to the definition of this Container in the ECU Configuration Parameter Definition. **Stereotypes:** atpIdentityContributor **Tags:**xml.sequenceOffset=-10
parameterValue	EcucParameterValue	*	aggr	Aggregates all ECU Configuration Values within this Container. atpVariation: [RS_ECUC_00079] **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=parameterValue.definition, parameter Value.variationPoint.shortLabel vh.latestBindingTime=postBuild
referenceValue	EcucAbstractReference Value	*	aggr	Aggregates all References with this container. atpVariation: [RS_ECUC_00079] **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=referenceValue.definition, reference Value.variationPoint.shortLabel vh.latestBindingTime=postBuild

▽

△

Class	EcucContainerValue			
subContainer	EcucContainerValue	*	aggr	Aggregates all sub-containers within this container. atpVariation: [RS_ECUC_00078] **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=subContainer.shortName, sub Container.definition, subContainer.variationPoint.short Label vh.latestBindingTime=postBuild

Table E.27: EcucContainerValue

Class	*EcucDefinitionElement* (abstract)			
Package	M2::AUTOSARTemplates::ECUCParameterDefTemplate			
Note	Common class used to express the commonalities of configuration parameters, references and containers. If not stated otherwise the default multiplicity is exactly one mandatory occurrence of the specified element.			
Base	*ARObject*, *AtpDefinition*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	*EcucCommonAttributes*, *EcucContainerDef*, EcucModuleDef			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
ecucCond	EcucCondition Specification	0..1	aggr	If it evaluates to true the Ecu Parameter definition shall be processed as specified. Otherwise the parameter definition shall be ignored. **Tags:**xml.sequenceOffset=100
ecucValidation Cond	EcucValidation Condition	*	aggr	Collection of validation conditions which all need to evaluate to true in order to indicate a valid validation condition of the EcucDefinitionElement.
lowerMultiplicity	PositiveInteger	0..1	attr	The lower multiplicity of the specified element. 0: optional 1: at least one occurrence n: at least n occurrences atpVariation: [RS_ECUC_00082] **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=codeGenerationTime xml.sequenceOffset=110
relatedTrace Item	Traceable	0..1	ref	This contains a sloppy reference to the Autosar compatible identifier of the element (EcucId). **Stereotypes:** atpUriDef **Tags:**xml.sequenceOffset=-10
scope	EcucScopeEnum	0..1	attr	Specifies the scope of this configuration element. **Tags:**xml.sequenceOffset=150

▽

△

Class	***EcucDefinitionElement*** (abstract)			
upperMultiplicity	PositiveInteger	0..1	attr	The upper multiplicity of the specified element. 0: no occurrence (used for VSMD) 1: at most one occurrence m: at most m occurrences If upperMultiplicity is set than upperMultiplicityInfinite shall not be used. atpVariation: [RS_ECUC_00082] **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=codeGenerationTime xml.sequenceOffset=120
upperMultiplicity Infinite	Boolean	0..1	attr	To express an infinite number of occurrences of this element this attribute has to be set to true. If upperMultiplicityInfinite is set than upperMultiplicity shall not be used. atpVariation: [RS_ECUC_00082] **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=codeGenerationTime xml.sequenceOffset=130

Table E.28: EcucDefinitionElement

Class	**EcucModuleConfigurationValues**			
Package	M2::AUTOSARTemplates::ECUCDescriptionTemplate			
Note	Head of the configuration of one Module. A Module can be a BSW module as well as the RTE and ECU Infrastructure. As part of the BSW module description, the EcucModuleConfigurationValues element has two different roles: The recommendedConfiguration contains parameter values recommended by the BSW module vendor. The preconfiguredConfiguration contains values for those parameters which are fixed by the implementation and cannot be changed. These two EcucModuleConfigurationValues are used when the base EcucModuleConfigurationValues (as part of the base ECU configuration) is created to fill parameters with initial values. **Tags:**atp.recommendedPackage=EcucModuleConfigurationValuess			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
container	EcucContainerValue	*	aggr	Aggregates all containers that belong to this module configuration. atpVariation: [RS_ECUC_00078] **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=container.shortName, container.definition, container.variationPoint.shortLabel vh.latestBindingTime=postBuild xml.sequenceOffset=10

▽

△

Class	EcucModuleConfigurationValues			
definition	EcucModuleDef	0..1	ref	Reference to the definition of this EcucModule ConfigurationValues element. Typically, this is a vendor specific module configuration. **Stereotypes:** atpIdentityContributor **Tags:**xml.sequenceOffset=-10
ecucDefEdition	RevisionLabelString	0..1	attr	This is the version info of the ModuleDef ECUC Parameter definition to which this values conform to / are based on. For the Definition of ModuleDef ECUC Parameters the AdminData shall be used to express the semantic changes. The compatibility rules between the definition and value revision labels is up to the module's vendor.
implementation ConfigVariant	EcucConfiguration VariantEnum	0..1	attr	Specifies the kind of deliverable this EcucModule ConfigurationValues element provides. If this element is not used in a particular role (e.g. preconfigured Configuration or recommendedConfiguration) then the value shall be one of VariantPreCompile, VariantLink Time, VariantPostBuild.
module Description	BswImplementation	0..1	ref	Referencing the BSW module description, which this EcucModuleConfigurationValues element is configuring. This is optional because the EcucModuleConfiguration Values element is also used to configure the ECU infrastructure (memory map) or Application SW-Cs. However in case the EcucModuleConfigurationValues are used to configure the module, the reference is mandatory in order to fetch module specific "common" published information.
postBuildVariant Used	Boolean	0..1	attr	Indicates whether a module implementation has or plans to have (i.e., introduced at link or post-build time) new post-build variation points. TRUE means yes, FALSE means no. If the attribute is not defined, FALSE semantics shall be assumed.

Table E.29: EcucModuleConfigurationValues

Class	EcucNumericalParamValue			
Package	M2::AUTOSARTemplates::ECUCDescriptionTemplate			
Note	Holding the value which is subject to variant handling.			
Base	*ARObject*, *EcucIndexableValue*, *EcucParameterValue*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
value	Numerical	0..1	attr	Value which is subject to variant handling. atpVariation: [RS_ECUC_00080] **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime

Table E.30: EcucNumericalParamValue

Class	*EcucParameterValue* (abstract)
Package	M2::AUTOSARTemplates::ECUCDescriptionTemplate
Note	Common class to all types of configuration values.
Base	*ARObject*, *EcucIndexableValue*
Subclasses	EcucAddInfoParamValue, EcucNumericalParamValue, EcucTextualParamValue

▽

△

Class	***EcucParameterValue*** (abstract)			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
annotation	Annotation	*	aggr	Possibility to provide additional notes while defining the ECU Configuration Parameter Values. These are not intended as documentation but are mere design notes. **Tags:**xml.sequenceOffset=10
definition	EcucParameterDef	0..1	ref	Reference to the definition of this EcucParameterValue subclasses in the ECU Configuration Parameter Definition. **Stereotypes:** atpIdentityContributor **Tags:**xml.sequenceOffset=-10
isAutoValue	Boolean	0..1	attr	If withAuto is set to "true" for this parameter definition the isAutoValue can be set to "true". If isAutoValue is set to "true" the actual value will not be considered during ECU Configuration but will be (re-)calculated by the code generator and stored in the value attribute afterwards. These implicit updated values might require a re-generation of other modules which reference these values. If isAutoValue is not present the default is "false". **Tags:**xml.sequenceOffset=20

Table E.31: EcucParameterValue

Class	**EcucReferenceDef**			
Package	M2::AUTOSARTemplates::ECUCParameterDefTemplate			
Note	Specify references within the ECU Configuration Description between parameter containers.			
Base	*ARObject*, *AtpDefinition*, *EcucAbstractInternalReferenceDef*, *EcucAbstractReferenceDef*, *EcucCommonAttributes*, *EcucDefinitionElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
destination	EcucContainerDef	0..1	ref	Exactly one reference to a parameter container is allowed as destination. **Stereotypes:** atpUriDef

Table E.32: EcucReferenceDef

Class	**EcucReferenceValue**			
Package	M2::AUTOSARTemplates::ECUCDescriptionTemplate			
Note	Used to represent a configuration value that has a parameter definition of type EcucAbstractReference Def (used for all of its specializations excluding EcucInstanceReferenceDef).			
Base	*ARObject*, *EcucAbstractReferenceValue*, *EcucIndexableValue*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
value	Referrable	0..1	ref	Specifies the destination of the reference.

Table E.33: EcucReferenceValue

Class	**EcucTextualParamValue**
Package	M2::AUTOSARTemplates::ECUCDescriptionTemplate
Note	Holding a value which is not subject to variation.
Base	*ARObject*, *EcucIndexableValue*, *EcucParameterValue*

▽

△

Class	EcucTextualParamValue			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
value	VerbatimString	0..1	attr	Value of the parameter, not subject to variant handling.

Table E.34: EcucTextualParamValue

Class	EnumerationMappingEntry			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling::AttributeValueVariationPoints			
Note	This class specifies the entry elements of the enumeration mapping table. Note that this class might be used in the extended meta-model only.			
Base	*ARObject*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
enumerator Value	NameToken	1	attr	This attribute specifies the symbolic value (e.g. in, out) of the enumeration entry. **Tags:**xml.sequenceOffset=20
numericalValue	PositiveInteger	1	attr	This attribute specifies the numerical value (e.g. 0, 1) of the enumeration entry. The numericalValue marks an index on M2 level. It is not used in C-Code or at runtime. The numericalValue is only given to be able to calculate a value that represents the enumerator literal in a numerical expression. **Tags:**xml.sequenceOffset=10

Table E.35: EnumerationMappingEntry

Class	EnumerationMappingTable			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling::AttributeValueVariationPoints			
Note	This class represents an attribute value variation point for Enumeration attributes. Note that this class might be used in the extended meta-model only. **Tags:**atp.recommendedPackage=EnumerationMappingTables			
Base	*ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
entry	EnumerationMapping Entry	*	aggr	Key-value pair mapping enumeration values to unique integers. **Tags:** xml.roleElement=true xml.roleWrapperElement=true xml.typeElement=false xml.typeWrapperElement=false

Table E.36: EnumerationMappingTable

Class	FMFeature
Package	M2::AUTOSARTemplates::FeatureModelTemplate
Note	A FMFeature describes an essential characteristic of a product. Each FMFeature is contained in exactly one FMFeatureModel. **Tags:**atp.recommendedPackage=FMFeatureModels
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*

▽

△

Class	**FMFeature**			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
attributeDef	FMAttributeDef	*	aggr	This defines the attributes of the given feature.
decomposition	FMFeature Decomposition	*	aggr	Lists the sub-features of a feature.
maximum IntendedBinding Time	BindingTimeEnum	0..1	attr	Defines an upper bound for the binding time of the variation points that are associated with the FMFeature. This attribute is meant as a hint for the development process.
minimum IntendedBinding Time	BindingTimeEnum	0..1	attr	Defines a lower bound for the binding time of the variation points that are associated with the FMFeature. This attribute is meant as a hint for the development process.
relation	FMFeatureRelation	*	aggr	Defines relations for FMFeatures, for example dependencies on other FMFeatures, or conflicts with other FMFeatures. A FMFeature can only be part of a FMFeatureSelectionSet if all its relations are fulfilled.
restriction	FMFeatureRestriction	*	aggr	Defines restrictions for FMFeatures. A FMFeature can only be part of a FMFeatureSelectionSet if at least one of its restrictions evaluates to true.

Table E.37: FMFeature

Class	**FMFeatureModel**			
Package	M2::AUTOSARTemplates::FeatureModelTemplate			
Note	A Feature model describes the features of a product line and their dependencies. Feature models are an optional part of an AUTOSAR model. **Tags:**atp.recommendedPackage=FMFeatureModels			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
feature	FMFeature	*	ref	"feature" holds the list of features of the feature model. No FMFeature may be contained twice in this list. Also, each FMFeature may be contained on only one feature model. **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=feature
root	FMFeature	0..1	ref	The features of a feature model define a tree. The attribute root points to the root of this tree.

Table E.38: FMFeatureModel

Class	***FibexElement*** (abstract)			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::FibexCore			
Note	ASAM FIBEX elements specifying Communication and Topology.			
Base	*ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Subclasses	*BusMirrorChannelMapping*, *CommunicationCluster*, ConsumedProvidedServiceInstanceGroup, CouplingElement, EcuInstance, EthernetWakeupSleepOnDatalineConfigSet, *Frame*, Gateway, Global TimeDomain, ISignal, ISignalGroup, ISignalIPduGroup, NmConfig, *Pdu*, PdurIPduGroup, Secure CommunicationPropsSet, ServiceInstanceCollectionSet, SoAdRoutingGroup, SocketConnectionIpdu IdentifierSet, *TpConfig*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table E.39: FibexElement

Class	**FlatMap**			
Package	M2::AUTOSARTemplates::CommonStructure::FlatMap			
Note	Contains a flat list of references to software objects. This list is used to identify instances and to resolve name conflicts. The scope is given by the RootSwCompositionPrototype for which it is used, i.e. it can be applied to a system, system extract or ECU-extract. An instance of FlatMap may also be used in a preliminary context, e.g. in the scope of a software component before integration into a system. In this case it is not referred by a RootSwComposition Prototype. **Tags:**atp.recommendedPackage=FlatMaps			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
instance	FlatInstanceDescriptor	1..*	aggr	A descriptor instance aggregated in the flat map. The variation point accounts for the fact, that the system in scope can be subject to variability, and thus the existence of some instances is variable. The aggregation has been made splitable because the content might be contributed by different stakeholders at different times in the workflow. Plus, the overall size might be so big that eventually it becomes more manageable if it is distributed over several files. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=instance.shortName, instance.variationPoint.shortLabel vh.latestBindingTime=postBuild

Table E.40: FlatMap

Class	<<atpVariation>> **FlexrayCommunicationController**			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::Fibex4Flexray::FlexrayTopology			
Note	FlexRay bus specific communication port attributes.			
Base	*ARObject*, *CommunicationController*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
accepted StartupRange	Integer	1	attr	Expanded range of measured clock deviation allowed for startup frames during integration. Unit:microtick
allowHaltDueTo Clock	Boolean	1	attr	Boolean flag that controls the transition to the POC:halt state due to a clock synchronization errors. If set to true, the Communication Controller is allowed to transition to POC:halt. If set to false, the Communication Controller will not transition to the POC:halt state but will enter or remain in the normal POC (passive State).
allowPassiveTo Active	Integer	0..1	attr	Number of consecutive even/odd cycle pairs that shall have valid clock correction terms before the Communication Controller will be allowed to transition from the POC:normal passive state to POC:normal active state. If set to 0, the Communication Controller is not allowed to transition from POC:norm
clusterDrift Damping	Integer	1	attr	The cluster drift damping factor used in clock synchronization rate correction in microticks
decoding Correction	Integer	1	attr	Value used by the receiver to calculate the difference between primary time reference point and secondary time reference point. Unit: Microticks (pDecodingCorrection)

△

Class	<<atpVariation>> **FlexrayCommunicationController**			
delay CompensationA	Integer	0..1	attr	Value used to compensate for reception delays on channel A Unit: Microticks. This optional parameter shall only be filled out if channel A is used.
delay CompensationB	Integer	0..1	attr	Value used to compensate for reception delays on channel B. Unit: Microticks. This optional parameter shall only be filled out if channel B is used.
externalSync	Boolean	0..1	attr	Flag indicating whether the node is externally synchronized (operating as Time Gateway Sink in an TT-E Time Triggered External Sync cluster) or locally synchronized.
externOffset Correction	Integer	0..1	attr	Fixed amount added or subtracted to the calculated offset correction term to facilitate external offset correction, expressed in node-local microticks.
externRate Correction	Integer	0..1	attr	Fixed amount added or subtracted to the calculated rate correction term to facilitate external rate correction, expressed in node-local microticks.
fallBackInternal	Boolean	0..1	attr	Flag indicating whether a Time Gateway Sink node will switch to local clock operation when synchronization with the Time Gateway Source node is lost (pFallBackInternal = true) or will instead go to POC:ready (pFallBackInternal = false).
flexrayFifo	FlexrayFifo Configuration	*	aggr	One First In First Out (FIFO) queued receive structure, defining the admittance criteria to the FIFO.
keySlotID	PositiveInteger	0..1	attr	ID of the slot used to transmit the startup frame, sync frame, or designated single slot frame. If the attributes keySlotUsedForStartUp, keySlotUsedForSync, or keySlot OnlyEnabled are set to true the key slot value is mandatory.
keySlotOnly Enabled	Boolean	1	attr	Flag indicating whether or not the node shall enter key slot only mode following startup.
keySlotUsedFor StartUp	Boolean	1	attr	Flag indicating whether the Key Slot is used to transmit a startup frame.
keySlotUsedFor Sync	Boolean	1	attr	Flag indicating whether the Key Slot is used to transmit a sync frame.
latestTX	Integer	1	attr	The number of the last minislot in which a transmission can start in the dynamic segment for the respective node
listenTimeout	Integer	1	attr	Value for the startup listen timeout and wakeup listen timeout. Although this is a node local parameter, the real time equivalent of this value should be the same for all nodes in the cluster. Unit: Microticks
macroInitial OffsetA	Integer	0..1	attr	Integer number of macroticks between the static slot boundary and the closest macrotick boundary of the secondary time reference point based on the nominal macrotick duration. (pMacroInitialOffset). This optional parameter shall only be filled out if channel A is used.
macroInitial OffsetB	Integer	0..1	attr	Integer number of macroticks between the static slot boundary and the closest macrotick boundary of the secondary time reference point based on the nominal macrotick duration. (pMacroInitialOffset). This optional parameter shall only be filled out if channel B is used.
maximum Dynamic PayloadLength	Integer	1	attr	Maximum payload length for the dynamic channel of a frame in 16 bit WORDS.

▽

△

Class	<<atpVariation>> **FlexrayCommunicationController**			
microInitial OffsetA	Integer	0..1	attr	Number of microticks between the closest macrotick boundary described by gMacroInitialOffset and the secondary time reference point. The parameter depends on pDelayCompensationA and therefore it has to be set independently for each channel. This optional parameter shall only be filled out if channel A is used.
microInitial OffsetB	Integer	0..1	attr	Number of microticks between the closest macrotick boundary described by gMacroInitialOffset and the secondary time reference point. The parameter depends on pDelayCompensationB and therefore it has to be set independently for each channel. This optional parameter shall only be filled out if channel B is used.
microPerCycle	Integer	1	attr	The nominal number of microticks in a communication cycle
microtick Duration	TimeValue	0..1	attr	Duration of a microtick. This attribute can be derived from samplePerMicrotick and gdSampleClockPeriod. Unit: seconds
nmVectorEarly Update	Boolean	0..1	attr	Flag indicating when the update of the Network Management Vector in the CHI shall take place. If set to false, the update shall take place after the NIT. If set to true, the update shall take place after the end of the static segment.
offsetCorrection Out	Integer	1	attr	Magnitude of the maximum permissible offset correction value. Unit:microtick (pOffsetCorrectionOut)
rateCorrection Out	Integer	1	attr	Magnitude of the maximum permissible rate correction value and the maximum drift offset between two nodes operating with unsynchronized clocks for one communication cycle. Unit:Microticks (pRateCorrection Out) Remarks: This parameter maps to FlexRay Protocol 2.1 Rev. A parameter pdMaxDrift.
samplesPer Microtick	Integer	0..1	attr	Number of samples per microtick
secondKeySlot Id	PositiveInteger	0..1	attr	ID of the second Key slot, in which a second startup frame shall be sent in TT-L Time Triggered Local Master Sync or TT-E Time Triggered External Sync mode. If this parameter is set to zero the node does not have a second key slot.
twoKeySlot Mode	Boolean	0..1	attr	Flag indicating whether node operates as a startup node in a TT-E Time Triggered External Sync or TT-L Time Triggered Local Master Sync cluster.
wakeUpPattern	Integer	1	attr	Number of repetitions of the Tx-wakeup symbol to be sent during the CC_WakeupSend state of this Node in the cluster

Table E.41: FlexrayCommunicationController

Primitive	**Float**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	An instance of Float is an element from the set of real numbers. **Tags:** xml.xsd.customType=FLOAT xml.xsd.type=double

Table E.42: Float

Class	<<atpMixedString>> ***FormulaExpression*** (abstract)			
Package	M2::AUTOSARTemplates::GenericStructure::FormulaLanguage			
Note	This class represents the syntax of the formula language. The class is modeled as an abstract class in order to be specialized into particular use cases. For each use case the referable objects might be specified in the specialization.			
Base	*ARObject*			
Subclasses	CompuGenericMath, EcucConditionFormula, EcucParameterDerivationFormula, *FMFormulaByFeatures AndAttributes*, *SwSystemconstDependentFormula*, TDEventOccurrenceExpressionFormula, Timing ConditionFormula			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
atpReference	Referrable	*	ref	The referable object shall yield a numerical / boolean value. **Stereotypes:** atpAbstract
atpString Reference	Referrable	*	ref	The referable object shall yield a string value. **Stereotypes:** atpAbstract

Table E.43: FormulaExpression

Class	**GenericModelReference**			
Package	M2::AUTOSARTemplates::GenericStructure::BuildActionManifest			
Note	This meta-class represents the ability to express a late binding reference to a model element. The model element can be from every model. Even if it is modeled according to the association representation, it is not limited to refer to AUTOSAR model elements.			
Base	*ARObject*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
base	NameToken	1	attr	This establishes the reference base. **Tags:**xml.attribute=true
dest	NameToken	1	attr	This attribute represents the class of the referenced model element. It is a String, since the model element can be in any model. Therefore we cannot have any assumption here. **Tags:**xml.attribute=true
ref	Ref	1	attr	This is the full qualified name of the model element **Tags:** xml.roleElement=true xml.roleWrapperElement=false xml.typeElement=false xml.typeWrapperElement=false

Table E.44: GenericModelReference

Class	***Implementation*** (abstract)			
Package	M2::AUTOSARTemplates::CommonStructure::Implementation			
Note	Description of an implementation a single software component or module.			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Subclasses	BswImplementation, SwcImplementation			
Attribute	**Type**	**Mult.**	**Kind**	**Note**

▽

△

Class	***Implementation*** (abstract)			
buildAction Manifest	BuildActionManifest	0..1	ref	A manifest specifying the intended build actions for the software delivered with this implementation. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=codeGenerationTime
codeDescriptor	Code	*	aggr	Specifies the provided implementation code.
compiler	Compiler	*	aggr	Specifies the compiler for which this implementation has been released
generated Artifact	DependencyOnArtifact	*	aggr	Relates to an artifact that will be generated during the integration of this Implementation by an associated generator tool. Note that this is an optional information since it might not always be in the scope of a single module or component to provide this information. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime
hwElement	HwElement	*	ref	The hardware elements (e.g. the processor) required for this implementation.
linker	Linker	*	aggr	Specifies the linker for which this implementation has been released.
mcSupport	McSupportData	0..1	aggr	The measurement & calibration support data belonging to this implementation. The aggregtion is <<atpSplitable>> because in case of an already exisiting BSW Implementation model, this description will be added later in the process, namely at code generation time. **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=mcSupport
programming Language	Programminglanguage Enum	0..1	attr	Programming language the implementation was created in.
requiredArtifact	DependencyOnArtifact	*	aggr	Specifies that this Implementation depends on the existance of another artifact (e.g. a library). This aggregation of DependencyOnArtifact is subject to variability with the purpose to support variability in the implementations. Different algorithms in the implementation might cause different dependencies, e.g. the number of used libraries. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime
required GeneratorTool	DependencyOnArtifact	*	aggr	Relates this Implementation to a generator tool in order to generate additional artifacts during integration. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime
resource Consumption	ResourceConsumption	0..1	aggr	All static and dynamic resources for each implementation are described within the ResourceConsumption class. **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=resourceConsumption.shortName
swcBsw Mapping	SwcBswMapping	0..1	ref	This allows a mapping between an SWC and a BSW behavior to be attached to an implementation description (for AUTOSAR Service, ECU Abstraction and Complex Driver Components). It is up to the methodology to define whether this reference has to be set for the Swc- or Bsw Implementtion or for both.
swVersion	RevisionLabelString	0..1	attr	Software version of this implementation. The numbering contains three levels (like major, minor, patch), its values are vendor specific.
usedCode Generator	String	0..1	attr	Optional: code generator used.

▽

△

Class	***Implementation*** (abstract)			
vendorId	PositiveInteger	0..1	attr	Vendor ID of this Implementation according to the AUTOSAR vendor list

Table E.45: Implementation

Class	**ImplementationDataType**			
Package	M2::AUTOSARTemplates::CommonStructure::ImplementationDataTypes			
Note	Describes a reusable data type on the implementation level. This will typically correspond to a typedef in C-code. **Tags:**atp.recommendedPackage=ImplementationDataTypes			
Base	*ARElement*, *ARObject*, *AbstractImplementationDataType*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *AutosarDataType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
dynamicArraySizeProfile	String	0..1	attr	Specifies the profile which the array will follow in case this data type is a variable size array.
isStructWithOptionalElement	Boolean	0..1	attr	This attribute is only valid if the attribute category is set to STRUCTURE. If set to True, this attribute indicates that the ImplementationDataType has been created with the intention to define at least one element of the structure as optional.
subElement (ordered)	ImplementationDataTypeElement	*	aggr	Specifies an element of an array, struct, or union data type. The aggregation of ImplementionDataTypeElement is subject to variability with the purpose to support the conditional existence of elements inside a Implementation DataType representing a structure. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime
symbolProps	SymbolProps	0..1	aggr	This represents the SymbolProps for the Implementation DataType. **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=symbolProps.shortName
typeEmitter	NameToken	0..1	attr	This attribute is used to control which part of the AUTOSAR toolchain is supposed to trigger data type definitions.

Table E.46: ImplementationDataType

Class	**ImplementationDataTypeElement**
Package	M2::AUTOSARTemplates::CommonStructure::ImplementationDataTypes
Note	Declares a data object which is locally aggregated. Such an element can only be used within the scope where it is aggregated. This element either consists of further subElements or it is further defined via its swDataDefProps. There are several use cases within the system of ImplementationDataTypes fur such a local declaration: • It can represent the elements of an array, defining the element type and array size • It can represent an element of a struct, defining its type • It can be the local declaration of a debug element.

▽

△

Class	**ImplementationDataTypeElement**			
Base	*ARObject*, *AbstractImplementationDataTypeElement*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
arrayImplPolicy	ArrayImplPolicyEnum	0..1	attr	This attribute controls the implementation of the payload of an array. It shall only be used if the enclosing ImplementationDataType constitutes an array.
arraySize	PositiveInteger	0..1	attr	The existence of this attributes (if bigger than 0) defines the size of an array and declares that this Implementation DataTypeElement represents the type of each single array element. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime
arraySize Handling	ArraySizeHandling Enum	0..1	attr	The way how the size of the array is handled in case of a variable size array.
arraySize Semantics	ArraySizeSemantics Enum	0..1	attr	This attribute controls the meaning of the value of the array size.
isOptional	Boolean	0..1	attr	This attribute represents the ability to declare the enclosing ImplementationDataTypeElement as optional. This means that, at runtime, the ImplementationDataType Element may or may not have a valid value and shall therefore be ignored. The underlying runtime software provides means to set the CppImplementationDataTypeElement as not valid at the sending end of a communication and determine its validity at the receiving end.
subElement (ordered)	ImplementationData TypeElement	*	aggr	Element of an array, struct, or union in case of a nested declaration (i.e. without using "typedefs"). The aggregation of ImplementionDataTypeElement is subject to variability with the purpose to support the conditional existence of elements inside a Implementation DataType representing a structure. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime
swDataDef Props	SwDataDefProps	0..1	aggr	The properties of this ImplementationDataTypeElement.

Table E.47: ImplementationDataTypeElement

Primitive	**Integer**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	An instance of Integer is an element in the set of integer numbers (..., -2, -1, 0, 1, 2, ...). The value can be expressed in decimal, octal, hexadecimal and binary representation. Negative numbers can only be expressed in decimal notation Range is from -2147483648 and 2147483647. **Tags:** xml.xsd.customType=INTEGER xml.xsd.pattern=0|[\+\-]?[1-9][0-9]*|0[xX][0-9a-fA-F]+|0[bB][0-1]+|0[0-7]+ xml.xsd.type=string

Table E.48: Integer

Class	***InternalBehavior*** (abstract)			
Package	M2::AUTOSARTemplates::CommonStructure::InternalBehavior			
Note	Common base class (abstract) for the internal behavior of both software components and basic software modules/clusters.			
Base	*ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	BswInternalBehavior, SwcInternalBehavior			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
constant Memory	ParameterData Prototype	*	aggr	Describes a read only memory object containing characteristic value(s) implemented by this Internal Behavior. The shortName of ParameterDataPrototype has to be equal to the ”C’ identifier of the described constant. The characteristic value(s) might be shared between Sw ComponentPrototypes of the same SwComponentType. The aggregation of constantMemory is subject to variability with the purpose to support variability in the software component or module implementations. Typically different algorithms in the implementation are requiring different number of memory objects. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=constantMemory.shortName, constant Memory.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
constantValue Mapping	ConstantSpecification MappingSet	*	ref	Reference to the ConstantSpecificationMapping to be applied for the particular InternalBehavior **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=constantValueMapping
dataType Mapping	DataTypeMappingSet	*	ref	Reference to the DataTypeMapping to be applied for the particular InternalBehavior **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=dataTypeMapping
exclusiveArea	ExclusiveArea	*	aggr	This specifies an ExclusiveArea for this InternalBehavior. The exclusiveArea is local to the component resp. module. The aggregation of ExclusiveAreas is subject to variability. Note: the number of ExclusiveAreas might vary due to the conditional existence of RunnableEntities or BswModuleEntities. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=exclusiveArea.shortName, exclusive Area.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
exclusiveArea NestingOrder	ExclusiveAreaNesting Order	*	aggr	This represents the set of ExclusiveAreaNestingOrder owned by the InternalBehavior. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=exclusiveAreaNestingOrder.shortName, exclusiveAreaNestingOrder.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
staticMemory	VariableDataPrototype	*	aggr	Describes a read and writeable static memory object representing measurerment variables implemented by this software component. The term "static" is used in the meaning of "non-temporary" and does not necessarily specify a linker encapsulation. This kind of memory is only supported if supportsMultipleInstantiation is FALSE. ▽

▽

△

Class	InternalBehavior (abstract)			
				△ The shortName of the VariableDataPrototype has to be equal with the ”C’ identifier of the described variable. The aggregation of staticMemory is subject to variability with the purpose to support variability in the software component’s implementations. Typically different algorithms in the implementation are requiring different number of memory objects. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=staticMemory.shortName, staticMemory.variationPoint.shortLabel vh.latestBindingTime=preCompileTime

Table E.49: InternalBehavior

Class	Keyword			
Package	M2::AUTOSARTemplates::CommonStructure::StandardizationTemplate::Keyword			
Note	This meta-class represents the ability to predefine keywords which may subsequently be used to construct names following a given naming convention, e.g. the AUTOSAR naming conventions. Note that such names is not only shortName. It could be symbol, or even longName. Application of keywords is not limited to particular names.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
abbrName	NameToken	1	attr	This attribute specifies an abbreviated name of a keyword. This abbreviation may e.g. be used for constructing valid shortNames according to the AUTOSAR naming conventions. Unlike shortName, it may contain any name token. E.g. it may consist of digits only.
classification	NameToken	*	attr	This attribute allows to attach classification to the Keyword such as MEAN, ACTION, CONDITION, INDEX, PREPOSITION

Table E.50: Keyword

Primitive	Limit			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes			
Note	This class represents the ability to express a numerical limit. Note that this is in fact a NumericalVariation Point but has the additional attribute intervalType. **Tags:** xml.xsd.customType=LIMIT-VALUE xml.xsd.pattern=(0[xX][0-9a-fA-F]+)|(0[0-7]+)|(0[bB][0-1]+)|(([+\-]?[1-9][0-9]+(\.[0-9]+)?|[+\-]?[0-9](\.[0-9]+)?)([eE]([+\-]?)[0-9]+)?)|\.0|INF|-INF|NaN xml.xsd.type=string			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
intervalType	IntervalTypeEnum	0..1	attr	This specifies the type of the interval. If the attribute is missing the interval shall be considered as "CLOSED". **Tags:**xml.attribute=true

Table E.51: Limit

Class	<<atpMixedString>> ***MixedContentForUnitNames*** (abstract)			
Package	M2::MSR::Documentation::TextModel::InlineTextModel			
Note	This is the text model for items with subscript and superscripts such as measurement unit designations. It is intended, that such models can easily be transcribed to a plain text model either by using appropriate characters or by transcribing like m^2.			
Base	*ARObject*			
Subclasses	SingleLanguageUnitNames			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
sub	Superscript	1	attr	This is subscript text. **Tags:**xml.sequenceOffset=40
sup	Superscript	1	attr	This is superscript text. **Tags:**xml.sequenceOffset=30

Table E.52: MixedContentForUnitNames

Class	**MsrQueryP2**			
Package	M2::MSR::Documentation::MsrQuery			
Note	This meta-class represents the ability to express a query which yields the content of a Documentation Block as a result.			
Base	*ARObject*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
msrQueryProps	MsrQueryProps	1	aggr	This is argument and properties of the Documentation Block query. **Tags:**xml.sequenceOffset=20
msrQueryResultP2	DocumentationBlock	0..1	aggr	This represents the result of the query. **Tags:**xml.sequenceOffset=30

Table E.53: MsrQueryP2

Primitive	**NameToken**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	This is an identifier as used in xml, e.g. xml-names. Typical usages are, for example, the names of type emitters, protocols, or profiles. For details see NMTOKEN definition on the W3C website (https://www.w3.org/TR/xml/#NT-Nmtoken). Note: Although NameToken supports a wide range of characters, the actually allowed patterns for a certain attribute typed by NameToken may be further restricted by the specification of that attribute. **Tags:** xml.xsd.customType=NMTOKEN-STRING xml.xsd.type=NMTOKEN

Table E.54: NameToken

Primitive	**Numerical**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes

▽

△

Primitive	**Numerical**
Note	This primitive specifies a numerical value. It can be denoted in different formats such as Decimal, Octal, Hexadecimal, Float. See the xsd pattern for details. The value can be expressed in octal, hexadecimal, binary representation. Negative numbers can only be expressed in decimal or float notation. **Tags:** xml.xsd.customType=NUMERICAL-VALUE xml.xsd.pattern=(0[xX][0-9a-fA-F]+)|(0[0-7]+)|(0[bB][0-1]+)|(([+\-]?[1-9][0-9]+(\.[0-9]+)?|[+\-]?[0-9](\.[0-9]+)?)([eE]([+\-]?)[0-9]+)?)|\.0|INF|-INF|NaN xml.xsd.type=string

Table E.55: Numerical

Class	**ParameterDataPrototype**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::DataPrototypes			
Note	A parameter element used for parameter interface and internal behavior, supporting signal like parameter and characteristic value communication patterns and parameter and characteristic value definition.			
Base	*ARObject*, *AtpFeature*, *AtpPrototype*, *AutosarDataPrototype*, *DataPrototype*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
initValue	ValueSpecification	0..1	aggr	Specifies initial value(s) of the ParameterDataPrototype

Table E.56: ParameterDataPrototype

Class	***PortInterface*** (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	Abstract base class for an interface that is either provided or required by a port of a software component.			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Subclasses	ClientServerInterface, *DataInterface*, ModeSwitchInterface, TriggerInterface			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
isService	Boolean	0..1	attr	This flag is set if the PortInterface is to be used for communication between an • ApplicationSwComponentType or • ServiceProxySwComponentType or • SensorActuatorSwComponentType or • ComplexDeviceDriverSwComponentType • ServiceSwComponentType • EcuAbstractionSwComponentType and a ServiceSwComponentType (namely an AUTOSAR Service) located on the same ECU. Otherwise the flag is not set.
serviceKind	ServiceProviderEnum	0..1	attr	This attribute provides further details about the nature of the applied service.

Table E.57: PortInterface

Class	**PortPrototype** (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	Base class for the ports of an AUTOSAR software component. The aggregation of PortPrototypes is subject to variability with the purpose to support the conditional existence of ports.			
Base	*ARObject*, *AtpBlueprintable*, *AtpFeature*, *AtpPrototype*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	*AbstractProvidedPortPrototype*, *AbstractRequiredPortPrototype*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
clientServer Annotation	ClientServerAnnotation	*	aggr	Annotation of this PortPrototype with respect to client/ server communication.
delegatedPort Annotation	DelegatedPort Annotation	0..1	aggr	Annotations on this delegated port.
ioHwAbstraction Server Annotation	IoHwAbstractionServer Annotation	*	aggr	Annotations on this IO Hardware Abstraction port.
logAndTrace Message CollectionSet	LogAndTraceMessage CollectionSet	0..1	ref	Reference to a collection of Log or Trace messages that will be used by the application. **Tags:**atp.Status=draft
modePort Annotation	ModePortAnnotation	*	aggr	Annotations on this mode port.
nvDataPort Annotation	NvDataPortAnnotation	*	aggr	Annotations on this non voilatile data port.
parameterPort Annotation	ParameterPort Annotation	*	aggr	Annotations on this parameter port.
senderReceiver Annotation	SenderReceiver Annotation	*	aggr	Collection of annotations of this ports sender/receiver communication.
triggerPort Annotation	TriggerPortAnnotation	*	aggr	Annotations on this trigger port.

Table E.58: PortPrototype

Class	**PortPrototypeBlueprint**			
Package	M2::AUTOSARTemplates::CommonStructure::StandardizationTemplate::BlueprintDedicated::Port ProtoypeBlueprint			
Note	This meta-class represents the ability to express a blueprint of a PortPrototype by referring to a particular PortInterface. This blueprint can then be used as a guidance to create particular PortPrototypes which are defined according to this blueprint. By this it is possible to standardize application interfaces without the need to also standardize software-components with PortPrototypes typed by the standardized Port Interfaces. **Tags:**atp.recommendedPackage=PortPrototypeBlueprints			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Collectable Element*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
initValue	PortPrototypeBlueprint InitValue	*	aggr	This specifies the init values for the dataElements in the particular PortPrototypeBlueprint.
interface	PortInterface	1	ref	This is the interface for which the blueprint is defined. It may be a blueprint itself or a standardized PortInterface
providedCom Spec	PPortComSpec	*	aggr	Provided communication attributes per interface element (data element or operation).
requiredCom Spec	RPortComSpec	*	aggr	Required communication attributes, one for each interface element.

Table E.59: PortPrototypeBlueprint

Primitive	**PositiveInteger**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	This is a positive integer which can be denoted in decimal, binary, octal and hexadecimal. The value is between 0 and 4294967295. **Tags:** xml.xsd.customType=POSITIVE-INTEGER xml.xsd.pattern=0|[\+]?[1-9][0-9]*|0[xX][0-9a-fA-F]+|0[bB][0-1]+|0[0-7]+ xml.xsd.type=string

Table E.60: PositiveInteger

Class	**RPortInCompositionInstanceRef**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Composition::InstanceRefs			
Note				
Base	*ARObject*, *AtpInstanceRef*, *PortInCompositionTypeInstanceRef*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
context Component	SwComponent Prototype	0..1	ref	**Tags:**xml.sequenceOffset=20
targetRPort	AbstractRequiredPort Prototype	0..1	ref	**Tags:**xml.sequenceOffset=30

Table E.61: RPortInCompositionInstanceRef

Primitive	**Ref**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes			
Note	This primitive denotes a name based reference. For detailed syntax see the xsd.pattern. • first slash (relative or absolute reference) [optional] • Identifier [required] • a sequence of slashes and Identifiers [optional] This primitive is used by the meta-model tools to create the references. **Tags:** xml.xsd.customType=REF xml.xsd.pattern=/?[a-zA-Z][a-zA-Z0-9_]{0,127}(/[a-zA-Z][a-zA-Z0-9_]{0,127})* xml.xsd.type=string			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
base	Identifier	0..1	attr	This attribute reflects the base to be used for this reference. **Tags:**xml.attribute=true
blueprintValue	String	0..1	attr	This represents a description that documents how the value shall be defined when deriving objects from the blueprint. **Tags:** atp.Status=draft xml.attribute=true
index	PositiveInteger	0..1	attr	This attribute supports the use case to point on specific elements in an array. This is in particular required if arrays are used to implement particular data objects. **Tags:**xml.attribute=true

Table E.62: Ref

Class	**RunnableEntity**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior			
Note	A RunnableEntity represents the smallest code-fragment that is provided by an AtomicSwComponent Type and are executed under control of the RTE. RunnableEntities are for instance set up to respond to data reception or operation invocation on a server.			
Base	*ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *ExecutableEntity*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
argument (ordered)	RunnableEntity Argument	*	aggr	This represents the formal definition of a an argument to a RunnableEntity.
asynchronous ServerCall ResultPoint	AsynchronousServer CallResultPoint	*	aggr	The server call result point admits a runnable to fetch the result of an asynchronous server call. The aggregation of AsynchronousServerCallResultPoint is subject to variability with the purpose to support the conditional existence of client server PortPrototypes and the variant existence of server call result points in the implementation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=asynchronousServerCallResultPoint.shortName, asynchronousServerCallResultPoint.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
canBeInvoked Concurrently	Boolean	0..1	attr	If the value of this attribute is set to "true" the enclosing RunnableEntity can be invoked concurrently (even for one instance of the corresponding AtomicSwComponent Type). This implies that it is the responsibility of the implementation of the RunnableEntity to take care of this form of concurrency.
dataRead Access	VariableAccess	*	aggr	RunnableEntity has implicit read access to dataElement of a sender-receiver PortPrototype or nv data of a nv data PortPrototype. The aggregation of dataReadAccess is subject to variability with the purpose to support the conditional existence of sender receiver ports or the variant existence of dataReadAccess in the implementation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=dataReadAccess.shortName, dataReadAccess.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
dataReceive PointBy Argument	VariableAccess	*	aggr	RunnableEntity has explicit read access to dataElement of a sender-receiver PortPrototype or nv data of a nv data PortPrototype. The result is passed back to the application by means of an argument in the function signature. The aggregation of dataReceivePointByArgument is subject to variability with the purpose to support the conditional existence of sender receiver PortPrototype or the variant existence of data receive points in the implementation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=dataReceivePointByArgument.shortName, dataReceivePointByArgument.variationPoint.shortLabel vh.latestBindingTime=preCompileTime

△

Class	**RunnableEntity**			
dataReceive PointByValue	VariableAccess	*	aggr	RunnableEntity has explicit read access to dataElement of a sender-receiver PortPrototype or nv data of a nv data PortPrototype. The result is passed back to the application by means of the return value. The aggregation of dataReceivePointBy Value is subject to variability with the purpose to support the conditional existence of sender receiver ports or the variant existence of data receive points in the implementation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=dataReceivePointByValue.shortName, data ReceivePointByValue.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
dataSendPoint	VariableAccess	*	aggr	RunnableEntity has explicit write access to dataElement of a sender-receiver PortPrototype or nv data of a nv data PortPrototype. The aggregation of dataSendPoint is subject to variability with the purpose to support the conditional existence of sender receiver PortPrototype or the variant existence of data send points in the implementation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=dataSendPoint.shortName, dataSend Point.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
dataWrite Access	VariableAccess	*	aggr	RunnableEntity has implicit write access to dataElement of a sender-receiver PortPrototype or nv data of a nv data PortPrototype. The aggregation of dataWriteAccess is subject to variability with the purpose to support the conditional existence of sender receiver ports or the variant existence of dataWriteAccess in the implementation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=dataWriteAccess.shortName, dataWrite Access.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
external TriggeringPoint	ExternalTriggeringPoint	*	aggr	The aggregation of ExternalTriggeringPoint is subject to variability with the purpose to support the conditional existence of trigger ports or the variant existence of external triggering points in the implementation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=externalTriggeringPoint.ident.shortName, externalTriggeringPoint.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
internal TriggeringPoint	InternalTriggeringPoint	*	aggr	The aggregation of InternalTriggeringPoint is subject to variability with the purpose to support the variant existence of internal triggering points in the implementation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=internalTriggeringPoint.shortName, internal TriggeringPoint.variationPoint.shortLabel vh.latestBindingTime=preCompileTime

▽

△

Class	**RunnableEntity**			
modeAccess Point	ModeAccessPoint	*	aggr	The runnable has a mode access point. The aggregation of ModeAccessPoint is subject to variability with the purpose to support the conditional existence of mode ports or the variant existence of mode access points in the implementation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=modeAccessPoint.ident.shortName, mode AccessPoint.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
modeSwitch Point	ModeSwitchPoint	*	aggr	The runnable has a mode switch point. The aggregation of ModeSwitchPoint is subject to variability with the purpose to support the conditional existence of mode ports or the variant existence of mode switch points in the implementation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=modeSwitchPoint.shortName, modeSwitch Point.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
parameter Access	ParameterAccess	*	aggr	The presence of a ParameterAccess implies that a RunnableEntity needs read only access to a Parameter DataPrototype which may either be local or within a Port Prototype. The aggregation of ParameterAccess is subject to variability with the purpose to support the conditional existence of parameter ports and component local parameters as well as the variant existence of Parameter Access (points) in the implementation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=parameterAccess.shortName, parameter Access.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
readLocal Variable	VariableAccess	*	aggr	The presence of a readLocalVariable implies that a RunnableEntity needs read access to a VariableData Prototype in the role of implicitInterRunnableVariable or explicitInterRunnableVariable. The aggregation of readLocalVariable is subject to variability with the purpose to support the conditional existence of implicitInterRunnableVariable and explicit InterRunnableVariable or the variant existence of read LocalVariable (points) in the implementation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=readLocalVariable.shortName, readLocal Variable.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
serverCallPoint	ServerCallPoint	*	aggr	The RunnableEntity has a ServerCallPoint. The aggregation of ServerCallPoint is subject to variability with the purpose to support the conditional existence of client server PortPrototypes or the variant existence of server call points in the implementation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=serverCallPoint.shortName, serverCall Point.variationPoint.shortLabel vh.latestBindingTime=preCompileTime

▽

△

Class	RunnableEntity			
symbol	CIdentifier	0..1	attr	The symbol describing this RunnableEntity's entry point. This is considered the API of the RunnableEntity and is required during the RTE contract phase.
waitPoint	WaitPoint	*	aggr	The WaitPoint associated with the RunnableEntity.
writtenLocal Variable	VariableAccess	*	aggr	The presence of a writtenLocalVariable implies that a RunnableEntity needs write access to a VariableData Prototype in the role of implicitInterRunnableVariable or explicitInterRunnableVariable. The aggregation of writtenLocalVariable is subject to variability with the purpose to support the conditional existence of implicitInterRunnableVariable and explicit InterRunnableVariable or the variant existence of written LocalVariable (points) in the implementation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=writtenLocalVariable.shortName, written LocalVariable.variationPoint.shortLabel vh.latestBindingTime=preCompileTime

Table E.63: RunnableEntity

Class	<<atpMixedString>> **SlOverviewParagraph**			
Package	M2::MSR::Documentation::TextModel::SingleLanguageData			
Note	MixedContentForOverviewParagraph in one particular language. The language is defined by the context. The attribute l is there only for backwards compatibility and shall be ignored.			
Base	*ARObject*, *MixedContentForOverviewParagraph*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table E.64: SlOverviewParagraph

Class	<<atpMixedString>> **SlParagraph**			
Package	M2::MSR::Documentation::TextModel::SingleLanguageData			
Note	This is the text for a paragraph in one particular language. The language is defined by the context. The attribute l is there only for backwards compatibility and shall be ignored.			
Base	*ARObject*, *MixedContentForParagraph*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table E.65: SlParagraph

Class	SwComponentPrototype			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Composition			
Note	Role of a software component within a composition.			
Base	*ARObject*, *AtpFeature*, *AtpPrototype*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
type	SwComponentType	0..1	tref	Type of the instance. **Stereotypes:** isOfType

Table E.66: SwComponentPrototype

Class	***SwComponentType*** (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	Base class for AUTOSAR software components.			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Subclasses	*AtomicSwComponentType*, CompositionSwComponentType, ParameterSwComponentType			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
consistency Needs	ConsistencyNeeds	*	aggr	This represents the collection of ConsistencyNeeds owned by the enclosing SwComponentType. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=consistencyNeeds.shortName, consistency Needs.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
port	PortPrototype	*	aggr	The PortPrototypes through which this SwComponent Type can communicate. The aggregation of PortPrototype is subject to variability with the purpose to support the conditional existence of PortPrototypes. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=port.shortName, port.variationPoint.short Label vh.latestBindingTime=preCompileTime
portGroup	PortGroup	*	aggr	A port group being part of this component. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime
swcMapping Constraint	SwComponentMapping Constraints	*	ref	Reference to constraints that are valid for this Sw ComponentType.
swComponent Documentation	SwComponent Documentation	0..1	aggr	This adds a documentation to the SwComponentType. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=swComponentDocumentation, sw ComponentDocumentation.variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=-10
unitGroup	UnitGroup	*	ref	This allows for the specification of which UnitGroups are relevant in the context of referencing SwComponentType.

Table E.67: SwComponentType

Class	<<atpVariation>> **SwDataDefProps**
Package	M2::MSR::DataDictionary::DataDefProperties
Note	This class is a collection of properties relevant for data objects under various aspects. One could consider this class as a "pattern of inheritance by aggregation". The properties can be applied to all objects of all classes in which SwDataDefProps is aggregated. Note that not all of the attributes or associated elements are useful all of the time. Hence, the process definition (e.g. expressed with an OCL or a Document Control Instance MSR-DCI) has the task of implementing limitations. SwDataDefProps covers various aspects: • Structure of the data element for calibration use cases: is it a single value, a curve, or a map, but also the recordLayouts which specify how such elements are mapped/converted to the Data

△

Class	<<atpVariation>> **SwDataDefProps**			
	△ Types in the programming language (or in AUTOSAR). This is mainly expressed by properties like swRecordLayout and swCalprmAxisSet • Implementation aspects, mainly expressed by swImplPolicy, swVariableAccessImplPolicy, sw AddrMethod, swPointerTagetProps, baseType, implementationDataType and additionalNative TypeQualifier • Access policy for the MCD system, mainly expressed by swCalibrationAccess • Semantics of the data element, mainly expressed by compuMethod and/or unit, dataConstr, invalidValue • Code generation policy provided by swRecordLayout **Tags:**vh.latestBindingTime=codeGenerationTime			
Base	*ARObject*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
additionalNative TypeQualifier	NativeDeclarationString	0..1	attr	This attribute is used to declare native qualifiers of the programming language which can neither be deduced from the baseType (e.g. because the data object describes a pointer) nor from other more abstract attributes. Examples are qualifiers like "volatile", "strict" or "enum" of the C-language. All such declarations have to be put into one string. **Tags:**xml.sequenceOffset=235
annotation	Annotation	*	aggr	This aggregation allows to add annotations (yellow pads ...) related to the current data object. **Tags:** xml.roleElement=true xml.roleWrapperElement=true xml.sequenceOffset=20 xml.typeElement=false xml.typeWrapperElement=false
baseType	SwBaseType	0..1	ref	Base type associated with the containing data object. **Tags:**xml.sequenceOffset=50
compuMethod	CompuMethod	0..1	ref	Computation method associated with the semantics of this data object. **Tags:**xml.sequenceOffset=180
dataConstr	DataConstr	0..1	ref	Data constraint for this data object. **Tags:**xml.sequenceOffset=190
displayFormat	DisplayFormatString	0..1	attr	This property describes how a number is to be rendered e.g. in documents or in a measurement and calibration system. **Tags:**xml.sequenceOffset=210
display Presentation	DisplayPresentation Enum	0..1	attr	This attribute controls the presentation of the related data for measurement and calibration tools.
implementation DataType	AbstractImplementation DataType	0..1	ref	This association denotes the ImplementationDataType of a data declaration via its aggregated SwDataDefProps. It is used whenever a data declaration is not directly referring to a base type. Especially • redefinition of an ImplementationDataType via a "typedef" to another ImplementationDatatype • the target type of a pointer (see SwPointerTarget Props), if it does not refer to a base type directly ▽

▽

△

Class	<<atpVariation>> **SwDataDefProps**			
				△ • the data type of an array or record element within an ImplementationDataType, if it does not refer to a base type directly • the data type of an SwServiceArg, if it does not refer to a base type directly **Tags:**xml.sequenceOffset=215
invalidValue	ValueSpecification	0..1	aggr	Optional value to express invalidity of the actual data element. **Tags:**xml.sequenceOffset=255
stepSize	Float	0..1	attr	This attribute can be used to define a value which is added to or subtracted from the value of a DataPrototype when using up/down keys while calibrating.
swAddrMethod	SwAddrMethod	0..1	ref	Addressing method related to this data object. Via an association to the same SwAddrMethod it can be specified that several DataPrototypes shall be located in the same memory without already specifying the memory section itself. **Tags:**xml.sequenceOffset=30
swAlignment	AlignmentType	0..1	attr	The attribute describes the intended typical alignment of the DataPrototype. If the attribute is not defined the alignment is determined by the swBaseType size and the memoryAllocationKeywordPolicy of the referenced Sw AddrMethod. **Tags:**xml.sequenceOffset=33
swBit Representation	SwBitRepresentation	0..1	aggr	Description of the binary representation in case of a bit variable. **Tags:**xml.sequenceOffset=60
swCalibration Access	SwCalibrationAccess Enum	0..1	attr	Specifies the read or write access by MCD tools for this data object. **Tags:**xml.sequenceOffset=70
swCalprmAxis Set	SwCalprmAxisSet	0..1	aggr	This specifies the properties of the axes in case of a curve or map etc. This is mainly applicable to calibration parameters. **Tags:**xml.sequenceOffset=90
swComparison Variable	SwVariableRefProxy	*	aggr	Variables used for comparison in an MCD process. **Tags:** xml.sequenceOffset=170 xml.typeElement=false
swData Dependency	SwDataDependency	0..1	aggr	Describes how the value of the data object has to be calculated from the value of another data object (by the MCD system). **Tags:**xml.sequenceOffset=200
swHostVariable	SwVariableRefProxy	0..1	aggr	Contains a reference to a variable which serves as a host-variable for a bit variable. Only applicable to bit objects. **Tags:** xml.sequenceOffset=220 xml.typeElement=false
swImplPolicy	SwImplPolicyEnum	0..1	attr	Implementation policy for this data object. **Tags:**xml.sequenceOffset=230

▽

△

Class	<<atpVariation>> **SwDataDefProps**			
swIntended Resolution	Numerical	0..1	attr	The purpose of this element is to describe the requested quantization of data objects early on in the design process. The resolution ultimately occurs via the conversion formula present (compuMethod), which specifies the transition from the physical world to the standardized world (and vice-versa) (here, "the slope per bit" is present implicitly in the conversion formula). In the case of a development phase without a fixed conversion formula, a pre-specification can occur through swIntendedResolution. The resolution is specified in the physical domain according to the property "unit". **Tags:**xml.sequenceOffset=240
swInterpolation Method	Identifier	0..1	attr	This is a keyword identifying the mathematical method to be applied for interpolation. The keyword needs to be related to the interpolation routine which needs to be invoked. **Tags:**xml.sequenceOffset=250
swIsVirtual	Boolean	0..1	attr	This element distinguishes virtual objects. Virtual objects do not appear in the memory, their derivation is much more dependent on other objects and hence they shall have a swDataDependency . **Tags:**xml.sequenceOffset=260
swPointerTarget Props	SwPointerTargetProps	0..1	aggr	Specifies that the containing data object is a pointer to another data object. **Tags:**xml.sequenceOffset=280
swRecord Layout	SwRecordLayout	0..1	ref	Record layout for this data object. **Tags:**xml.sequenceOffset=290
swRefresh Timing	MultidimensionalTime	0..1	aggr	This element specifies the frequency in which the object involved shall be or is called or calculated. This timing can be collected from the task in which write access processes to the variable run. But this cannot be done by the MCD system. So this attribute can be used in an early phase to express the desired refresh timing and later on to specify the real refresh timing. **Tags:**xml.sequenceOffset=300
swTextProps	SwTextProps	0..1	aggr	the specific properties if the data object is a text object. **Tags:**xml.sequenceOffset=120
swValueBlock Size	Numerical	0..1	attr	This represents the size of a Value Block **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=80

▽

△

Class	<<atpVariation>> **SwDataDefProps**			
swValueBlock SizeMult (ordered)	Numerical	*	attr	This attribute is used to specify the dimensions of a value block (VAL_BLK) for the case that that value block has more than one dimension. The dimensions given in this attribute are ordered such that the first entry represents the first dimension, the second entry represents the second dimension, and so on. For one-dimensional value blocks the attribute swValue BlockSize shall be used and this attribute shall not exist. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime
unit	Unit	0..1	ref	Physical unit associated with the semantics of this data object. This attribute applies if no compuMethod is specified. If both units (this as well as via compuMethod) are specified the units shall be compatible. **Tags:**xml.sequenceOffset=350
valueAxisData Type	ApplicationPrimitive DataType	0..1	ref	The referenced ApplicationPrimitiveDataType represents the primitive data type of the value axis within a compound primitive (e.g. curve, map). It supersedes CompuMethod, Unit, and BaseType. **Tags:**xml.sequenceOffset=355

Table E.68: SwDataDefProps

Class	**SwcInternalBehavior**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior			
Note	The SwcInternalBehavior of an AtomicSwComponentType describes the relevant aspects of the software-component with respect to the RTE, i.e. the RunnableEntities and the RTEEvents they respond to.			
Base	*ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *InternalBehavior*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
arTypedPer Instance Memory	VariableDataPrototype	*	aggr	Defines an AUTOSAR typed memory-block that needs to be available for each instance of the SW-component. This is typically only useful if supportsMultipleInstantiation is set to "true" or if the component defines NVRAM access via permanent blocks. The aggregation of arTypedPerInstanceMemory is subject to variability with the purpose to support variability in the software component's implementations. Typically different algorithms in the implementation are requiring different number of memory objects. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=arTypedPerInstanceMemory.shortName, arTypedPerInstanceMemory.variationPoint.shortLabel vh.latestBindingTime=preCompileTime

▽

△

Class	**SwcInternalBehavior**			
event	RTEEvent	*	aggr	This is a RTEEvent specified for the particular Swc InternalBehavior. The aggregation of RTEEvent is subject to variability with the purpose to support the conditional existence of RTE events. Note: the number of RTE events might vary due to the conditional existence of PortPrototypes using Data ReceivedEvents or due to different scheduling needs of algorithms. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=event.shortName, event.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
exclusiveArea Policy	SwcExclusiveArea Policy	*	aggr	Options how to generate the ExclusiveArea related APIs. When no SwcExclusiveAreaPolicy is specified for an ExclusiveArea the default values apply. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=exclusiveAreaPolicy, exclusiveArea Policy.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
explicitInter Runnable Variable	VariableDataPrototype	*	aggr	Implement state message semantics for establishing communication among runnables of the same component. The aggregation of explicitInterRunnable Variable is subject to variability with the purpose to support variability in the software components implementations. Typically different algorithms in the implementation are requiring different number of memory objects. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=explicitInterRunnableVariable.shortName, explicitInterRunnableVariable.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
handle TerminationAnd Restart	HandleTerminationAnd RestartEnum	0..1	attr	This attribute controls the behavior with respect to stopping and restarting. The corresponding AtomicSw ComponentType may either not support stop and restart, or support only stop, or support both stop and restart.
implicitInter Runnable Variable	VariableDataPrototype	*	aggr	Implement state message semantics for establishing communication among runnables of the same component. The aggregation of implicitInterRunnable Variable is subject to variability with the purpose to support variability in the software components implementations. Typically different algorithms in the implementation are requiring different number of memory objects. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=implicitInterRunnableVariable.shortName, implicitInterRunnableVariable.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
includedData TypeSet	IncludedDataTypeSet	*	aggr	The includedDataTypeSet is used by a software component for its implementation. **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=includedDataTypeSet

▽

△

Class	**SwcInternalBehavior**			
includedMode DeclarationGroupSet	IncludedMode DeclarationGroupSet	*	aggr	This aggregation represents the included Mode DeclarationGroups **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=includedModeDeclarationGroupSet
instantiation DataDefProps	InstantiationDataDef Props	*	aggr	The purpose of this is that within the context of a given SwComponentType some data def properties of individual instantiations can be modified. The aggregation of InstantiationDataDefProps is subject to variability with the purpose to support the conditional existence of Port Prototypes and component local memories like "per InstanceParameter" or "arTypedPerInstanceMemory". **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=instantiationDataDefProps, instantiationData DefProps.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
perInstance Memory	PerInstanceMemory	*	aggr	Defines a per-instance memory object needed by this software component. The aggregation of PerInstance Memory is subject to variability with the purpose to support variability in the software components implementations. Typically different algorithms in the implementation are requiring different number of memory objects. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=perInstanceMemory.shortName, perInstance Memory.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
perInstance Parameter	ParameterData Prototype	*	aggr	Defines parameter(s) or characteristic value(s) that needs to be available for each instance of the software-component. This is typically only useful if supportsMultipleInstantiation is set to "true". The aggregation of perInstanceParameter is subject to variability with the purpose to support variability in the software components implementations. Typically different algorithms in the implementation are requiring different number of memory objects. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=perInstanceParameter.shortName, per InstanceParameter.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
portAPIOption	PortAPIOption	*	aggr	Options for generating the signature of port-related calls from a runnable to the RTE and vice versa. The aggregation of PortPrototypes is subject to variability with the purpose to support the conditional existence of ports. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=portAPIOption, portAPIOption.variation Point.shortLabel vh.latestBindingTime=preCompileTime

▽

△

Class	**SwcInternalBehavior**			
runnable	RunnableEntity	*	aggr	This is a RunnableEntity specified for the particular Swc InternalBehavior. The aggregation of RunnableEntity is subject to variability with the purpose to support the conditional existence of RunnableEntities. Note: the number of RunnableEntities might vary due to the conditional existence of Port Prototypes using DataReceivedEvents or due to different scheduling needs of algorithms. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=runnable.shortName, runnable.variation Point.shortLabel vh.latestBindingTime=preCompileTime
service Dependency	SwcService Dependency	*	aggr	Defines the requirements on AUTOSAR Services for a particular item. The aggregation of SwcServiceDependency is subject to variability with the purpose to support the conditional existence of ports as well as the conditional existence of ServiceNeeds. The SwcServiceDependency owned by an SwcInternal Behavior can be located in a different physical file in order to support that SwcServiceDependency might be provided in later development steps or even by different expert domain (e.g OBD expert for Obd related Service Needs) tools. Therefore the aggregation is <<atp Splitable>>. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=serviceDependency.shortName, service Dependency.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
shared Parameter	ParameterData Prototype	*	aggr	Defines parameter(s) or characteristic value(s) shared between SwComponentPrototypes of the same Sw ComponentType The aggregation of sharedParameter is subject to variability with the purpose to support variability in the software components implementations. Typically different algorithms in the implementation are requiring different number of memory objects. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=sharedParameter.shortName, shared Parameter.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
supports Multiple Instantiation	Boolean	0..1	attr	Indicate whether the corresponding software-component can be multiply instantiated on one ECU. In this case the attribute will result in an appropriate component API on programming language level (with or without instance handle).
variationPoint Proxy	VariationPointProxy	*	aggr	Proxy of a variation points in the C/C++ implementation. **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=variationPointProxy.shortName

Table E.69: SwcInternalBehavior

Class	**System**			
Package	M2::AUTOSARTemplates::SystemTemplate			
Note	The top level element of the System Description. The System description defines five major elements: Topology, Software, Communication, Mapping and Mapping Constraints. The System element directly aggregates the elements describing the Software, Mapping and Mapping Constraints; it contains a reference to an ASAM FIBEX description specifying Communication and Topology. **Tags:**atp.recommendedPackage=Systems			
Base	*ARElement*, *ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
clientId DefinitionSet	ClientIdDefinitionSet	*	ref	Set of Client Identifiers that are used for inter-ECU client-server communication in the System.
containerIPdu HeaderByte Order	ByteOrderEnum	0..1	attr	Defines the byteOrder of the header in ContainerIPdus.
ecuExtract Version	RevisionLabelString	0..1	attr	Version number of the Ecu Extract.
fibexElement	FibexElement	*	ref	Reference to ASAM FIBEX elements specifying Communication and Topology. All Fibex Elements used within a System Description shall be referenced from the System Element. atpVariation: In order to describe a product-line, all Fibex Elements can be optional. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=postBuild
interpolation Routine MappingSet	InterpolationRoutine MappingSet	*	ref	This reference identifies the InterpolationRoutineMapping Sets that are relevant in the context of the enclosing System.
j1939Shared AddressCluster	J1939SharedAddress Cluster	*	aggr	Collection of J1939Clusters that share a common address space for the routing of messages. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=j1939SharedAddressCluster.shortName, j1939SharedAddressCluster.variationPoint.shortLabel vh.latestBindingTime=postBuild
mapping	SystemMapping	*	aggr	Aggregation of all mapping aspects (mapping of SW components to ECUs, mapping of data elements to signals, and mapping constraints). In order to support OEM / Tier 1 interaction and shared development for one common System this aggregation is atpSplitable and atpVariation. The content of System Mapping can be provided by several parties using different names for the SystemMapping. This element is not required when the System description is used for a network-only use-case. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=mapping.shortName, mapping.variation Point.shortLabel vh.latestBindingTime=postBuild
pncVector Length	PositiveInteger	0..1	attr	Length of the partial networking request release information vector (in bytes).
pncVectorOffset	PositiveInteger	0..1	attr	Absolute offset (with respect to the NM-PDU) of the partial networking request release information vector that is defined in bytes as an index starting with 0.

△

Class	**System**			
rootSoftware Composition	RootSwComposition Prototype	0..1	aggr	Aggregation of the root software composition, containing all software components in the System in a hierarchical structure. This element is not required when the System description is used for a network-only use-case. atpVariation: The RootSwCompositionPrototype can vary. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=rootSoftwareComposition.shortName, root SoftwareComposition.variationPoint.shortLabel vh.latestBindingTime=systemDesignTime
swCluster	CpSoftwareCluster	*	ref	CP Software Clusters of this System **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=swCluster.cpSoftwareCluster, sw Cluster.variationPoint.shortLabel atp.Status=draft vh.latestBindingTime=systemDesignTime
system Documentation	Chapter	*	aggr	Possibility to provide additional documentation while defining the System. The System documentation can be composed of several chapters. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=systemDocumentation.shortName, system Documentation.variationPoint.shortLabel vh.latestBindingTime=systemDesignTime xml.sequenceOffset=-10
systemVersion	RevisionLabelString	1	attr	Version number of the System Description.

Table E.70: System

Primitive	**TimeValue**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	This primitive type is taken for expressing time values. The numerical value is supposed to be interpreted in the physical unit second. **Tags:** xml.xsd.customType=TIME-VALUE xml.xsd.type=double

Table E.71: TimeValue

Class	***TimingExtension*** (abstract)			
Package	M2::AUTOSARTemplates::CommonStructure::Timing			
Note	The abstract parent class of the different template specific timing extensions. Depending on the specific timing extension the timing descriptions and timing constraints, that can be used to specify the timing behavior, are restricted.			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Subclasses	BswCompositionTiming, BswModuleTiming, EcuTiming, SwcTiming, SystemTiming, VfbTiming			
Attribute	*Type*	*Mult.*	*Kind*	*Note*

▽

△

Class	***TimingExtension*** (abstract)			
timingCondition	TimingCondition	*	aggr	The timing condition specifies a specific condition. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=timingCondition.shortName, timing Condition.variationPoint.shortLabel vh.latestBindingTime=postBuild
timing Description	TimingDescription	*	aggr	The timing descriptions that belong to a specific timing specification. In order to support different timing description variants within a timing specification, the aggregation is marked with the stereotype "atpVariation". **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=timingDescription.shortName, timing Description.variationPoint.shortLabel vh.latestBindingTime=postBuild
timing Guarantee	TimingConstraint	*	aggr	The timing constraints that belong to a specific timing specification in the role of a timing guarantee. In order to support different timing constraint variants within a timing specification, the aggregation is marked with the stereotype "atpVariation". **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=timingGuarantee.shortName, timing Guarantee.variationPoint.shortLabel vh.latestBindingTime=postBuild
timing Requirement	TimingConstraint	*	aggr	The timing constraints that belong to a specific timing specification in the role of a timing requirement. In order to support different timing constraint variants within a timing specification, the aggregation is marked with the stereotype "atpVariation". **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=timingRequirement.shortName, timing Requirement.variationPoint.shortLabel vh.latestBindingTime=postBuild
timingResource	TimingExtension Resource	0..1	aggr	The timing resource contains all instance references referred from within a timing condition formula of a timing view. **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=timingResource.shortName

Table E.72: TimingExtension

Class	<<atpMixed>> **TopicContent**			
Package	M2::MSR::Documentation::Chapters			
Note	This meta-class represents the content of a topic. It is mainly a documentation block, but can also be a table.			
Base	*ARObject*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
blockLevel Content	DocumentationBlock	1	aggr	This is that part of the content which may also occur in a table cell. **Tags:**xml.roleElement=false

▽

△

Class	<<atpMixed>> **TopicContent**			
table	Table	0..1	aggr	This represents a table within a topic. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=postBuild
traceableTable	TraceableTable	1	aggr	This represents a traceable table within a topic.

Table E.73: TopicContent

Class	**Unit**			
Package	M2::MSR::AsamHdo::Units			
Note	This is a physical measurement unit. All units that might be defined should stem from SI units. In order to convert one unit into another factor and offset are defined. For the calculation from SI-unit to the defined unit the factor (factorSiToUnit) and the offset (offsetSiTo Unit) are applied as follows: x [{unit}] := y * [{siUnit}] * factorSiToUnit [[unit]/{siUnit}] + offsetSiToUnit [{unit}] For the calculation from a unit to SI-unit the reciprocal of the factor (factorSiToUnit) and the negation of the offset (offsetSiToUnit) are applied. y {siUnit} := (x*{unit} - offsetSiToUnit [{unit}]) / (factorSiToUnit [[unit]/{siUnit}] **Tags:**atp.recommendedPackage=Units			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
displayName	SingleLanguageUnit Names	0..1	aggr	This specifies how the unit shall be displayed in documents or in user interfaces of tools.The displayName corresponds to the Unit.Display in an ASAM MCD-2MC file. **Tags:**xml.sequenceOffset=20
factorSiToUnit	Float	0..1	attr	This is the factor for the conversion from SI Units to units. The inverse is used for conversion from units to SI Units. **Tags:**xml.sequenceOffset=30
offsetSiToUnit	Float	0..1	attr	This is the offset for the conversion from and to siUnits. **Tags:**xml.sequenceOffset=40
physical Dimension	PhysicalDimension	0..1	ref	This association represents the physical dimension to which the unit belongs to. Note that only values with units of the same physical dimensions might be converted. **Tags:**xml.sequenceOffset=50

Table E.74: Unit

Primitive	**UnlimitedInteger**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	An instance of UnlimitedInteger is an element in the set of integer numbers (..., -2, -1, 0, 1, 2, ...). The range is limited by constraint 2534. The value can be expressed in decimal, octal, hexadecimal and binary representation. Negative numbers can only be expressed in decimal notation. **Tags:** xml.xsd.customType=UNLIMITED-INTEGER xml.xsd.pattern=0|[\+\-]?[1-9][0-9]*|0[xX][0-9a-fA-F]+|0[bB][0-1]+|0[0-7]+ xml.xsd.type=string

Table E.75: UnlimitedInteger

F Examples

This chapter contains more detailed information for examples which were shown inside the preceding chapters of the specification.

F.1 ShortLabels in VariationPoints

F.1.1 Identifiables with identical shortNames

The following ARXML files illustrate the use of `shortLabel`s in case of identical `shortName`s and distribution in three partial models, respectively three different files.

For example the `InternalBehavior` (Ib_MySWC) of a software component is subject to variation. In this case the `VariationPoint` needs to be considered as well [TPS_GST_00047].

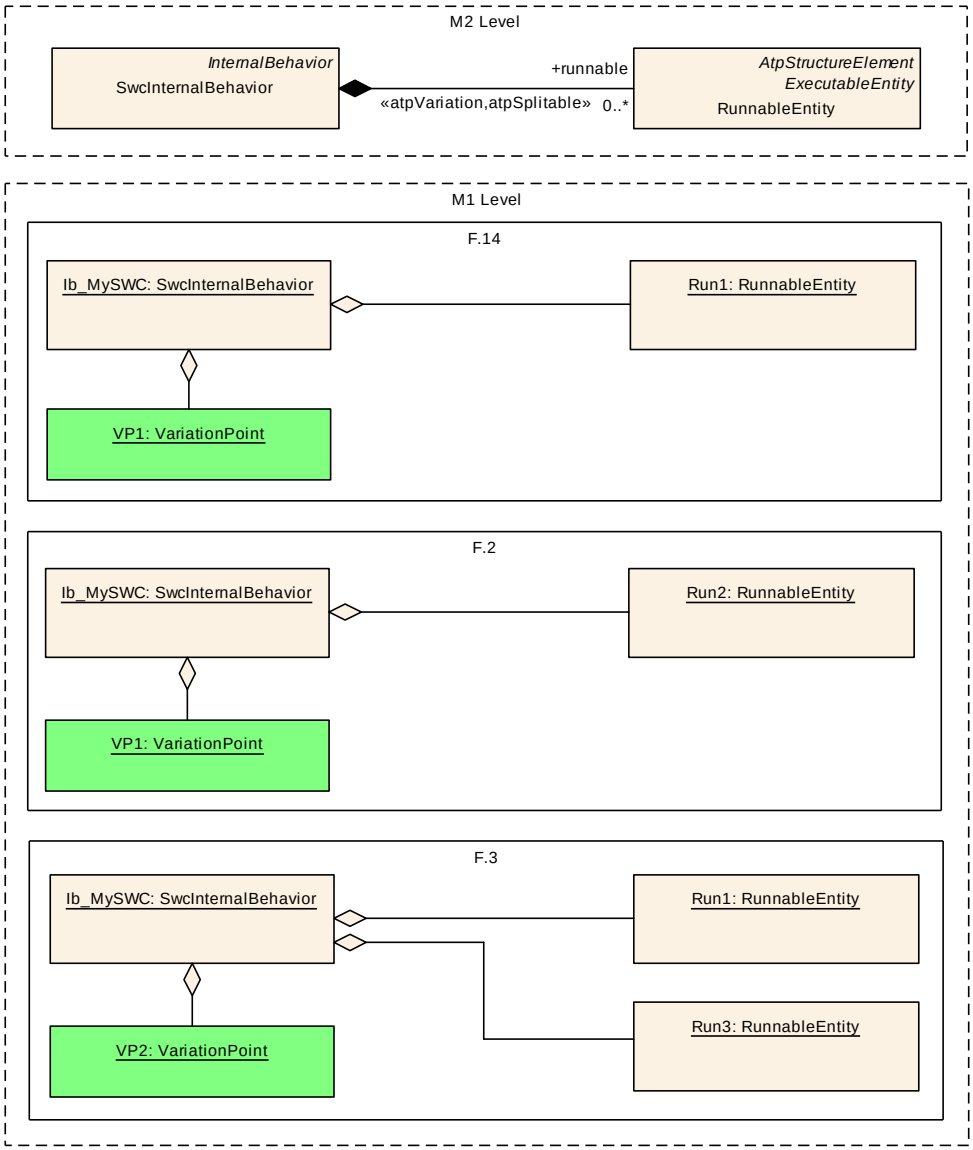


Figure F.1: M1 Model with variants and splitable elements

The figure F.1 shows two `VariationPoint`s (VP1, VP2) of an `InternalBehavior` (Ib_MySWC) distributed in three files (listings) F.1, F.2 and F.3.

Listing F.1 contains the identical `shortName`s as in listing F.2 but both exist in separate partial models. Both have the same `atp.splitkey` given by the combination of the `shortName` (Ib_MySwc) and the `shortLabel` (VP1).

Listing F.1: Example for identical `shortName`s in partial model 1

```
<?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http:
    //autosar.org/schema/r4.0" xsi:schemaLocation="http://autosar.org/schema
    /r4.0␣AUTOSAR_00046.xsd">
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>Demo</SHORT-NAME>
      <AR-PACKAGES>
```

```
        <AR-PACKAGE>
          <SHORT-NAME>SwComponentTypes</SHORT-NAME>
          <ELEMENTS>
            <APPLICATION-SW-COMPONENT-TYPE>
              <SHORT-NAME>MySWC</SHORT-NAME>
              <INTERNAL-BEHAVIORS>
                <SWC-INTERNAL-BEHAVIOR>
                  <SHORT-NAME>Ib_MySWC</SHORT-NAME>
                  <RUNNABLES>
                    <RUNNABLE-ENTITY>
                      <SHORT-NAME>Run1</SHORT-NAME>
                    </RUNNABLE-ENTITY>
                  </RUNNABLES>
                  <VARIATION-POINT>
                    <SHORT-LABEL>VP1</SHORT-LABEL>
                    <SW-SYSCOND BINDING-TIME="CODE-GENERATION-TIME">
                      <SYSC-REF DEST="SW-SYSTEMCONST">/Demo/SystemConstants
                          /SY_TURBO</SYSC-REF>== 0</SW-SYSCOND>
                  </VARIATION-POINT>
                </SWC-INTERNAL-BEHAVIOR>
              </INTERNAL-BEHAVIORS>
            </APPLICATION-SW-COMPONENT-TYPE>
          </ELEMENTS>
        </AR-PACKAGE>
        <AR-PACKAGE>
          <SHORT-NAME>SystemConstants</SHORT-NAME>
          <ELEMENTS>
            <SW-SYSTEMCONST>
              <SHORT-NAME>SY_TURBO</SHORT-NAME>
              <SW-DATA-DEF-PROPS>
                <SW-DATA-DEF-PROPS-VARIANTS>
                  <SW-DATA-DEF-PROPS-CONDITIONAL>
                    <SW-CALIBRATION-ACCESS>NOT-ACCESSIBLE</SW-CALIBRATION-
                        ACCESS>
                  </SW-DATA-DEF-PROPS-CONDITIONAL>
                </SW-DATA-DEF-PROPS-VARIANTS>
              </SW-DATA-DEF-PROPS>
            </SW-SYSTEMCONST>
          </ELEMENTS>
        </AR-PACKAGE>
      </AR-PACKAGES>
    </AR-PACKAGE>
  </AR-PACKAGES>
</AUTOSAR>
```

Listing F.2: Example for identical shortNames in partial model 2

```
<?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http:
    //autosar.org/schema/r4.0" xsi:schemaLocation="http://autosar.org/schema
    /r4.0␣AUTOSAR_00046.xsd">
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>Demo</SHORT-NAME>
      <AR-PACKAGES>
        <AR-PACKAGE>
```

```
          <SHORT-NAME>SwComponentTypes</SHORT-NAME>
          <ELEMENTS>
            <APPLICATION-SW-COMPONENT-TYPE>
              <SHORT-NAME>MySWC</SHORT-NAME>
              <INTERNAL-BEHAVIORS>
                <SWC-INTERNAL-BEHAVIOR>
                  <SHORT-NAME>Ib_MySWC</SHORT-NAME>
                  <RUNNABLES>
                    <RUNNABLE-ENTITY>
                      <SHORT-NAME>Run2</SHORT-NAME>
                    </RUNNABLE-ENTITY>
                  </RUNNABLES>
                  <VARIATION-POINT>
                    <SHORT-LABEL>VP1</SHORT-LABEL>
                  </VARIATION-POINT>
                </SWC-INTERNAL-BEHAVIOR>
              </INTERNAL-BEHAVIORS>
            </APPLICATION-SW-COMPONENT-TYPE>
          </ELEMENTS>
        </AR-PACKAGE>
      </AR-PACKAGES>
    </AR-PACKAGE>
  </AR-PACKAGES>
</AUTOSAR>
```

The listing F.3 has the combined `atp.splitkey` (Ib_MySWC/VP2) and the `shortLabel` (VP2). It contains the identical `shortName` as in the listing F.1 but a different `shortLabel` (VP2). This `VariationPoint` provides two `RunnableEntity`s (Run1, Run3) in listing F.3.

Listing F.3: Example for identical `shortName`s but different `shortLabel`

```
<?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http:
    //autosar.org/schema/r4.0" xsi:schemaLocation="http://autosar.org/schema
    /r4.0␣AUTOSAR_00046.xsd">
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>Demo</SHORT-NAME>
      <AR-PACKAGES>
        <AR-PACKAGE>
          <SHORT-NAME>SwComponentTypes</SHORT-NAME>
          <ELEMENTS>
            <APPLICATION-SW-COMPONENT-TYPE>
              <SHORT-NAME>MySWC</SHORT-NAME>
              <INTERNAL-BEHAVIORS>
                <SWC-INTERNAL-BEHAVIOR>
                  <SHORT-NAME>Ib_MySWC</SHORT-NAME>
                  <RUNNABLES>
                    <RUNNABLE-ENTITY>
                      <SHORT-NAME>Run1</SHORT-NAME>
                    </RUNNABLE-ENTITY>
                    <RUNNABLE-ENTITY>
                      <SHORT-NAME>Run3</SHORT-NAME>
                    </RUNNABLE-ENTITY>
                  </RUNNABLES>
```

```
                <VARIATION-POINT>
                  <SHORT-LABEL>VP2</SHORT-LABEL>
                  <SW-SYSCOND BINDING-TIME="CODE-GENERATION-TIME">
                    <SYSC-REF DEST="SW-SYSTEMCONST">/Demo/SystemConstants
                        /SY_TURBO</SYSC-REF>== 1</SW-SYSCOND>
                </VARIATION-POINT>
              </SWC-INTERNAL-BEHAVIOR>
            </INTERNAL-BEHAVIORS>
          </APPLICATION-SW-COMPONENT-TYPE>
        </ELEMENTS>
      </AR-PACKAGE>
    </AR-PACKAGES>
  </AR-PACKAGE>
 </AR-PACKAGES>
</AUTOSAR>
```

Due to [constr_2512] the `shortName` and the `shortLabel` of a variant element shall be unique within the name space established by the surrounding Identifiable. If additionally those `shortName`s are described in partial models the `shortLabel`s in the partial models indicate which elements belong together and it is required to repeat the `shortLabel` consistently.

After merging the splitables (listings) F.1, F.2, F.3 the merged model (listing) F.4 contains again the two `VariationPoint`s (VP1, VP2) of the `InternalBehavior` (Ib_MySWC).

Note: The merged model exists only internally in the AUTOSAR tool. The listing is shown only for illustration.

Listing F.4: Example for Merged Model

```
<?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http:
    //autosar.org/schema/r4.0" xsi:schemaLocation="http://autosar.org/schema
    /r4.0␣AUTOSAR_00046.xsd">
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>Demo</SHORT-NAME>
      <AR-PACKAGES>
        <AR-PACKAGE>
          <SHORT-NAME>SwComponentTypes</SHORT-NAME>
          <ELEMENTS>
            <APPLICATION-SW-COMPONENT-TYPE>
              <SHORT-NAME>MySWC</SHORT-NAME>
              <INTERNAL-BEHAVIORS>
                <SWC-INTERNAL-BEHAVIOR>
                  <SHORT-NAME>Ib_MySWC</SHORT-NAME>
                  <RUNNABLES>
                    <RUNNABLE-ENTITY>
                      <SHORT-NAME>Run1</SHORT-NAME>
                    </RUNNABLE-ENTITY>
                    <RUNNABLE-ENTITY>
                      <SHORT-NAME>Run2</SHORT-NAME>
                    </RUNNABLE-ENTITY>
                  </RUNNABLES>
```

```
            <VARIATION-POINT>
              <SHORT-LABEL>VP1</SHORT-LABEL>
              <!-- <SW-SYSCOND BINDING-TIME="CODE-GENERATION-TIME">
                <SYSC-REF DEST="SW-SYSTEMCONST">SystemConstants/
                    SY_TURBO</SYSC-REF>== 0</SW-SYSCOND> -->
            </VARIATION-POINT>
          </SWC-INTERNAL-BEHAVIOR>
          <SWC-INTERNAL-BEHAVIOR>
            <SHORT-NAME>Ib_MySWC</SHORT-NAME>
            <RUNNABLES>
              <RUNNABLE-ENTITY>
                <SHORT-NAME>Run1</SHORT-NAME>
              </RUNNABLE-ENTITY>
              <RUNNABLE-ENTITY>
                <SHORT-NAME>Run3</SHORT-NAME>
              </RUNNABLE-ENTITY>
            </RUNNABLES>
            <VARIATION-POINT>
              <SHORT-LABEL>VP2</SHORT-LABEL>
              <SW-SYSCOND BINDING-TIME="CODE-GENERATION-TIME">
                <SYSC-REF DEST="SW-SYSTEMCONST">SystemConstants/
                    SY_TURBO</SYSC-REF>== 1</SW-SYSCOND>
            </VARIATION-POINT>
          </SWC-INTERNAL-BEHAVIOR>
        </INTERNAL-BEHAVIORS>
      </APPLICATION-SW-COMPONENT-TYPE>
    </ELEMENTS>
  </AR-PACKAGE>
 </AR-PACKAGES>
</AR-PACKAGE>
</AR-PACKAGES>
</AUTOSAR>
```

Finally, after binding the variants, only the elements remain where the variation condition is true.

F.2 Splitable in use

F.2.1 Introduction Example

The following ARXML files illustrate the example given in section 8.1 Introduction [1].

The Composition A does not contain any Atomic Software Components and will be persisted in one ARXML file, see listing F.5.

Listing F.5: Example for Composition A (DOC_GST_ExampleForSplitableCompA.arxml)

```
<?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns="http://autosar.org/schema/r4.0" xmlns:xsi="http://www.w3.
    org/2001/XMLSchema-instance" xsi:schemaLocation="http://autosar.org/
    schema/r4.0␣AUTOSAR_00046.xsd">
```

[1] The first listing is printed with XML declaration and the root element. The further listings without.

```
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>Composition</SHORT-NAME>
      <ELEMENTS>
        <COMPOSITION-SW-COMPONENT-TYPE>
          <SHORT-NAME>CompositionA</SHORT-NAME>
          <PORTS>
            <P-PORT-PROTOTYPE>
              <SHORT-NAME>PPort_CompositionA</SHORT-NAME>
            </P-PORT-PROTOTYPE>
            <R-PORT-PROTOTYPE>
              <SHORT-NAME>RPort_CompositionA</SHORT-NAME>
            </R-PORT-PROTOTYPE>
          </PORTS>
        </COMPOSITION-SW-COMPONENT-TYPE>
      </ELEMENTS>
    </AR-PACKAGE>
  </AR-PACKAGES>
</AUTOSAR>
```

The Atomic Software Component (SW-C X) will be persisted in a dedicated ARXML file, see listing F.6.

Listing F.6: Example for SW-C X (DOC_GST_ExampleForSplitableSWCX.arxml)

```
<AR-PACKAGE>
  <SHORT-NAME>SwComponentTypes</SHORT-NAME>
  <ELEMENTS>
    <APPLICATION-SW-COMPONENT-TYPE>
      <SHORT-NAME>SWCX</SHORT-NAME>
      <PORTS>
        <P-PORT-PROTOTYPE>
          <SHORT-NAME>PPort_SWCX</SHORT-NAME>
        </P-PORT-PROTOTYPE>
        <R-PORT-PROTOTYPE>
          <SHORT-NAME>RPort_SWCX</SHORT-NAME>
        </R-PORT-PROTOTYPE>
      </PORTS>
      <INTERNAL-BEHAVIORS>
        <SWC-INTERNAL-BEHAVIOR>
          <SHORT-NAME>SwcInternalBehavior_SWCX</SHORT-NAME>
          <RUNNABLES>
            <RUNNABLE-ENTITY>
              <SHORT-NAME>Runnable1_SWCX</SHORT-NAME>
              <SYMBOL>RunnableEntity_X1_func</SYMBOL>
            </RUNNABLE-ENTITY>
            <RUNNABLE-ENTITY>
              <SHORT-NAME>Runnable2_SWCX</SHORT-NAME>
              <SYMBOL>RunnableEntity_X2_func</SYMBOL>
            </RUNNABLE-ENTITY>
          </RUNNABLES>
        </SWC-INTERNAL-BEHAVIOR>
      </INTERNAL-BEHAVIORS>
    </APPLICATION-SW-COMPONENT-TYPE>
  </ELEMENTS>
</AR-PACKAGE>
```

The same is done for the Atomic Software Component SW-C Y, see listing F.7.

Listing F.7: Example for SW-C Y (DOC_GST_ExampleForSplitableSWCY.arxml)

```
<AR-PACKAGE>
  <SHORT-NAME>SwComponentTypes</SHORT-NAME>
  <ELEMENTS>
    <APPLICATION-SW-COMPONENT-TYPE>
      <SHORT-NAME>SWCY</SHORT-NAME>
      <INTERNAL-BEHAVIORS>
        <SWC-INTERNAL-BEHAVIOR>
          <SHORT-NAME>SwcInternalBehavior_SWCY</SHORT-NAME>
        </SWC-INTERNAL-BEHAVIOR>
      </INTERNAL-BEHAVIORS>
    </APPLICATION-SW-COMPONENT-TYPE>
  </ELEMENTS>
</AR-PACKAGE>
```

The ARPackage SW-C Y is split e.g. in separate port definitions. They ensure the interaction with the Composition A and the Atomic Software Component (SW-C X), see listing F.8 and listing F.9.

Listing F.8: Example for P-Port of SW-C Y (DOC_GST_ExampleForSplitableSWCY_PP.arxml)

```
<AR-PACKAGE>
  <SHORT-NAME>SwComponentTypes</SHORT-NAME>
  <ELEMENTS>
    <APPLICATION-SW-COMPONENT-TYPE>
      <SHORT-NAME>SWCY</SHORT-NAME>
      <PORTS>
        <P-PORT-PROTOTYPE>
          <SHORT-NAME>PPort_SWCY</SHORT-NAME>
        </P-PORT-PROTOTYPE>
      </PORTS>
    </APPLICATION-SW-COMPONENT-TYPE>
  </ELEMENTS>
</AR-PACKAGE>
```

Listing F.9: Example for R-Port of SW-C Y (DOC_GST_ExampleForSplitableSWCY_RP.arxml)

```
<AR-PACKAGE>
  <SHORT-NAME>SwComponentTypes</SHORT-NAME>
  <ELEMENTS>
    <APPLICATION-SW-COMPONENT-TYPE>
      <SHORT-NAME>SWCY</SHORT-NAME>
      <PORTS>
        <R-PORT-PROTOTYPE>
          <SHORT-NAME>RPort_SWCY</SHORT-NAME>
        </R-PORT-PROTOTYPE>
      </PORTS>
    </APPLICATION-SW-COMPONENT-TYPE>
  </ELEMENTS>
</AR-PACKAGE>
```

Due to further optimization two splitable artifacts are useful for the runnables of SW-C Y (see listing F.10 and listing F.11).

Listing F.10: Example for Runnable 1 of SW-C Y (DOC_GST_ExampleForSplitableSWCY_IBRun1.arxml)

```
<AR-PACKAGE>
  <SHORT-NAME>SwComponentTypes</SHORT-NAME>
  <ELEMENTS>
    <APPLICATION-SW-COMPONENT-TYPE>
      <SHORT-NAME>SWCY</SHORT-NAME>
      <INTERNAL-BEHAVIORS>
        <SWC-INTERNAL-BEHAVIOR>
          <SHORT-NAME>SwcInternalBehavior_SWCY</SHORT-NAME>
          <RUNNABLES>
            <RUNNABLE-ENTITY>
              <SHORT-NAME>Runnable1_SWCY</SHORT-NAME>
              <SYMBOL>RunnableEntity_Y1_func</SYMBOL>
            </RUNNABLE-ENTITY>
          </RUNNABLES>
        </SWC-INTERNAL-BEHAVIOR>
      </INTERNAL-BEHAVIORS>
    </APPLICATION-SW-COMPONENT-TYPE>
  </ELEMENTS>
</AR-PACKAGE>
```

Listing F.11: Example for Runnable 2 of SW-C Y (DOC_GST_ExampleForSplitableSWCY_IBRun2.arxml)

```
<AR-PACKAGE>
  <SHORT-NAME>SwComponentTypes</SHORT-NAME>
  <ELEMENTS>
    <APPLICATION-SW-COMPONENT-TYPE>
      <SHORT-NAME>SWCY</SHORT-NAME>
      <INTERNAL-BEHAVIORS>
        <SWC-INTERNAL-BEHAVIOR>
          <SHORT-NAME>SwcInternalBehavior_SWCY</SHORT-NAME>
          <RUNNABLES>
            <RUNNABLE-ENTITY>
              <SHORT-NAME>Runnable2_SWCY</SHORT-NAME>
              <SYMBOL>RunnableEntity_Y2_func</SYMBOL>
            </RUNNABLE-ENTITY>
          </RUNNABLES>
        </SWC-INTERNAL-BEHAVIOR>
      </INTERNAL-BEHAVIORS>
    </APPLICATION-SW-COMPONENT-TYPE>
  </ELEMENTS>
</AR-PACKAGE>
```

Finally the "merged model" of the SWC-Y is shown in listing F.12.

Note: The merged model exists only internally in the AUTOSAR tool. The listing is shown only for illustration.

Listing F.12: Example for Merged Model of SW-C Y (DOC_GST_ExampleForSplitableSWCY_Merged.arxml)

```
<AR-PACKAGE>
  <SHORT-NAME>SwComponentTypes</SHORT-NAME>
  <ELEMENTS>
```

```
      <APPLICATION-SW-COMPONENT-TYPE>
        <SHORT-NAME>SWCY</SHORT-NAME>
        <PORTS>
          <P-PORT-PROTOTYPE>
            <SHORT-NAME>PPort_SWCY</SHORT-NAME>
          </P-PORT-PROTOTYPE>
          <R-PORT-PROTOTYPE>
            <SHORT-NAME>RPort_SWCY</SHORT-NAME>
          </R-PORT-PROTOTYPE>
        </PORTS>
        <INTERNAL-BEHAVIORS>
          <SWC-INTERNAL-BEHAVIOR>
            <SHORT-NAME>SwcInternalBehavior_SWCY</SHORT-NAME>
            <RUNNABLES>
              <RUNNABLE-ENTITY>
                <SHORT-NAME>Runnable1_SWCY</SHORT-NAME>
                <SYMBOL>RunnableEntity_Y1_func</SYMBOL>
              </RUNNABLE-ENTITY>
              <RUNNABLE-ENTITY>
                <SHORT-NAME>Runnable2_SWCY</SHORT-NAME>
                <SYMBOL>RunnableEntity_Y2_func</SYMBOL>
              </RUNNABLE-ENTITY>
            </RUNNABLES>
          </SWC-INTERNAL-BEHAVIOR>
        </INTERNAL-BEHAVIORS>
      </APPLICATION-SW-COMPONENT-TYPE>
    </ELEMENTS>
  </AR-PACKAGE>
```

For illustrative reasons the ARXML of the complete "merged model" of the above example is shown in listing F.13.

Listing F.13: Example for Merged Model of Comp A (DOC_GST_ExampleForSplitableCompA_Merged.arxml)

```
  <AR-PACKAGE>
    <SHORT-NAME>Composition</SHORT-NAME>
    <ELEMENTS>
      <COMPOSITION-SW-COMPONENT-TYPE>
        <SHORT-NAME>CompositionA</SHORT-NAME>
        <PORTS>
          <P-PORT-PROTOTYPE>
            <SHORT-NAME>PPort_CompositionA</SHORT-NAME>
          </P-PORT-PROTOTYPE>
          <R-PORT-PROTOTYPE>
            <SHORT-NAME>RPort_CompositionA</SHORT-NAME>
          </R-PORT-PROTOTYPE>
        </PORTS>
        <COMPONENTS>
          <SW-COMPONENT-PROTOTYPE>
            <SHORT-NAME>CPT_SWCX</SHORT-NAME>
            <TYPE-TREF DEST="APPLICATION-SW-COMPONENT-TYPE">/
              SwComponentTypes/SWCX</TYPE-TREF>
          </SW-COMPONENT-PROTOTYPE>
          <SW-COMPONENT-PROTOTYPE>
            <SHORT-NAME>CPT_SWCY</SHORT-NAME>
```

```
            <TYPE-TREF DEST="APPLICATION-SW-COMPONENT-TYPE">/
               SwComponentTypes/SWCY</TYPE-TREF>
          </SW-COMPONENT-PROTOTYPE>
        </COMPONENTS>
      </COMPOSITION-SW-COMPONENT-TYPE>
    </ELEMENTS>
  </AR-PACKAGE>
  <AR-PACKAGE>
    <SHORT-NAME>SwComponentTypes</SHORT-NAME>
    <ELEMENTS>
      <APPLICATION-SW-COMPONENT-TYPE>
        <SHORT-NAME>SWCX</SHORT-NAME>
        <PORTS>
          <P-PORT-PROTOTYPE>
            <SHORT-NAME>PPort_SWCX</SHORT-NAME>
          </P-PORT-PROTOTYPE>
          <R-PORT-PROTOTYPE>
            <SHORT-NAME>RPort_SWCX</SHORT-NAME>
          </R-PORT-PROTOTYPE>
        </PORTS>
        <INTERNAL-BEHAVIORS>
          <SWC-INTERNAL-BEHAVIOR>
            <SHORT-NAME>SwcInternalBehavior_SWCX</SHORT-NAME>
            <RUNNABLES>
              <RUNNABLE-ENTITY>
                <SHORT-NAME>Runnable1_SWCX</SHORT-NAME>
                <SYMBOL>RunnableEntity_X1_func</SYMBOL>
              </RUNNABLE-ENTITY>
              <RUNNABLE-ENTITY>
                <SHORT-NAME>Runnable2_SWCX</SHORT-NAME>
                <SYMBOL>RunnableEntity_X2_func</SYMBOL>
              </RUNNABLE-ENTITY>
            </RUNNABLES>
          </SWC-INTERNAL-BEHAVIOR>
        </INTERNAL-BEHAVIORS>
      </APPLICATION-SW-COMPONENT-TYPE>
      <APPLICATION-SW-COMPONENT-TYPE>
        <SHORT-NAME>SWCY</SHORT-NAME>
        <PORTS>
          <P-PORT-PROTOTYPE>
            <SHORT-NAME>PPort_SWCY</SHORT-NAME>
          </P-PORT-PROTOTYPE>
          <R-PORT-PROTOTYPE>
            <SHORT-NAME>RPort_SWCY</SHORT-NAME>
          </R-PORT-PROTOTYPE>
        </PORTS>
        <INTERNAL-BEHAVIORS>
          <SWC-INTERNAL-BEHAVIOR>
            <SHORT-NAME>SwcInternalBehavior_SWCY</SHORT-NAME>
            <RUNNABLES>
              <RUNNABLE-ENTITY>
                <SHORT-NAME>Runnable1_SWCY</SHORT-NAME>
                <SYMBOL>RunnableEntity_Y1_func</SYMBOL>
              </RUNNABLE-ENTITY>
              <RUNNABLE-ENTITY>
                <SHORT-NAME>Runnable2_SWCY</SHORT-NAME>
```

```
            <SYMBOL>RunnableEntity_Y2_func</SYMBOL>
          </RUNNABLE-ENTITY>
        </RUNNABLES>
      </SWC-INTERNAL-BEHAVIOR>
    </INTERNAL-BEHAVIORS>
  </APPLICATION-SW-COMPONENT-TYPE>
 </ELEMENTS>
</AR-PACKAGE>
```

Name of splitable element	Splitkey	Value of Splitkey path in example
SwComponentType.port	shortName, variationPoint.shortLabel	("SwComponentTypes","SWCY","PPort_SWCY"), ("SwComponentTypes","SWCY","RPort_SWCY")
SwcInternalBehavior.runnable	shortName, variationPoint.shortLabel	("SwComponentTypes","SWCY","SwcInternal-Behavior_SWCY","Runnable1_SWCY"), ("SwComponentTypes","SWCY","SwcInternal-Behavior_SWCY","Runnable2_SWCY")

Table F.1: Splitable elements of the above example

The figure F.2 illustrates the classes of the meta-model used in the above example.

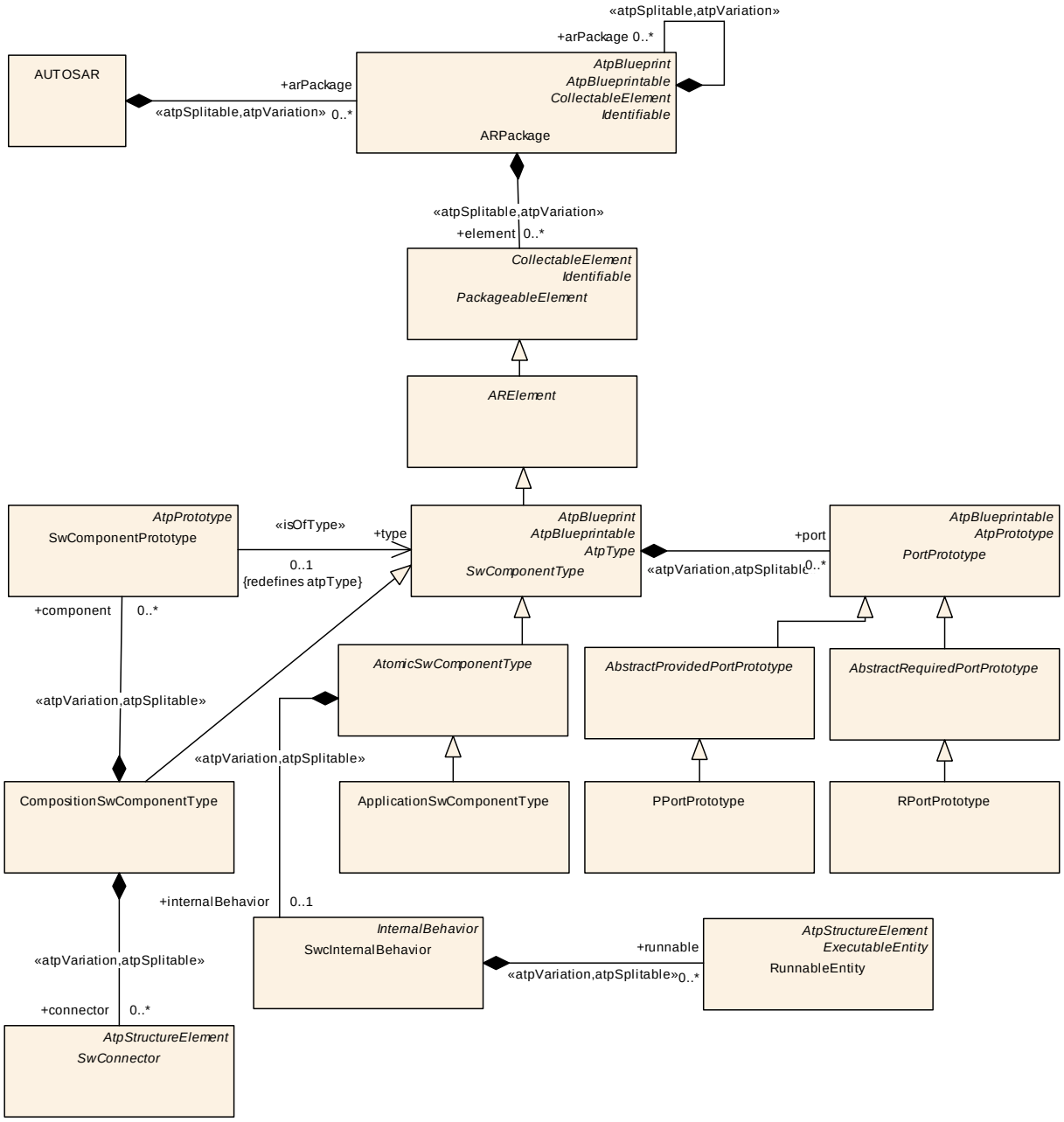


Figure F.2: Meta-model with splitables

F.2.2 Splitkey for aggregation with upper multiplicity 1

According to [TPS_GST_00047], the role of aggregation shall be used as the splitkey for aggregations with upper multiplicity 1. An example for this use case is the measurement & calibration support data belonging to an implementation. The aggregation is ≪atpSplitable≫ because in case of an already existing BswImplementation model, the description mcSupport will be added later in the process.

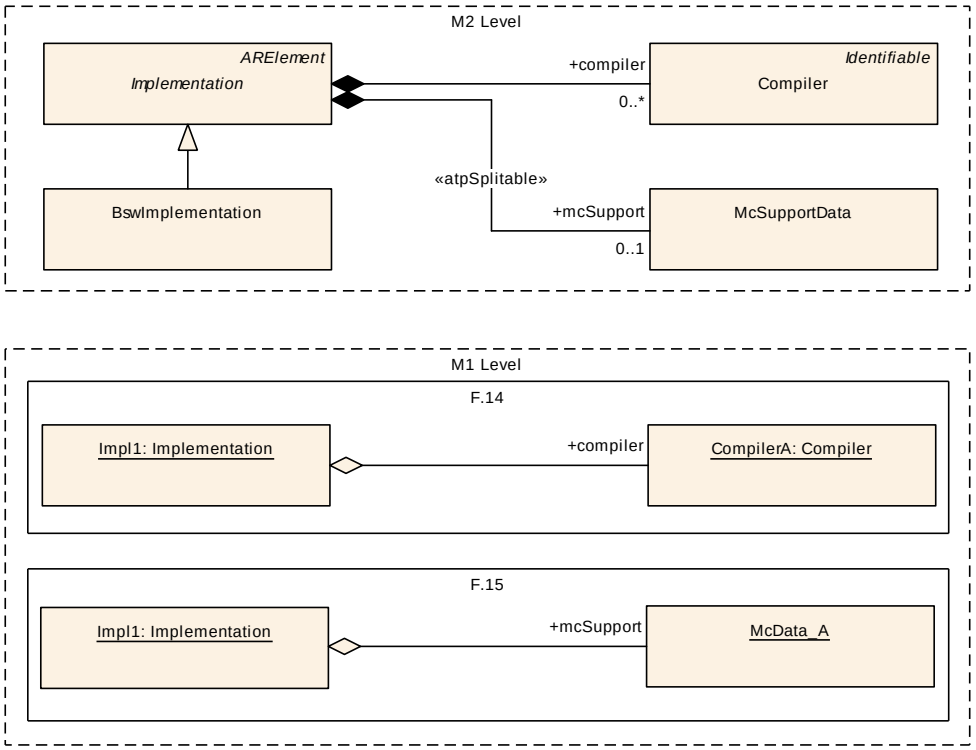


Figure F.3: M1 Model splitable elements with Aggregation

The figure F.3 shows in the upper part an existing `BswImplementation` (*Impl1*) with an aggregation to `Compiler`. This `BswImplementation` will be extended with a description of `mcSupport` (lower part). The splitkey is the role name "mcSupport".

The following ARXML files illustrate this use case and the distribution in two partial models. Listing F.14 contains an existing part of `BswImplementation` (*Impl1*).

Listing F.14: Example for splitable (aggregation) in partial model 1

```
<?xml version="1.0"?>
<AUTOSAR xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http:
    //autosar.org/schema/r4.0" xsi:schemaLocation="http://autosar.org/schema
    /r4.0␣AUTOSAR_00046.xsd">
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>Supplier</SHORT-NAME>
      <ELEMENTS>
        <BSW-IMPLEMENTATION>
          <SHORT-NAME>Impl1</SHORT-NAME>
          <COMPILERS>
            <COMPILER>
              <SHORT-NAME>CompilerA</SHORT-NAME>
            </COMPILER>
          </COMPILERS>
        </BSW-IMPLEMENTATION>
      </ELEMENTS>
    </AR-PACKAGE>
  </AR-PACKAGES>
</AUTOSAR>
```

Listing F.15 contains the extending part of BswImplementation (*Impl1*) with mcSupport.

Listing F.15: Example for splitable (aggregation) in partial model 2

```
<?xml version="1.0"?>
<AUTOSAR xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http:
    //autosar.org/schema/r4.0" xsi:schemaLocation="http://autosar.org/schema
    /r4.0␣AUTOSAR_00046.xsd">
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>Supplier</SHORT-NAME>
      <ELEMENTS>
        <BSW-IMPLEMENTATION>
          <SHORT-NAME>Impl1</SHORT-NAME>
          <MC-SUPPORT>
            <MC-PARAMETER-INSTANCES>
              <MC-DATA-INSTANCE>
                <SHORT-NAME>McData_A</SHORT-NAME>
              </MC-DATA-INSTANCE>
            </MC-PARAMETER-INSTANCES>
          </MC-SUPPORT>
        </BSW-IMPLEMENTATION>
      </ELEMENTS>
    </AR-PACKAGE>
```

After merging the splitables F.14, F.15 the merged model F.16 contains both parts of BswImplementation.

Note: The merged model exists only internally in the AUTOSAR tool. The listing is shown only for illustration.

Listing F.16: Example for splitable (aggregation)

```
<?xml version="1.0"?>
<AUTOSAR xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http:
    //autosar.org/schema/r4.0" xsi:schemaLocation="http://autosar.org/schema
    /r4.0␣AUTOSAR_00046.xsd">
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>Supplier</SHORT-NAME>
      <ELEMENTS>
        <BSW-IMPLEMENTATION>
          <SHORT-NAME>Impl1</SHORT-NAME>
          <COMPILERS>
            <COMPILER>
              <SHORT-NAME>CompilerA</SHORT-NAME>
            </COMPILER>
          </COMPILERS>
          <MC-SUPPORT>
            <MC-PARAMETER-INSTANCES>
              <MC-DATA-INSTANCE>
                <SHORT-NAME>McData_A</SHORT-NAME>
              </MC-DATA-INSTANCE>
            </MC-PARAMETER-INSTANCES>
          </MC-SUPPORT>
        </BSW-IMPLEMENTATION>
```

```
      </ELEMENTS>
    </AR-PACKAGE>
  </AR-PACKAGES>
</AUTOSAR>
```

F.2.3 Splitkey for Association

If the ≪atpSplitable≫ property represents an association, the value of the splitkey is given by the role name of the association and the full qualified path name of the referenced element.

For example the FMFeatureModel (FM) refers to two FMFeature elements (Feature1, Feature2).

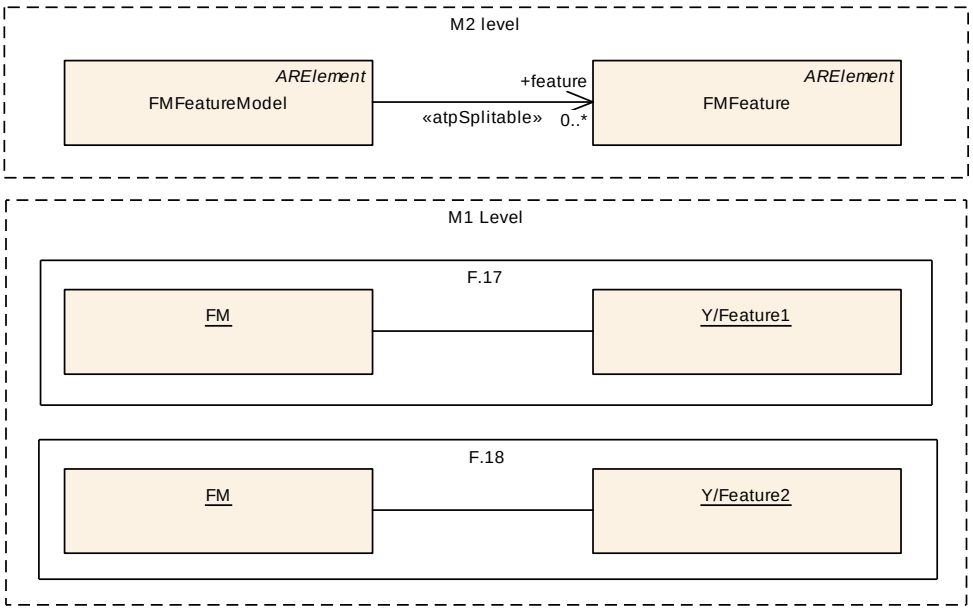


Figure F.4: M1 Model splitable elements with Association

The figure F.4 shows one FMFeatureModel (FM) with two references to FMFeature (Feature1, Feature2) distributed in two files F.17 and F.18. The multiplicity is 2 and the split key is the role name "feature" with the full qualified path names (/Y/Feature1, /Y/Feature2).

The following ARXML files illustrate the use case and the distribution in two partial models. Listing F.17 contains reference to FMFeature (Feature1).

Listing F.17: Example for splitable (Role of assciation) in partial model 1

```
<?xml version="1.0"?>
<AUTOSAR xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http:
    //autosar.org/schema/r4.0" xsi:schemaLocation="http://autosar.org/schema
    /r4.0␣AUTOSAR_00046.xsd">
     <AR-PACKAGES>
         <AR-PACKAGE>
             <SHORT-NAME>x</SHORT-NAME>
```

```
            <ELEMENTS>
                <FM-FEATURE-MODEL>
                    <SHORT-NAME>FM</SHORT-NAME>
                    <FEATURE-REFS>
                        <FEATURE-REF DEST="FM-FEATURE">/Y/Feature1</FEATURE
                            -REF>
                    </FEATURE-REFS>
                </FM-FEATURE-MODEL>
            </ELEMENTS>
        </AR-PACKAGE>
    </AR-PACKAGES>
</AUTOSAR>
```

Listing F.18 contains reference to `FMFeature` (Feature2).

Listing F.18: Example for splitable (Role of assciation) in partial model 2

```
<?xml version="1.0"?>
<AUTOSAR xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http:
    //autosar.org/schema/r4.0" xsi:schemaLocation="http://autosar.org/schema
    /r4.0␣AUTOSAR_00046.xsd">
    <AR-PACKAGES>
        <AR-PACKAGE>
            <SHORT-NAME>x</SHORT-NAME>
            <ELEMENTS>
                <FM-FEATURE-MODEL>
                    <SHORT-NAME>FM</SHORT-NAME>
                    <FEATURE-REFS>
                        <FEATURE-REF DEST="FM-FEATURE">/Y/Feature2</FEATURE
                            -REF>
                    </FEATURE-REFS>
                </FM-FEATURE-MODEL>
            </ELEMENTS>
        </AR-PACKAGE>
    </AR-PACKAGES>
</AUTOSAR>
```

After merging the splitables F.17 and F.18 the merged model F.19 contains again the two references to `FMFeature` (Feature1, Feature2).

Note: The merged model exists only internally in the AUTOSAR tool. The listing is shown only for illustration.

Listing F.19: Example for Merged Model (Role of assciation)

```
<?xml version="1.0"?>
<AUTOSAR xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http:
    //autosar.org/schema/r4.0" xsi:schemaLocation="http://autosar.org/schema
    /r4.0␣AUTOSAR_00046.xsd">
    <AR-PACKAGES>
        <AR-PACKAGE>
            <SHORT-NAME>x</SHORT-NAME>
            <ELEMENTS>
                <FM-FEATURE-MODEL>
                    <SHORT-NAME>FM</SHORT-NAME>
                    <FEATURE-REFS>
```

```
                        <FEATURE-REF DEST="FM-FEATURE">/Y/Feature1</FEATURE
                            -REF>
                        <FEATURE-REF DEST="FM-FEATURE">/Y/Feature2</FEATURE
                            -REF>
                    </FEATURE-REFS>
                </FM-FEATURE-MODEL>
            </ELEMENTS>
        </AR-PACKAGE>
    </AR-PACKAGES>
</AUTOSAR>
```

F.2.4 ECUC Parameter

It is allowed to split definitions of type EcucContainerValue to different ECUC Value files, because the aggregation in the meta-model is marked with stereotype ≪atpSplitable≫. Furthermore, EcucParameterValues aggregated inside an EcucContainerValue can again be split to separate files. For example the NvmBlockDescriptor is of type EcucContainerValue and therefore is splitable in several files.

The instantiation in figure F.5 illustrates this in the listings F.20 and F.21. Both files hold the EcucContainerValue of NvM_ConfigId. The upper part in figure F.5 reflects the ParamDef perspective.

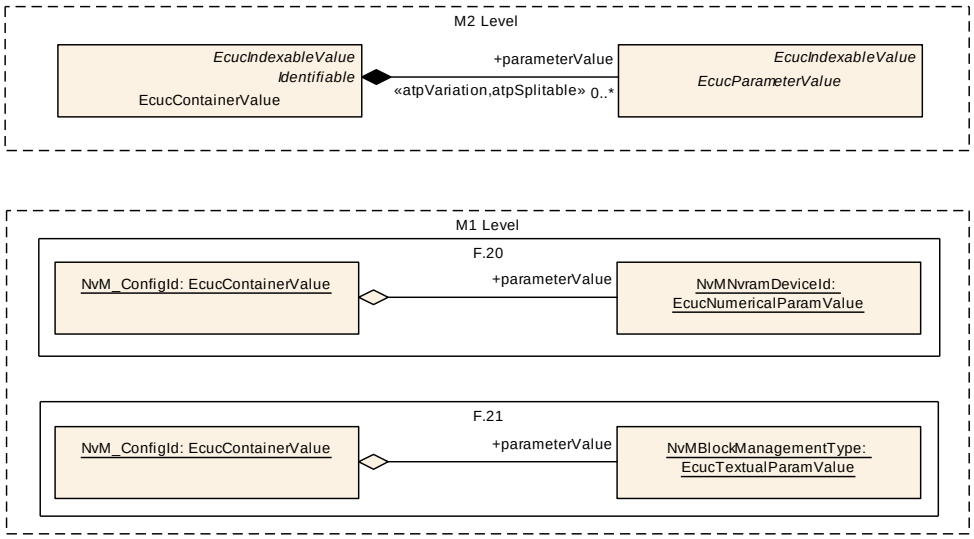


Figure F.5: M1 Model splitable elements in ECUC Value files

The input of the first file (listing) F.20 is the EcucNumericalParamValue (NvMBlockDescriptor/NvMNvramDeviceId) with the value 0.

Listing F.20: Example for splitable with ECUC Parameters in partial model 1

```
<?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http:
    //autosar.org/schema/r4.0" xsi:schemaLocation="http://autosar.org/schema
    /r4.0␣AUTOSAR_00046.xsd">
```

```
  <AR-PACKAGES>
    <AR-PACKAGE>
     <SHORT-NAME>EcucModuleConfigurationValuess</SHORT-NAME>
      <ELEMENTS>
       <ECUC-MODULE-CONFIGURATION-VALUES>
        <SHORT-NAME>NvM</SHORT-NAME>
        <DEFINITION-REF DEST="ECUC-MODULE-DEF">/AUTOSAR/EcucDefs/NvM</
            DEFINITION-REF>
        <CONTAINERS>
         <ECUC-CONTAINER-VALUE>
          <SHORT-NAME>NvM_ConfigId</SHORT-NAME>
          <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/
              EcucDefs/NvM/NvMBlockDescriptor</DEFINITION-REF>
           <PARAMETER-VALUES>
            <ECUC-NUMERICAL-PARAM-VALUE>
             <DEFINITION-REF DEST="ECUC-INTEGER-PARAM-DEF">/AUTOSAR/
                 EcucDefs/NvM/NvMBlockDescriptor/NvMNvramDeviceId</
                 DEFINITION-REF>
              <VALUE>0</VALUE>
            </ECUC-NUMERICAL-PARAM-VALUE>
           </PARAMETER-VALUES>
         </ECUC-CONTAINER-VALUE>
        </CONTAINERS>
       </ECUC-MODULE-CONFIGURATION-VALUES>
      </ELEMENTS>
    </AR-PACKAGE>
  </AR-PACKAGES>
</AUTOSAR>
```

The input of the second file (listing) F.21 is the `EcucTextualParamValue` (NvMBlockDescriptor/NvMBlockManagementType) with the value NVM_BLOCK_NATIVE.

Listing F.21: Example for splitable with ECUC Parameters in partial model 2

```
<?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http:
    //autosar.org/schema/r4.0" xsi:schemaLocation="http://autosar.org/schema
    /r4.0␣AUTOSAR_00046.xsd">
  <AR-PACKAGES>
    <AR-PACKAGE>
     <SHORT-NAME>EcucModuleConfigurationValuess</SHORT-NAME>
      <ELEMENTS>
       <ECUC-MODULE-CONFIGURATION-VALUES>
        <SHORT-NAME>NvM</SHORT-NAME>
        <DEFINITION-REF DEST="ECUC-MODULE-DEF">/AUTOSAR/EcucDefs/NvM</
            DEFINITION-REF>
        <CONTAINERS>
         <ECUC-CONTAINER-VALUE>
          <SHORT-NAME>NvM_ConfigId</SHORT-NAME>
          <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/
              EcucDefs/NvM/NvMBlockDescriptor</DEFINITION-REF>
           <PARAMETER-VALUES>
            <ECUC-TEXTUAL-PARAM-VALUE>
             <DEFINITION-REF DEST="ECUC-ENUMERATION-PARAM-DEF">/AUTOSAR/
                 EcucDefs/NvM/NvMBlockDescriptor/NvMBlockManagementType</
                 DEFINITION-REF>
```

```
            <VALUE>NVM_BLOCK_NATIVE</VALUE>
           </ECUC-TEXTUAL-PARAM-VALUE>
          </PARAMETER-VALUES>
         </ECUC-CONTAINER-VALUE>
        </CONTAINERS>
       </ECUC-MODULE-CONFIGURATION-VALUES>
      </ELEMENTS>
    </AR-PACKAGE>
  </AR-PACKAGES>
</AUTOSAR>
```

EcucNumericalParamValue and EcucTextualParamValue are not splitable. The multiplicity of both classes is set to 1. For example an EcucTextualParamValue can't be split up to separate files one describing the definition the other describing the value, because those elements are not marked with ≪atpSplitable≫ refer to [constr_2525].

Even if both files (listings) (F.20, F.21) would contain EcucTextualParamValue NvMBlockDescriptor/NvMBlockManagementType with different values, e.g. NVM_BLOCK_NATIVE and NVM_BLOCK_REDUNDANT, the validation of the tool will throw an error due to the multiplicity of 1.

After merging the splitables F.20 and F.21 the merged model F.22 contains both EcucContainerValues, EcucNumericalParamValue and EcucTextualParamValue.

Note: The merged model exists only internally in the AUTOSAR tool. The listing is shown only for illustration.

Listing F.22: Example for Merged Model (ECUC Parameters)

```
<?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http:
    //autosar.org/schema/r4.0" xsi:schemaLocation="http://autosar.org/schema
    /r4.0␣AUTOSAR_00046.xsd">
  <AR-PACKAGES>
   <AR-PACKAGE>
     <SHORT-NAME>EcucModuleConfigurationValuess</SHORT-NAME>
      <ELEMENTS>
       <ECUC-MODULE-CONFIGURATION-VALUES>
        <SHORT-NAME>NvM</SHORT-NAME>
        <DEFINITION-REF DEST="ECUC-MODULE-DEF">/AUTOSAR/EcucDefs/NvM</
            DEFINITION-REF>
        <CONTAINERS>
         <ECUC-CONTAINER-VALUE>
          <SHORT-NAME>NvM_ConfigId</SHORT-NAME>
          <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/
              EcucDefs/NvM/NvMBlockDescriptor</DEFINITION-REF>
           <PARAMETER-VALUES>
            <ECUC-NUMERICAL-PARAM-VALUE>
             <DEFINITION-REF DEST="ECUC-INTEGER-PARAM-DEF">/AUTOSAR/
                 EcucDefs/NvM/NvMBlockDescriptor/NvMNvramDeviceId</
                 DEFINITION-REF>
             <VALUE>0</VALUE>
            </ECUC-NUMERICAL-PARAM-VALUE>
```

```
            <ECUC-TEXTUAL-PARAM-VALUE>
             <DEFINITION-REF DEST="ECUC-ENUMERATION-PARAM-DEF">/AUTOSAR/
                EcucDefs/NvM/NvMBlockDescriptor/NvMBlockManagementType</
                DEFINITION-REF>
             <VALUE>NVM_BLOCK_NATIVE</VALUE>
            </ECUC-TEXTUAL-PARAM-VALUE>
           </PARAMETER-VALUES>
          </ECUC-CONTAINER-VALUE>
         </CONTAINERS>
        </ECUC-MODULE-CONFIGURATION-VALUES>
       </ELEMENTS>
     </AR-PACKAGE>
   </AR-PACKAGES>
</AUTOSAR>
```