| Document Title | Specification of Abstract Platform |
|---|---|
| **Document Owner** | AUTOSAR |
| **Document Responsibility** | AUTOSAR |
| **Document Identification No** | 947 |

| | |
|---|---|
| **Document Status** | published |
| **Part of AUTOSAR Standard** | Adaptive Platform |
| **Part of Standard Release** | R19-11 |

| Document Change History | | | |
|---|---|---|---|
| **Date** | **Release** | **Changed by** | **Description** |
| 2019-11-28 | R19-11 | AUTOSAR Release Management | • Initial release |

**Disclaimer**

# Table of Contents

# References

[1] Standardization Template
AUTOSAR_TPS_StandardizationTemplate

[2] Software Component Template
AUTOSAR_TPS_SoftwareComponentTemplate

[3] Generic Structure Template
AUTOSAR_TPS_GenericStructureTemplate

# 1 Introduction

## 1.1 Document Structure

This document contains the specification of the design of an *abstract platform*. The specification document (although currently placed in the *AUTOSAR adaptive platform* (See 1.4.1)) should be seen as being *abstract* of the *AUTOSAR adaptive platform* and *AUTOSAR classic platform*.

The document is structured in the following way:

Section 1 (this chapter) documents the terms, abbreviations, conventions; scope and limitations in the specification and requirement tracing.

Section 2 provides a description of the big picture, sets the background reasons and motivation for the specification and usage principles for intended stakeholders. Additionally, the general modeling approach and modeling decisions are described.

Section 3 dives into the design aspects of an *abstract platform*. The modeling is described along with constraints and requirement specifics. The sub-sections follow the main use-cases: introduction of new meta-classes and description of existing meta-classes to realize the design of an abstract platform and...

Section 4 ...annotation and traceability of requirements.

## 1.2 Abbreviations

The following table contains a list of abbreviations used in the scope of this document along with the spelled-out meaning of each of the abbreviations.

| *Abbreviation* | *Meaning* |
|---|---|
| AP | Adaptive Platform |
| API | Application Programming Interface |
| ARXML | AutosarR XML |
| ASD | Abstract System Description |
| CP | Classic Platform |
| ECU | Electrical Control Unit |
| GENIVI | GENeva In-Vehicle Infotainment |
| IO | Interface Description Language |
| IO | Input/Output |
| JSON | JavaScript Object Notation |

▽

△

| *Abbreviation* | *Meaning* |
|---|---|
| NVM | Non Volatile Memory |
| OEM | Original Equipment Manufacturer |
| OS | Operating System |
| RPC | Remote Procedure Call |
| SOA | Service-Oriented Architecture |
| SWC | Software Component |
| SYSML | Systems Modelling Language |
| VFB | Virtual Functional Bus |
| VISS | Vehicle Information Service Specification |
| W3C | World Wide Consortium |
| XML | Extensible Markup Language |
| XSD | XML Schema Definition |

**Table 1.1: Abbreviations used in the scope of this Document**

## 1.3 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:

| *Class* | **AUTOSAR** | | | |
|---|---|---|---|---|
| ***Package*** | M2::AUTOSARTemplates::AutosarTopLevelStructure | | | |
| ***Note*** | Root element of an AUTOSAR description, also the root element in corresponding XML documents.<br>**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.<br>**Tags:**xml.sequenceOffset=10 |
| arPackage | ARPackage | * | aggr | This is the top level package in an AUTOSAR model.<br>**Stereotypes:** atpSplitable; atpVariation<br>**Tags:**<br>atp.Splitkey=shortName, variationPoint.shortLabel<br>vh.latestBindingTime=blueprintDerivationTime<br>xml.sequenceOffset=30 |
| fileInfo Comment | FileInfoComment | 0..1 | aggr | This represents a possibility to provide a structured comment in an AUTOSAR file.<br>**Stereotypes:** atpStructuredComment<br>**Tags:**<br>xml.roleElement=true<br>xml.sequenceOffset=-10<br>xml.typeElement=false |
| introduction | DocumentationBlock | 0..1 | aggr | This represents an introduction on the Autosar file. It is intended for example to rpresent disclaimers and legal notes.<br>**Tags:**xml.sequenceOffset=20 |

**Table 1.2: AUTOSAR**

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 ([1]).

The representation of requirements in AUTOSAR documents follows the table specified in [TPS_STDT_00078], see Standardization Template, chapter Support for Traceability ([1]).

## 1.4 Scope and Limitations

An *abstract platform* description is purely functional description and not topological.

As with any system description using the AUTOSAR model, an *abstract platform* description has its natural borders. The base of an *abstract platform* description shall be an AUTOSAR `System` description. The depth of the *abstract platform* description is down to the definition of application level data types.

### 1.4.1 Caveats in 19-11

The effect of the UML Tag **"'mmt.RestrictToStandards"'** as specified in [TPS_GST_00372] in the context of an *abstract platform* shall be taken over provisionally. This specification is currently placed in the *AUTOSAR adaptive platform*, but shall in future releases be in an as yet undefined platform.

The reader should therefore be aware that the appearance and descriptions of some model elements in generated artifacts may be viewed as erroneous because they are tagged currently with "'AP"' (*meaning AUTOSAR adaptive platform*) with the caveat that this will in future changed to an *abstract platform* specific value for **"'mmt.RestrictToStandards"'**. Due to this caveat, it means there may not always be an AUTOSAR constraint to restrict the usage of the artifact to an as yet undefined platform as this is deemed to be a temporary case. This caveat is further grounded by the current status of this specification as *DRAFT*.

# 2 Methodology

## 2.1 Background

The existing AUTOSAR meta-model provides a means to comprehensively design and deploy applications on *AUTOSAR classic platform* `ECU`s and *AUTOSAR adaptive platform* `Machine`s. Depending on the intended chosen platform for concrete deployment, the feature/function design model is (intentionally) tightly coupled to the choice of platform. A system designer is drawn *a priori* into a concrete decision whether to design and deploy on *AUTOSAR adaptive platform* or *AUTOSAR classic platform* or indeed non-AUTOSAR platform. The design choices become therefore biased by the intended deployment platform.

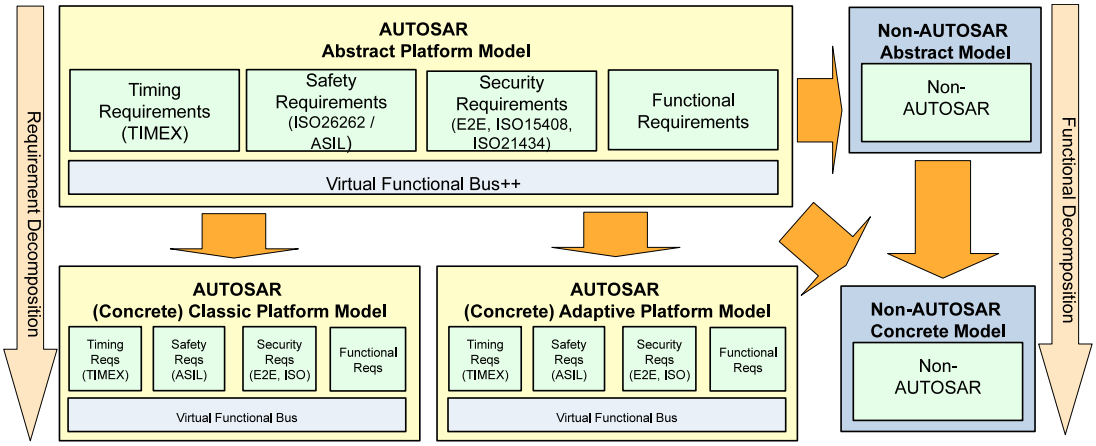


**Figure 2.1: Placement of an *abstract platform***

An system designer at an early **software** design stage may not necessarily care about for example what type of concrete component shall implement the function, or, which type of concrete interface provides the required data; rather the designer just wants to model the interaction between the functional software blocks: i.e. the signal name carrying the data, the directional flow of the data (providers/consumers) and the physical unit of the data on a high level and leave the further refinement of the design or indeed implementation details to a downstream stage, i.e. separation of concerns.

## 2.2 Usage

The specification aims to provide a software based system description of a functional model. It further allows requirement annotation and general traceability of model elements including requirements and functional elements. The abstract description may provide a higher level view of a system, to help a system designers "'step back"' from early decisions about deployment, or indeed whether to defer that decision to a downstream design step or to a supplier(s).

While the principal use-cases are founded for *AUTOSAR adaptive platform* and *AUTOSAR classic platform*, it is not (by design) intended to be exclusive to those platforms. Usage with *other* automotive or non-automotive domains should also be possible as shown in Figure 2.2

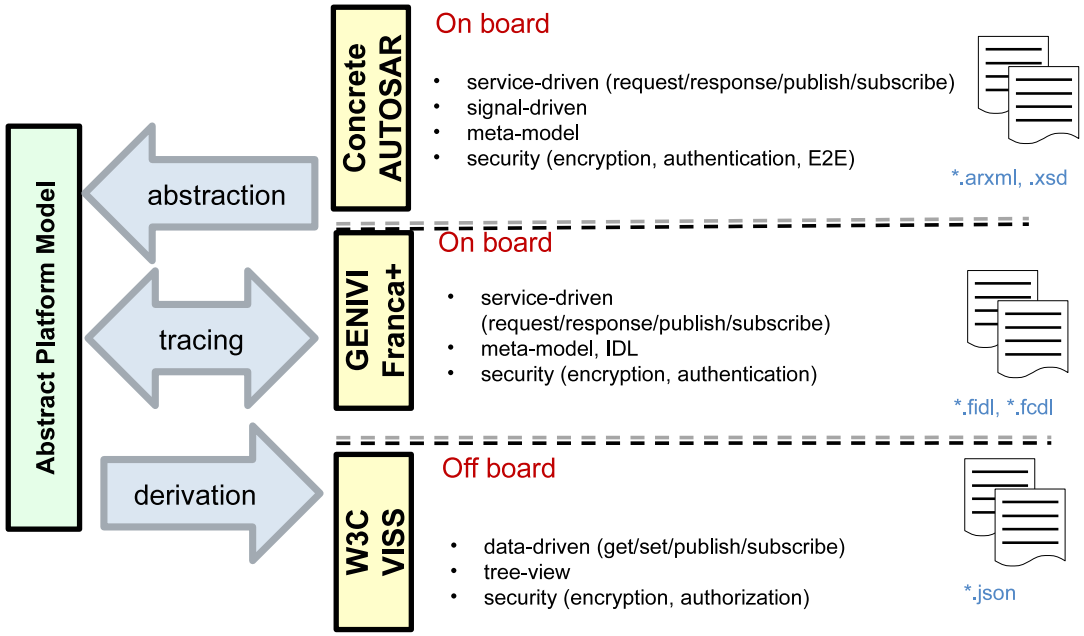


**Figure 2.2: Relationship to non-AUTOSAR systems**

There is not a *hard* modeling dependency between an *abstract platform* and a *AUTOSAR concrete platform* view in the sense that the concrete level depends on the abstract. The methodological approach does not forbid a system designer bypassing entirely an *abstract platform* model and designing only in an *AUTOSAR concrete platform* model to achieve the desired design and deployment. Nevertheless, with the support for traceability, it should be similarly entirely possible to create an *abstract platform* out of an *AUTOSAR concrete platform* description.

## 2.3 Modeling Approach

### 2.3.1 Model Choice

If the goal is to allow design of an abstract platform, it could be argued that the chosen model should also be abstract (of AUTOSAR). However, while the abstract design should be open to designers of *non-AUTOSAR platform*s to utilize, the primary focus is usage within the AUTOSAR domain i.e. *AUTOSAR adaptive platform* and *AUTOSAR classic platform*s. For that reason, the argumentation of using the AUTOSAR meta-model as the basis for the modeling approach is solidified. The *abstract platform* is designed using the AUTOSAR meta-model, but should not restrict usage of abstract designs to AUTOSAR.

### 2.3.2 Bottom-up vs Top-down

Based on the assumption of the model choice in 2.3.1, the next point is how to approach the creation of an abstraction. In very general terms, there are two possible approaches to designing an abstraction layer: bottom-up and top-down.

A bottom-up approach essentially involves analyzing the existing *AUTOSAR adaptive platform*, *AUTOSAR classic platform* and those *non-AUTOSAR platform*s considered in the current scope and creating an *abstract platform* design based on (but not exclusive to) the needs of these *concrete platform*s. With this approach, the *abstract platform* design is better guaranteed to fit well with the existing platforms. A potential disadvantage is that the design is still somewhat coupled to the existing *AUTOSAR concrete platform*s.

A top-down approach involves designing a new green-fields *abstract platform* and 'making it fit' with the existing *AUTOSAR adaptive platform*, *AUTOSAR classic platform* and those *non-AUTOSAR platform*s. While offering more freedom to design, there is a risk of specifying an *abstract platform* which, in the end, is too distant from the needs of the existing platforms. The general approach therefore is to favor the bottom-up method.

### 2.3.3 Meta-class selection

Having decided on the general approach for the design of the abstract platform, the next question is which approach to use regarding meta-class selection, i.e. re-use existing meta-classes or re-design new meta-classes.

While the *AUTOSAR adaptive platform* and *AUTOSAR classic platform* are based on different architecture principles, they mostly share the same modeling principles on VFB level and thus the VFB modeling. The approach is therefore to examine the VFB level model in both platforms as a primary basis and the non-AUTOSAR platforms as a secondary basis. The existing AUTOSAR meta-model, especially the specification of the AUTOSAR Software-Component Template [2] already provides a good basis to comprehensively design a software component model. The principles therein may also be found in other more generic non-AUTOSAR component models.

The conclusion is to opt for re-use in so far as it makes theoretical sense on a case-by-case basis. It may be that any given identical meta-class may be used in both the *abstract platform* and *AUTOSAR concrete platform*. This approach is similar to that used when designing the *AUTOSAR adaptive platform* meta-model, and similarly, it will then be necessary to either extend meta-classes with *abstract platform* specifics, or where applicable, constrain them to the *abstract platform*.

## 2.4 Model Creation

An *abstract platform* description can either be created from scratch or by abstraction.

If an *abstract platform* description is created from scratch, then a designer has a free hand to model the vehicle communications using a *green fields* approach. Due to the manual overhead in creating such a description from scratch, this is perhaps less of a likely scenario than the 2nd approach.

If an *abstract platform* description is created by abstraction a designer could create it by taking an existing *concrete platform* model as a basis for the content. This in practice means that this form of *abstract platform* description is immediately *more* valid than the former approach because it already has a basis in a *concrete platform* description, the same holds for tracing between the *concrete platform* and *abstract platform* models. This approach would also allow for an automated creation of an *abstract platform* description.

# 3 Abstract System Design

## 3.1 Abstract Platform Design

An abstract platform system description provides the possibility to achieve a higher-level software view on the system. An architect can decide during design time which type of downstream AUTOSAR system description to use.

A level of architectural freedom through abstraction is attained by formally describing the functional interactions on a *component model* level, but without fixing details of any downstream implementation platform.

**[TPS_APSD_01000]**{DRAFT} **Principle of an abstract platform system description** ⌈The content of an *abstract platform* system description shall allow a platform independent specification of the functional interactions of software components in the vehicle communications matrix.⌋*()*

**[TPS_APSD_01001]**{DRAFT} **Modeling of vehicle communications in an abstract platform** ⌈The abstract platform description should encompass formal model elements needed to derive a vehicle communications abstraction.⌋*()*

**[TPS_APSD_01002]**{DRAFT} **Agnosticism of deployment modeling artifacts in an abstract platform** ⌈There shall be no architecture/platform specific modeling appear at this level, except when explicitly intended and annotated/described explicitly though model artifacts.⌋*()*

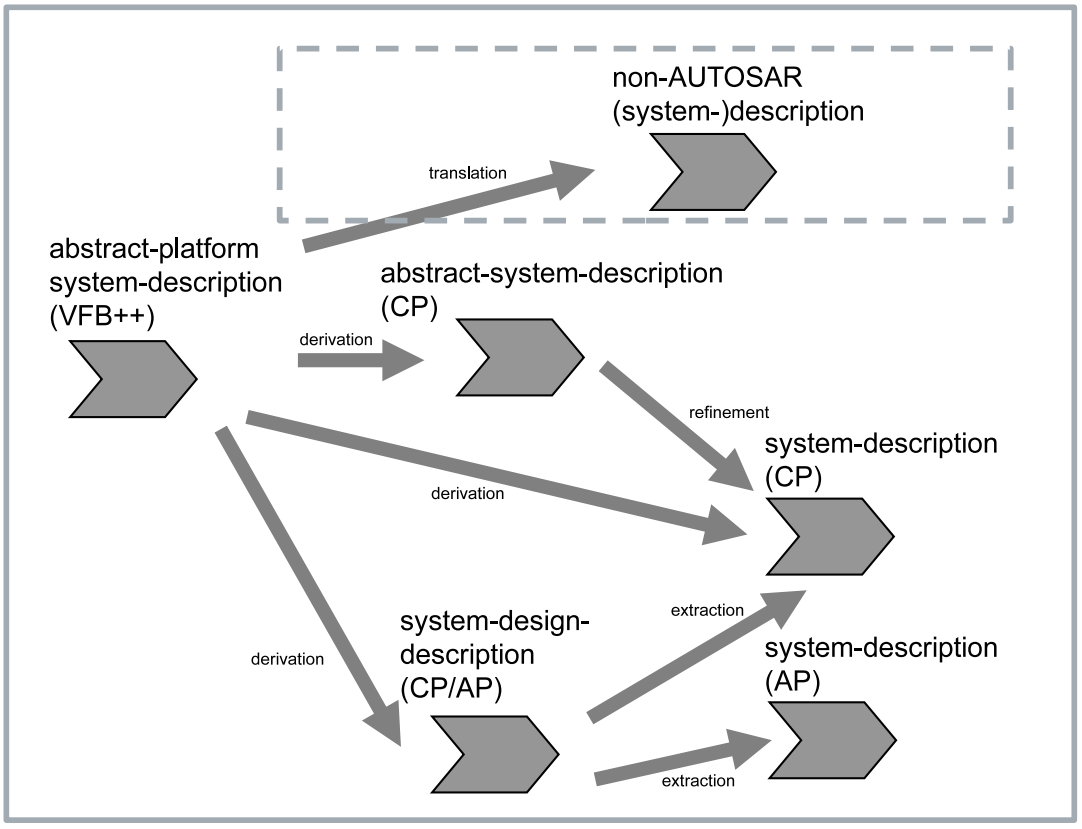


**Figure 3.1: Abstract Platform System Description in Methodology**

**[TPS_APSD_01003]**{DRAFT} **Exclusion of abstract platform artifacts to an AUTOSAR concrete platform** ⌈To allow a completely independent design of an *AUTOSAR concrete platform*, it should be avoided that meta-model artifacts defined within the context of an *abstract platform* are used in a *AUTOSAR concrete platform*.⌋ *()*

## 3.2 System Description

As per existing system descriptions in *AUTOSAR adaptive platform* and *AUTOSAR classic platform*s, an *abstract platform* needs its own system description to distinguish *abstract platform* content from other types of system descriptions/extracts. The basis for all AUTOSAR system descriptions/extracts is the meta-class `System` and as with other AUTOSAR system descriptions, the `category` shall be used to identify the content.

**[TPS_APSD_01004]**{DRAFT} **System `category` for a system description with Abstract Platform content** ⌈The `System` element that contains design artifacts that are relevant for an Abstract Platform shall have the `category` `ABSTRACT_PLATFORM_SYSTEM_DESCRIPTION`.⌋*()*

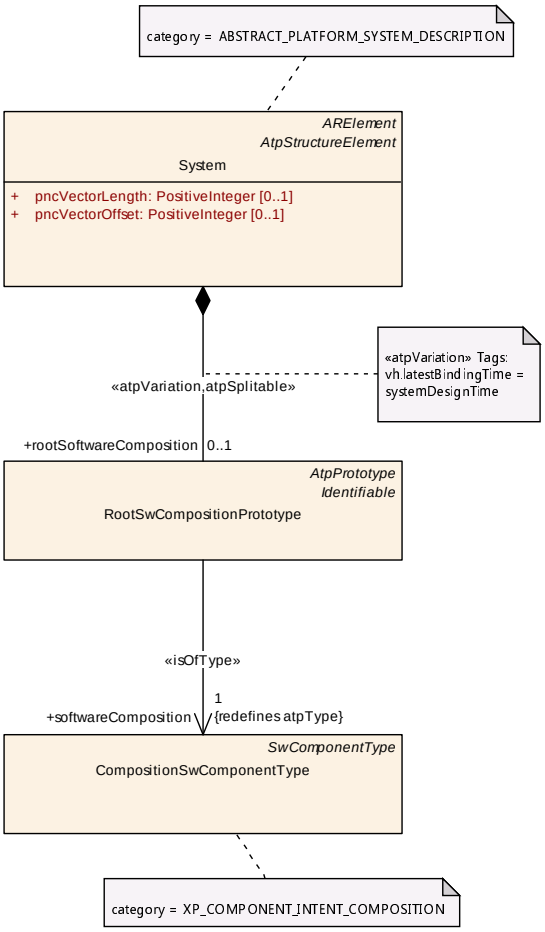


**Figure 3.2: Modeling of an Abstract Platform `System`**

| *Class* | **System** | | | |
|---|---|---|---|---|
| ***Package*** | M2::AUTOSARTemplates::SystemTemplate | | | |
| ***Note*** | The top level element of the System Description.<br>**Tags:**atp.recommendedPackage=Systems | | | |
| ***Base*** | *ARElement*, *ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable* | | | |
| ***Attribute*** | ***Type*** | ***Mult.*** | ***Kind*** | ***Note*** |
| fibexElement | FibexElement | * | ref | Reference to ASAM FIBEX elements specifying Communication and Topology.<br>All Fibex Elements used within a System Description shall be referenced from the System Element.<br>atpVariation: In order to describe a product-line, all Fibex Elements can be optional.<br>**Stereotypes:** atpVariation<br>**Tags:**vh.latestBindingTime=postBuild |
| mapping | SystemMapping | * | aggr | Aggregation of all mapping aspects relevant in the System Description.<br>**Stereotypes:** atpSplitable; atpVariation<br>**Tags:**<br>atp.Splitkey=shortName, variationPoint.shortLabel<br>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. |
| 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.<br>atpVariation: The RootSwCompositionPrototype can vary.<br>**Stereotypes:** atpSplitable; atpVariation<br>**Tags:**<br>atp.Splitkey=shortName, variationPoint.shortLabel<br>vh.latestBindingTime=systemDesignTime |

**Table 3.1: System**

**[constr_6800]**{DRAFT} **Non-relevance of `FibexElement` and `SystemMapping` for a `System` description with Abstract Platform content** ⌈Any `System` with the `category` set to `ABSTRACT_PLATFORM_SYSTEM_DESCRIPTION` shall not:

- reference a `FibexElement` in the role `fibexElement`.
- aggregate a `SystemMapping`.

⌋()

**[constr_6801]**{DRAFT} **Non-relevance of the attributes `pncVectorLength`, `pncVectorOffset` for a `System` description with Abstract Platform content** ⌈Any `System` with the `category` set to `ABSTRACT_PLATFORM_SYSTEM_DESCRIPTION` shall ignore the attributes `pncVectorLength`, `pncVectorOffset`.⌋()

| *Class* | **RootSwCompositionPrototype** | | | |
|---|---|---|---|---|
| *Package* | M2::AUTOSARTemplates::SystemTemplate | | | |
| *Note* | The RootSwCompositionPrototype represents the top-level-composition of software components within a given System. According to the use case of the System, this may for example be the a more or less complete VFB description, the software of a System Extract or the software of a flat ECU Extract with only atomic SWCs.<br>Therefore the RootSwComposition will only occasionally contain all atomic software components that are used in a complete VFB System. The OEM is primarily interested in the required functionality and the interfaces defining the integration of the Software Component into the System. The internal structure of such a component contains often substantial intellectual property of a supplier. Therefore a top-level software composition will often contain empty compositions which represent subsystems.<br>The contained SwComponentPrototypes are fully specified by their SwComponentTypes (including Port Prototypes, PortInterfaces, VariableDataPrototypes, SwcInternalBehavior etc.), and their ports are interconnected using SwConnectorPrototypes. | | | |
| *Base* | *ARObject*, *AtpFeature*, *AtpPrototype*, *Identifiable*, *MultilanguageReferrable*, *Referrable* | | | |
| ***Attribute*** | ***Type*** | ***Mult.*** | ***Kind*** | ***Note*** |
| software Composition | CompositionSw ComponentType | 1 | tref | We assume that there is exactly one top-level composition that includes all Component instances of the system<br>**Stereotypes:** isOfType |

**Table 3.2: RootSwCompositionPrototype**

**[constr_6802]**{DRAFT} **Restriction of the `category` of a `CompositionSwComponentType` which types a `RootSwCompositionPrototype` in a `System` description with Abstract Platform content** ⌈Any `System` with the `category` set to `ABSTRACT_PLATFORM_SYSTEM_DESCRIPTION` which aggregates a `RootSwCompositionPrototype` which in turn is typed by a `CompositionSwComponentType` shall have the `category` set to:

- `XP_COMPONENT_INTENT_COMPOSITION`: the `CompositionSwComponentType` represents an actual root software composition.

⌋()

## 3.3 Component Design

Generic software component models are (generally) quite similar in nature. The AUTOSAR `VFB` level component model follows those basic principles.

**[TPS_APSD_01005]**{DRAFT} **Identification of component types in an abstract platform** ⌈The *abstract platform* shall exploit the `category` of the component as a means to optionally identify the intent behind the component.⌋()

The abstract component design shall focus on allowing a component design which does not force any intended downstream usage to the designer, but nevertheless allows a limited set of indicators [TPS_APSD_01005] to identify the intent of the component.

There are two main areas to considered in an abstract component model: compositions and typing of components.

**[TPS_APSD_01006]**{DRAFT} **Recursive component definition in an abstract platform** ⌈An abstract component design shall support recursive depth-wise definition of components.⌋*()*

This [TPS_APSD_01006] is no different than in *AUTOSAR adaptive platform* and *AUTOSAR classic platform* which handle compositions the same. The *abstract platform* shall take over the existing meta-classes used in the AUTOSAR meta-model as the basis for the abstract component model.

In an *AUTOSAR adaptive platform*, a designer should have the freedom to design recursive abstract components and defer decomposition to a later stage in the process. Unlike in an *AUTOSAR concrete platform*, components are always composite; atomic components (or meta-classes thereof) are not explicitly identifiable.

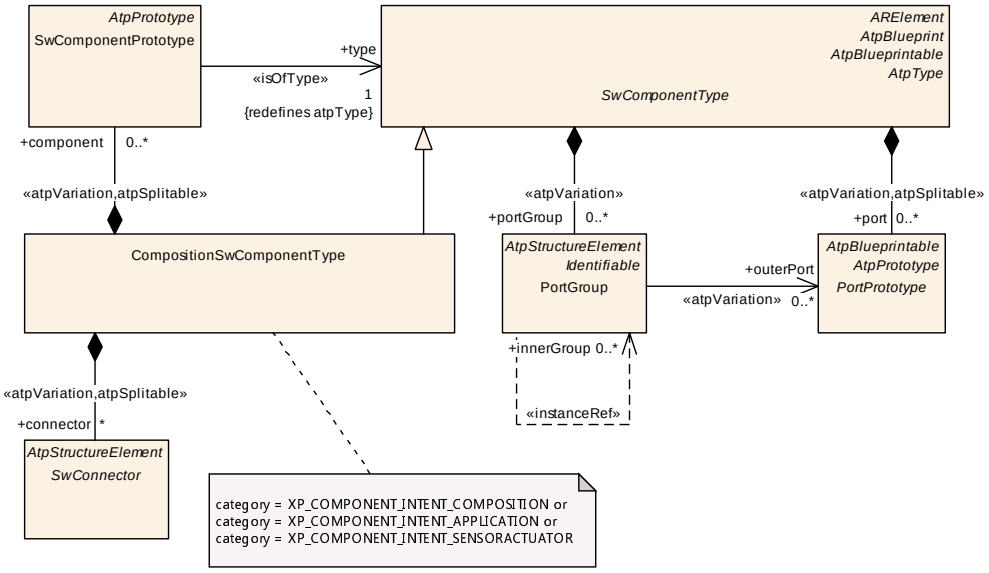


**Figure 3.3: Modeling of Abstract Platform Components**

| ***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.<br>**Tags:**atp.recommendedPackage=SwComponentTypes | | | |
| ***Base*** | *ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *SwComponentType* | | | |
| ***Attribute*** | ***Type*** | ***Mult.*** | ***Kind*** | ***Note*** |

▽

△

| *Class* | **CompositionSwComponentType** | | | |
|---|---|---|---|---|
| component | SwComponent Prototype | * | aggr | The instantiated components that are part of this composition.<br>**Stereotypes:** atpSplitable; atpVariation<br>**Tags:**<br>atp.Splitkey=shortName, variationPoint.shortLabel<br>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.<br>The aggregation of SwConnectors is subject to variability with the purpose to support variant data flow.<br>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.<br>**Stereotypes:** atpSplitable; atpVariation<br>**Tags:**<br>atp.Splitkey=shortName, variationPoint.shortLabel<br>vh.latestBindingTime=postBuild |
| constantValue Mapping | ConstantSpecification MappingSet | * | ref | Reference to the ConstantSpecificationMapping to be applied for initValues of PPortComSpecs and RPortCom Spec.<br>**Stereotypes:** atpSplitable<br>**Tags:**atp.Splitkey=constantValueMapping |
| dataType Mapping | DataTypeMappingSet | * | ref | Reference to the DataTypeMapping to be applied for the used ApplicationDataTypes in ServiceInterfaces.<br>**Stereotypes:** atpSplitable<br>**Tags:**atp.Splitkey=dataTypeMapping |

**Table 3.3: CompositionSwComponentType**

**[constr_6813]**{DRAFT} **Restriction of `SwComponentType`s in an Abstract Platform** ⌈In a `System` with the `category` set to `ABSTRACT_PLATFORM_SYSTEM_DESCRIPTION` any `SwComponentPrototype` shall not reference:

- a `AtomicSwComponentType` in the role `type`.
- a `ParameterSwComponentType` in the role `type`.

Such that these types are excluded from typing a `SwComponentPrototype`⌋*()*

**[constr_6803]**{DRAFT} **Restriction of the `category` of a `CompositionSwComponentType` which references a `SwComponentPrototype` in a `System` description with Abstract Platform content** ⌈In a `System` with the `category` set to `ABSTRACT_PLATFORM_SYSTEM_DESCRIPTION` any `CompositionSwComponentType` which is referenced by a `SwComponentPrototype` in the `type` shall have the `category` set to:

- `unspecified`: the `CompositionSwComponentType` represents an as yet unknown or unspecified software component.

- XP_COMPONENT_INTENT_COMPOSITION: the CompositionSwComponentType represents an actual composite software component.
- XP_COMPONENT_INTENT_APPLICATION: the CompositionSwComponentType represents an application software component.
- XP_COMPONENT_INTENT_SENSORACTUATOR: the CompositionSwComponentType represents a sensor or actuator software component.

⌋()

Rationale for the existence of [constr_6803]: Without such a indicator, it is very arbitrary how to trace between an *abstract platform* and *concrete platform* - foreseeably the abstract component could only be derived by default to say an arbitrary representation in a downstream platform and it would be a pure manual step and not allow for any future automation. The intent should therefore allow an architect to avoid specifics but facilitate automation of these decisions.

**[constr_6804]**{DRAFT} **Non-relevance of ConstantSpecificationMappingSet and DataTypeMappingSet for a CompositionSwComponentType in an Abstract Platform** ⌈In a System with the category set to ABSTRACT_PLATFORM_SYSTEM_DESCRIPTION any CompositionSwComponentType which is referenced by a SwComponentPrototype in the role type shall not reference:

- a ConstantSpecificationMappingSet in the role constantValueMapping.
- a DataTypeMappingSet in the role dataTypeMapping.

⌋()

## 3.4 Port Design

Abstract ports shall follow the general rule of being designated input or output at design time. Assigning a port to be both input and output shall not be possible. Note: it would be still possible to group two ports (one input, one output) into a logical port grouping which could be represented as a single IO port in a *concrete platform*. This [TPS_APSD_01007] may change in future releases.

**[TPS_APSD_01007]**{DRAFT} **Prototyping of ports in an abstract platform** ⌈An abstract level port is *either* in the role of consumer or provider but not both.⌋()

One important difference with an abstract port in contrast to a concrete port is that of [TPS_APSD_01008]. It does not imply a SOA as in a *AUTOSAR adaptive platform* ServiceInterface. Neither does it hard-type a particular functional usage to a peer port as in for example a PersistencyInterface or NvDataInterface.

Nevertheless, it is important at an early abstract design stage to be able to model that a port at a later design stage is intended for a certain use. For this reason it is possible

to optionally indicate the port intent via the `category`. This serves as a hint which may be optionally considered when deriving (if it has a semantical meaning on the downstream platform).

**[TPS_APSD_01008]**{DRAFT} **Generic typing of interfaces in an abstract platform** ⌈An abstract level port interface shall be generic.⌋*()*

Port grouping is fairly standard in component models, though it is really at the discretion of the model itself what the semantic meaning of a port group is. Several scenarios are possible such as limiting inclusion of discrete ports in discrete groups or allowing discrete ports to be mapped into different groups.

Some models define an abstract port group as being a composition which may be further decomposed in a downstream platform. The AUTOSAR model allows ports to be optionally contained in groups, and can be reused in an *abstract platform*.

**[TPS_APSD_01009]**{DRAFT} **Grouping of ports in an abstract platform** ⌈Assigning discrete ports to zero or more port groups shall be possible.⌋*()*

Further rationale for [TPS_APSD_01009] is that not only since this represents a logical grouping in the abstract level, but also to trace to a port grouping in a *concrete platform* if it is supported.

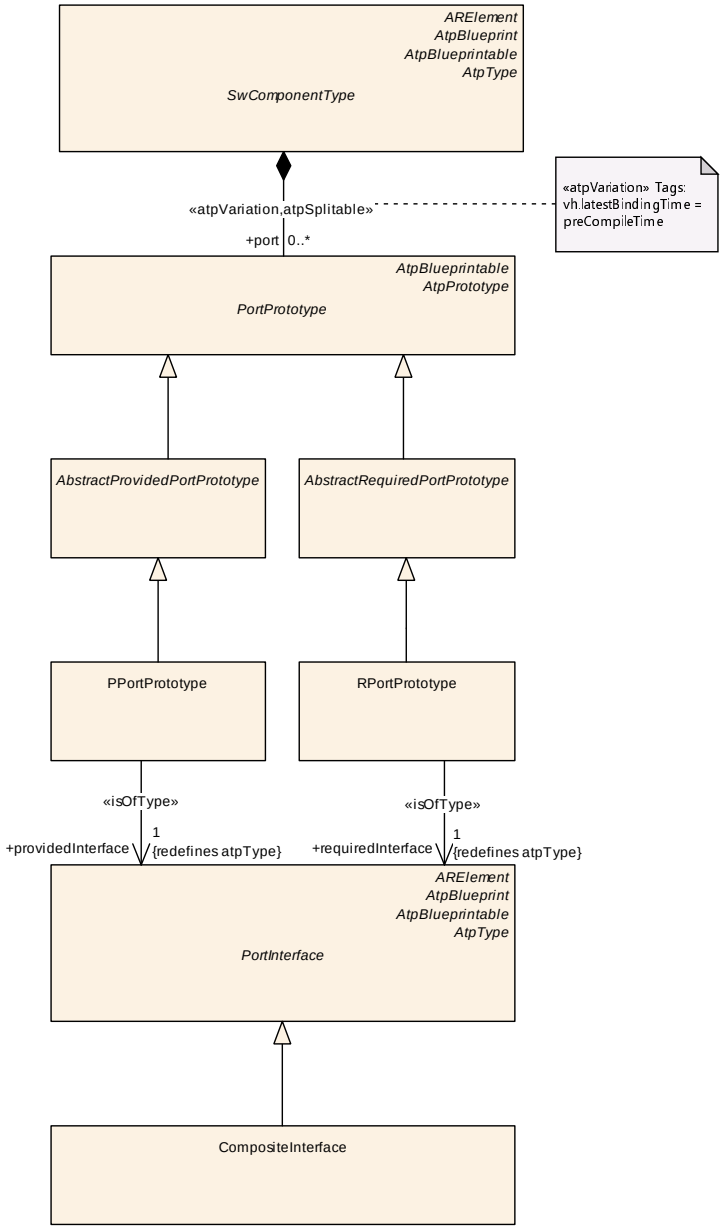


**Figure 3.4: Modeling of Abstract Platform Ports**

| | | | | |
|---|---|---|---|---|
| ***Class*** | ***PortPrototype*** (abstract) | | | |
| ***Package*** | M2::AUTOSARTemplates::SWComponentTemplate::Components | | | |
| ***Note*** | Base class for the ports of an AUTOSAR software component.<br>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*** |

△

| *Class* | ***PortPrototype*** (abstract) | | | |
|---|---|---|---|---|
| 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. |
| 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. |
| portPrototype Props | PortPrototypeProps | 0..1 | aggr | This attribute allows for the definition of further qualification of the semantics of a PortPrototype.<br>**Tags:**atp.Status=draft |
| senderReceiver Annotation | SenderReceiver Annotation | * | aggr | Collection of annotations of this ports sender/receiver communication. |
| triggerPort Annotation | TriggerPortAnnotation | * | aggr | Annotations on this trigger port. |

**Table 3.4: PortPrototype**

| *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, CompositeInterface, *DataInterface*, *DiagnosticPortInterface*, ModeSwitchInterface, *PersistencyInterface*, *PlatformHealthManagementInterface*, RawDataStreamInterface, RestService Interface, ServiceInterface, *TimeSynchronizationInterface*, TriggerInterface | | | |
| ***Attribute*** | ***Type*** | ***Mult.*** | ***Kind*** | ***Note*** |
| namespace (ordered) | SymbolProps | * | aggr | This represents the SymbolProps used for the definition of a hierarchical namespace applicable for the generation of code artifacts out of the definition of a ServiceInterface.<br>**Stereotypes:** atpSplitable<br>**Tags:**<br>atp.Splitkey=shortName<br>atp.Status=draft |

**Table 3.5: PortInterface**

| *Class* | **PortGroup** |
|---|---|
| ***Package*** | M2::AUTOSARTemplates::SWComponentTemplate::Components |

▽

△

| *Class* | **PortGroup** | | | |
|---|---|---|---|---|
| ***Note*** | Group of ports which share a common functionality, e.g. need specific network resources. This information shall be available on the VFB level in order to delegate it properly via compositions. When propagated into the ECU extract, this information is used as input for the configuration of Services like the Communication Manager. A PortGroup is defined locally in a component (which can be a composition) and refers to the "outer" ports belonging to the group as well as to the "inner" groups which propagate this group into the components which are part of a composition. A PortGroup within an atomic SWC cannot be linked to inner groups. | | | |
| ***Base*** | *ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable* | | | |
| ***Attribute*** | ***Type*** | ***Mult.*** | ***Kind*** | ***Note*** |
| innerGroup | PortGroup | * | iref | Links a PortGroup in a composition to another PortGroup, that is defined in a component which is part of this CompositionSwComponentType. |
| outerPort | PortPrototype | * | ref | Outer PortPrototype of this AtomicSwComponentType which belongs to the group. A port can belong to several groups or to no group at all.<br>**Stereotypes:** atpVariation<br>**Tags:**vh.latestBindingTime=preCompileTime |

**Table 3.6: PortGroup**

**[constr_6805]**{DRAFT} **Non-relevance of `PRPortPrototype` for a `System` with Abstract Platform content** ⌈An `CompositeInterface` shall not be referenced by a `PRPortPrototype` in the role `providedRequiredInterface`.⌋*()*

Rationale for [constr_6805]: The usage of `PRPortPrototype` is tightly coupled to reading/writing to `NVM` which is out of scope for an *abstract platform* description.

**[constr_6806]**{DRAFT} **Restriction of the `category` of a `PortInterface` for a `System` description with Abstract Platform content** ⌈In a `System` with the `category` set to `ABSTRACT_PLATFORM_SYSTEM_DESCRIPTION` any `PortInterface` may have one of the following categories:

- `unspecified`: the precise usage of the `PortInterface` is as yet unknown or unspecified.
- `XP_PORT_INTENT_CTRL_SECURITY`: the `PortInterface` represents a control port to a security entity: e.g. a cryptographic or authentication entity.
- `XP_PORT_INTENT_CTRL_TIMESYNC`: the `PortInterface` represents a control port to a time synchronization entity: e.g. an *AUTOSAR adaptive platform* `TimeSynchronizationInterface`
- `XP_PORT_INTENT_DATA_STORAGE`: the `PortInterface` represents a port to a storage entity used to hold persistent data: e.g. an *AUTOSAR adaptive platform* `PersistencyInterface` or *AUTOSAR classic platform* `NvDataInterface`
- `XP_PORT_INTENT_DATA_APPLICATION`: the `PortInterface` represents a general application data port.

⌋*()*

## 3.5 Interface Design

**[TPS_APSD_01010]**{DRAFT} **Agnosticism of abstract platform interfaces to middleware deployments** ⌈An *abstract platform* interface shall be agnostic of both architecture and any middleware deployment options.⌋*()*

Unlike in an *AUTOSAR concrete platform* interface, the interface type shall not convey anything relating to the contractual port usage or about the underlying signaling architecture between ports [TPS_APSD_01010].

Nevertheless, in designing an *abstract platform* interface, it is prudent to be aware of design features from *concrete platform* interfaces which could benefit the design. One principal feature which shall be used is the composition/aggregation of elements within the interface.

**[TPS_APSD_01011]**{DRAFT} **Aggregation of interface elements in an abstract platform interface** ⌈An *abstract platform* interface shall aggregate all exchange elements associated with that interface.⌋*()*

This [TPS_APSD_01011] is the design approach normally taken in `SOA` and is in practice in *AUTOSAR adaptive platform* `ServiceInterface` and shall form the basis of the design. An alternative approach would be to use *atomic* interfaces (an interface with only a singular message exchange element) as used in practice in the *AUTOSAR classic platform* `ClientServerInterface` and `SenderReceiverInterface`. However, a *composite* interface is a super-set of the *atomic* in aggregation, offers more flexibility and can be directly applied better to modern architectures.

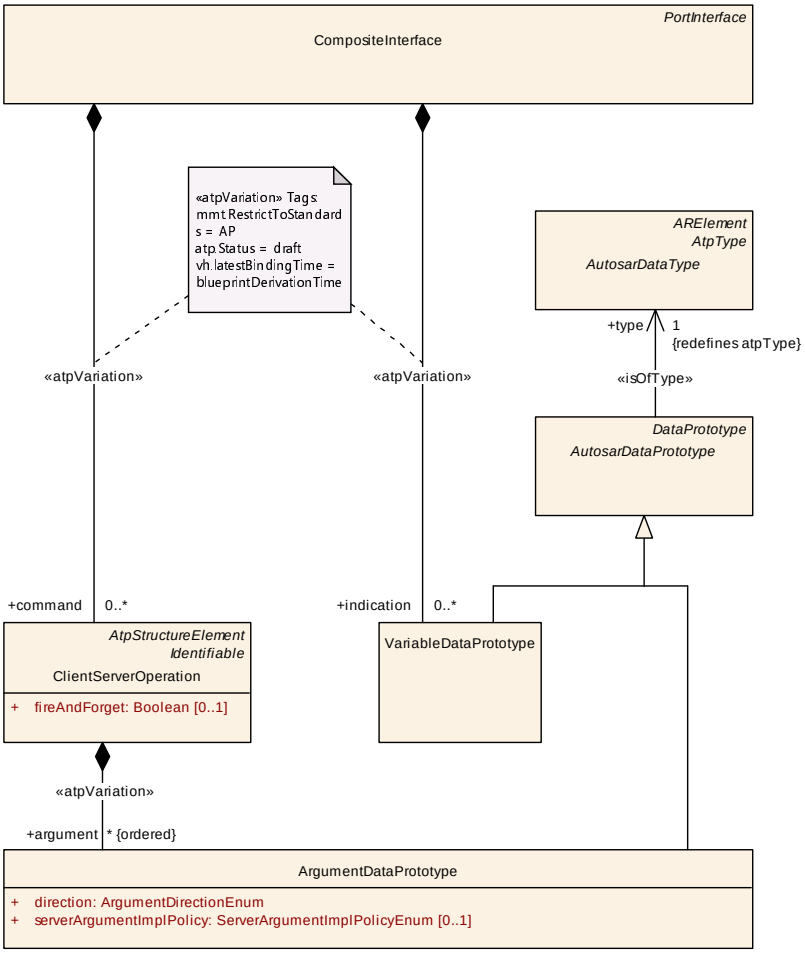


**Figure 3.5: Modeling of Abstract Platform interfaces**

| ***Class*** | **CompositeInterface** | | | |
|---|---|---|---|---|
| ***Package*** | M2::AUTOSARTemplates::AbstractPlatform | | | |
| ***Note*** | This represents the ability to define a PortInterface that consists of a composition of commands and indications.<br>**Tags:**<br>atp.Status=draft<br>atp.recommendedPackage=CompositeInterfaces | | | |
| ***Base*** | *ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable* | | | |
| ***Attribute*** | ***Type*** | ***Mult.*** | ***Kind*** | ***Note*** |
| command | ClientServerOperation | * | aggr | This represents the collection of methods defined in the context of a ServiceInterface.<br>**Stereotypes:** atpVariation<br>**Tags:**<br>atp.Status=draft<br>vh.latestBindingTime=blueprintDerivationTime |

△

| Class | CompositeInterface | | | |
|---|---|---|---|---|
| indication | VariableDataPrototype | * | aggr | This represents the collection of events defined in the context of a ServiceInterface.<br>**Stereotypes:** atpVariation<br>**Tags:**<br>atp.Status=draft<br>vh.latestBindingTime=blueprintDerivationTime |

**Table 3.7: CompositeInterface**

| 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<br>**Stereotypes:** atpVariation<br>**Tags:**vh.latestBindingTime=blueprintDerivationTime |
| fireAndForget | Boolean | 0..1 | attr | This attribute defines whether this method is a fire&forget method (true) or not (false).<br>**Tags:**atp.Status=draft |
| possibleApError | ApApplicationError | * | ref | This reference identifies AdaptivePlatformApplication Errors as a possible error raised by the enclosing Client ServerOperation.<br>**Tags:**atp.Status=draft |
| possibleApError Set | ApApplicationErrorSet | * | ref | This reference represents the ability to refer to an entire group of ApApplicationErrors as one model element instead of having to refer to all the represented Ap ApplicationErrors separately.<br>**Tags:**atp.Status=draft |

**Table 3.8: ClientServerOperation**

| Class | VariableDataPrototype | | | |
|---|---|---|---|---|
| Package | M2::AUTOSARTemplates::SWComponentTemplate::Datatype::DataPrototypes | | | |
| Note | A VariableDataPrototype is used to contain values in an ECU application. This means that most likely a VariableDataPrototype allocates "static" memory on the ECU. In some cases optimization strategies might lead to a situation where the memory allocation can be avoided.<br>In particular, the value of a VariableDataPrototype is likely to change as the ECU on which it is used executes. | | | |
| 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 VariableDataPrototype |

**Table 3.9: VariableDataPrototype**

| *Class* | **ArgumentDataPrototype** | | | |
|---|---|---|---|---|
| ***Package*** | M2::AUTOSARTemplates::SWComponentTemplate::PortInterface | | | |
| ***Note*** | An argument of an operation, much like a data element, but also carries direction information and is owned by a particular ClientServerOperation. | | | |
| ***Base*** | *ARObject*, *AtpFeature*, *AtpPrototype*, *AutosarDataPrototype*, *DataPrototype*, *Identifiable*, *MultilanguageReferrable*, *Referrable* | | | |
| ***Attribute*** | ***Type*** | ***Mult.*** | ***Kind*** | ***Note*** |
| direction | ArgumentDirectionEnum | 1 | attr | This attribute specifies the direction of the argument prototype. |
| serverArgumentImplPolicy | ServerArgumentImplPolicyEnum | 0..1 | attr | This defines how the argument type of the servers RunnableEntity is implemented.<br>If the attribute is not defined this has the same semantics as if the attribute is set to the value useArgumentType for primitive arguments and structures. |

**Table 3.10: ArgumentDataPrototype**

**[constr_6807]**{DRAFT} **Exclusivity of an `CompositeInterface` to an Abstract Platform** ⌈A `CompositeInterface` shall not type a `PortPrototype` unless the `category` of the `System` is `ABSTRACT_PLATFORM_SYSTEM_DESCRIPTION`.⌋*()*

Rationale for [constr_6807]: is grounded in [TPS_APSD_01003].

Within an `CompositeInterface` there shall be 2 forms of exchanging data between ports:

- `command`: a message that shall be exchanged via a command call out (`RPC`) with optional arguments.
- `indication`: a message that shall be exchanged (indicated) with optional arguments.

Further exchange types may be added in future releases. There is no particular behavior associated with these exchange types in the abstract level, rather they are there to inform the *concrete platform* how to realize the modeling and more importantly the implementation in the respective middleware.

**[constr_6808]**{DRAFT} **Non-relevance of the attribute `fireAndForget` for a `ClientServerOperation` used in a `CompositeInterface`** ⌈A `ClientServerOperation` aggregated in a `CompositeInterface` in the role `command` shall ignore the attribute `fireAndForget`.⌋*()*

**[constr_6809]**{DRAFT} **Non-relevance of `ApApplicationError` and `ApApplicationErrorSet` for a `ClientServerOperation` in the context of a `CompositeInterface`** ⌈Any `ClientServerOperation` aggregated in a `CompositeInterface` with the role `command` shall not:

- reference a `ApApplicationError` in the role `possibleApError`.
- reference a `ApApplicationErrorSet` in the role `possibleApErrorSet`.

⌋*()*

Note it is entirely possible that the downstream platform only supports atomic interface types, in this case, in the derivation engineering step the designer must take steps to decompose the composite interface to discrete atomic interfaces. Obviously this has an impact on ports which would need to be created or alternatively some facade pattern employed to aggregate the atomic interfaces.

## 3.6 Connector Design

While support for modeling of port connectors in an *abstract platform* entirely makes sense for certain architectures in others it doesn't. Especially for SOA based platforms it can be argued that they are superfluous - SOA middlewares typically only create the "'connection"' when the service provider is "'found"' during run time after the other side has initiated a search.

The *abstract platform* model has connectors, though it does not imply that they must be relevant in a downstream platform. Theoretically the downstream platform may still not be decided, but a designer may wish to still model connectors for those downstream platforms that do support connectors.

**[TPS_APSD_01012]**{DRAFT} **Modeling of connectors in an abstract platform** ⌈It shall be possible to model port connectors in the *abstract platform* model.⌋*()*

The proviso for [TPS_APSD_01012] is that, (as with any *abstract platform* element,) they can be ignored for downstream platforms which do not support them. The basis for connectors in an *abstract platform* model shall be the meta-class SwConnector.

| ***Class*** | ***SwConnector*** (abstract) | | | |
|---|---|---|---|---|
| ***Package*** | M2::AUTOSARTemplates::SWComponentTemplate::Composition | | | |
| ***Note*** | The base class for connectors between ports. Connectors have to be identifiable to allow references from the system constraint template. | | | |
| ***Base*** | *ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable* | | | |
| ***Subclasses*** | AssemblySwConnector, DelegationSwConnector, PassThroughSwConnector | | | |
| ***Attribute*** | ***Type*** | ***Mult.*** | ***Kind*** | ***Note*** |
| mapping | PortInterfaceMapping | 0..1 | ref | Reference to a PortInterfaceMapping specifying the mapping of unequal named PortInterface elements of the two different PortInterfaces typing the two PortPrototypes which are referenced by the ConnectorPrototype. |

**Table 3.11: SwConnector**

| Class | DelegationSwConnector | | | |
|---|---|---|---|---|
| Package | M2::AUTOSARTemplates::SWComponentTemplate::Composition | | | |
| Note | A delegation connector delegates one inner PortPrototype (a port of a component that is used inside the composition) to a outer PortPrototype of compatible type that belongs directly to the composition (a port that is owned by the composition). | | | |
| Base | *ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *SwConnector* | | | |
| **Attribute** | **Type** | **Mult.** | **Kind** | **Note** |
| innerPort | PortPrototype | 1 | iref | The port that belongs to the ComponentPrototype in the composition<br>**Tags:**xml.typeElement=true |
| outerPort | PortPrototype | 1 | ref | The port that is located on the outside of the Composition Type |

**Table 3.12: DelegationSwConnector**

| 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. |
| requester | AbstractRequiredPort Prototype | 0..1 | iref | Instance of requiring port. |

**Table 3.13: AssemblySwConnector**

| Class | PassThroughSwConnector | | | |
|---|---|---|---|---|
| Package | M2::AUTOSARTemplates::SWComponentTemplate::Composition | | | |
| Note | This kind of SwConnector can be used inside a CompositionSwComponentType to connect two delegation PortPrototypes. | | | |
| Base | *ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *SwConnector* | | | |
| **Attribute** | **Type** | **Mult.** | **Kind** | **Note** |
| providedOuter Port | AbstractProvidedPort Prototype | 1 | ref | This represents the provided outer delegation Port Prototype of the PassThroughSwConnector. |
| requiredOuter Port | AbstractRequiredPort Prototype | 1 | ref | This represents the required outer delegation Port Prototype of the PassThroughSwConnector. |

**Table 3.14: PassThroughSwConnector**

## 3.7 Data Type Design

With reference to [TPS_APSD_01013], this is where any further description of types in the abstract view stops. Given the existing high-level nature of the AUTOSAR appli-

cation types, it is possible to take over a limited model of the the existing AUTOSAR meta-classes as the basis for a model of data type definition in an *abstract platform*.

**[TPS_APSD_01013]**{DRAFT} **Abstraction of implementation details of data types in an abstract platform** ⌈Data types in the abstract level should be concerned with at most a high-level typing of identifiers and the physical meaning behind them and not in any way convey implementation details.⌋*()*

The AUTOSAR data type model starts with `AutosarDataType`. The meta-class `AutosarDataType` inherits from `Identifiable` which provides the identifying attributes needed: `longName`, `shortName`. The `category` is then used to indicate the application level data type.

**[TPS_APSD_01014]**{DRAFT} **Allowed data types in an abstract platform** ⌈The *abstract platform* shall support a high-level specification of data typing of:

- single values in the form of `category VALUE`
- structures in the form of `category ARRAY`
- arrays in the form of `category ARRAY`

⌋*()*

The basis for allowed data types in an *abstract platform* are those application data types cited in AUTOSAR Software-Component Template [2] chapter "'Data Categories"'. `category`. It is necessary to lock out the other unsupported data types via constraints.

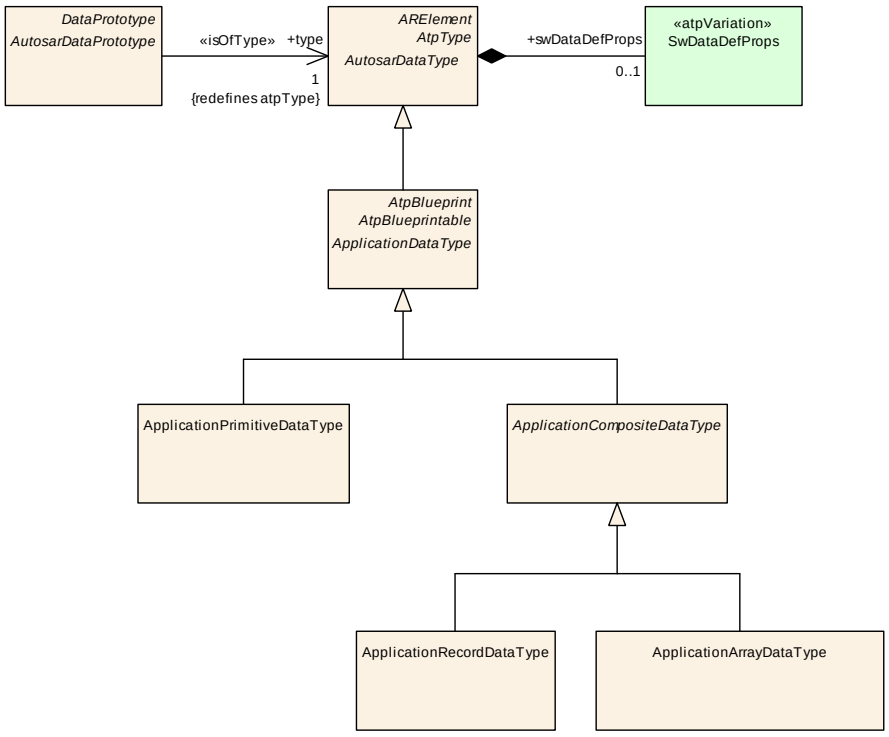


**Figure 3.6: Modeling of Abstract Platform data types**

| **Class** | ***AutosarDataType*** (abstract) | | | |
|---|---|---|---|---|
| **Package** | M2::AUTOSARTemplates::SWComponentTemplate::Datatype::Datatypes | | | |
| **Note** | Abstract base class for user defined AUTOSAR data types for ECU software. | | | |
| **Base** | *ARElement*, *ARObject*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable* | | | |
| **Subclasses** | *AbstractImplementationDataType*, *ApplicationDataType* | | | |
| **Attribute** | **Type** | **Mult.** | **Kind** | **Note** |
| swDataDefProps | SwDataDefProps | 0..1 | aggr | The properties of this AutosarDataType. |

**Table 3.15: AutosarDataType**

| **Class** | ***ApplicationDataType*** (abstract) | | | |
|---|---|---|---|---|
| **Package** | M2::AUTOSARTemplates::SWComponentTemplate::Datatype::Datatypes | | | |
| **Note** | ApplicationDataType defines a data type from the application point of view. Especially it should be used whenever something "physical" is at stake.<br>An ApplicationDataType represents a set of values as seen in the application model, such as measurement units. It does not consider implementation details such as bit-size, endianess, etc.<br>It should be possible to model the application level aspects of a VFB system by using ApplicationData Types only. | | | |
| **Base** | *ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *AutosarDataType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable* | | | |
| **Subclasses** | *ApplicationCompositeDataType*, ApplicationPrimitiveDataType | | | |
| **Attribute** | **Type** | **Mult.** | **Kind** | **Note** |
| – | – | – | – | – |

**Table 3.16: ApplicationDataType**

**[constr_6810] Applicable categories for data types in an abstract platform** ⌈Table 3.17 defines the applicable data type `category`s relating to applicable meta-model classes.⌋*()*

| Category | Applicable to ... | | | | | | | Description |
|---|---|---|---|---|---|---|---|---|
| | ApplicationDataType | ApplicationArrayDataType | ApplicationRecordDataType | ApplicationPrimitiveDataType | ApplicationRecordElement | ApplicationArrayElement | ApplicationValueSpecification | |
| VALUE | | | | x | x | x | x | Contains a single value. |
| STRUCTURE | | | x | | x | x | | Holds one or several further elements which can have different AutosarDataTypes. |
| UNION | | | | | | | | Can hold values of different data types.<br>A UNION data prototype can contain only one of its elements at a time. The size of the UNION is at least the size of the largest member. |
| ARRAY | | x | | | x | x | | A fixed-sized array of sub-elements of the same type. |
| DEFERRED | x | | | | | | | An as yet unspecified type, to be defined before, or latest during, an implementation stage. |

**Table 3.17: Usage of category for Data Types**

### 3.7.1 Deferred Data Type

Further to the data types taken over from the AUTOSAR Software-Component Template [2] chapter "'Data Categories"', the *abstract platform* introduces a further exclusive type called DEFERRED. Due to the fact that a data type may not yet be known in the *abstract platform* or shall be defined later in the design in a downstream stage, the typing shall be deferred with the proviso that it shall be defined during derivation to a *concrete platform*.

This allows an extra level of flexibility in abstract modeling just the signals between ports and any for example a unit, but leaving the implementation details to the downstream stage.

**[TPS_APSD_01015]**{DRAFT} **Deferral of the category of an ApplicationDataType typing in an abstract platform** ⌈It shall be possible to defer typing of an ApplicationDataType to a later design stage.⌋*()*

**[TPS_APSD_01016]**{DRAFT} **Concrete definition of a deferred type** ⌈Deferred ApplicationDataTypes should be defined before or latest during data type mapping to an ImplementationDataType in an *AUTOSAR concrete platform*.⌋*()*

**[TPS_APSD_01017]**{DRAFT} **The category of a deferred type in an abstract platform** ⌈The value ApplicationDataType.category shall be DEFERRED.⌋*()*

**[constr_6811]**{DRAFT} **Exclusivity of `ApplicationDataType.category DEFERRED` to the *abstract platform*** ⌈Usage of `ApplicationDataType.category DEFERRED` shall be limited to the *abstract platform*.⌋*()*

Rationale for [constr_6811]: is grounded in [TPS_APSD_01003].

### 3.7.2 Attributes of SwDataDefProps

Currently those `category`s taken over from the AUTOSAR Software-Component Template [2] chapter "'Data Categories"' have retained their attributes in the *abstract platform*, this is subject to change in the future.

**[constr_6812]**{DRAFT} **`SwDataDefProps` applicable to `ApplicationDataType`s exclusive to the *abstract platform*** ⌈A complete list of the allowed `SwDataDefProps` attributes and their multiplicities which are allowed for a given `category` is shown in table 3.18.⌋*()*

| Attributes of SwDataDefProps | Root Elem. | | | Attribute Existence per Category | | | | |
|---|---|---|---|---|---|---|---|---|
| | ApplicationDataType | ApplicationRecordElement | ApplicationArrayElement | VALUE | DEFERRED | STRUCTURE | ARRAY | |
| annotation | x | x | x | * | * | * | * | |
| compuMethod | x | | | 0..1 | | | | |
| dataConstr.dataConstrRule.physConstrs | x | x | x | 0..1 | | | 0..1 | |
| dataConstr.dataConstrRule.internalConstrs | x | x | x | d/c[1] | | | d/c | |
| displayFormat | x | x | x | 0..1 | | | 0..1 | |
| invalidValue | x | | | 0..1 | | | | |
| swImplPolicy | x | | | 0..1 | | 0..1 | 0..1 | |
| swIntendedResolution | x | x | x | 0..1 | | | | |
| swTextProps | x | | | | | | | |
| unit | x | | | 0..1 | 0..1 | | | |
| **Other Attributes below the Root Element** | | | | | | | | |
| element: ApplicationRecordElement | x | x | x | | | 1..* | | |
| element: ApplicationArrayElement | x | x | x | | | | 1 | |
| ApplicationArrayElement.maxNumberOfElements | x | | | | | | 1 | |

**Table 3.18: Allowed Attributes vs. `category` for `ApplicationDataType`s**

[1] don't care

**[TPS_APSD_01018]**{DRAFT} **Exclusion of type mapping in an abstract platform** ⌈Since the *abstract platform* shall work only with types on the level of `ApplicationDataType`s, there shall be no type mapping in an *abstract platform* (see [constr_6804]).⌋*()*

# 4 Requirements

## 4.1 General

The AUTOSAR meta-model already provides a healthy set of meta-classes for the topic of requirements in the AUTOSAR Standardization Template [1] [TPS_STDT_00060]. For requirements engineering (annotation, documentation, rationalization, traceability) in an *abstract platform*, they can be directly applied.

## 4.2 Requirement Annotation

The intent of [TPS_APSD_01100] is to allow (during the design of an *abstract platform*) a designer to early annotate an *abstract platform* description with requirements. During any time in the design stage, a parent requirement can be recursively broken-down (decomposed) into N x *child* level requirements and annotated to an *abstract platform* description.

**[TPS_APSD_01100]**{DRAFT} **Requirement annotation in an abstract platform** ⌈It shall be possible to specify any number of levels of engineering requirements and annotate those to an *abstract platform* description accordingly.⌋*()*

It is at the discretion of the designer how and when to do this step and to decide when the current decomposition level is sufficient. During the *concrete platform* implementation stage a developer would then implement according to the requirements.

There are no restrictions on what a requirement is, nor on the number of decompositions of a requirement. The meta-class `StructuredReq` may be reused directly for requirement specification.

| ***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.<br>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.<br>**Tags:**<br>xml.namePlural=APPLIES-TO-DEPENDENCIES<br>xml.sequenceOffset=25 |
| conflicts | DocumentationBlock | 0..1 | aggr | This represents an informal specification of conflicts.<br>**Tags:**xml.sequenceOffset=40 |

▽

△

| *Class* | **StructuredReq** | | | |
|---|---|---|---|---|
| date | DateTime | 1 | attr | This represents the date when the requirement was initiated.<br>**Tags:**xml.sequenceOffset=5 |
| dependencies | DocumentationBlock | 0..1 | aggr | This represents an informal specifiaction of dependencies. Note that upstream tracing should be formalized in the property trace provided by the superclass Traceable.<br>**Tags:**xml.sequenceOffset=30 |
| description | DocumentationBlock | 0..1 | aggr | Ths represents the general description of the requirement.<br>**Tags:**xml.sequenceOffset=10 |
| importance | String | 1 | attr | This allows to represent the importance of the requirement.<br>**Tags:**xml.sequenceOffset=8 |
| issuedBy | String | 1 | attr | This represents the person, organization or authority which issued the requirement.<br>**Tags:**xml.sequenceOffset=6 |
| rationale | DocumentationBlock | 0..1 | aggr | This represents the rationale of the requirement.<br>**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.<br>**Tags:**xml.sequenceOffset=60 |
| supporting Material | DocumentationBlock | 0..1 | aggr | This represents an informal specifiaction of the supporting material.<br>**Tags:**xml.sequenceOffset=50 |
| testedItem | Traceable | * | ref | This assocation represents the ability to trace on the same specification level. This supports for example the of acceptance tests.<br>**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.<br>**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.<br>**Tags:**xml.sequenceOffset=35 |

**Table 4.1: StructuredReq**

## 4.3 Requirement Traceability

Tracing in a *abstract platform* description is relevant in 2 forms:

- engineering requirements tracing: See [TPS_APSD_01101].
- functional tracing: See [TPS_APSD_01102]. Here there are 2 possibilities:

- If an existing *abstract platform* description is used to derive parts of an existing *concrete platform* description (e.g: `VFB` level elements in a *AUTOSAR classic platform* model), those elements 'derived from' or 'related to' can be traced between the models.
- Inversely, if a *abstract platform* shall be created out of an existing *concrete platform* description (e.g: *adaptive platform* `ApplicationManifest` level elements in a *AUTOSAR adaptive platform* model), those elements 'abstracted out of' or 'related to' can be traced between the models.

**[TPS_APSD_01101]**{DRAFT} **Requirements tracing in an abstract platform** ⌈Decomposed requirements specified as part of the requirements annotation engineering step may be traced within the model description to their composite source.⌋ *()*

**[TPS_APSD_01102]**{DRAFT} **Functional tracing in an abstract platform** ⌈It shall be possible to utilize two way tracing between an *abstract platform* description and *concrete platform* description.⌋ *()*

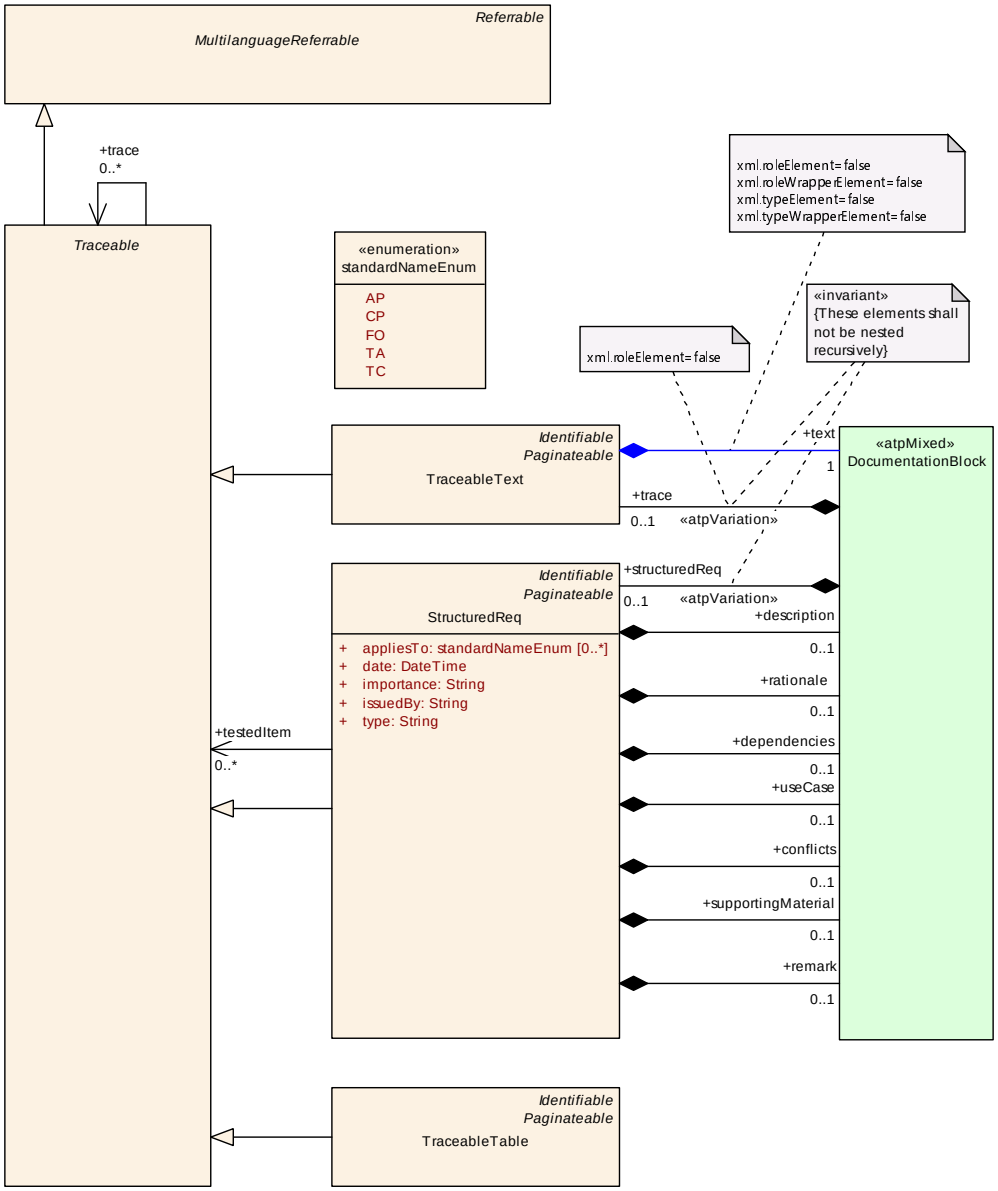


**Figure 4.1: Requirements and Tracing model in an Abstract Platform**

| | | | | |
|---|---|---|---|---|
| ***Class*** | ***TraceReferrable*** (abstract) | | | |
| ***Package*** | M2::MSR::Documentation::BlockElements::RequirementsTracing | | | |
| ***Note*** | This meta class is intended to add the category to the subclasses of Traceable.<br>Even if the model seems to be a bit awkward, it ensures backwards compatibility of the schema.<br>This approach allows to have subclasses of Traceable which are either Identifiable or only Referrable while still maintaining the consistent sequence of shortName, longName, category. | | | |
| ***Base*** | *ARObject*, *MultilanguageReferrable*, *Referrable* | | | |
| ***Subclasses*** | | | | |
| ***Attribute*** | ***Type*** | ***Mult.*** | ***Kind*** | ***Note*** |

▽

△

| *Class* | ***TraceReferrable*** (abstract) | | | |
|---|---|---|---|---|
| – | – | – | – | – |

**Table 4.2: TraceReferrable**

| *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.<br>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*** |
| trace | Traceable | * | ref | This assocation represents the ability to trace to upstream requirements / constraints. This supports for example the bottom up tracing<br>ProjectObjectives <- MainRequirements <- Features <- RequirementSpecs <- BSW/AI<br>**Tags:**xml.sequenceOffset=20 |

**Table 4.3: Traceable**

# A 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*** | **ApApplicationError** | | | |
|---|---|---|---|---|
| ***Package*** | M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface | | | |
| ***Note*** | This meta-class represents the ability to formally specify the semantics of an application error on the AUTOSAR adaptive platform<br>**Tags:**<br>atp.Status=draft<br>atp.recommendedPackage=ApplicationErrors | | | |
| ***Base*** | *ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*, *UploadablePackageElement* | | | |
| ***Attribute*** | ***Type*** | ***Mult.*** | ***Kind*** | ***Note*** |
| errorCode | Integer | 1 | attr | This attribute has the ability to specify the error code value within the enclosing AdaptivePlatformApplication Error. |
| errorDomain | ApApplicationError Domain | 1 | ref | This reference represents the error domain of the Ap ApplicationError.<br>**Tags:**atp.Status=draft |

**Table A.1: ApApplicationError**

| ***Class*** | **ApApplicationErrorSet** | | | |
|---|---|---|---|---|
| ***Package*** | M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface | | | |
| ***Note*** | This meta-class acts as a reference target that represents an entire collection of APApplicationErrors. This takes the burden from ClientServerOperations that reference a larger number of ApApplication Errors.<br>**Tags:**<br>atp.Status=draft<br>atp.recommendedPackage=ApplicationErrorSets | | | |
| ***Base*** | *ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*, *UploadablePackageElement* | | | |
| ***Attribute*** | ***Type*** | ***Mult.*** | ***Kind*** | ***Note*** |
| apApplication Error | ApApplicationError | * | ref | Thi reference represents the collection of ApApplication Error represented by the enclosing ApApplicationErrorSet<br>**Tags:**atp.Status=draft |

**Table A.2: ApApplicationErrorSet**

| ***Class*** | **ApplicationArrayDataType** |
|---|---|
| ***Package*** | M2::AUTOSARTemplates::SWComponentTemplate::Datatype::Datatypes |
| ***Note*** | An application data type which is an array, each element is of the same application data type.<br>**Tags:**atp.recommendedPackage=ApplicationDataTypes |

△

| Class | ApplicationArrayDataType | | | |
|---|---|---|---|---|
| Base | ARElement, ARObject, ApplicationCompositeDataType, ApplicationDataType, 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 if it is a variable size array. |
| element | ApplicationArrayElement | 1 | aggr | This association implements the concept of an array element. That is, in some cases it is necessary to be able to identify single array elements, e.g. as input values for an interpolation routine. |

**Table A.3: ApplicationArrayDataType**

| Class | ApplicationArrayElement | | | |
|---|---|---|---|---|
| Package | M2::AUTOSARTemplates::SWComponentTemplate::Datatype::DataPrototypes | | | |
| Note | Describes the properties of the elements of an application array data type. | | | |
| Base | ARObject, ApplicationCompositeElementDataPrototype, AtpFeature, AtpPrototype, DataPrototype, Identifiable, MultilanguageReferrable, Referrable | | | |
| **Attribute** | **Type** | **Mult.** | **Kind** | **Note** |
| arraySizeHandling | ArraySizeHandlingEnum | 0..1 | attr | The way how the size of the array is handled. |
| arraySizeSemantics | ArraySizeSemanticsEnum | 0..1 | attr | This attribute controls how the information about the array size shall be interpreted. |
| indexDataType | ApplicationPrimitiveDataType | 0..1 | ref | This reference can be taken to assign a CompuMethod of category TEXTTABLE to the array. The texttable entries associate a textual value to an index number such that the element with that index number is represented by a symbolic name. |
| maxNumberOfElements | PositiveInteger | 0..1 | attr | The maximum number of elements that the array can contain.<br>**Stereotypes:** atpVariation<br>**Tags:**vh.latestBindingTime=preCompileTime |

**Table A.4: ApplicationArrayElement**

| Class | ApplicationPrimitiveDataType | | | |
|---|---|---|---|---|
| Package | M2::AUTOSARTemplates::SWComponentTemplate::Datatype::Datatypes | | | |
| Note | A primitive data type defines a set of allowed values.<br>**Tags:**atp.recommendedPackage=ApplicationDataTypes | | | |
| Base | ARElement, ARObject, ApplicationDataType, AtpBlueprint, AtpBlueprintable, AtpClassifier, AtpType, AutosarDataType, CollectableElement, Identifiable, MultilanguageReferrable, PackageableElement, Referrable | | | |
| **Attribute** | **Type** | **Mult.** | **Kind** | **Note** |
| – | – | – | – | – |

**Table A.5: ApplicationPrimitiveDataType**

| Class | ApplicationRecordDataType | | | |
|---|---|---|---|---|
| Package | M2::AUTOSARTemplates::SWComponentTemplate::Datatype::Datatypes | | | |
| Note | An application data type which can be decomposed into prototypes of other application data types.<br>**Tags:**atp.recommendedPackage=ApplicationDataTypes | | | |
| Base | *ARElement*, *ARObject*, *ApplicationCompositeDataType*, *ApplicationDataType*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *AutosarDataType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable* | | | |
| **Attribute** | **Type** | **Mult.** | **Kind** | **Note** |
| element (ordered) | ApplicationRecordElement | 1..* | aggr | Specifies an element of a record.<br>The aggregation of ApplicationRecordElement is subject to variability with the purpose to support the conditional existence of elements inside a ApplicationrecordDataType.<br>**Stereotypes:** atpVariation<br>**Tags:**vh.latestBindingTime=preCompileTime |

**Table A.6: ApplicationRecordDataType**

| Class | ApplicationRecordElement | | | |
|---|---|---|---|---|
| Package | M2::AUTOSARTemplates::SWComponentTemplate::Datatype::DataPrototypes | | | |
| Note | Describes the properties of one particular element of an application record data type. | | | |
| Base | *ARObject*, *ApplicationCompositeElementDataPrototype*, *AtpFeature*, *AtpPrototype*, *DataPrototype*, *Identifiable*, *MultilanguageReferrable*, *Referrable* | | | |
| **Attribute** | **Type** | **Mult.** | **Kind** | **Note** |
| isOptional | Boolean | 0..1 | attr | This attribute represents the ability to declare the enclosing ApplicationRecordElement as optional. This means the that, at runtime, the ApplicationRecordElement may or may not have a valid value and shall therefore be ignored.<br>The underlying runtime software provides means to set the ApplicationRecordElement as not valid at the sending end of a communication and determine its validity at the receiving end. |

**Table A.7: ApplicationRecordElement**

| Class | ApplicationValueSpecification | | | |
|---|---|---|---|---|
| Package | M2::AUTOSARTemplates::CommonStructure::Constants | | | |
| Note | This meta-class represents values for DataPrototypes typed by ApplicationDataTypes (this includes in particular compound primitives).<br>For further details refer to ASAM CDF 2.0. This meta-class corresponds to some extent with SW-INSTANCE in ASAM CDF 2.0. | | | |
| Base | *ARObject*, *ValueSpecification* | | | |
| **Attribute** | **Type** | **Mult.** | **Kind** | **Note** |
| category | Identifier | 1 | attr | Specifies to which category of ApplicationDataType this ApplicationValueSpecification can be applied (e.g. as an initial value), thus imposing constraints on the structure and semantics of the contained values, see [constr_1006] and [constr_2051]. |

▽

△

| *Class* | **ApplicationValueSpecification** | | | |
|---|---|---|---|---|
| swAxisCont (ordered) | SwAxisCont | * | aggr | This represents the axis values of a Compound Primitive Data Type (curve or map).<br>The first swAxisCont describes the x-axis, the second sw AxisCont describes the y-axis, the third swAxisCont describes the z-axis. In addition to this, the axis can be denoted in swAxisIndex. |
| swValueCont | SwValueCont | 0..1 | aggr | This represents the values of a Compound Primitive Data Type. |

**Table A.8: ApplicationValueSpecification**

| *Class* | ***AtomicSwComponentType*** (abstract) | | | |
|---|---|---|---|---|
| ***Package*** | M2::AUTOSARTemplates::SWComponentTemplate::Components | | | |
| ***Note*** | An atomic software component is atomic in the sense that it cannot be further decomposed and distributed across multiple ECUs. | | | |
| ***Base*** | *ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *SwComponentType* | | | |
| ***Subclasses*** | ApplicationSwComponentType, ComplexDeviceDriverSwComponentType, EcuAbstractionSwComponent Type, NvBlockSwComponentType, SensorActuatorSwComponentType, ServiceProxySwComponent Type, ServiceSwComponentType | | | |
| ***Attribute*** | ***Type*** | ***Mult.*** | ***Kind*** | ***Note*** |
| internalBehavior | SwcInternalBehavior | 0..1 | aggr | The SwcInternalBehaviors owned by an AtomicSw ComponentType can be located in a different physical file. Therefore the aggregation is <<atpSplitable>>.<br>**Stereotypes:** atpSplitable; atpVariation<br>**Tags:**<br>atp.Splitkey=internalBehavior, variationPoint.shortLabel<br>vh.latestBindingTime=preCompileTime |
| symbolProps | SymbolProps | 0..1 | aggr | This represents the SymbolProps for the AtomicSw ComponentType.<br>**Stereotypes:** atpSplitable<br>**Tags:**atp.Splitkey=shortName |

**Table A.9: AtomicSwComponentType**

| *Class* | **ClientIdDefinitionSet** | | | |
|---|---|---|---|---|
| ***Package*** | M2::AUTOSARTemplates::SystemTemplate | | | |
| ***Note*** | Set of Client Identifiers that are used for inter-ECU client-server communication in the System.<br>**Tags:**atp.recommendedPackage=ClientIdDefinitionSets | | | |
| ***Base*** | *ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable* | | | |
| ***Attribute*** | ***Type*** | ***Mult.*** | ***Kind*** | ***Note*** |

▽

△

| *Class* | **ClientIdDefinitionSet** | | | |
|---|---|---|---|---|
| clientId Definition | ClientIdDefinition | * | aggr | Definition of a Client Identifier that will be used by the RTE in a inter-ECU client-server communication.<br>**Stereotypes:** atpSplitable; atpVariation<br>**Tags:**<br>atp.Splitkey=shortName, variationPoint.shortLabel<br>vh.latestBindingTime=postBuild |

**Table A.10: ClientIdDefinitionSet**

| *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.<br>**Tags:**atp.recommendedPackage=PortInterfaces | | | |
| *Base* | *ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable* | | | |
| ***Attribute*** | ***Type*** | ***Mult.*** | ***Kind*** | ***Note*** |
| operation | ClientServerOperation | 1..* | aggr | ClientServerOperation(s) of this ClientServerInterface.<br>**Stereotypes:** atpVariation<br>**Tags:**vh.latestBindingTime=blueprintDerivationTime |
| possibleError | ApplicationError | * | aggr | Application errors that are defined as part of this interface. |

**Table A.11: ClientServerInterface**

| *Class* | **ConstantSpecificationMappingSet** | | | |
|---|---|---|---|---|
| *Package* | M2::AUTOSARTemplates::CommonStructure::Constants | | | |
| *Note* | This meta-class represents the ability to map two ConstantSpecifications to each others. One Constant Specification is supposed to be described in the application domain and the other should be described in the implementation domain.<br>**Tags:**atp.recommendedPackage=ConstantSpecificationMappingSets | | | |
| *Base* | *ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable* | | | |
| ***Attribute*** | ***Type*** | ***Mult.*** | ***Kind*** | ***Note*** |
| mapping | ConstantSpecification Mapping | 1..* | aggr | ConstantSpecificationMappings owned by the Constant SpecificationMappingSet. |

**Table A.12: ConstantSpecificationMappingSet**

| *Class* | **DataConstr** | | | |
|---|---|---|---|---|
| *Package* | M2::MSR::AsamHdo::Constraints::GlobalConstraints | | | |
| *Note* | This meta-class represents the ability to specify constraints on data.<br>**Tags:**atp.recommendedPackage=DataConstrs | | | |
| *Base* | *ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *CollectableElement*, *Identifiable*, *Multilanguage Referrable*, *PackageableElement*, *Referrable* | | | |
| ***Attribute*** | ***Type*** | ***Mult.*** | ***Kind*** | ***Note*** |

▽

△

| Class | DataConstr | | | |
|---|---|---|---|---|
| dataConstrRule | DataConstrRule | * | aggr | This is one particular rule within the data constraints.<br>**Tags:**<br>xml.roleElement=true<br>xml.roleWrapperElement=true<br>xml.sequenceOffset=30<br>xml.typeElement=false<br>xml.typeWrapperElement=false |

**Table A.13: DataConstr**

| Class | DataConstrRule | | | |
|---|---|---|---|---|
| Package | M2::MSR::AsamHdo::Constraints::GlobalConstraints | | | |
| Note | This meta-class represents the ability to express one specific data constraint rule. | | | |
| Base | *ARObject* | | | |
| **Attribute** | **Type** | **Mult.** | **Kind** | **Note** |
| constrLevel | Integer | 0..1 | attr | This attribute describes the category of a constraint. One of its functions is in the area of constraint violation, where it can be used from a certain level, to produce error messages.<br>The lower the level, the more stringent the check.<br>Used to distinguish hard or soft limits.<br>**Tags:**xml.sequenceOffset=20 |
| internalConstrs | InternalConstrs | 0..1 | aggr | Describes the limitations applicable on the internal domain (as opposed to the physical domain).<br>**Tags:**xml.sequenceOffset=40 |
| physConstrs | PhysConstrs | 0..1 | aggr | Describes the limitations applicable on the physical domain (as opposed to the internal domain).<br>**Tags:**xml.sequenceOffset=30 |

**Table A.14: DataConstrRule**

| Class | DataTypeMappingSet | | | |
|---|---|---|---|---|
| Package | M2::AUTOSARTemplates::SWComponentTemplate::Datatype::Datatypes | | | |
| Note | This class represents a list of mappings between ApplicationDataTypes and ImplementationDataTypes. In addition, it can contain mappings between ImplementationDataTypes and ModeDeclarationGroups.<br>**Tags:**atp.recommendedPackage=DataTypeMappingSets | | | |
| Base | *ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable* | | | |
| **Attribute** | **Type** | **Mult.** | **Kind** | **Note** |
| dataTypeMap | DataTypeMap | * | aggr | This is one particular association between an ApplicationDataType and its AbstractImplementationDataType. |
| modeRequestTypeMap | ModeRequestTypeMap | * | aggr | This is one particular association between an ModeDeclarationGroup and its AbstractImplementationDataType. |

**Table A.15: DataTypeMappingSet**

| **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, DltMessageCollectionSet, EcuInstance, *Frame*, Gateway, GlobalTimeDomain, ISignal, ISignalGroup, ISignalIPduGroup, MachineDesign, NmConfig, *Pdu*, PdurIPduGroup, Secure CommunicationPropsSet, ServiceInstanceCollectionSet, SoAdRoutingGroup, SocketConnectionIpdu IdentifierSet, *TpConfig* | | | |
| ***Attribute*** | ***Type*** | ***Mult.*** | ***Kind*** | ***Note*** |
| – | – | – | – | – |

**Table A.16: FibexElement**

| **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, *AbstractEvent*, *AbstractImplementationDataTypeElement*, *AbstractServiceInstance*, *AbstractSignalBasedToISignalTriggeringMapping*, *AdaptiveModuleInstantiation*, AdaptiveSwcInternal Behavior, ApplicationEndpoint, ApplicationError, ApplicationPartitionToEcuPartitionMapping, AsynchronousServerCallResultPoint, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpFeature*, Autosar OperationArgumentInstance, AutosarVariableInstance, BswInternalTriggeringPoint, BswModule Dependency, *BuildActionEntity*, BuildActionEnvironment, CanTpAddress, CanTpChannel, CanTpNode, Chapter, CheckpointTransition, ClassContentConditional, ClientIdDefinition, ClientServerOperation, Code, *CollectableElement*, ComManagementMapping, *CommConnectorPort*, *Communication Connector*, *CommunicationController*, Compiler, ConsistencyNeeds, ConsumedEventGroup, Coupling Port, *CouplingPortStructuralElement*, CryptoKeySlot, *CryptoServiceMapping*, DataPrototypeGroup, Data Transformation, DdsRpcServiceDeployment, DependencyOnArtifact, DeterministicClientResourceNeeds, *DiagEventDebounceAlgorithm*, DiagnosticConnectedIndicator, DiagnosticDataElement, Diagnostic FunctionInhibitSource, DiagnosticMasterToSlaveEventMapping, *DiagnosticRoutineSubfunction*, Dlt Argument, DltLogChannel, DltMessage, DoIpInterface, DoIpLogicAddress, E2EProfileConfiguration, EC UMapping, *EOCExecutableEntityRefAbstract*, EcuPartition, EcucContainerValue, *EcucDefinition Element*, EcucDestinationUriDef, EcucEnumerationLiteralDef, EcucQuery, EcucValidationCondition, End2EndEventProtectionProps, EndToEndProtection, EventMapping, ExclusiveArea, *ExecutableEntity*, *ExecutionTime*, FMAttributeDef, FMFeatureMapAssertion, FMFeatureMapCondition, FMFeatureMap Element, FMFeatureRelation, FMFeatureRestriction, FMFeatureSelection, FieldMapping, FireAndForget Mapping, FlatInstanceDescriptor, FlexrayArTpNode, FlexrayTpConnectionControl, FlexrayTpNode, FlexrayTpPduPool, *FrameTriggering*, GeneralParameter, GlobalTimeGateway, *GlobalTimeMaster*, *GlobalTimeSlave*, *HealthChannel*, *HeapUsage*, HwAttributeDef, HwAttributeLiteralDef, HwPin, HwPin Group, IPSecRule, IPv6ExtHeaderFilterList, ISignalToIPduMapping, ISignalTriggering, *IdentCaption*, InterfaceMapping, InternalTriggeringPoint, J1939SharedAddressCluster, J1939TpNode, Keyword, Life CycleState, LinScheduleTable, LinTpNode, Linker, MacMulticastGroup, McDataInstance, Memory Section, MethodMapping, ModeDeclaration, ModeDeclarationMapping, ModeSwitchPoint, Network Endpoint, *NmCluster*, *NmNode*, NvBlockDescriptor, *PackageableElement*, ParameterAccess, PduTo FrameMapping, PduTriggering, PerInstanceMemory, PersistencyFileProxy, PersistencyKeyValuePair, *PhmActionItem*, PhmActionList, PhmLogicalExpression, PhmRule, *PhmSupervision*, *PhysicalChannel*, PortGroup, *PortInterfaceMapping*, PossibleErrorReaction, ProcessDesignToMachineDesignMapping, ProcessToMachineMapping, Processor, ProcessorCore, PskIdentityToKeySlotMapping, RawDataStream MethodDeployment, ResourceConsumption, ResourceGroup, *RestAbstractEndpoint*, RestElementDef, RestResourceDef, RootSwClusterDesignComponentPrototype, RootSwComponentPrototype, RootSw CompositionPrototype, RptComponent, RptContainer, RptExecutableEntity, RptExecutableEntityEvent, RptExecutionContext, RptProfile, RptServicePoint, RunnableEntityGroup, *SdgAttribute*, SdgClass, Sec ▽ |

▽

△

| **Class** | ***Identifiable*** (abstract) | | | |
|---|---|---|---|---|
| | △<br>OcJobMapping, SecOcJobRequirement, *SecureComProps*, SecureCommunicationAuthenticationProps, *SecureCommunicationDeployment*, SecureCommunicationFreshnessProps, *ServerCallPoint*, *ServiceEventDeployment*, *ServiceFieldDeployment*, ServiceInstanceToSignalMapping, *ServiceInterfaceElementMapping*, ServiceInterfaceElementSecureComConfig, ServiceInterfaceMapping, *ServiceMethodDeployment*, *ServiceNeeds*, SignalServiceTranslationEventProps, SignalServiceTranslationProps, SocketAddress, SoftwarePackageStep, SomeipEventGroup, SomeipProvidedEventGroup, SomeipTpChannel, *SpecElementReference*, *StackUsage*, StartupConfig, StaticSocketConnection, StructuredReq, SupervisionCheckpoint, SwGenericAxisParamType, SwServiceArg, SwcServiceDependency, SwcToApplicationPartitionMapping, SwcToEcuMapping, SwcToImplMapping, SystemMapping, SystemMemoryUsage, TcpOptionFilterList, *TimeBaseResource*, TimingCondition, *TimingConstraint*, *TimingDescription*, TimingExtensionResource, TimingModeInstance, TlsCryptoCipherSuite, TlsJobMapping, Topic1, TpAddress, TraceableTable, TraceableText, *TracedFailure*, *TransformationProps*, TransformationPropsToServiceInterfaceElementMapping, TransformationTechnology, Trigger, UcmDescription, UcmStep, VariableAccess, VariationPointProxy, VehicleRolloutStep, ViewMap, VlanConfig, WaitPoint | | | |
| ***Attribute*** | ***Type*** | ***Mult.*** | ***Kind*** | ***Note*** |
| adminData | AdminData | 0..1 | aggr | This represents the administrative data for the identifiable object.<br>**Tags:**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.<br>**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.<br>**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.<br>More elaborate documentation, (in particular how the object is built or used) should go to "introduction".<br>**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.<br>**Tags:**xml.sequenceOffset=-30 |
| 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<br>▽ | |

▽

△

| Class | Identifiable (abstract) | | | |
|---|---|---|---|---|
| | | | | △<br>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.<br>**Tags:**xml.attribute=true |

**Table A.17: Identifiable**

| 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.<br>**Tags:**atp.recommendedPackage=ImplementationDataTypes | | | |
| Base | *ARElement*, *ARObject*, *AbstractImplementationDataType*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *AutosarDataType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable* | | | |
| **Attribute** | **Type** | **Mult.** | **Kind** | **Note** |
| dynamicArray SizeProfile | String | 0..1 | attr | Specifies the profile which the array will follow in case this data type is a variable size array. |
| isStructWith Optional Element | Boolean | 0..1 | attr | This attribute is only valid if the attribute category is set to STRUCTURE.<br>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) | ImplementationData TypeElement | * | aggr | Specifies an element of an array, struct, or union data type.<br>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.<br>**Stereotypes:** atpVariation<br>**Tags:**vh.latestBindingTime=preCompileTime |
| symbolProps | SymbolProps | 0..1 | aggr | This represents the SymbolProps for the Implementation DataType.<br>**Stereotypes:** atpSplitable<br>**Tags:**atp.Splitkey=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 A.18: ImplementationDataType**

| **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 | MultilanguageLong Name | 0..1 | aggr | This specifies the long name of the object. Long name is targeted to human readers and acts like a headline. |

**Table A.19: MultilanguageReferrable**

| **Class** | **NvDataInterface** | | | |
|---|---|---|---|---|
| **Package** | M2::AUTOSARTemplates::SWComponentTemplate::PortInterface | | | |
| **Note** | A non volatile data interface declares a number of VariableDataPrototypes to be exchanged between non volatile block components and atomic software components.<br>**Tags:**atp.recommendedPackage=PortInterfaces | | | |
| **Base** | *ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *DataInterface*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable* | | | |
| ***Attribute*** | ***Type*** | ***Mult.*** | ***Kind*** | ***Note*** |
| nvData | VariableDataPrototype | 1..* | aggr | The VariableDataPrototype of this nv data interface. |

**Table A.20: NvDataInterface**

| **Class** | **PRPortPrototype** | | | |
|---|---|---|---|---|
| **Package** | M2::AUTOSARTemplates::SWComponentTemplate::Components | | | |
| **Note** | This kind of PortPrototype can take the role of both a required and a provided PortPrototype. | | | |
| **Base** | *ARObject*, *AbstractProvidedPortPrototype*, *AbstractRequiredPortPrototype*, *AtpBlueprintable*, *AtpFeature*, *AtpPrototype*, *Identifiable*, *MultilanguageReferrable*, *PortPrototype*, *Referrable* | | | |
| ***Attribute*** | ***Type*** | ***Mult.*** | ***Kind*** | ***Note*** |
| provided Required Interface | PortInterface | 1 | tref | This represents the PortInterface used to type the PRPort Prototype<br>**Stereotypes:** isOfType |

**Table A.21: PRPortPrototype**

| **Class** | **ParameterSwComponentType** | | | |
|---|---|---|---|---|
| **Package** | M2::AUTOSARTemplates::SWComponentTemplate::Components | | | |
| **Note** | The ParameterSwComponentType defines parameters and characteristic values accessible via provided Ports. The provided values are the same for all connected SwComponentPrototypes<br>**Tags:**atp.recommendedPackage=SwComponentTypes | | | |
| **Base** | *ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *SwComponentType* | | | |
| ***Attribute*** | ***Type*** | ***Mult.*** | ***Kind*** | ***Note*** |

▽

△

| *Class* | **ParameterSwComponentType** | | | |
|---|---|---|---|---|
| constant Mapping | ConstantSpecification MappingSet | * | ref | Reference to the ConstanSpecificationMapping to be applied for the particular ParameterSwComponentType<br>**Stereotypes:** atpSplitable<br>**Tags:**atp.Splitkey=constantMapping |
| dataType Mapping | DataTypeMappingSet | * | ref | Reference to the DataTypeMapping to be applied for the particular ParameterSwComponentType<br>**Stereotypes:** atpSplitable<br>**Tags:**atp.Splitkey=dataTypeMapping |
| 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.<br>The aggregation of InstantiationDataDefProps is subject to variability with the purpose to support the conditional existence of PortPrototypes<br>**Stereotypes:** atpVariation<br>**Tags:**vh.latestBindingTime=preCompileTime |

**Table A.22: ParameterSwComponentType**

| *Class* | ***PersistencyInterface*** (abstract) | | | |
|---|---|---|---|---|
| ***Package*** | M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface | | | |
| ***Note*** | This meta-class provides the abstract ability to define a PortInterface for the support of persistency use cases.<br>**Tags:**atp.Status=draft | | | |
| ***Base*** | *ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable* | | | |
| ***Subclasses*** | PersistencyFileProxyInterface, PersistencyKeyValueDatabaseInterface | | | |
| ***Attribute*** | ***Type*** | ***Mult.*** | ***Kind*** | ***Note*** |
| minimum SustainedSize | PositiveInteger | 0..1 | attr | The value of this attribute represents the minimum size required at design time for the enclosing Persistency Interface. |
| redundancy | PersistencyRedundancy Enum | 0..1 | attr | This attribute represents a requirement towards the redundancy of storage. |
| updateStrategy | PersistencyCollection LevelUpdateStrategy Enum | 0..1 | attr | This attribute can be used to specify the update strategy of the respective PersistencyInterface as a whole. |

**Table A.23: PersistencyInterface**

| *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* |

▽

△

| *Class* | ***Referrable*** (abstract) | | | |
|---|---|---|---|---|
| ***Subclasses*** | *AtpDefinition*, BswDistinguishedPartition, *BswModuleCallPoint*, BswModuleClientServerEntry, BswVariableAccess, CouplingPortTrafficClassAssignment, *CppImplementationDataTypeContextTarget*, DiagnosticDebounceAlgorithmProps, *DiagnosticEnvModeElement*, EthernetPriorityRegeneration, EventHandler, ExclusiveAreaNestingOrder, *HwDescriptionEntity*, *ImplementationProps*, LinSlaveConfigIdent, ModeTransition, *MultilanguageReferrable*, *NetworkConfiguration*, NmNetworkHandle, PduActivationRoutingGroup, PncMappingIdent, *SingleLanguageReferrable*, SoConIPduIdentifier, SocketConnectionBundle, SomeipRequiredEventGroup, 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.<br>**Tags:**<br>xml.enforceMinMultiplicity=true<br>xml.sequenceOffset=-100 |
| shortName Fragment | ShortNameFragment | * | aggr | This specifies how the Referrable.shortName is composed of several shortNameFragments.<br>**Tags:**xml.sequenceOffset=-90 |

**Table A.24: Referrable**

| *Class* | **SenderReceiverInterface** | | | |
|---|---|---|---|---|
| ***Package*** | M2::AUTOSARTemplates::SWComponentTemplate::PortInterface | | | |
| ***Note*** | A sender/receiver interface declares a number of data elements to be sent and received.<br>**Tags:**atp.recommendedPackage=PortInterfaces | | | |
| ***Base*** | *ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *DataInterface*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable* | | | |
| ***Attribute*** | ***Type*** | ***Mult.*** | ***Kind*** | ***Note*** |
| dataElement | VariableDataPrototype | 1..* | aggr | The data elements of this SenderReceiverInterface. |
| invalidation Policy | InvalidationPolicy | * | aggr | InvalidationPolicy for a particular dataElement |
| metaDataItem Set | MetaDataItemSet | * | aggr | This aggregation defines fixed sets of meta-data items associated with dataElements of the enclosing Sender ReceiverInterface |

**Table A.25: SenderReceiverInterface**

| *Class* | **ServiceInterface** | | | |
|---|---|---|---|---|
| ***Package*** | M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface | | | |
| ***Note*** | This represents the ability to define a PortInterface that consists of a heterogeneous collection of methods, events and fields.<br>**Tags:**<br>atp.Status=draft<br>atp.recommendedPackage=ServiceInterfaces | | | |
| ***Base*** | *ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable* | | | |
| ***Attribute*** | ***Type*** | ***Mult.*** | ***Kind*** | ***Note*** |

▽

△

| *Class* | **ServiceInterface** | | | |
|---|---|---|---|---|
| event | VariableDataPrototype | * | aggr | This represents the collection of events defined in the context of a ServiceInterface.<br>**Stereotypes:** atpVariation<br>**Tags:**<br>atp.Status=draft<br>vh.latestBindingTime=blueprintDerivationTime<br>xml.sequenceOffset=30 |
| field | Field | * | aggr | This represents the collection of fields defined in the context of a ServiceInterface.<br>**Stereotypes:** atpVariation<br>**Tags:**<br>atp.Status=draft<br>vh.latestBindingTime=blueprintDerivationTime<br>xml.sequenceOffset=40 |
| majorVersion | PositiveInteger | 0..1 | attr | Major version of the service contract.<br>**Tags:**<br>atp.Status=draft<br>xml.sequenceOffset=10 |
| method | ClientServerOperation | * | aggr | This represents the collection of methods defined in the context of a ServiceInterface.<br>**Stereotypes:** atpVariation<br>**Tags:**<br>atp.Status=draft<br>vh.latestBindingTime=blueprintDerivationTime<br>xml.sequenceOffset=50 |
| minorVersion | PositiveInteger | 0..1 | attr | Minor version of the service contract.<br>**Tags:**<br>atp.Status=draft<br>xml.sequenceOffset=20 |

**Table A.26: ServiceInterface**

| *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 | 1 | tref | Type of the instance.<br>**Stereotypes:** isOfType |

**Table A.27: 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* |

▽

△

| *Class* | ***SwComponentType*** (abstract) | | | |
|---|---|---|---|---|
| ***Subclasses*** | AdaptiveApplicationSwComponentType, *AtomicSwComponentType*, CompositionSwComponentType, ParameterSwComponentType | | | |
| ***Attribute*** | ***Type*** | ***Mult.*** | ***Kind*** | ***Note*** |
| port | PortPrototype | * | aggr | The PortPrototypes through which this SwComponent Type can communicate.<br>The aggregation of PortPrototype is subject to variability with the purpose to support the conditional existence of PortPrototypes.<br>**Stereotypes:** atpSplitable; atpVariation<br>**Tags:**<br>atp.Splitkey=shortName, variationPoint.shortLabel<br>vh.latestBindingTime=preCompileTime |
| portGroup | PortGroup | * | aggr | A port group being part of this component.<br>**Stereotypes:** atpVariation<br>**Tags:**vh.latestBindingTime=preCompileTime |
| swComponent Documentation | SwComponent Documentation | 0..1 | aggr | This adds a documentation to the SwComponentType.<br>**Stereotypes:** atpSplitable; atpVariation<br>**Tags:**<br>atp.Splitkey=swComponentDocumentation, variation Point.shortLabel<br>vh.latestBindingTime=preCompileTime<br>xml.sequenceOffset=-10 |

**Table A.28: 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.<br>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.<br>SwDataDefProps covers various aspects:<br>• 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 Types in the programming language (or in AUTOSAR). This is mainly expressed by properties like swRecordLayout and swCalprmAxisSet<br>• Implementation aspects, mainly expressed by swImplPolicy, swVariableAccessImplPolicy, sw AddrMethod, swPointerTagetProps, baseType, implementationDataType and additionalNative TypeQualifier<br>• Access policy for the MCD system, mainly expressed by swCalibrationAccess<br>• Semantics of the data element, mainly expressed by compuMethod and/or unit, dataConstr, invalidValue<br>• Code generation policy provided by swRecordLayout<br>**Tags:**vh.latestBindingTime=codeGenerationTime | | | |
| ***Base*** | *ARObject* | | | |
| ***Attribute*** | ***Type*** | ***Mult.*** | ***Kind*** | ***Note*** |

▽

△

| *Class* | <<atpVariation>> **SwDataDefProps** | | | |
|---|---|---|---|---|
| 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.<br>**Tags:**xml.sequenceOffset=235 |
| annotation | Annotation | * | aggr | This aggregation allows to add annotations (yellow pads ...) related to the current data object.<br>**Tags:**<br>xml.roleElement=true<br>xml.roleWrapperElement=true<br>xml.sequenceOffset=20<br>xml.typeElement=false<br>xml.typeWrapperElement=false |
| baseType | SwBaseType | 0..1 | ref | Base type associated with the containing data object.<br>**Tags:**xml.sequenceOffset=50 |
| compuMethod | CompuMethod | 0..1 | ref | Computation method associated with the semantics of this data object.<br>**Tags:**xml.sequenceOffset=180 |
| dataConstr | DataConstr | 0..1 | ref | Data constraint for this data object.<br>**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.<br>**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<br>• redefinition of an ImplementationDataType via a "typedef" to another ImplementationDatatype<br>• the target type of a pointer (see SwPointerTarget Props), if it does not refer to a base type directly<br>• the data type of an array or record element within an ImplementationDataType, if it does not refer to a base type directly<br>• the data type of an SwServiceArg, if it does not refer to a base type directly<br>**Tags:**xml.sequenceOffset=215 |
| invalidValue | ValueSpecification | 0..1 | aggr | Optional value to express invalidity of the actual data element.<br>**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. |

▽

△

| *Class* | <<atpVariation>> **SwDataDefProps** | | | |
|---|---|---|---|---|
| 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.<br>**Tags:**xml.sequenceOffset=30 |
| swAlignment | AlignmentType | 0..1 | attr | The attribute describes the intended alignment of the DataPrototype. If the attribute is not defined the alignment is determined by the swBaseType size and the memory AllocationKeywordPolicy of the referenced SwAddr Method.<br>**Tags:**xml.sequenceOffset=33 |
| swBit Representation | SwBitRepresentation | 0..1 | aggr | Description of the binary representation in case of a bit variable.<br>**Tags:**xml.sequenceOffset=60 |
| swCalibration Access | SwCalibrationAccess Enum | 0..1 | attr | Specifies the read or write access by MCD tools for this data object.<br>**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.<br>**Tags:**xml.sequenceOffset=90 |
| swComparison Variable | SwVariableRefProxy | * | aggr | Variables used for comparison in an MCD process.<br>**Tags:**<br>xml.sequenceOffset=170<br>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).<br>**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.<br>**Tags:**<br>xml.sequenceOffset=220<br>xml.typeElement=false |
| swImplPolicy | SwImplPolicyEnum | 0..1 | attr | Implementation policy for this data object.<br>**Tags:**xml.sequenceOffset=230 |
| 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.<br>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).<br>▽ |

▽

△

| *Class* | <<atpVariation>> **SwDataDefProps** | | | |
|---|---|---|---|---|
| | | | | △<br>In the case of a development phase without a fixed conversion formula, a pre-specification can occur through swIntendedResolution.<br>The resolution is specified in the physical domain according to the property "unit".<br>**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.<br>**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 .<br>**Tags:**xml.sequenceOffset=260 |
| swPointerTarget Props | SwPointerTargetProps | 0..1 | aggr | Specifies that the containing data object is a pointer to another data object.<br>**Tags:**xml.sequenceOffset=280 |
| swRecord Layout | SwRecordLayout | 0..1 | ref | Record layout for this data object.<br>**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.<br>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.<br>**Tags:**xml.sequenceOffset=300 |
| swTextProps | SwTextProps | 0..1 | aggr | the specific properties if the data object is a text object.<br>**Tags:**xml.sequenceOffset=120 |
| swValueBlock Size | Numerical | 0..1 | attr | This represents the size of a Value Block<br>**Stereotypes:** atpVariation<br>**Tags:**<br>vh.latestBindingTime=preCompileTime<br>xml.sequenceOffset=80 |
| 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.<br>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.<br>For one-dimensional value blocks the attribute swValue BlockSize shall be used and this attribute shall not exist.<br>**Stereotypes:** atpVariation<br>**Tags:**vh.latestBindingTime=preCompileTime |

▽

△

| *Class* | <<atpVariation>> **SwDataDefProps** | | | |
|---|---|---|---|---|
| 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.<br>**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.<br>**Tags:**xml.sequenceOffset=355 |

**Table A.29: SwDataDefProps**

| *Class* | **SystemMapping** | | | |
|---|---|---|---|---|
| *Package* | M2::AUTOSARTemplates::SystemTemplate | | | |
| *Note* | The system mapping aggregates all mapping aspects that are relevant in the System Description. | | | |
| *Base* | *ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable* | | | |
| *Attribute* | *Type* | *Mult.* | *Kind* | *Note* |
| pncMapping | PncMapping | * | aggr | Mappings between Virtual Function Clusters and Partial Network Clusters.<br>**Stereotypes:** atpVariation<br>**Tags:**vh.latestBindingTime=systemDesignTime |

**Table A.30: SystemMapping**

| *Class* | ***TimeSynchronizationInterface*** (abstract) | | | |
|---|---|---|---|---|
| *Package* | M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface | | | |
| *Note* | This meta-class provides the abstract ability to define a PortInterface for the interaction with Time Synchronization.<br>**Tags:**atp.Status=draft | | | |
| *Base* | *ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable* | | | |
| *Subclasses* | TimeSynchronizationMasterInterface, TimeSynchronizationPureLocalInterface, TimeSynchronization SlaveInterface | | | |
| *Attribute* | *Type* | *Mult.* | *Kind* | *Note* |
| – | – | – | – | – |

**Table A.31: TimeSynchronizationInterface**

# B History of Constraints and Specification Items

Please note that the lists in this chapter also include constraints and specification items that have been removed from the specification in a later version. These constraints and specification items do not appear as hyperlinks in the document.

## B.1 Constraint and Specification Item History of this document according to AUTOSAR Release 19-11

### B.1.1 Added Traceables in 19-11

| Number | Heading |
|---|---|
| [TPS_APSD_01000] | Principle of an abstract platform system description |
| [TPS_APSD_01001] | Modeling of vehicle communications in an abstract platform |
| [TPS_APSD_01002] | Agnosticism of deployment modeling artifacts in an abstract platform |
| [TPS_APSD_01003] | Exclusion of abstract platform artifacts to an AUTOSAR concrete platform |
| [TPS_APSD_01004] | System `category` for a system description with Abstract Platform content |
| [TPS_APSD_01005] | Identification of component types in an abstract platform |
| [TPS_APSD_01006] | Recursive component definition in an abstract platform |
| [TPS_APSD_01007] | Prototyping of ports in an abstract platform |
| [TPS_APSD_01008] | Generic typing of interfaces in an abstract platform |
| [TPS_APSD_01009] | Grouping of ports in an abstract platform |
| [TPS_APSD_01010] | Agnosticism of abstract platform interfaces to middleware deployments |
| [TPS_APSD_01011] | Aggregation of interface elements in an abstract platform interface |
| [TPS_APSD_01012] | Modeling of connectors in an abstract platform |
| [TPS_APSD_01013] | Abstraction of implementation details of data types in an abstract platform |
| [TPS_APSD_01014] | Allowed data types in an abstract platform |
| [TPS_APSD_01015] | Deferral of the `category` of an `ApplicationDataType` typing in an abstract platform |
| [TPS_APSD_01016] | Concrete definition of a deferred type |
| [TPS_APSD_01017] | The `category` of a deferred type in an abstract platform |
| [TPS_APSD_01018] | Exclusion of type mapping in an abstract platform |
| [TPS_APSD_01100] | Requirement annotation in an abstract platform |
| [TPS_APSD_01101] | Requirements tracing in an abstract platform |
| [TPS_APSD_01102] | Functional tracing in an abstract platform |

**Table B.1: Added Traceables in 19-11**

### B.1.2 Changed Traceables in 19-11

none

### B.1.3 Deleted Traceables in 19-11

none

### B.1.4 Added Constraints in 19-11

| Number | Heading |
|---|---|
| [constr_6800] | Non-relevance of `FibexElement` and `SystemMapping` for a `System` description with Abstract Platform content |
| [constr_6801] | Non-relevance of the attributes `pncVectorLength`, `pncVectorOffset` for a `System` description with Abstract Platform content |
| [constr_6802] | Restriction of the `category` of a `CompositionSwComponentType` which types a `RootSwCompositionPrototype` in a `System` description with Abstract Platform content |
| [constr_6803] | Restriction of the `category` of a `CompositionSwComponentType` which references a `SwComponentPrototype` in a `System` description with Abstract Platform content |
| [constr_6804] | Non-relevance of `ConstantSpecificationMappingSet` and `DataTypeMappingSet` for a `CompositionSwComponentType` in an Abstract Platform |
| [constr_6805] | Non-relevance of `PRPortPrototype` for a `System` with Abstract Platform content |
| [constr_6806] | Restriction of the `category` of a `PortInterface` for a `System` description with Abstract Platform content |
| [constr_6807] | Exclusivity of an `CompositeInterface` to an Abstract Platform |
| [constr_6808] | Non-relevance of the attribute `fireAndForget` for a `ClientServerOperation` used in a `CompositeInterface` |
| [constr_6809] | Non-relevance of `ApApplicationError` and `ApApplicationErrorSet` for a `ClientServerOperation` in the context of a `CompositeInterface` |
| [constr_6810] | Applicable categories for data types in an abstract platform |
| [constr_6811] | Exclusivity of `ApplicationDataType`.`category` `DEFERRED` to the *abstract platform* |
| [constr_6812] | `SwDataDefProps` applicable to `ApplicationDataType`s exclusive to the *abstract platform* |
| [constr_6813] | Restriction of `SwComponentType`s in an Abstract Platform |

**Table B.2: Added Constraints in 19-11**

### B.1.5 Changed Constraints in 19-11

none

### B.1.6 Deleted Constraints in 19-11

none

# C Splitable Elements in the Scope of this Document

This chapter contains a table of all model elements stereotyped ≪atpSplitable≫ in the scope of this document.

Each entry in the table consists of the identification of the specific model element itself and the applicable value of the tagged value atp.Splitkey.

For more information about the concept of splitable model elements and how these shall be treated please refer to [3].

# D Variation Points in the Scope of this Document

This chapter contains a table of all model elements stereotyped ≪atpVariation≫ in the scope of this document.

Each entry in the table consists of the identification of the model element itself and the applicable value of the tagged value vh.latestBindingTime.

For more information about the concept of variation points and how model elements that contain variation points shall be treated please refer to [3].

| Variation Point | Latest Binding Time |
|---|---|
| CompositeInterface.command | blueprintDerivationTime |
| CompositeInterface.indication | blueprintDerivationTime |

**Table D.1: Usage of variation points**