Document Title	Application Interfaces User Guide
Document Category	Explanation (EXP)
Document UID	442
Document Status	published
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	R26-11

Document Change History

Date	Release	Changed by	Description
2026-11-26	R26-11	AUTOSAR Release Management	• Complete rework of all chapters - Part 2 • Milestone propagation added • Editorial corrections
2025-11-27	R25-11	AUTOSAR Release Management	• Complete rework of all chapters • Transformation to AI-Tool
2024-11-27	R24-11	AUTOSAR Release Management	• No content changes
2023-11-23	R23-11	AUTOSAR Release Management	• No content changes
2022-11-24	R22-11	AUTOSAR Release Management	• No content changes
2021-11-25	R21-11	AUTOSAR Release Management	• No content changes
2020-11-30	R20-11	AUTOSAR Release Management	• No content changes
2019-11-28	R19-11	AUTOSAR Release Management	• Editorial changes • Set Document to obsolete, as methodology is implemented in AI-Tool incl. online help
2018-10-31	4.4.0	AUTOSAR Release Management	• Editorial changes
2017-12-08	4.3.1	AUTOSAR Release Management	• Editorial changes



△

2016-11-30	4.3.0	AUTOSAR Release Management	• Add chapter about implementation of data types as integer or floating point data types – Chapter ID 4.2.3.3. Bugzilla # 72021
2015-07-31	4.2.2	AUTOSAR Release Management	• Updated explanation of the COMPU_METHOD reusage • Updated the Linear Conversion Example
2014-10-31	4.2.1	AUTOSAR Release Management	• Sensors and Actuators Pattern adopted in the AI Domain • Obsolete AI Table substituted by new official AI Tool for content development phase and arxml generation • Enhanced collections arxml deliverables structure
2013-10-31	4.1.2	AUTOSAR Release Management	• New ARXML file distribution feature
2013-03-15	4.1.1	AUTOSAR Administration	• Updated categories of model elements (Data Constraints and Keywords to Blueprints) • Introduced description of Backward Compatibility, Lifecyle State and Variant Handling (Views) concepts • Deliverables from Application Interfaces updated
2011-12-22	4.0.3	AUTOSAR Administration	• Description of Categories of model elements created • Synchronization of Update of XML package structure especially regarding Port Blueprints • Synchronization to updates of AUTOSAR meta model • Description of Naming conventions for connectors
2011-04-15	4.0.2	AUTOSAR Administration	• Initial Release

Disclaimer

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

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

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

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

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

Table of Contents

1 Purpose of this document 7
1.1 Document Overview . . . 7
2 Definition of terms and acronyms 8
2.1 Acronyms and abbreviations . . . 8
3 Related Documentation 9
3.1 Input documents & related standards and norms . . . 9
4 Introduction to Application Interfaces 11
5 AUTOSAR Methodology 14
5.1 Overview on available documents . . . 16
5.1.1 Software Component Template . . . 16
5.1.2 Standardization Template . . . 16
5.1.3 Generic Structure Template . . . 17
5.1.4 AI Specification and AI Specification Examples . . . 17
5.1.5 Modeling Guide for Application Interfaces . . . 17
5.1.6 AUTOSAR Methodology . . . 17
5.1.7 Explanation of Application Interfaces for Domain Body Comfort . . . 17
5.1.8 Explanation of Application Interfaces for Domain Powertrain . . . 17
5.1.9 Explanation of Application Interfaces for Domain Chassis . . . 17
5.1.10 Explanation of Application Interfaces for Domain Occupant and Pedestrian Safety . . . 18
5.1.11 Explanation of Application Interfaces for Domain Multimedia, Telematics, Human Machine Interface . . . 18
6 Metamodel representation of the contents maintained by the AI-Tool 19
6.1 Category of model elements . . . 19
6.2 Meta model diagrams and their application in the AI-Tool data structure . 20
6.2.1 Components . . . 20
6.2.2 Multiple Instantiation . . . 24
6.2.3 Blueprint Mapping & BlueprintMappingSet . . . 27
6.2.4 Ports . . . 29
6.2.5 Connectors . . . 32
6.2.6 PortInterfaces . . . 35
6.2.7 DataTypes . . . 40
6.2.8 Physical Units . . . 47
6.2.9 Computation Methods . . . 48
6.2.10 Keyword and KeywordSet . . . 49
7 Backward Compatibility 53
7.1 Introduction . . . 53
7.2 Backward Compatibility Definition . . . 54

7.3 Summary . . . 57
8 Life Cycles and Workflow 58
8.1 Introduction . . . 58
8.2 Life-Cycle States . . . 58
8.2.1 Representation in Metamodel and .arxml . . . 59
8.2.2 Representation in AI-Tool . . . 61
8.3 Milestones . . . 62
8.4 Work-Flow and Milestone propagation . . . 63
8.4.1 Application Interfaces Work-Flow . . . 63
8.4.2 Milestone Propagation . . . 65
9 View Concept in Application Interfaces (Variant Handling) 68
9.1 Introduction . . . 68
9.2 Implementation in Application Interfaces and Meta Model Representation 68
10 Data Structure of Application Interface Definitions in the AI-Tool 74
10.1 Main Views of the AI-Tool . . . 74
10.1.1 Explorer View . . . 74
10.1.2 Editor View . . . 75
10.1.3 Working Panel View . . . 76
10.2 Overview: Table AI Specification in Tabular View . . . 77
10.3 Application Array Data Types . . . 78
10.4 Application Primitive Continuous Data Types . . . 82
10.5 Application Primitive Enum Data Types . . . 87
10.6 Application Record Data Types . . . 90
10.7 Computation Methods . . . 94
10.7.1 Computation Methods of type IDENTICAL . . . 96
10.7.2 Computation Methods of type LINEAR . . . 99
10.7.3 Computation Methods of type TEXTTABLE . . . 101
10.7.4 Computation Methods of type BITFIELD_TEXTTABLE . . . 104
10.7.5 Computation Methods of type RAT_FUNC . . . 105
10.7.6 Computation Methods of type SCALE_LINEAR . . . 105
10.7.7 Computation Methods of type SCALE_LINEAR_AND_TEXTTABLE 105
10.7.8 Computation Methods of type SCALE_RAT_FUNC . . . 106
10.7.9 Computation Methods of type SCALE_RATIONAL_AND_TEXTTABLE 106
10.7.10 Computation Methods of type TAB_NOINTP . . . 106
10.8 Port Interfaces . . . 106
10.8.1 Client-Server Interfaces . . . 107
10.8.2 Sender-Receiver Interfaces . . . 112
10.9 Data Constraints . . . 115
10.10 Keywords . . . 119
10.11 Port Prototypes . . . 122
10.12 SW Components . . . 127
10.13 Connectors . . . 133

10.13.1 DelegationSwConnector 134
10.13.2 AssemblySwConnector 135
10.14 Physical Units 137
10.15 Physical Dimensions 139
11 Relationship between AI-Tool data and .arxml Output 144
11.1 Overview 144
11.1.1 Dependencies of XML Generation 144
11.1.2 Contents of Generated XML 144
11.1.3 Schema Structure 149
11.2 PortPrototypeBlueprints 151
11.3 PortInterfaces 151
11.3.1 Sender-Receiver-Interface 152
11.4 Life Cycle State 153
11.5 Blueprint Mapping Sets 157

1 Purpose of this document

AUTOSAR Application Interfaces aims at the delivery of functionality through communicating Software-Components, which can be placed nearly arbitrarily on a network of ECUs. To ensure the interoperability of Software-Components from different sources (i.e. vendors), the interfaces of these shall be unified.

The content collected in the CP_MOD_AISpecification[1] and in CP_MOD_AISpecificationExamples [2] which are maintained via the AI-Tool is the specification of interfaces of several automotive domains. The goal is to define and publish stable and widely accepted application interfaces.

This document aims at explaining all relevant details about the information structure used to collect the necessary information to define these standardized application interfaces, especially for AUTOSAR WG AIF members, developing and maintaining the definitions in the AI-Tool. Experienced AI-Tool users can skip the Chapter 5, AUTOSAR Methodology. Some sections contain extracts from other AUTOSAR documents. In case of differences in the contents then the original AUTOSAR documents are valid.

1.1 Document Overview

This document gives an overview of the methodological background of the application interfaces. It gives an overview of the content and structure of the Application Interface Specification (CP_MOD_AISpecification) and of CP_MOD_AISpecificationExamples (CP_MOD_AISpecificationExamples), collections of arxml-files containing the final output of the maintenance process which is supported by the AI-Tool. The document also describes the work-flow for the maintenance of the contents of the AI-Tool and of the resulting .arxml-files.

Note: The figures and screen shots shown in the following sections are only snapshots which are used to explain certain facts. They may not match exactly to the latest AI-Tool views.

2 Definition of terms and acronyms

2.1 Acronyms and abbreviations

The content of this chapter is informative in nature and shall not be considered as normative content. 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.

Terms and acronyms that are defined more generally in AUTOSAR are collected in: [3].

Abbreviation / Acronym:	Description:
.ai, .ai-file	file maintained by the AI-Tool, collecting all information for process of the definition of Application Interfaces
.arxml	AUTOSAR Extensible Markup Language File
AI-Tool	Tool supporting the process of Application Interfaces definition
CPU	Central Processing Unit
ECU	Electronic Control Unit
HMI	Human Machine Interface
MS	Milestone
RTE	Run-Time Environment
SW-C	Software Component
SW	Software
VFB	Virtual Function Bus
WG	Working Group

Table 2.1: Acronyms and abbreviations used in the scope of this Document

3 Related Documentation

3.1 Input documents & related standards and norms

[1] XML Specification of Application Interfaces
AUTOSAR_CP_MOD_AISpecification

[2] Application Interface Examples
AUTOSAR_CP_MOD_AISpecificationExamples

[3] Glossary
AUTOSAR_FO_TR_Glossary

[4] Software Component Template
AUTOSAR_CP_TPS_SoftwareComponentTemplate

[5] Standardization Template
AUTOSAR_FO_TPS_StandardizationTemplate

[6] SW-C and System Modeling Guide
AUTOSAR_CP_TR_SWCModelingGuide

[7] Methodology for Classic Platform
AUTOSAR_CP_TR_Methodology

[8] Generic Structure Template
AUTOSAR_FO_TPS_GenericStructureTemplate

[9] Explanation of Application Interfaces of the Body and Comfort Domain
AUTOSAR_CP_EXP_AIBodyAndComfort

[10] Explanation of Application Interfaces of the Powertrain Engine Domain
AUTOSAR_CP_EXP_AIPowertrain

[11] Explanation of Application Interfaces of the Chassis Domain
AUTOSAR_CP_EXP_AIChassis

[12] Explanation of Application Interfaces of Occupant and Pedestrian Safety Systems Domain
AUTOSAR_CP_EXP_AIOccupantAndPedestrianSafety

[13] Explanation of Application Interfaces of the HMI, Multimedia and Telematics Domain
AUTOSAR_CP_EXP_AIHMIMultimediaAndTelematics

[14] Meta Model-generated XML Schema
AUTOSAR_FO_MMOD_XMLSchema

[15] Meta Model
AUTOSAR_FO_MMOD_MetaModel

[16] Collection of blueprints for AUTOSAR M1 models

AUTOSAR_FO_MOD_GeneralBlueprints

[17] Supplementary material of general blueprints for AUTOSAR
AUTOSAR_CP_TR_GeneralBlueprintsSupplement

[18] ARXML Serialization Rules
AUTOSAR_FO_TPS_ARXMLSerializationRules

4 Introduction to Application Interfaces

The Application Interface Tool (AI-Tool) is a user interface dedicated to manage all the data, which define the Application Interfaces (AI). The tool enables maintenance, validation, and generation of work product output. Input to the tool is data based on AUTOSAR defined methodology meta-model. The output of the tool is a collection of .arxml (AUTOSAR XML) files containing detailed information of all standardized Application Interfaces transferred to a commonly readable format, which can be the input for e.g. authoring tools at the development departments of SW developing companies.

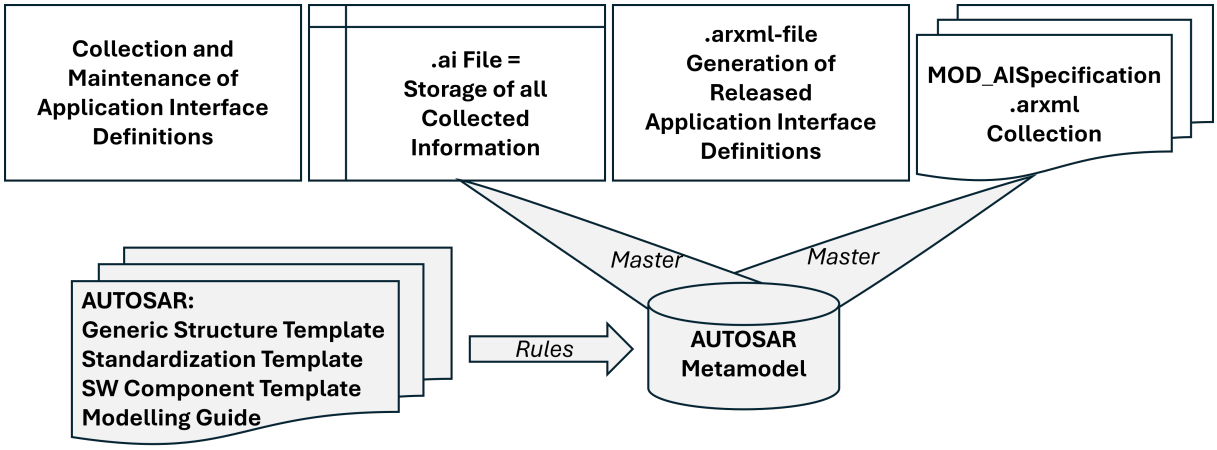


Figure 4.1: The Application Interfaces Process

The AI-Tool enables the manipulation of AI definitions in order to produce the outcome: the .arxml file for data exchange.

The AUTOSAR Software Component Template [4] tells us what can be modeled. The Standardization Template [5] explains and supports the Blueprint approach.

The Application Interface definitions apply the rules defined by AUTOSAR Software Component Modeling Guide [6].

All definitions are finally provided in the AUTOSAR_CP_MOD_AISpecification [1] and AUTOSAR_CP_MOD_AISpecificationExamples [2].

In order to standardize Application Interfaces a number of SW Components are described within the AI-Tool decomposed into bigger function groups (formerly called domains) and their main functions. Nevertheless, these components must be seen as examples only. They are necessary in order to specify the ports/port prototypes in a consistent way. Each Port/PortPrototype needs a connection to a Component to be specified in a proper way.

The structure within Figure 4.2 illustrates the decomposition of Software-Compositions using the AUTOSAR Software Component Template [4].

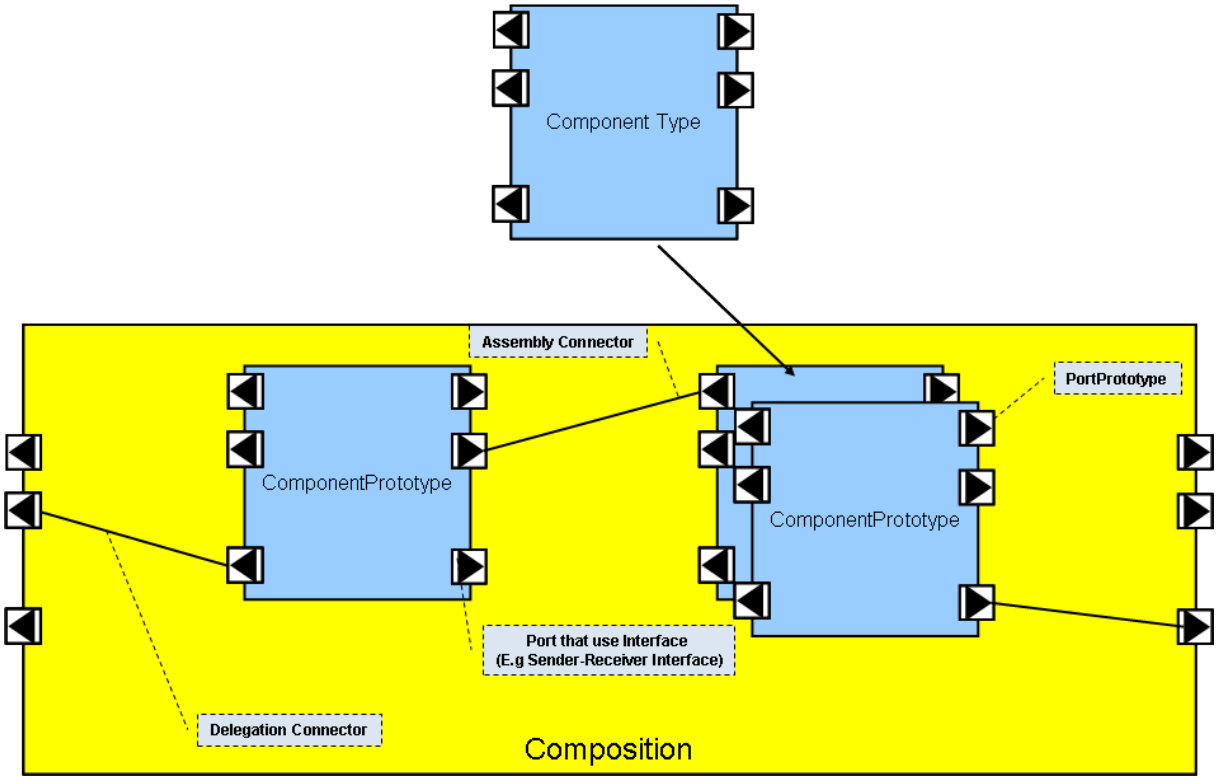


Figure 4.2: Decomposition of a component using the concepts defined in the software component template

Note:The yellow block in the figure represents a Composition that is decomposed into other Compositions/Components. These Component Types and ComponentPrototypes are described within the same ArPackages (see blue blocks in the figure). A SwComponentPrototype implements the usage of a SwComponentType in a specific role.

SwComponentPrototypes are only used for implementing SwComponentTypes in a specific role, i.e. they are used to instantiate the SwComponentType.

Example: a SwComponentPrototype "LeftDoorControl" fulfills the role of implementing the SwComponentType "DoorControl" for the left door of a vehicle while the SwComponentPrototype "RightDoorControl" fulfills the role of the SwComponentType "DoorControl" for the right door.

The AI-Tool Explorer View contains so called ArPackages where different elements of a complete Application Interfaces definition are collected and maintained. Within these ArPackages the following application interface relevant information is handled:

- Collections: Main collections of elements which are generated differently during the AUTOSAR Release Build:
  - 1 AISpecification [ARPackage]

  - 2 Physical Units [ARPackage]

The AISpecification Package contains the following collections of definitions:

- ApplicationDataTypes_Blueprint [ARPackage]
- ApplicationDataTypes_Example [ARPackage]
- CompuMethods_Blueprint [ARPackage]
- CompuMethods_Example [ARPackage]
- DataConstrs_Blueprint [ARPackage]
- DataConstrs_Example [ARPackage]
- KeywordSets_Blueprint [ARPackage]
- PhysicalDimensions_obsolete [ARPackage]
- PortInterfaces_Blueprint [ARPackage]
- PortInterfaces_Example [ARPackage]
- PortPrototypeBlueprints_Blueprint [ARPackage]
- SwComponentTypes_Example [ARPackage]
- Systems [ARPackage]
- Units_obsolete [ARPackage]

The PhysicalUnits Package contains the following collections of definitions:

- PhysicalDimensions_Blueprint [ARPackage]
- Units_Blueprint

5 AUTOSAR Methodology

AUTOSAR requires a formal technical approach for some steps of system development. This approach is called the "AUTOSAR Methodology". The AUTOSAR Methodology is neither a complete process description nor a business model and "roles" and "responsibilities" are not defined in this methodology. Furthermore, it does not prescribe a precise order in which activities should be carried out. The methodology is a mere work-product flow: it defines the dependencies of activities on work-products.

During system design, the software components and the hardware have to be selected, and overall system constraints have to be identified. AUTOSAR intends to ease the formal description of these initial system design decisions via the information exchange format and the use of templates. Defining the System Configuration Input means to fill out or edit the appropriate templates. AUTOSAR methodology allows for a high degree of reuse in this context. In any case, this editing is assumed to be supported by editing tools. Here is a brief description of activities:

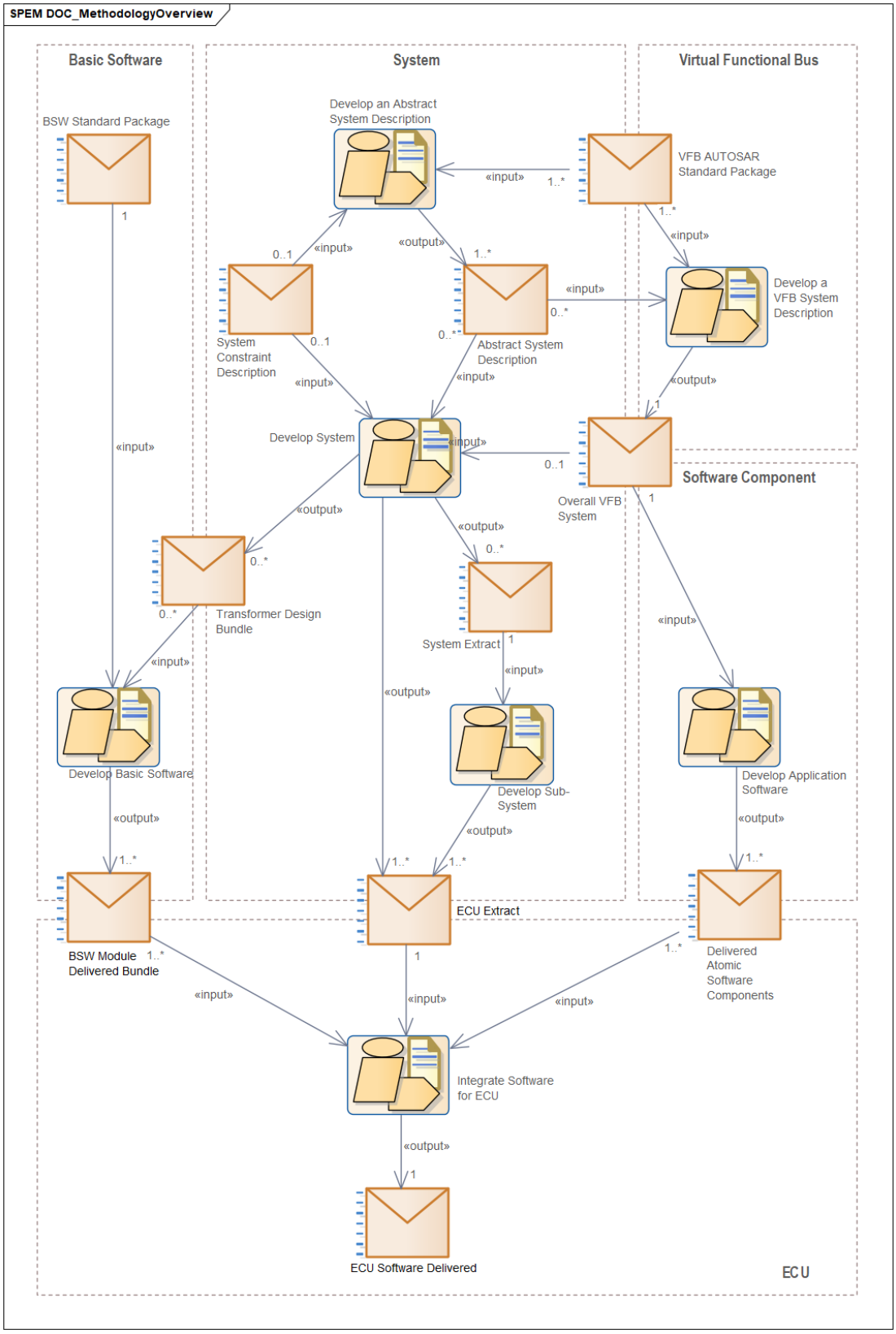


Figure 5.1: Overview of the AUTOSAR Methodology with the Virtual Functional Bus (VFB)

- **Develop a VFB System Description:** The Virtual Functional Bus (VFB) view of a system shows how the systems software and hardware functions interact independent of any network topology or deployment of features across multiple ECUs. This activity includes the following subactivities:
- **Develop System:** Develop the description of an overall AUTOSAR System as a basis to deliver system or ECU extracts.
- **Develop SubSystem:** Develop the description of a sub-system based on a given System Extract.
- **Develop Application Software:** This pattern describes the workflow and the necessary activities in terms of the AUTOSAR methodology for the development of application software components. The workflow shall allow a more or less independent development of the software component core functionality. These activities have to be performed for every application software component.

  The AI-Tool information schema and its functionality base on the principles of the VFB System Description Development. They are applied to collect all necessary information of Application Interface definitions (Data Model Development) and Examples of SwComponents (Component Model Development).
- **Develop Basic Software:** Describes the overall activities to develop Basic Software, starting from the design down to delivery of modules.

  According to these principles, the development and implementation of a software component is more or less independent of ECU configuration.

The general concepts of this chapter are an extract of the detailed AUTOSAR Methodology [7].

5.1 Overview on available documents

For detailed information, following documents are available.

5.1.1 Software Component Template

This document [4] provides introductory description and rationale for the part of the AUTOSAR meta-model relevant for the definition of Software Components.

5.1.2 Standardization Template

This document [5] is intended to support the delivery of standardized model elements by AUTOSAR. This document also refines the blueprint approach for standardization.

5.1.3 Generic Structure Template

This document [8] acts as a supplement for the formal definition provided by the AUTOSAR meta model. This document provides the introductory description and rationale for the parts of the AUTOSAR meta model relevant for all AUTOSAR templates.

5.1.4 AI Specification and AI Specification Examples

AUTOSAR_CP_MOD_AISpecification and MOD_AISpecificationExamples are the outputs of the standardization of Application Interfaces ([1], [2]). The outputs are delivered as a set of .arxml files.

5.1.5 Modeling Guide for Application Interfaces

This document [6] gives guidelines and conventions on using the AUTOSAR model elements in order to build AUTOSAR systems. It does not contain guidelines for the AUTOSAR meta-model.

5.1.6 AUTOSAR Methodology

See above [7].

5.1.7 Explanation of Application Interfaces for Domain Body Comfort

The document [9] explains design decisions and boundary conditions that lead to the Application Interfaces of the domain Body and Comfort.

5.1.8 Explanation of Application Interfaces for Domain Powertrain

The document [10] explains design decisions and boundary conditions that lead to the Application Interfaces of the domain Powertrain.

5.1.9 Explanation of Application Interfaces for Domain Chassis

The document [11] explains design decisions and boundary conditions that lead to the Application Interfaces of the domain Chassis.

5.1.10 Explanation of Application Interfaces for Domain Occupant and Pedestrian Safety

The document [12] explains design decisions and boundary conditions that lead to the Application Interfaces of the domain Occupant and pedestrian safety.

5.1.11 Explanation of Application Interfaces for Domain Multimedia, Telematics, Human Machine Interface

The document [13] explains design decisions and boundary conditions that lead to the Application Interfaces of the domain Multimedia, Telematics, Human Machine Interface.

6 Metamodel representation of the contents maintained by the AI-Tool

This section describes the relation between meta-model elements (AUTOSAR Meta Model [14], [15]) and their representation within the AI-Tool, in AUTOSAR_CP_MOD_AISpecification, and in AUTOSAR_CP_MOD_AISpecificationExamples.

The AUTOSAR Meta Model conceptually is defined as 'M2' level, which describes the entities which are used to describe, e.g. Software Components and Ports. The relations between those entities as well as their semantics are part of this model.

6.1 Category of model elements

All Application Interface model elements are classified into different categories. These are:

Category	Description	Categorized elements
BLUEPRINT	Blueprints are predefined model elements, which form the basis for further modeling. Blueprints are model elements from which other model elements can be derived by copying. These elements are not complete in all aspects. They act as a template for projects to create the real elements.	Application Data Types ([1]) Computation Methods ([1]) Data Constraints ([1]) Keyword Sets ([1]) Port Interfaces ([1]) Port Prototypes Blueprints ([1]) Physical Dimensions (FO_MOD_PhysicalDimensions_Blueprint) Units (FO_MOD_Units_Blueprint) Collections ([1]
EXAMPLE	Elements that are also reusable, but are also helpful for understanding are created as category EXAMPLE. These elements were created as help for the users to actually create their project specific implementations. They represent one out of many possible ways of implementation.	SwComponentTypes ([2]) Application Data Types ([2]) Blueprint Mapping Sets ([2]) Computation Methods ([2]) Data Constraints ([2]) Port Interfaces ([2]) SW Design Patterns ([2])
STANDARD	All elements, which can be used as they are defined by just including them in the project, belong to the category STANDARD. These elements need no modifications before they are used in the projects.	LifeCycleInfoSets PhysicalDimensions_obsolete Units_obsolete

Table 6.1: Categories of model elements

(The listed elements which are marked as obsolete are still generated during Release Build for Backward Compatibility (BWC) needs.)

ApplicationDataTypes, CompuMethods, DataConstrs and PortInterfaces categorized as EXAMPLE are the derived elements from their blueprints elements.

6.2 Meta model diagrams and their application in the AI-Tool data structure

This section describes the relationship between the AUTOSAR Meta-Model (AUTOSAR_FO_MMOD_MetaModel (M2)) diagrams and the AI-Tool contents to enable the development of application software-components.

6.2.1 Components

Data for the definition of software component types are collected in the AI-Tool in the package SwComponentTypes_Example. It is not in the scope of the AUTOSAR Application Interfaces to define complete system compositions. All software component composition types are defined in a package with category EXAMPLE and they are meant only as illustration of usage of the standardized elements.

This package contains another package SwDesignPatterns where example implementations for template software components are collected. On the same hierarchical level in the Explorer View as this package, the composition types are listed. The levels below the composition software components represent the inner structure of the compositions, the contained components, it's ports and the connectors between it's ports and the ports of the inner components.

6.2.1.1 Composition

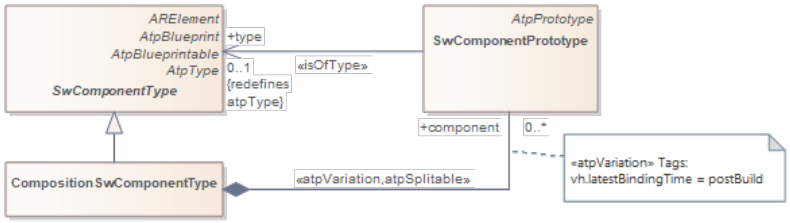


Figure 6.1: The recursive relation of software-components and compositions

The purpose of an AUTOSAR CompositionSwComponentType is to allow the encapsulation of specific functionality by aggregating existing SoftwareComponentPrototypes which are instances of predefined SwComponentTypes. Since a CompositionSwComponentType is also a SwComponentType, it may be aggregated again in further CompositionSwComponentTypes. This recursive relation is formally expressed in Figure 6.1.

It is important to understand that while compositions allow for (sub-) system abstraction, they are solely an architectural element for the implementation of model scalability. They simply group existing software-components and thereby take away complexity when viewing or designing logical system architecture. In the AI-Tool Explorer View all SwComponentTypes found in AISpecification/SwComponentTypes_Example are defined as CompositionSwComponentTypes because even the simplest component here

must contain at least one AtomicSwComponent describing the internal behavior of the SwComponent.

Meta Model Reference:
M2::AUTOSARTemplates::SWComponentTemplate::Composition [4]

The following XML snippet shows the XML generated for the Pt (Powertrain) CompositionComponent

Generated .arxml Example:

```
<COMPOSITION-SW-COMPONENT-TYPE>
  <SHORT-NAME>Pt</SHORT-NAME>
  <LONG-NAME>
     <L-4 L="EN">Powertrain</L-4>
   </LONG-NAME>
   <DESC>
     <L-2 L="EN">Powertrain</L-2>
   </DESC>
   <PORTS>
     <P-PORT-PROTOTYPE>
       <SHORT-NAME>TrsmStAxleFrntActr</SHORT-NAME>
        ...
     </P-PORT-PROTOTYPE>
     <R-PORT-PROTOTYPE>
       <SHORT-NAME>AbsCtrlIntvg</SHORT-NAME>
          ...
     </R-PORT-PROTOTYPE>
  </PORTS>
  <COMPONENTS>
   <SW-COMPONENT-PROTOTYPE>
      <SHORT-NAME>PtCoorr</SHORT-NAME>
      <LONG-NAME>
      <L-4 L="EN">Powertrain Coordinator</L-4>
         ...
   </COMPONENTS>
  <CONNECTORS>
    <ASSEMBLY-SW-CONNECTOR>
       <SHORT-NAME>AccrPedlMeasdOfVehMtnForPtToAccrPedlMeasdOfPtCoorr
       </SHORT-NAME>
       <PROVIDER-IREF>
         <CONTEXT-COMPONENT-REF BASE="SwComponentTypes"
           DEST="SW-COMPONENT-PROTOTYPE">Pt/VehMtnForPt
         </CONTEXT-COMPONENT-REF>
         <TARGET-P-PORT-REF BASE="SwComponentTypes"
           DEST="P-PORT-PROTOTYPE">VehMtnForPt/AccrPedlMeasd
         </TARGET-P-PORT-REF>
        </PROVIDER-IREF>
        <REQUESTER-IREF>
        ...
  </CONNECTORS>
</COMPOSITION-SW-COMPONENT-TYPE>
```

AI-Tool Reference:
AI-Tool Project Explorer: ARPackage: SwComponentTypes_Example
AI-Tool Tabular View: Table Sw Component Type Examples

AUTOSAR_MOD_AISpecification Reference:
MOD_AISpecification Reference: AUTOSAR_CP_MOD_AISpecificationExamples

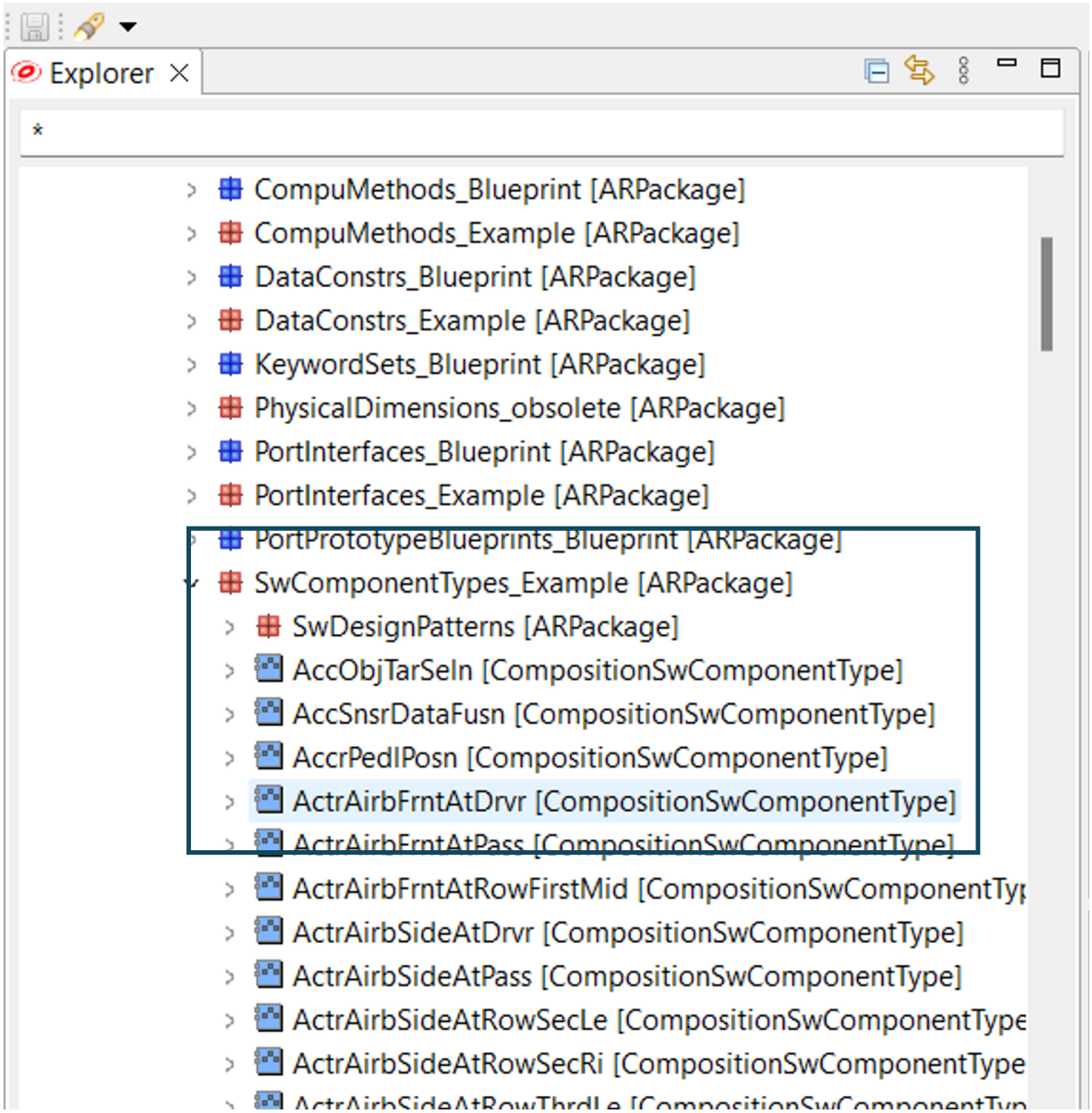
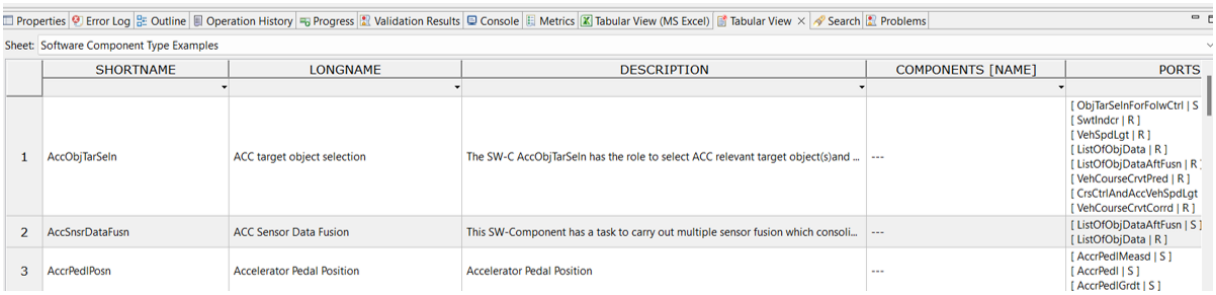


Figure 6.2: SwComponentType Examples in the Explorer View of the AI-Tool



	SHORTNAME	LONGNAME	DESCRIPTION	COMPONENTS [NAME]	PORTS
1	AccObjTarSeln	ACC target object selection	The SW-C AccObjTarSeln has the role to select ACC relevant target object(s)and ...	---	[ObjTarSelnForFolwCtrl | S [SwtIndcr | R] [VehSpdLgt | R] [ListOfObjData | R] [ListOfObjDataAftFusn | R [VehCourseCrvtPred | R] [CrsCtrlAndAccVehSpdLgt [VehCourseCrvtCorrd | R]
2	AccSnsrDataFusn	ACC Sensor Data Fusion	This SW-Component has a task to carry out multiple sensor fusion which consoli...	---	[ListOfObjDataAftFusn | S] [ListOfObjData | R]
3	AccrPedlPosn	Accelerator Pedal Position	Accelerator Pedal Position	---	[AccrPedlMeasd | S] [AccrPedl | S] [AccrPedlGrdt | S]

Figure 6.3: SwComponentType Examples in the Tabular View of the AI-Tool

Example: In the case of Exterior light [CompositionSwComponentType] found in the AI-Tool Explorer View:
In AISpecification/SwComponentTypes_Example (Figure 6.4) the component ExtrLi is a CompositionSwComponent type, composed of different SwComponentPrototypes like ActrOfHdlampLvlgLe, ActrOfHdlampLvlgRi, AdprCornrg, AdprHomeCmngAndHomeLvng, BeamHiLe, BeamHiRi,..., ExtrLiMgr,..., etc.

Example: The "Composition"SwComponentType ExtrLiMgr (Figure 6.4) is not decomposed further into SwComponentPrototypes. Therefore, it is just a SwComponentType. The SwComponentPrototype ExtrLiMgr found in ExtrLi [CompositionSwComponentType] is of the type ExtrLiMgr [CompositionSwComponentType]. Here the type and prototype have the same ShortName.

Example: The SwComponentPrototypes ActrOfHdlampLvlgLe and ActrOfHdlampLvlgRi found in ExtrLi (Figure 6.4) are of the type ActrOfHdlampLvlg [CompositionSwComponentType]. Here the type and prototype have different ShortName and the type is instantiated twice.

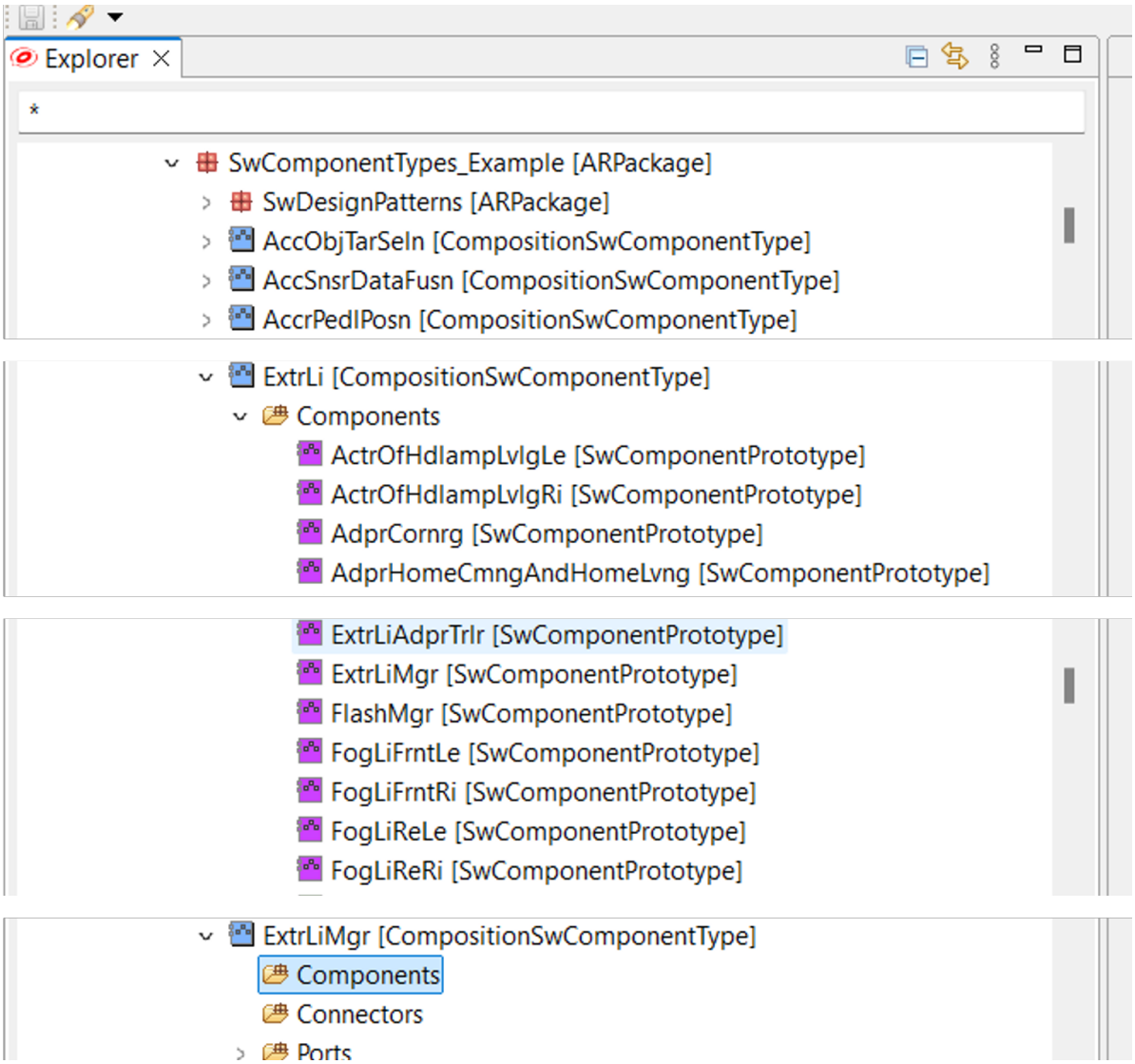


Figure 6.4: Exterior Light Components Decomposition

There can be arbitrary numbers of SwComponentPrototypes that refer to specific SwComponentTypes created. Note that a CompositionSwComponentType also aggregates the abstract meta-class SwConnector to describe the connection of the SwComponentPrototypes exchanging information via Ports and their Interfaces.

6.2.2 Multiple Instantiation

When designing a system it is often the case that elements in the runtime space share the same structure. A well-known example domain is object-oriented programming, where objects instantiated from the same class all have the same structure specified by that class. The ability to specify a structure once and then use it in multiple places in the design is called multiple instantiation.

The same concept is used in the AI-Tool for the definition of SwComponentTypes (Classes) for multiple SwComponentPrototypes (Instances) which might exist in several CompositionSwComponentTypes.

In the example shown below (Figure 6.5), the SwComponentPrototypes WshrFrnt, WshrRe and WshrHdlamp which are only existing as subcomponents of the CompositionSwComponentType WiperWashr are created via instantiation of the SwComponentType Wshr.

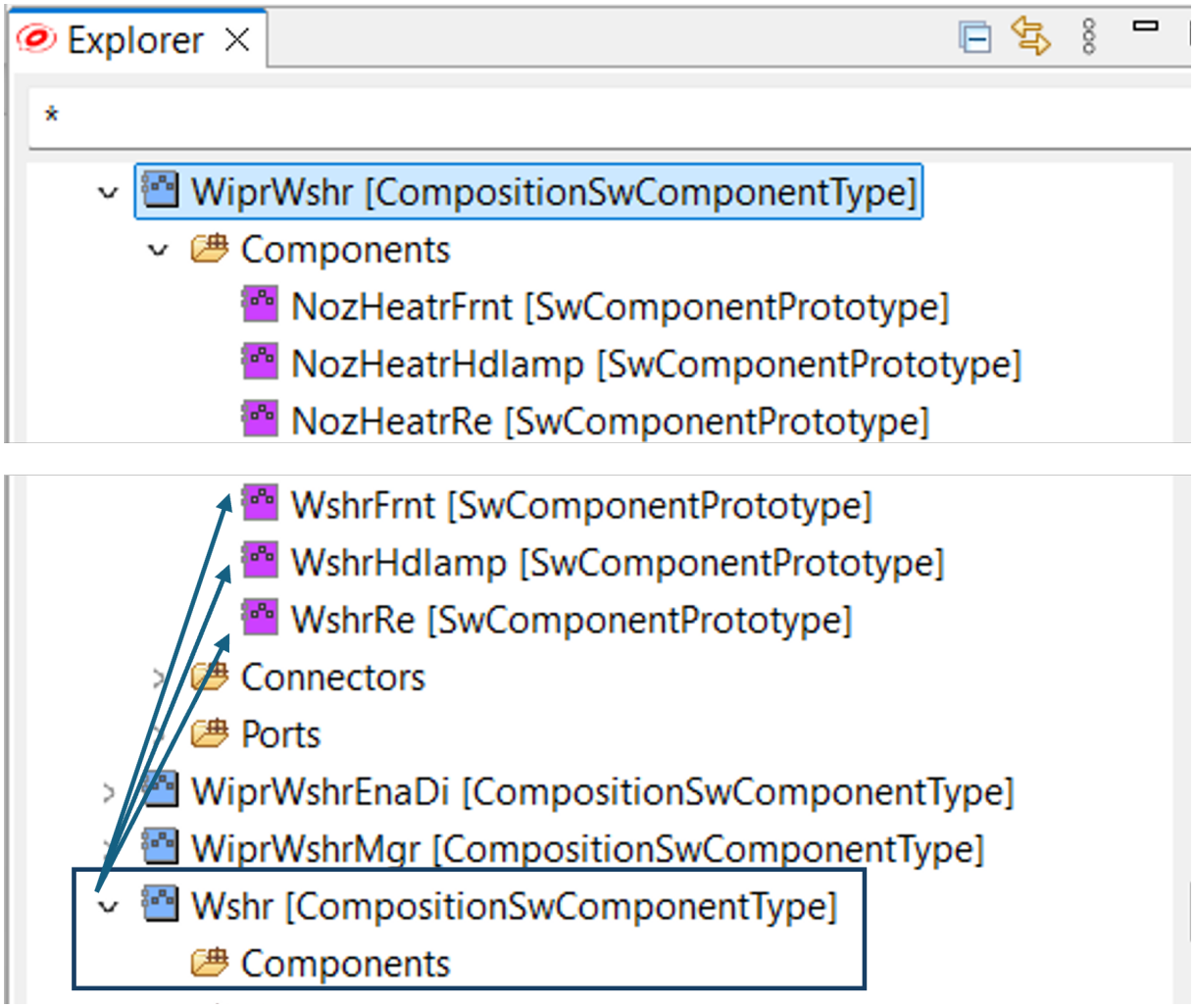


Figure 6.5: WiperWasher - Example of Multiple Instantiation

All the SwComponentPrototypes derived from the SwComponentType by instantiation will inherit the properties of the SwComponentType. In other words: if a SwComponentType is defined with 2 Provider PortPrototypes and 3 Receiver PortPrototypes then all the SwComponentPrototypes derived will have the same number of Provider and Receiver PortPrototypes.

Via a TYPE-TREF, the component prototype references a composition type, which is generated according to the prototype definition.

Example: In the following listing the reference between the SwComponentPrototype PtCoorr and the SwComponentType PtCoorr is described.

```
<COMPONENTS>
   <SW-COMPONENT-PROTOTYPE>
      <SHORT-NAME>PtCoorr</SHORT-NAME>
      <LONG-NAME>
         <L-4 L="EN">Powertrain Coordinator</L-4>
      </LONG-NAME>
      <DESC>
         <L-2 L="EN">Powertrain Coordinator</L-2>
      </DESC>
      <TYPE-TREF BASE="SwComponentTypes"
        DEST="COMPOSITION-SW-COMPONENT-TYPE">PtCoorr
      </TYPE-TREF>
   </SW-COMPONENT-PROTOTYPE>
```

If two SwComponentTypes are connected to each other and both SwComponentTypes are multiply instantiated then it is possible that the connections between these SwComponentTypes are ambiguous.

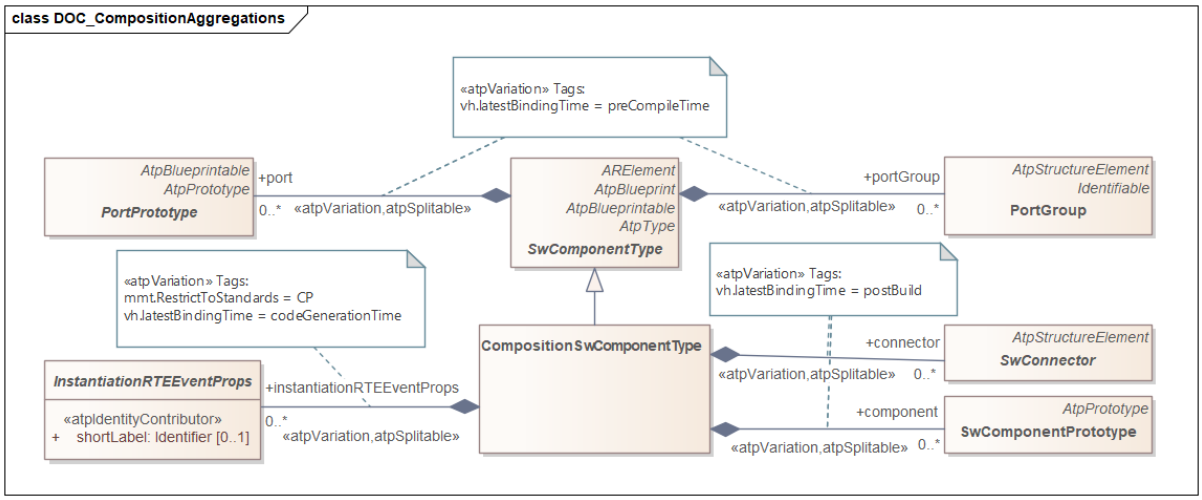


Figure 6.6: Composition - Aggregations

Meta Model Reference:
M2::AUTOSARTemplates::SWComponentTemplate::Software Components, Ports, and Interfaces [4]

AUTOSAR_MOD_AISpecification Reference:
AUTOSAR_CP_MOD_AISpecificationExamples

Note that being a SwComponentType, a CompositionSwComponentType also exposes PortPrototypes to the outside world. However, the PortPrototypes are only delegated and do not play the same role as PortPrototypes attached to AtomicSwComponentTypes (AtomicSwComponentTypes encapsulate the implementation of their functionality and behavior and merely expose well defined connection points, called PortProto-

types, to the outside world.). For more details, refer to the SW Component Template [4].

6.2.3 Blueprint Mapping & BlueprintMappingSet

It is not in the scope of the AUTOSAR Application Interfaces to define complete system compositions. All software component composition types are defined in a package with category EXAMPLE and meant only as illustration of usage of the standardized elements. However, it is in the scope of the Application Interfaces to describe the roles that interfaces can play in compositions. This can be done using Blueprints of elements which shall be standardized, like, e.g. PortPrototypeBlueprints, which define potential ports of a component type and can carry more attributes to pre-define values for usages of the blueprint, e.g. an initial value.

A blueprint mapping acts as reference between the blueprint and the derived element. Blueprint mapping describes the relationship between the blueprinted element and the actual blueprint. It also validates the derived element against the blueprint. Aggregation of these BlueprintMappings is a BlueprintMappingSet. The Figure below shows the BlueprintMapping and BlueprintMappingSet. The AI-Tool records the Blueprint Mapping automatically when a prototyped element is derived from a blueprint element.

Meta Model Reference:
M2::AUTOSARTemplates::CommonStructure::SoftwareComponentTemplate::Blueprints [4]

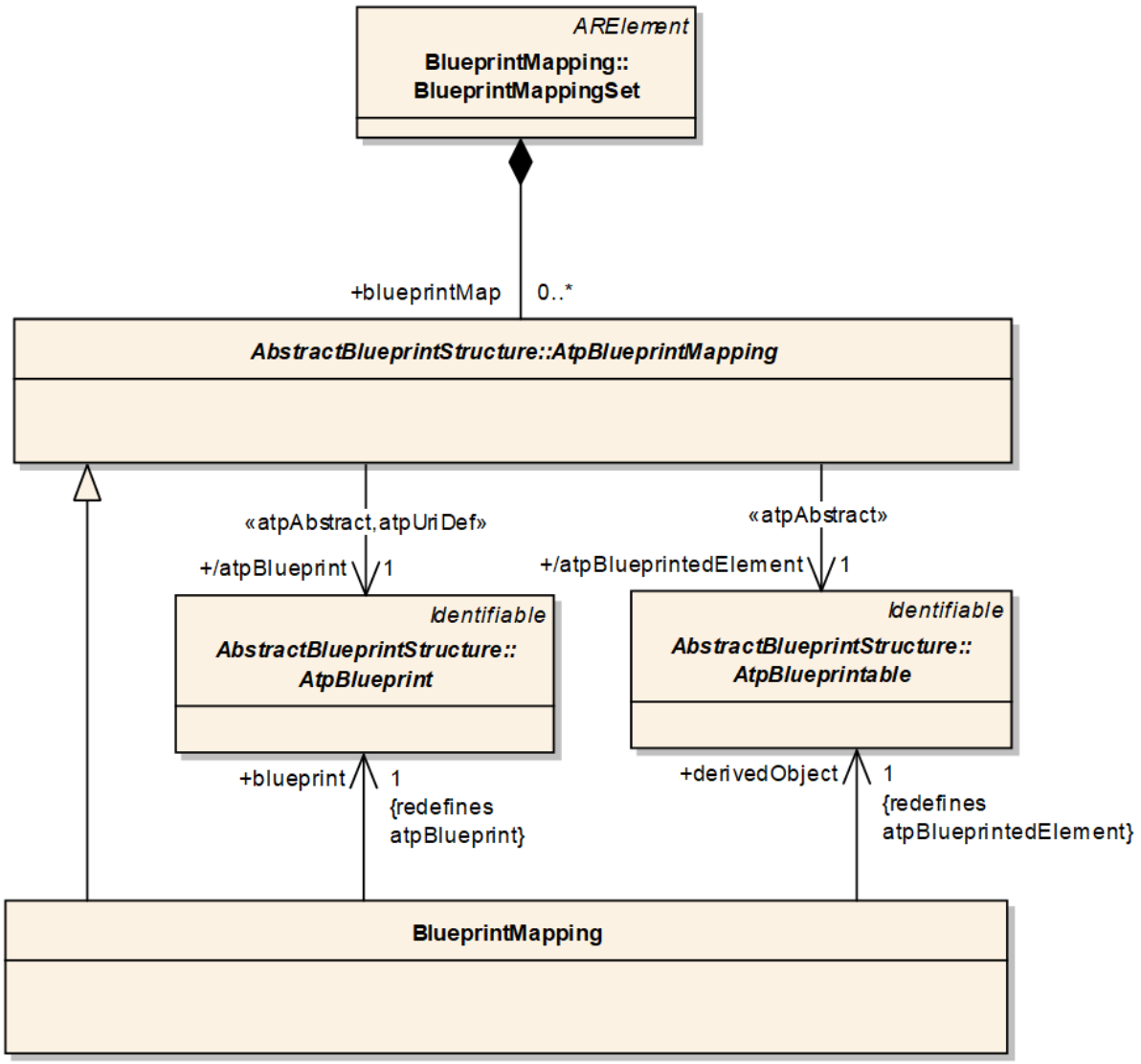


Figure 6.7: BlueprintMapping and BlueprintMappingSet

AUTOSAR_MOD_AISpecification Reference:
AUTOSAR_CP_MOD_AISpecificationExamples, Subpackage BlueprintMappingSets_Example

AI-Tool Reference:
BlueprintMappings are represented by the list of "Derived Elements", e.g., in the PortPrototypeBlueprint definition editor, or in the "Derived from:" information of the element editors of EXAMPLE elements of the AI-Tool.

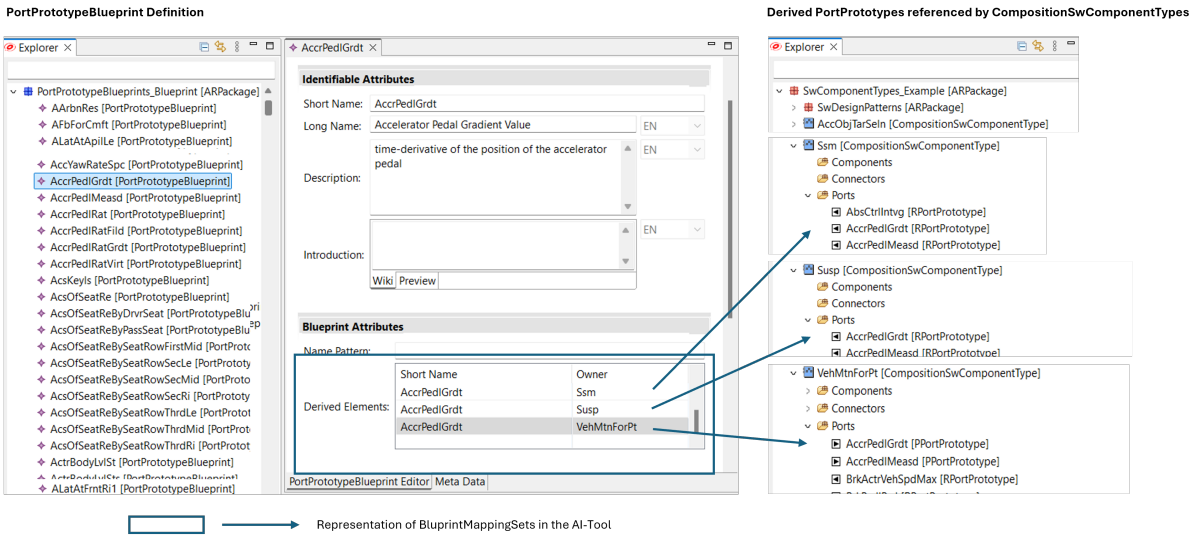


Figure 6.8: Blueprint Mapping Representation in the AI-Tool

6.2.4 Ports

PortPrototypes also sometimes referred as Ports are the well-defined connection points for communication between different software-components. Two SwComponentPrototypes are eventually connected by hooking up a PPortPrototype of one SwComponentPrototype to a compatible RPortPrototype of the other SwComponentPrototype.

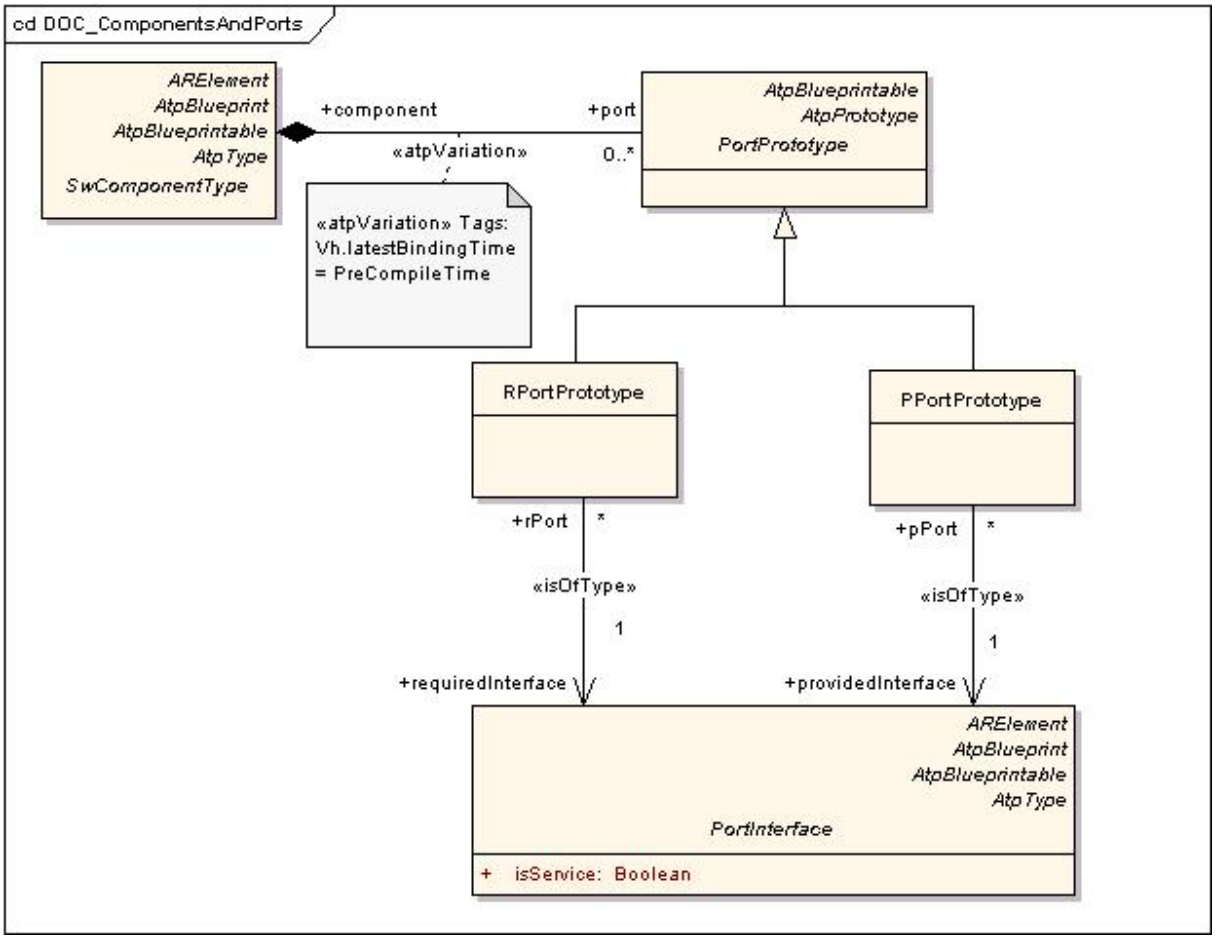


Figure 6.9: PortPrototypes

Depending on its integration in the SwComponentType any PortPrototype is either required type, or provided type or provided and required type.

Type	Name	Description
Required Port	RPortPrototype	Requires certain services or data from other SwComponentTypes, represents an input information to the SWComponentType
Provided	PPortPrototype	Provides certain services or data of the SwComponentType to other SwComponentTypes, represents an output information of the SWComponentType
Required and provided	PRPortPrototype	Both requires and provides certain services or data of the SwComponentType to and from other SwComponents, and, e.g., as feedback, to the SwComponent itself

Table 6.2: Types of PortPrototypes

6.2.4.1 PortPrototypeBlueprints

A PortPrototypeBlueprint is an ARElement and acts as a blueprint for the creation of PortPrototypes. A user can pick a specific PortPrototypeBlueprint and create PortPrototype out of it.

A PortPrototypeBlueprint is not related to any SwComponentType. The PortPrototypeBlueprints are collected in a separate package in the AUTOSAR_CP_MOD_AISpecification.

PortPrototypeBlueprints can be seen as a library, from which the user can choose a PortPrototypeBlueprint as a template to create a PortPrototype of a SwComponentPrototype. Consequently, the PortPrototypeBlueprints are just a collection of PortPrototypes without any architectural relation. As soon as a PortPrototypeBlueprint is attached to a SwComponentPrototype, it becomes a PortPrototype.

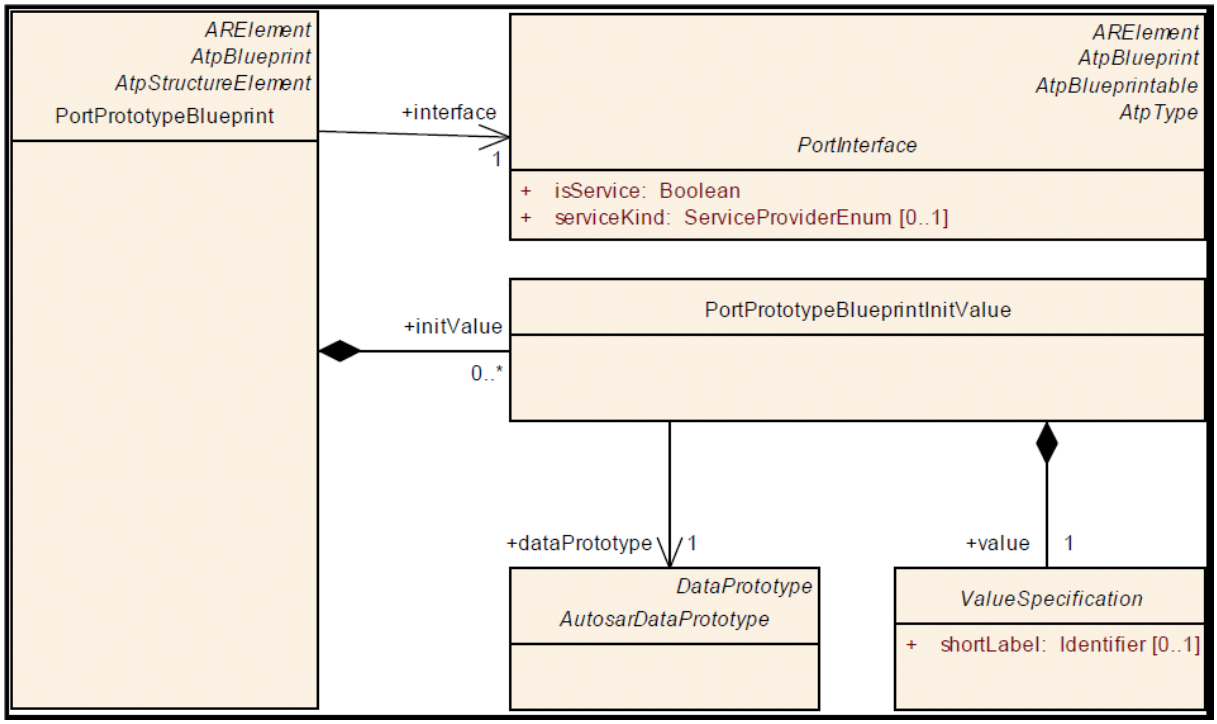


Figure 6.10: PortPrototypeBlueprints

6.2.4.2 BlueprintMapping of PortPrototypeBlueprints

The process of creating a PortPrototype from the available PortPrototypeBlueprints is called BlueprintMapping. BlueprintMapping is demonstrated in Figure 6.7. The mapping between PortPrototypes and PortPrototypeBlueprints can be found in the BlueprintMappingSet "PortPrototypeBlueprintMappings" within the package "BlueprintMappingSets_Example".

Meta Model Reference:
M2::AUTOSARTemplates::SWComponentTemplate::Ports [4]

AI-Tool Reference:
AI-Tool Project Explorer: ARPackage: PortPrototypeBlueprints_Blueprint
AI-Tool Tabular View: Table Port Prototype Blueprints

AUTOSAR_MOD_AISpecification Reference:
AUTOSAR_CP_MOD_AISpecification_PortPrototypeBlueprint_Blueprint
AUTOSAR_CP_MOD_AISpecificationExamples

6.2.5 Connectors

A CompositionSwComponentType also aggregates the abstract meta-class SwConnector to connect the SwComponentPrototypes interfering with each other.

CompositionSwComponentTypes contain three types of SwConnectors:

1. AssemblySwConnectors to interconnect PortPrototypes of SwComponentPrototypes that are part of the CompositionSwComponentType
2. DelegationSwConnectors to connect from "inner" PortPrototypes to delegated "outer" PortPrototypes and vice versa.
3. PassTroughSwConnectors to connect an "outer" R-PortPrototypes to an "outer" R-PortPrototype typed by the same Interface directly. The

In the case that the outer PortPrototype is referenced by multiple DelegationSwConnectors, the semantic is the multiplication of the AssemblySwConnectors referencing the outer PortPrototypes.

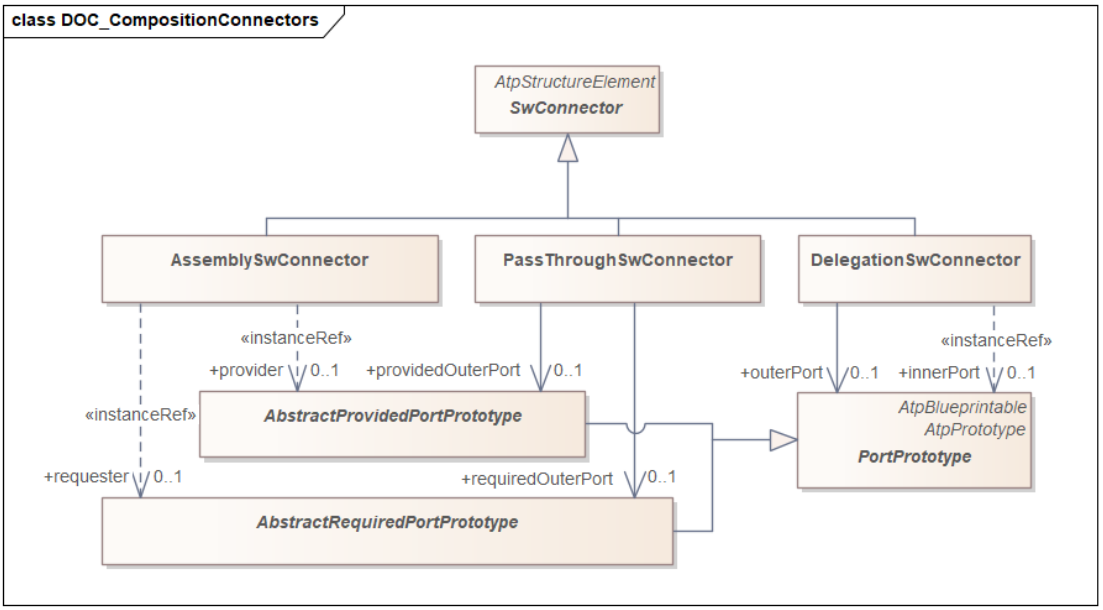


Figure 6.11: Composition - Connectors

Example: In the case of Exterior light decomposition, "ExtrLi" software component prototype is the composite type which provides an "outer" PPortPrototype "TrlrSts", which is delegated from "inner" PPortPrototype of software component prototype "ExtrLiAd-

prTrlr". The same PPortPrototype is connected to RPortPrototype of "ExtrLiAdprReLe" and "ExtrLiAdprReRi" through an assembly connector prototype.

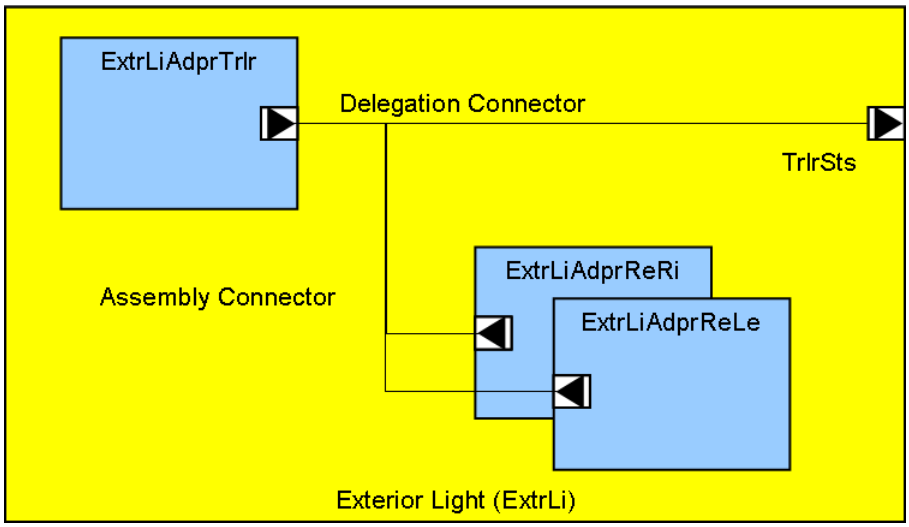


Figure 6.12: : Exterior Light Decomposition example

The AI-Tool creates automatically Delegation Connectors for the Ports having the same PortPrototypeBlueprint name and the same direction.

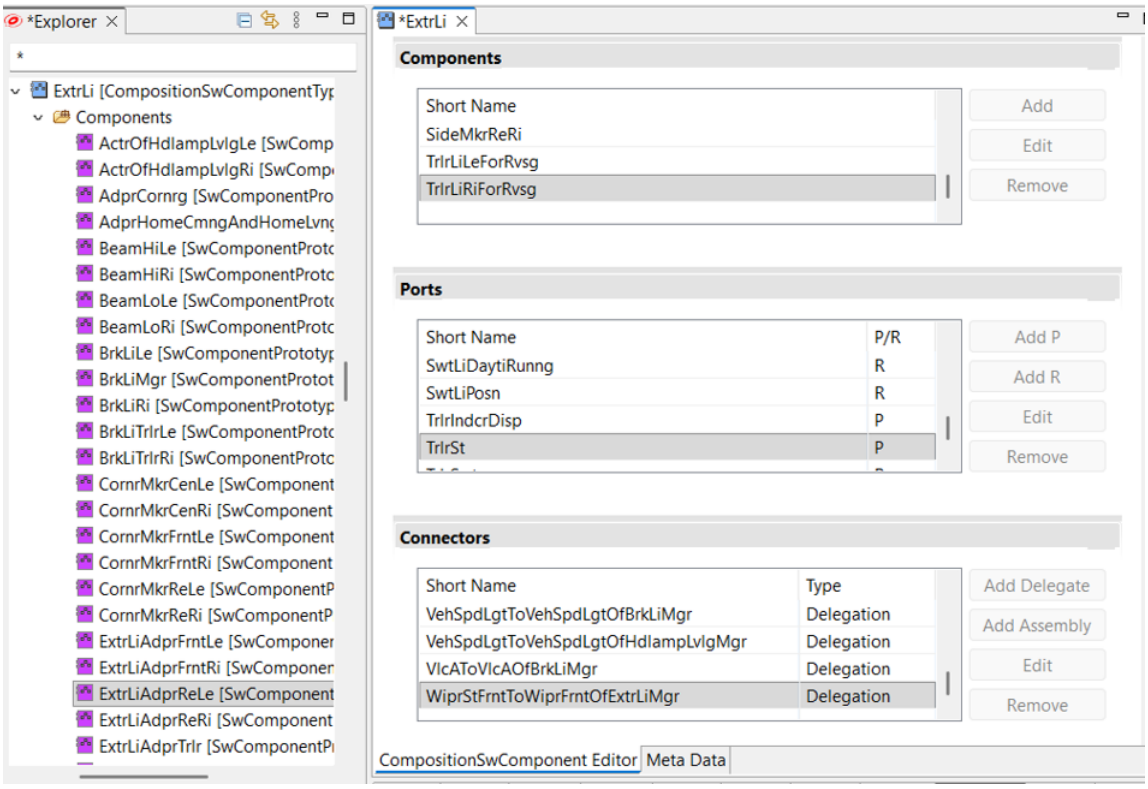


Figure 6.13: : Exterior Light Connectors in AI-Tool

```
<DELEGATION-SW-CONNECTOR>
  <SHORT-NAME>WipgSpdIntlFromHmiToWipgSpdIntlFromHmiOfWiprWshrMgr</SHORT-
    NAME>
```

```
  <INNER-PORT-IREF>
    <R-PORT-IN-COMPOSITION-INSTANCE-REF>
      <CONTEXT-COMPONENT-REF DEST="SW-COMPONENT-PROTOTYPE" BASE="
          SwComponentTypes">WiprWshr/WiprWshrMgr</CONTEXT-COMPONENT-REF>
      <TARGET-R-PORT-REF DEST="R-PORT-PROTOTYPE" BASE="SwComponentTypes">
          WiprWshrMgr/WipgSpdIntlFromHmi</TARGET-R-PORT-REF>
    </R-PORT-IN-COMPOSITION-INSTANCE-REF>
  </INNER-PORT-IREF>
  <OUTER-PORT-REF DEST="R-PORT-PROTOTYPE" BASE="SwComponentTypes">WiprWshr/
      WipgSpdIntlFromHmi</OUTER-PORT-REF>
</DELEGATION-SW-CONNECTOR>
```

The generator creates the short name of the delegation connector according to the rule defined below;

```
<outerPortName>To<innerPortName>Of<InnerComponentPrototypeName>
```

In the above example <outerPortName> is WipgSpdIntlFromHmi, <innerPortName> is WipgSpdIntlFromHmi, and <InnerComponentPrototypeName> is WiprWshrMgr. Therefore, the short name of the delegation connector shown above is WipgSpdIntlFromHmiToWipgSpdIntlFromHmiOfWiprWshrMgr.

The inner port references the port from the component prototype using an instance reference.

Note, that the context of the IREF is the component prototype WiprWshrMgr inside the composition WiprWshr, while the target port references the component type WiprWshrMgr in the general package for all composition types SwComponentTypes, thus the different path prefixes. The outer port references the port from the composition type WiprWshr.

Assembly Connectors are created for each required port and the corresponding P-Port of internal component prototypes

```
<ASSEMBLY-SW-CONNECTOR>
  <!-- sheet=050108_WiperWasher, row=43, p_col=28, r_col=48-->
  <SHORT-NAME>
      ActvnOfWshngCmdOfWshrFrntOfWiprWshrMgrToActvnOfWshngCmdOfWshrFrnt</
      SHORT-NAME>
  <PROVIDER-IREF>
    <CONTEXT-COMPONENT-REF DEST="SW-COMPONENT-PROTOTYPE" BASE="
        SwComponentTypes">WiprWshr/WiprWshrMgr</CONTEXT-COMPONENT-REF>
    <TARGET-P-PORT-REF DEST="P-PORT-PROTOTYPE" BASE="SwComponentTypes">
        WiprWshrMgr/ActvnOfWshngCmdOfWshrFrnt</TARGET-P-PORT-REF>
  </PROVIDER-IREF>
  <REQUESTER-IREF>
    <CONTEXT-COMPONENT-REF DEST="SW-COMPONENT-PROTOTYPE" BASE="
        SwComponentTypes">WiprWshr/WshrFrnt</CONTEXT-COMPONENT-REF>
    <TARGET-R-PORT-REF DEST="R-PORT-PROTOTYPE" BASE="SwComponentTypes">Wshr
        /ActvnOfWshngCmd</TARGET-R-PORT-REF>
  </REQUESTER-IREF>
</ASSEMBLY-SW-CONNECTOR>
```

As can be seen from the snippet, both ports are referenced via instance references.

The generator creates the name of the assembly connector according to the rule below;

`<P-Port-ShortName>Of<P-Port-SW-C-Prototype-ShortName>To<R-Port-ShortName>Of<R-Port-SW-C-Prototype-ShortName>`

In the context of the above example <P-Port-ShortName> is ActvnOfWshngCmdOfWshrFrnt, <P-Port-SW-C-Prototype-ShortName>, is WiprWshrMgr, <R-Port-ShortName> is ActvnOfWshngCmd, and <R-Port_SW-C-Prototype-ShortName> is WshrFrnt. Therefore, name of the assembly connector is ActvnOfWshngCmdOfWshrFrntOfWiprWshrMgrToActvnOfWshngCmdOfWshrFrnt.

6.2.6 PortInterfaces

The PortInterface defines the kind of information transported between two PortPrototypes.

PortInterfaces are used to support a design-by-contract workflow, i.e. they provide a means to formally verify structural and dynamic compatibility between software-components. In other words, PortInterfaces represent a pivotal point in the AUTOSAR concept.

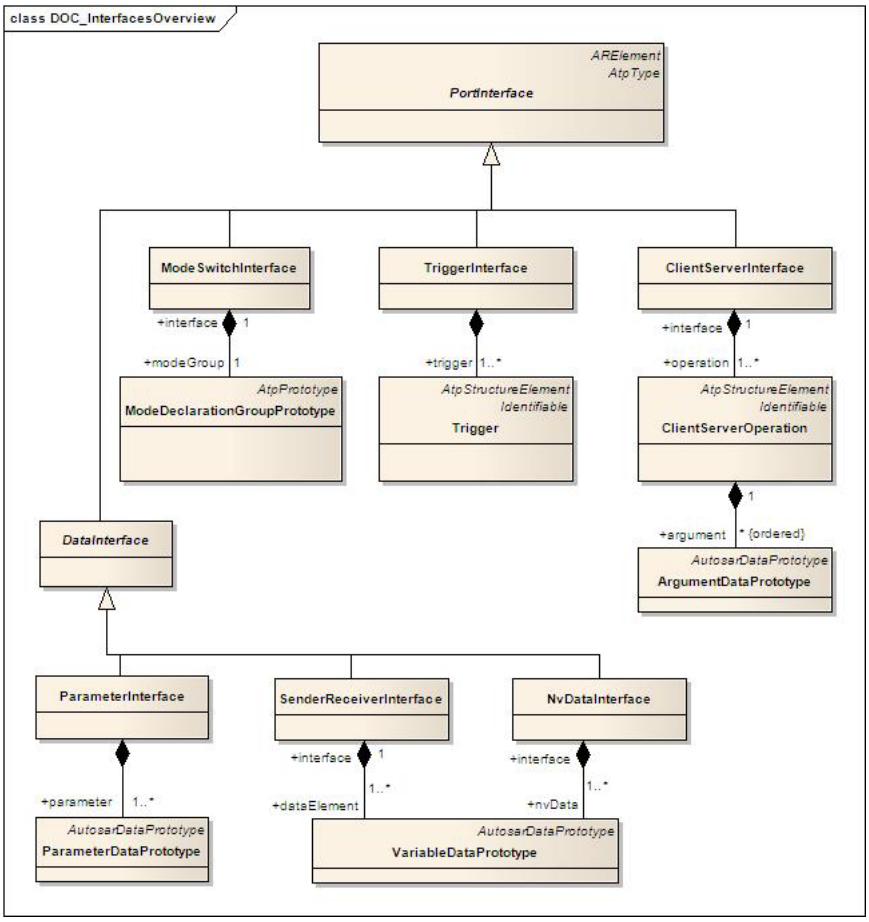


Figure 6.14: Interfaces Overview

Meta Model Reference:
M2::AUTOSARTemplates::SWComponentTemplate::PortInterface [4]

AI-Tool Reference:
AI-Tool Project Explorer: ARPackage: PortInterfaces_Blueprint
ARPackage: PortInterfaces_Example
AI-Tool Tabular View: Sender-Receiver Interface Blueprints
Sender-Receiver Interface Examples
Client-Server Interface Blueprints
Client-Server Interface Examples

AUTOSAR_CP_MOD_AISpecification Reference:
AUTOSAR_CP_MOD_AISpecification_PortInterface_Blueprint
AUTOSAR_CP_MOD_AISpecificationExamples

6.2.6.1 Sender Receiver Communication

SenderReceiverInterfaces allow for the specification of the typically asynchronous communication pattern where a sender provides data that is required by one or more receivers. While the actual communication takes place via the respective PortPrototypes, a SenderReceiverInterface allows for formally describing what kind of information is sent and received.

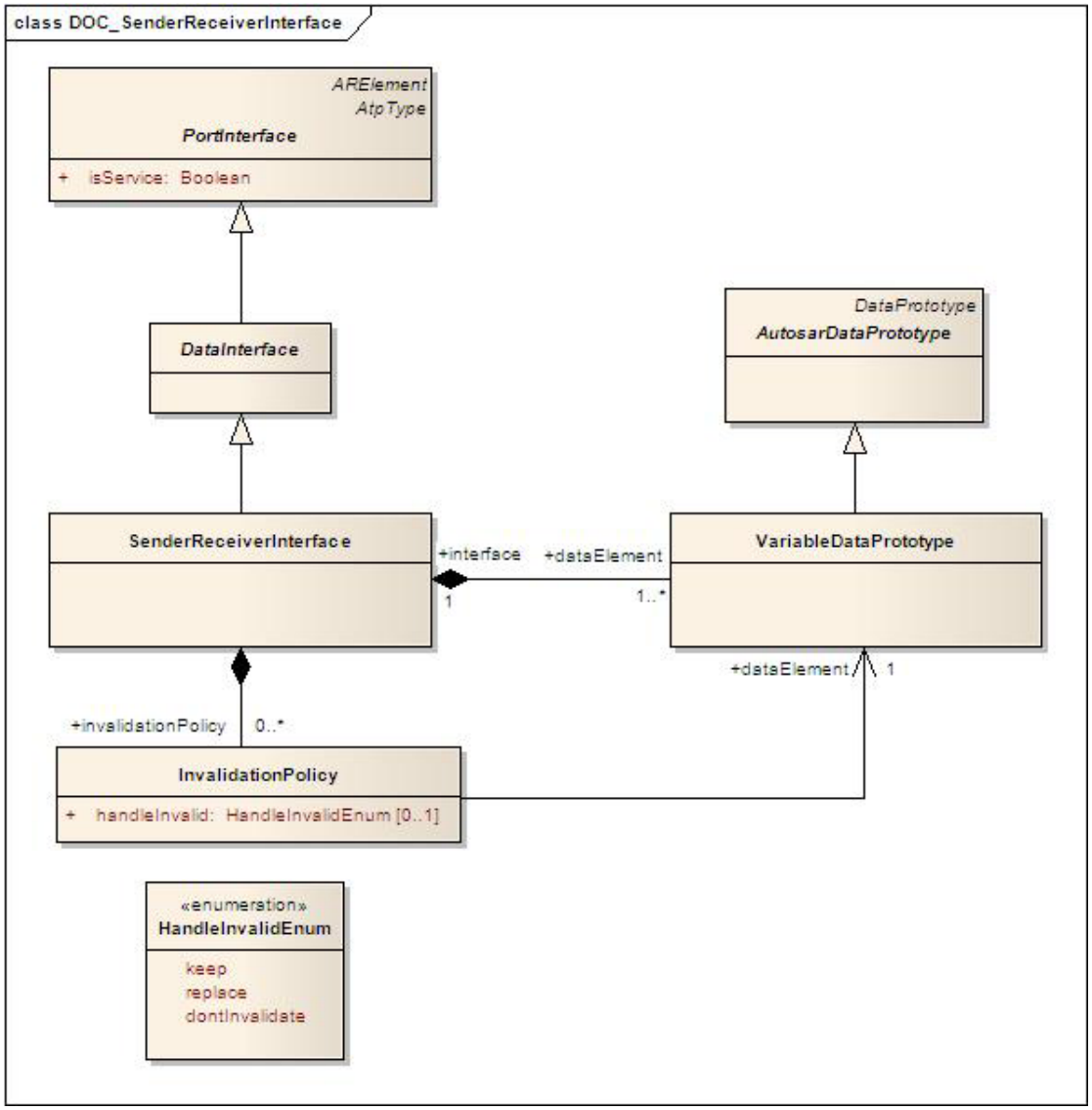


Figure 6.15: SenderReceiverInterface

A SenderReceiverInterface declares a number of data elements (VariableDataPrototype) to be sent and received. A SenderReceiverInterface focuses on the description of information items represented by VariableDataPrototypes. A VariableDataPrototype aggregated in the role of dataElement represents an atomic piece of information transmitted among PortPrototypes typed by a SenderReceiverInterface.

AI-Tool Reference:
AI-Tool Project Explorer: PortInterfaces_Blueprint
PortInterfaces_Example
AI-Tool Tabular View: Sender-Receiver Interface Blueprints
Sender-Receiver Interface Examples

Sheet: Sender-Receiver Interface Blueprints

	SHORTNAME	LONGNAME	DESCRIPTION	DATA ELEMENT...	DATA TYP...	IS SE...	MILESTONE
1	AArbnRes1	Acceleration Arbitration Result 1	Feedback from Vehicle Longitudinal Control to driver assistant sy...	[AArbnRes]	[A1]	false	MS4 - Approval Readines
2	ACenForOccptPedSfty1	Acceleration Central for Occupant Pedestrian Safety	High-g acceleration, measured in center by different acceleration...	[A]	[A8]	false	MS4 - Approval Readines
3	ACenForOccptPedSfty2	Acceleration Central for Occupant Pedestrian Safety	High-g acceleration, measured in center by different acceleration...	[A]	[A13]	false	MS4 - Approval Readines
4	AExtForOccptPedSfty1	Acceleration External for Occupant Pedestrian Safety	High-g acceleration, measured by different acceleration measure...	[A]	[A9]	false	MS4 - Approval Readines
5	AExtForOccptPedSfty2	Acceleration External for Occupant Pedestrian Safety	High-g acceleration, measured by different acceleration measure...	[A]	[A14]	false	MS4 - Approval Readines
6	AFbForCmft1	Acceleration Feedback for Comfort	Cluster of information regarding acceleration and acceleration sa...	[AFbForCmft]	[AFbForCmf...	false	MS4 - Approval Readines
7	ALat1	Acceleration Lateral 1	Generic interface for Lateral acceleration	[ALat]	[A4]	false	MS4 - Approval Readines

Figure 6.16: Sender-Receiver Interfaces in Tabular View

Example: "TrlrSts1" is a SenderReceiver interface that has one data element "TrlrSts" of Type "Boolean".

6.2.6.2 ClientServer Communication

The underlying semantics of a client/server communication is that a client may initiate the execution of an operation by a server that supports the operation. The server executes the operation and immediately provides the client with the result (synchronous operation call) or else the client checks for the completion of the operation by itself (asynchronous operation call).

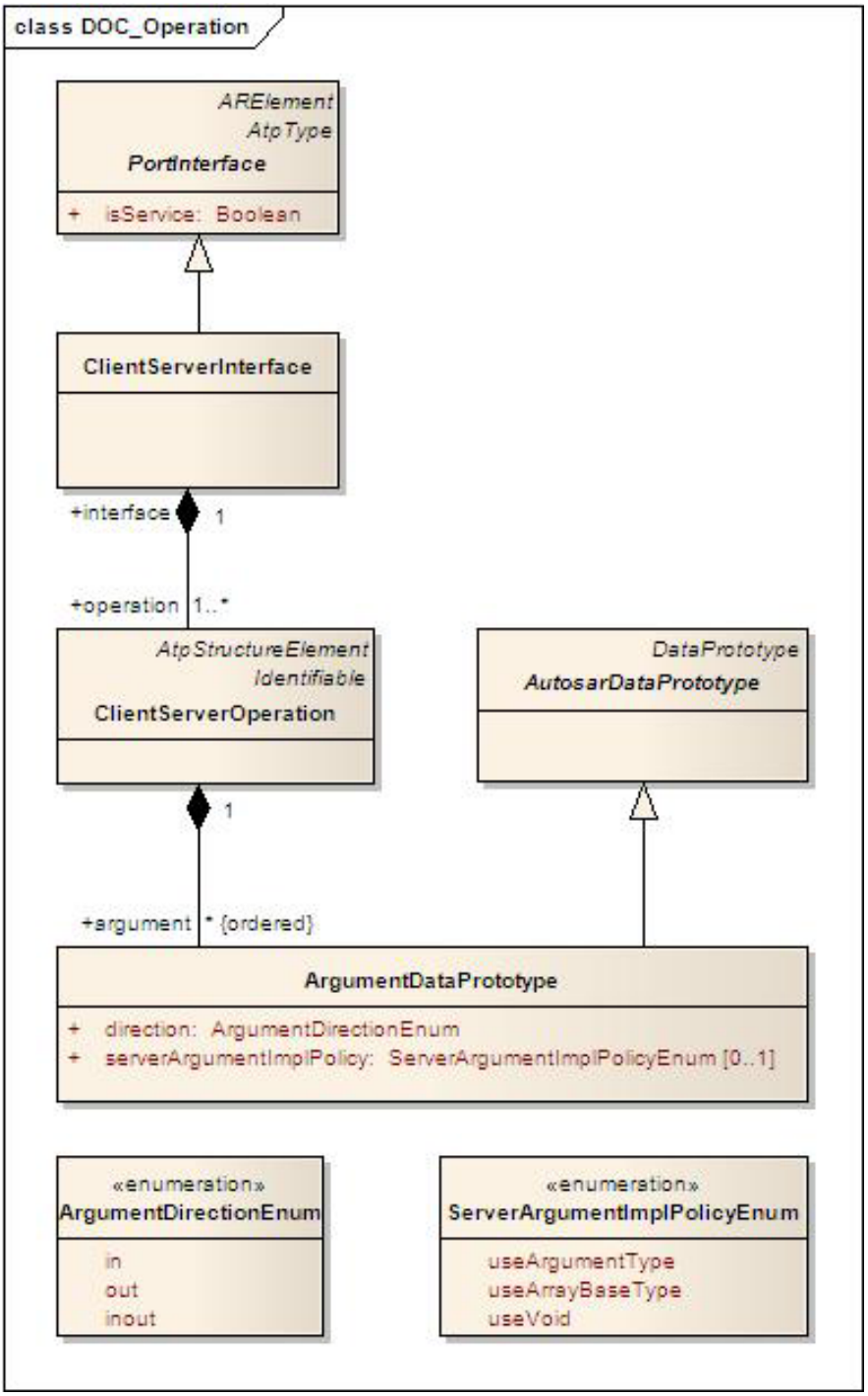


Figure 6.17: ClientServerInterface

ClientServerInterface, therefore to some extent is a counterpart to the SenderReceiverInterface. Instead of defining pieces of information to be transferred among softwarecomponents, a ClientServerInterface defines a collection of ClientServerOperations.

As depicted in Figure 6.17, a ClientServerInterface is composed of ClientServerOperations, i.e. a ClientServerOperation cannot be reused in the context of a different ClientServerInterface. A ClientServerOperation consists of 0 to many ArgumentDataPrototypes. The latter may be

- Passed to the operation
- Passed to and returned from the operation
- Returned from the operation.

AI-Tool Reference:
AI-Tool Project Explorer: PortInterfaces_Blueprint
PortInterfaces_Example
AI-Tool Tabular View: Client-Server Interface Blueprints
Client-Server Interface Examples

6.2.7 DataTypes

It is possible to describe data provided by a software component from the application as well as implementation point of view. The common concept behind this is expressed by the abstract meta-class AutosarDataType, from which an ApplicationDataType and an ImplementationDataType are derived. Figure 6.18 shows a summary of the basic meta-classes used for the definition of AutosarDataTypes.

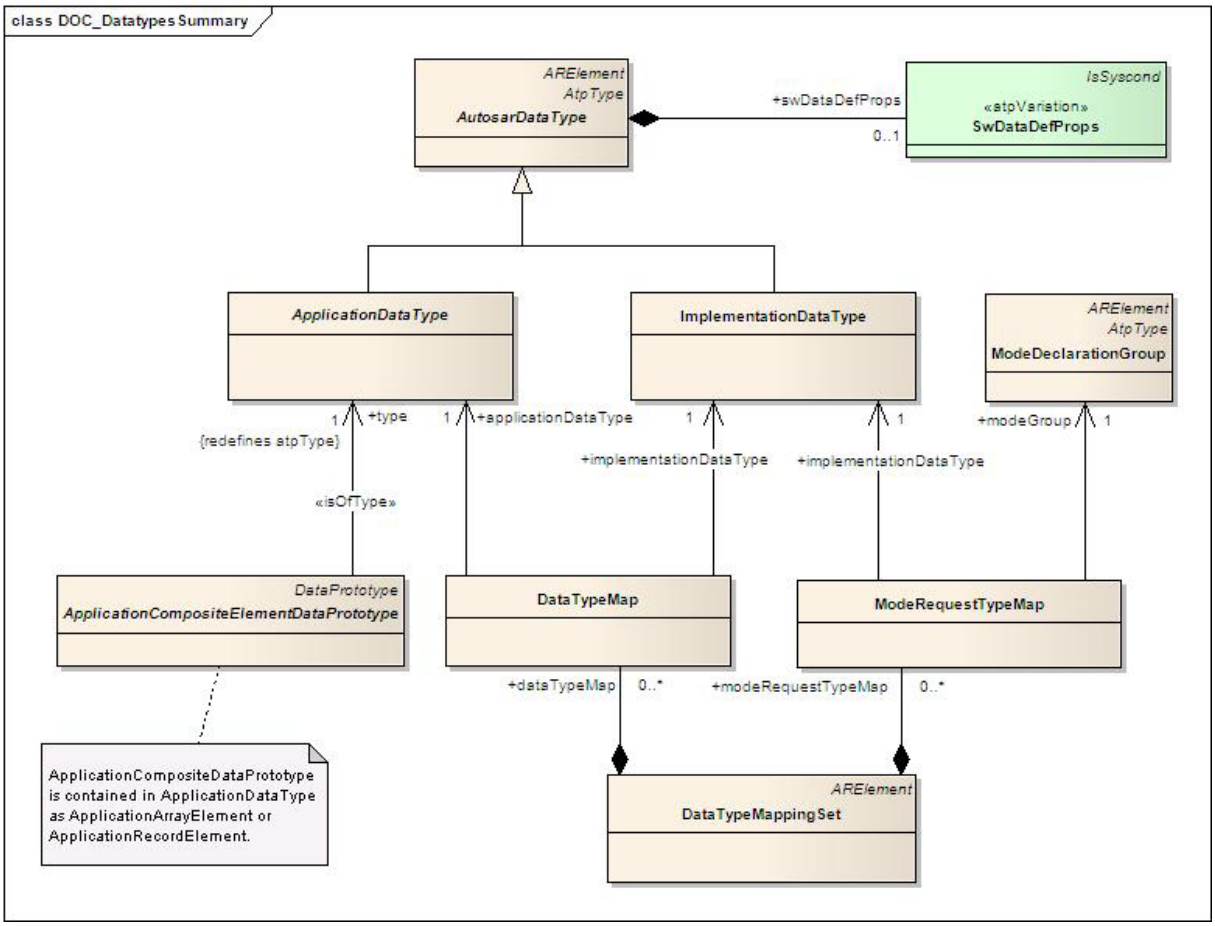


Figure 6.18: DataTypes Overview

An ApplicationDataType can be composed (in form of a record or an array) of elements, which themselves are typed by another ApplicationDataType. This is expressed by the meta-class ApplicationCompositeElementDataPrototype, which is shown in Figure 6.18 for completeness. An ImplementationDataType can be composed too, but in this case, no type/prototype concept has been applied.

6.2.7.1 ApplicationDataTypes

The abstract meta-class ApplicationDataType is further derived into an ApplicationPrimitiveDataType and an ApplicationCompositeDataType. Like any AutosarDataType, a primitive and composite type on application level is characterized by its category and its SwDataDefProps. For a given category, only a limited set of attributes of the SwDataDefProps make sense.

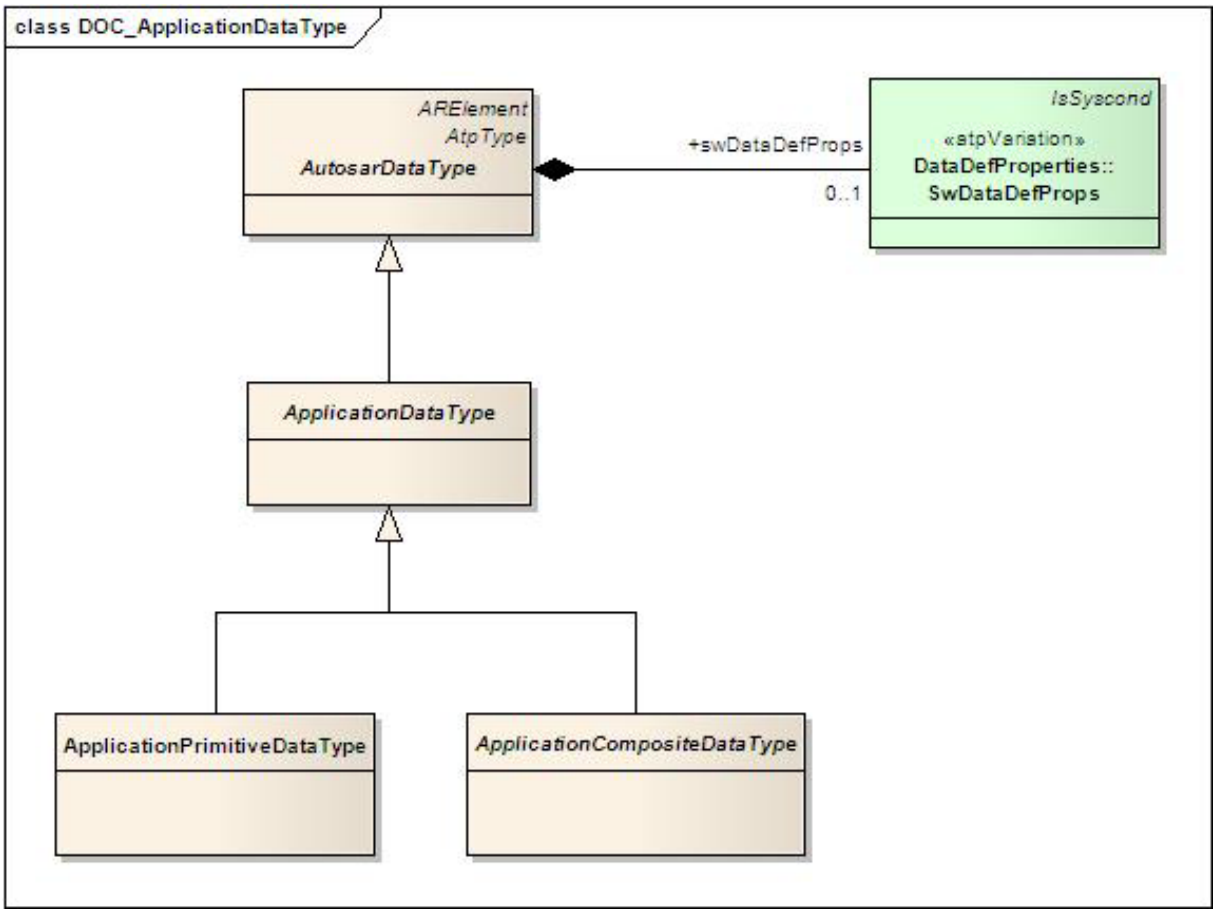


Figure 6.19: Application Data Type

6.2.7.2 Application Primitive Data Types - Continuous Data types

This chapter defines the primitive application data types that may be used for the data prototypes of PortInterfaces or composite application data types. They are usually ap-

plied to represent numerical values of variables describing physical quantities. The different data type types are typified via the CompuMethods and Units they are referencing.

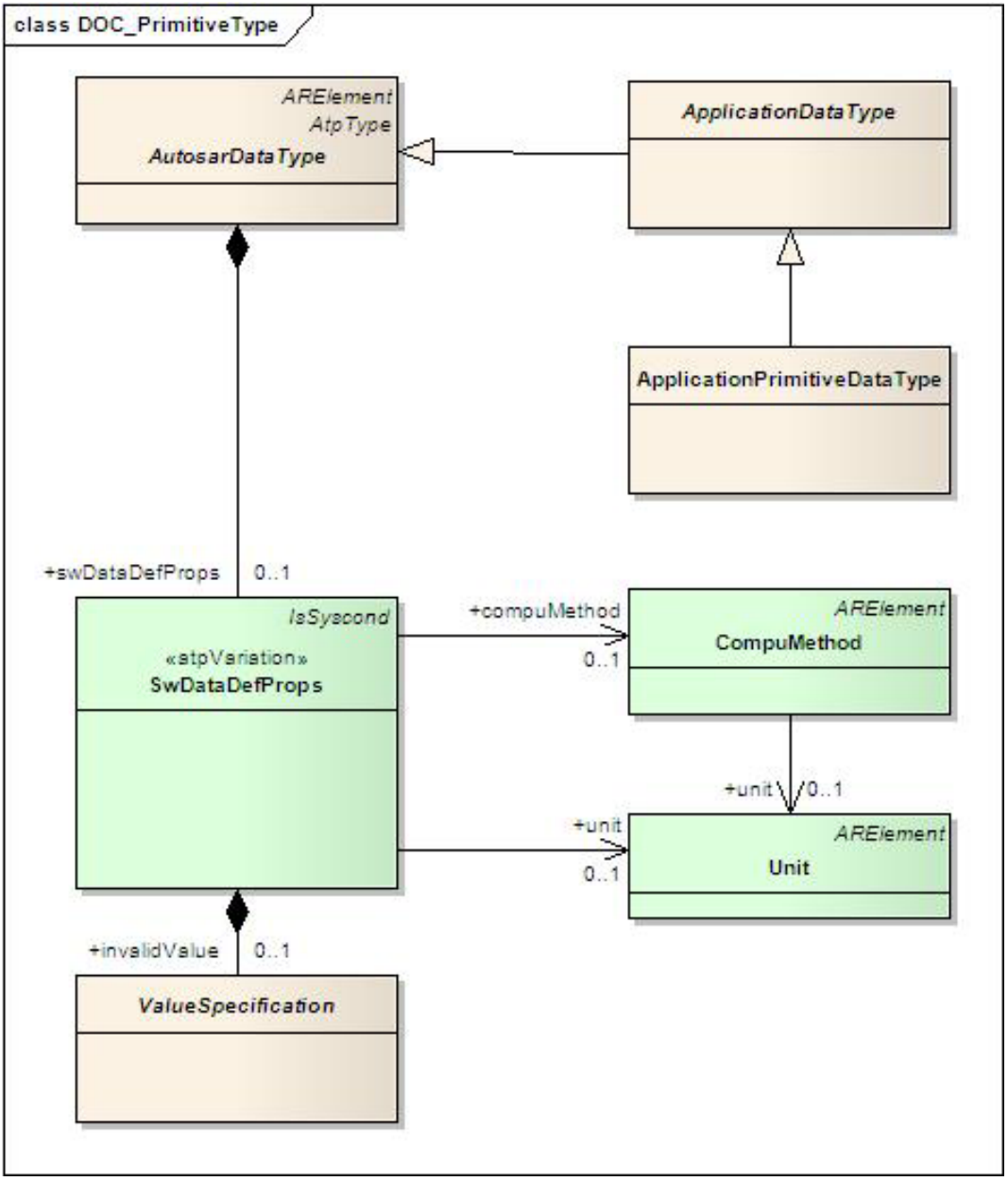


Figure 6.20: Primitive Data Types

Meta Model Reference:
M2::AUTOSARTemplates::SWComponentTemplate::DataType::DataTypes [4]

AI-Tool Reference:
AI-Tool Project Explorer: ApplicationDataTypes_Blueprint
ApplicationDataTypes_Example

AI-Tool Reference:
AI-Tool Project Explorer: ApplicationDataTypes_Blueprint
ApplicationDataTypes_Example
AI-Tool Tabular View: Application Continuous Data Type Blueprints
Application Continuous Data Type Examples

6.2.7.3 Application Primitive Data Types - Enumeration

This chapter defines the primitive application data types that may be used for the data prototypes of PortInterfaces or composite application data types. This type of data types can be used to describe the states of a state machine or boolean information.

CompuMethod (TextTable)

AUTOSAR - AISpecification - CompuMethods_Blueprint - TrsmPosnGearLvr1

Identifiable Attributes

Short Name: TrsmPosnGearLvr1
Long Name: Transmission Position Gear Lever

Literals

Value	Name	Description
0	Neut	0 = Neut (neutral position)
1	Prkg	1 = Prkg (parking position)
2	Rvs	2 = Rvs (reverse gear position)
3	Drv	3 = Drv (driving position)
4	Spt	4 = Spt (sportive position)
5	Drv1	5 = Drv1 (driving position gear 1)
6	Drv2	6 = Drv2 (driving position gear 2)
7	Drv3	7 = Drv3 (driving position gear 3)

CompuMethod (TextTable) | Meta Data

Figure 6.21: Example of a Computation Method for an Enumeration DataType

Example: In the above screen shot, the Computation Method TrsmPosnGearLvr1 of an Enumeration DataType TrsmPosnGearLvr is defined. All permissible values of the Enumeration DataType are defined, here.

AI-Tool Reference:
AI-Tool Project Explorer: ApplicationDataTypes_Blueprint
ApplicationDataTypes_Example
AI-Tool Tabular View: Application Enum Data Type Blueprints
Application Enum Data Type Examples

6.2.7.4 Application Composite Data Types

The meta-classes ApplicationArrayDataType and ApplicationRecordDataType provide the means to define composite DataTypes. Such a composite DataType is required, if the application software wants to have access to the individual elements of the composite as well as to do operations with the whole composite, e.g. wants to communicate the complete record or array in a single transaction.

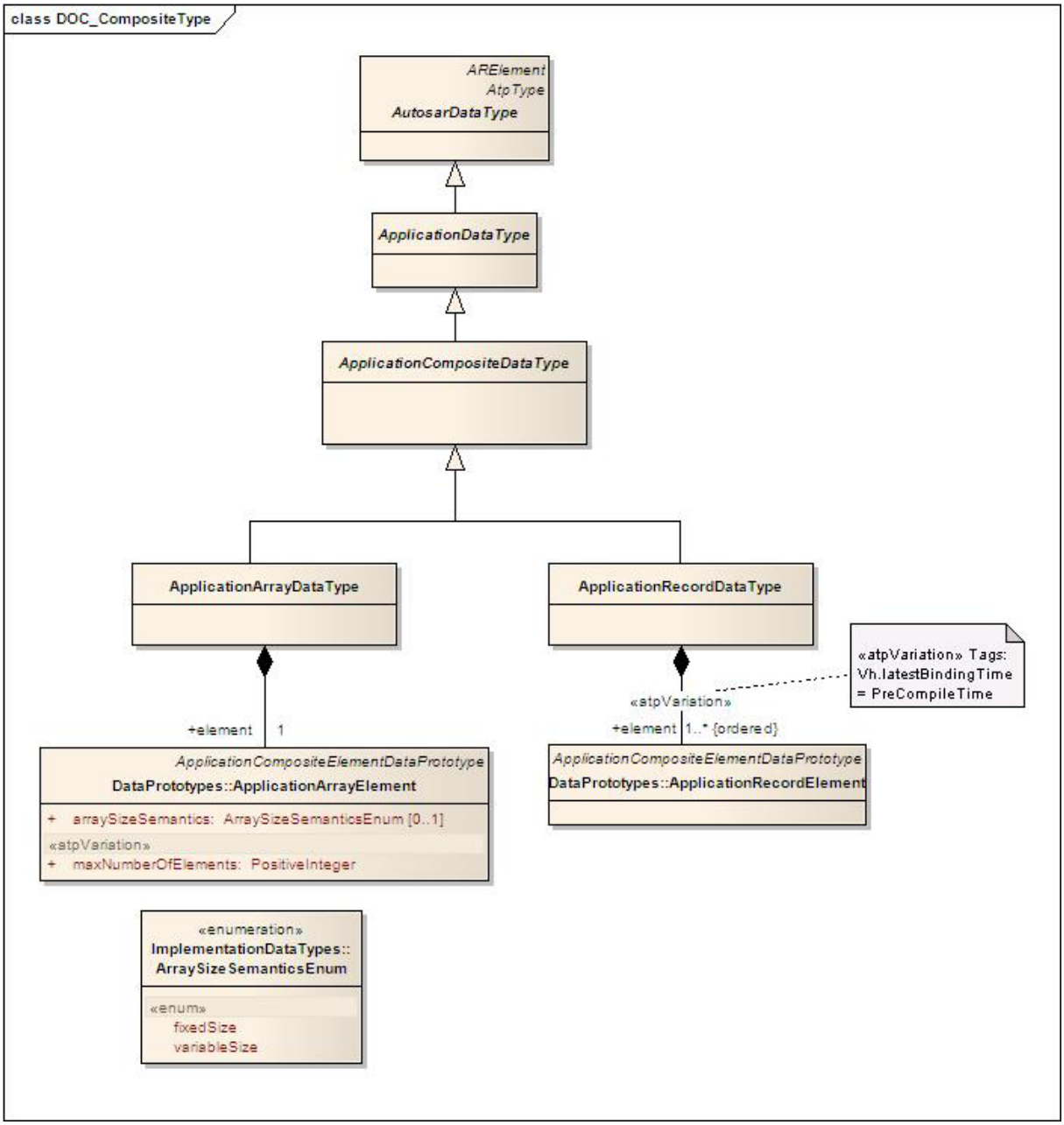


Figure 6.22: Data Types Composite

It is possible to use a combination of ApplicationArrayDataType and ApplicationRecordDataType, so that an ApplicationArrayDataType could be defined as ApplicationRecordElement of an ApplicationRecordDataType and in the same manner, an ApplicationRecordDataType could be used as the base type of an ApplicationArrayDataType. The creation of nested ApplicationComposite DataTypes is also possible.

6.2.7.4.1 ApplicationArrayDataType

An ApplicationArrayDataType may contain maxNumberOfElements of ApplicationArrayElements. Each of these ApplicationArrayElements must have the same type.

When referring to an element of an array within the software-component descriptions, the element index runs from 0 to (maxNumberOfElements-1).

AI-Tool Reference:
AI-Tool Project Explorer: ApplicationDataTypes_Blueprint
ApplicationDataTypes_Example
AI-Tool Tabular View: Application Array Data Type Blueprints
Application Array Data Type Examples

Example: The standard variables of DataType array used in application component development are listed here for reference. Array DataType "TirePPerWhl1" contains 4 elements, each of DataType 'P1'.

6.2.7.4.2 ApplicationRecordDataType

A declaration of ApplicationRecordDataType describes a nonempty set of objects, each of which has a unique identifier with respect to the ApplicationRecordDataType and each has its own ApplicationDataType. The ShortName of each ApplicationRecordElement within the scope of an ApplicationRecordDataType must be unique.

AI-Tool Reference:
AI-Tool Project Explorer: ApplicationDataTypes_Blueprint
ApplicationDataTypes_Example
AI-Tool Tabular View: Application Record Data Type Blueprints
Application Record Data Type Examples

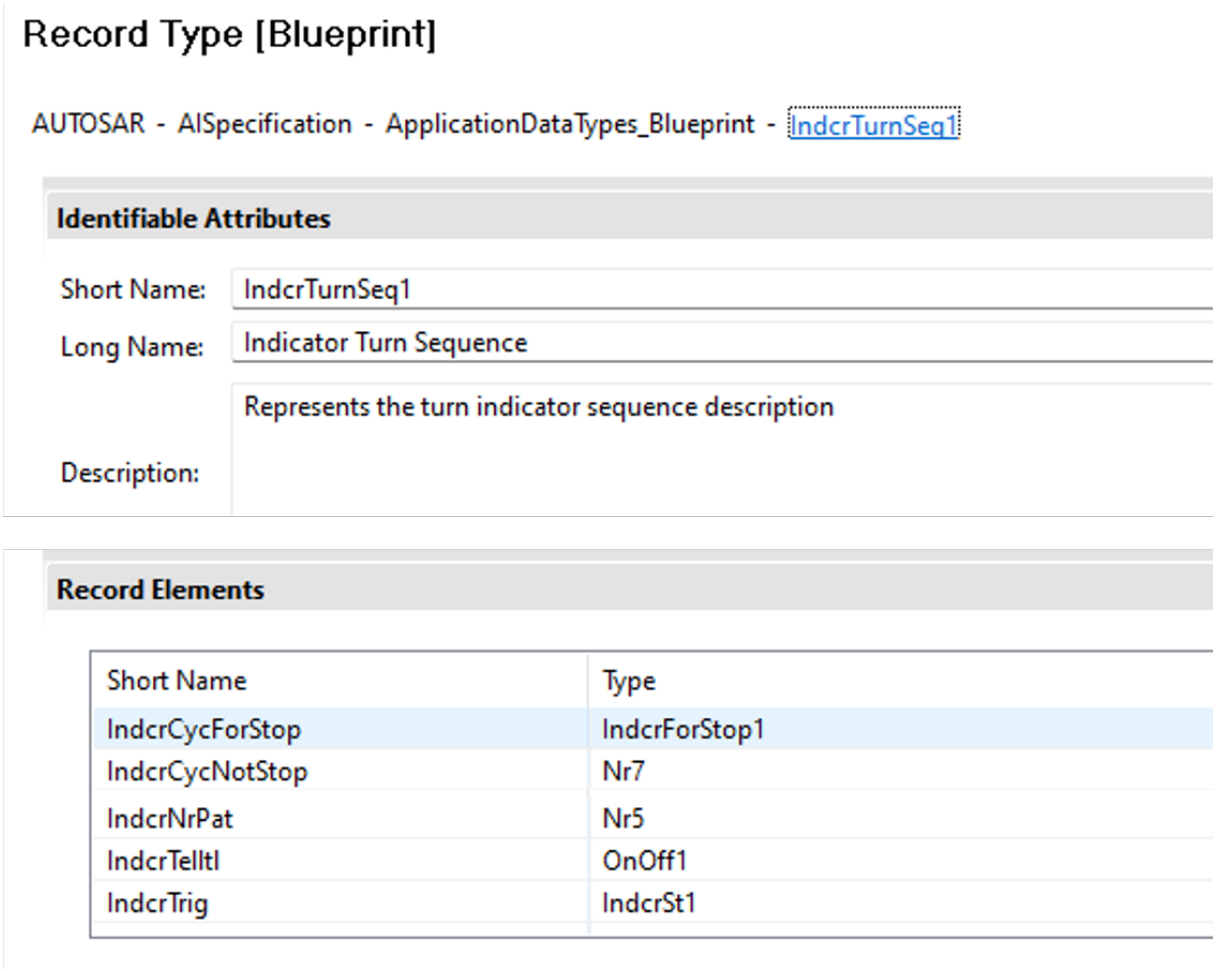


Figure 6.23: Example of a Record DataType Definition

Example: The "IndcrTurnSeq1" (see Figure 6.23) is a record type variable to represent the turn indicator sequence. The record DataType 'IndcrTurnSeq1' contains 5 elements.

6.2.8 Physical Units

An important part of the semantics associated with a DataType is its physical dimension. Units are used to augment the value with additional information like m/s or liter. This is necessary for a correct interpretation of the physical value for input and output processes. The unit involves information about its physical dimensions.

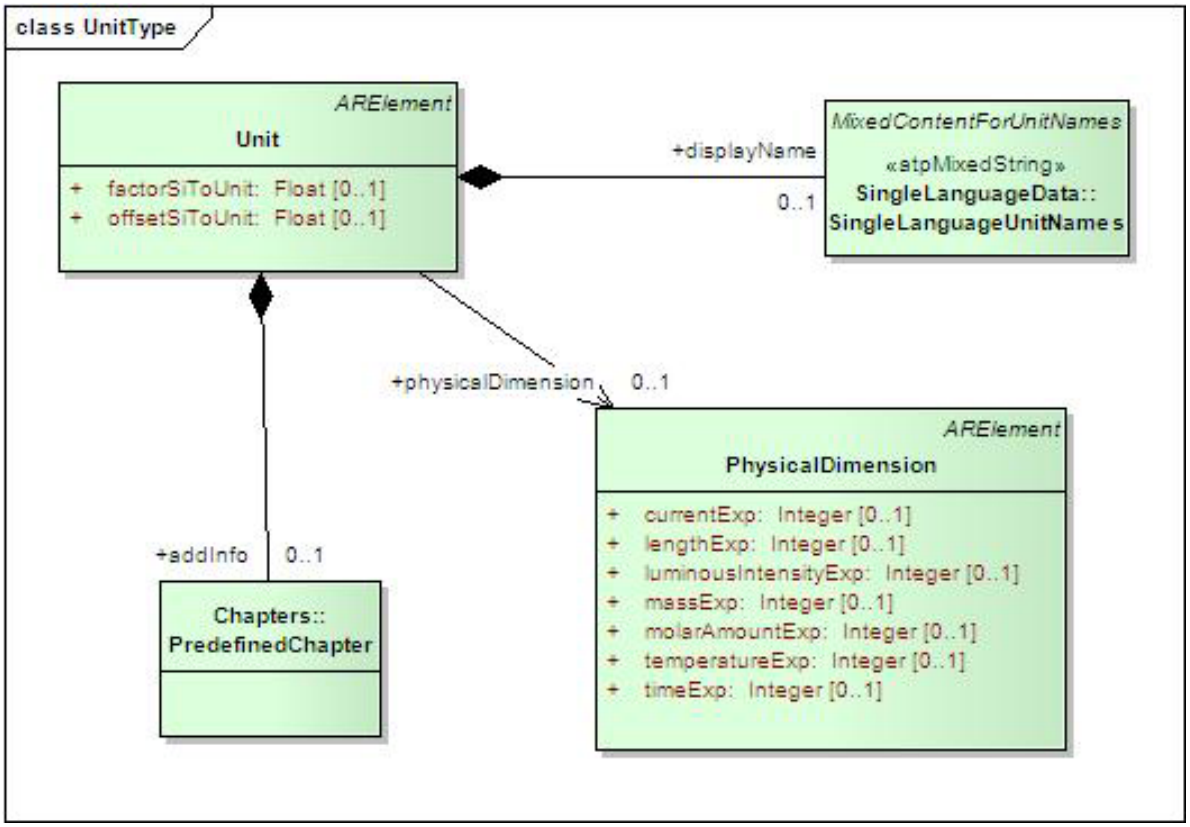


Figure 6.24: Units

The unit references one physical dimension. If the physical dimensions of two units are identical, a conversion between them is possible.

Meta Model Reference:
M2::AUTOSARTemplates::SWComponentTemplate::DataType::Units [4]

AI-Tool Reference:
AI-Tool Project Explorer: PhysicalUnits::PhysicalDimensions_Blueprint
PhysicalUnits::Units_Blueprint
AI-Tool Tabular View: Physical Dimensions
Units

6.2.9 Computation Methods

This meta-class represents the ability to express the relationship between a physical value and the mathematical representation.

Note that this is still independent of the technical implementation in data types. It only specifies the formula how the internal value corresponds to its physical pendant.

CompuMethods in general shall be reuseable and not fixed to a specific data type. CompuMethods can be referenced by any kind of datatype which need such described

computational features; the reuse is strongly recommended and mandatory for float datatypes definition. The AI-Tool supports such cross referencing of parameters. For more information, Refer SW-C and System Modeling Guide [6].

6.2.9.1 Example of Linear Conversion

The following example illustrates how a linear conversion is specified using CompuMethod.

$$F_{[kmh]} = 30_{[kmh]} + 2_{[kmh]} * x$$

```
<COMPU-METHOD>
  <SHORT-NAME>LinearExample</SHORT-NAME>
  <CATEGORY>LINEAR</CATEGORY>
  <UNIT-REF DEST="UNIT">kmh</UNIT-REF>
  <COMPU-INTERNAL-TO-PHYS>
    <COMPU-SCALES>
      <COMPU-SCALE>
        <COMPU-RATIONAL-COEFFS>
          <COMPU-NUMERATOR>
            <V>30</V>
            <V>2</V>
          </COMPU-NUMERATOR>
          <COMPU-DENOMINATOR>
            <V>1</V>
          </COMPU-DENOMINATOR>
        </COMPU-RATIONAL-COEFFS>
      </COMPU-SCALE>
    </COMPU-SCALES>
  </COMPU-INTERNAL-TO-PHYS>
</COMPU-METHOD>
```

6.2.10 Keyword and KeywordSet

An important part of defining short names for component types, ports, PortInterfaces or data elements is to make use of the predefined keywords in AUTOSAR and their abbreviations. The advantage is, that this results in relatively short names with established meaning. The Keywords are aggregated under the category KeywordSets_Blueprint.

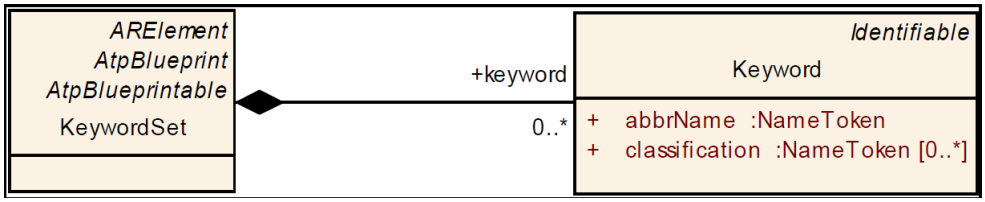


Figure 6.25: Class diagram for Keyword and KeywordSet

Meta Model Reference:
M2::AUTOSARTemplates::StandardizationTemplate::Keyword, KeywordSet [5]

AI-Tool Reference:
AI-Tool Project Explorer: KeywordSets_Blueprint
AI-Tool Tabular View: Table Keywords

AUTOSAR_MOD_AISpecification Reference:
AUTOSAR_MOD_AISpecification_KeywordSet_Blueprint

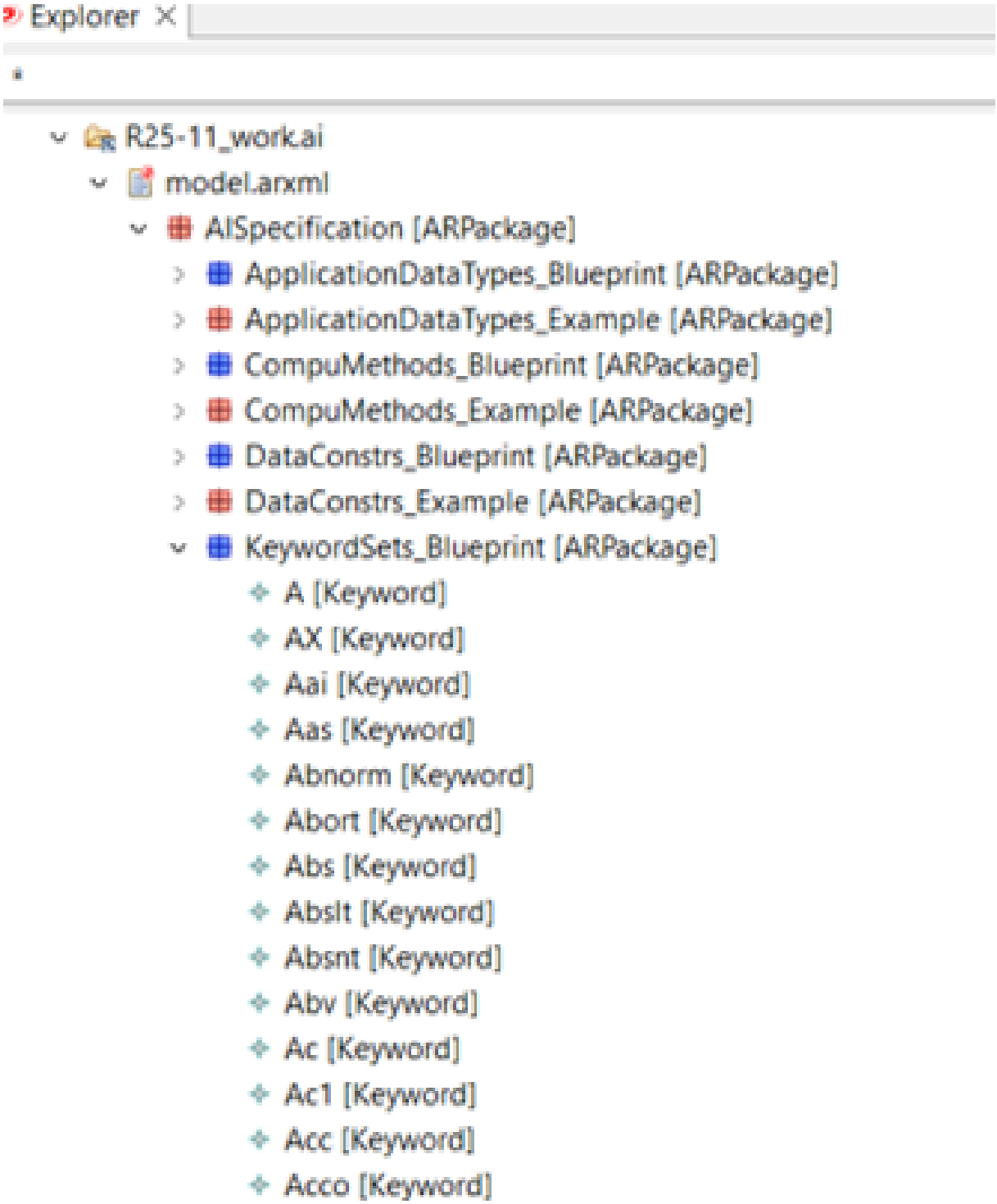


Figure 6.26: KeywordSets in the Explorer View of the AI-Tool

If not differently specified in the rest of the document the term keyword will refer to the longName of the keyword, while the abbreviated name could be referred as "abbrName attribute" or "keyword abbreviation" as well.

The resulting xml outputs is as under:

```
<SHORT-NAME>AISpecification</SHORT-NAME>
```

```
<AR-PACKAGES>
  <AR-PACKAGE>
    <SHORT-NAME>KeywordSets_Blueprint</SHORT-NAME>
    <CATEGORY>BLUEPRINT</CATEGORY>
    <ELEMENTS>
      <KEYWORD-SET>
        <SHORT-NAME>KeywordList</SHORT-NAME>
        <LONG-NAME><L-4 L="EN">AUTOSAR Keywords and Keywords
            Abbreviations</L-4></LONG-NAME>
        <KEYWORDS>
          <KEYWORD>
            <SHORT-NAME>Prepn</SHORT-NAME>
            <LONG-NAME><L-4 L="EN">Preparation</L-4></LONG-NAME>
            <DESC><L-2 L="EN">this characteristic is used to express a
                general status of preparation, e.g. processing of certain
                tasks before an activation of a certain component or
                functionality</L-2></DESC>
            <ABBR-NAME>Prepn</ABBR-NAME>
            <CLASSIFICATIONS>
              <CLASSIFICATION>Condition-Qualifier</CLASSIFICATION>
            </CLASSIFICATIONS>
          </KEYWORD>
...
```

In order to build readable and understandable names, keywords shall be arranged according to semantic rules. Such rules define **Semantic Fields** that must be used in a defined sequence. These rules are described further in [6].

7 Backward Compatibility

7.1 Introduction

In course of the AUTOSAR standard development, it was recognised with newer releases that the standard was not compatible with its predecessor version. This in turn resulted in severe integration issues especially for configurations that intended to use certain modules from the updated version of the specification, see ([5].

Therefore, AUTOSAR introduced the Backward Compatibility requirement for new concepts to ensure smooth and error free operation of software with mixed versions of implementations.

Three use-cases induce three kinds of compatibility statements that will be provided by AUTOSAR- the respective document owners provide list of changes with analysis of impact on the three kinds of Backward Compatibility.

- Specification-wise backwards compatibility
- Bus backwards compatibility
- Application backwards compatibility

For the Application wise backward compatibility, it considers that set of modules of an AUTOSAR release that has an effect on the interaction of application software components.

Hence, the use case definition for Application Interfaces is derived to be:

Horizontal "application-compatibility" (only standardized Application Interfaces are regarded)

- Old scenario: Application development based on standardized Application Interfaces defined in e.g. release R4.0.3
- New scenario: The Applications shall be updated with respect to the standardized Application Interfaces of e.g. release R4.1.1
- Question: Do the new standardized Application interfaces work with the older (unchanged) standardized Application interfaces without adaptations?

7.2 Backward Compatibility Definition

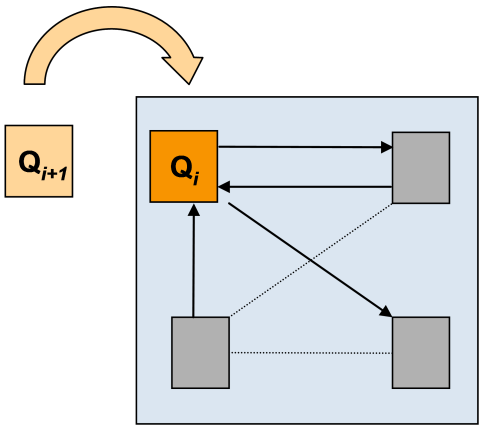


Figure 7.1: Illustrative representation of Backward Compatibility

According to AUTOSAR;

- A product Q_{i+1} is said to be compatible to product Q_i if Q_{i+1} is able to take the place of Q_i that again is interacting with other products (which were left untouched) designed for product Q_i.
- A product Q_{i+1} is said to be backwards compatible to product Q_i if Q_{i+1} is compatible to Q_i and Q_{i+1} is a successor of Q_i.

Therefore, the definition in the context of Application Interfaces is:

- A blueprint Q_{i+1} is said to be compatible to blueprint Q_i if blueprint Q_{i+1} is able to take the place of blueprint Q_i that again is referenced by ports (which were left untouched).
- Application Interface Q_{i+1} is said to be compatible to application interface Q_i if application interface Q_{i+1} is able to take the place of application interface Q_i that again is referenced by a system (which were left untouched) and which was designed according to application interface Q_i.

Example:

PortCompliance is:

- ShortNames are equal
- InterfaceBlueprint of Portblueprint complies to interface of port

InterfaceCompliance is:

- interfaces have the same number of dataElementPrototypes
- names of dataElementPrototypes are the same
- Datatypes of the dataElementPrototypes are compatible

Explanations

The Backward Compatibility requirement is a necessary precondition in order to enable exchange of SWC with new standardized Application Interfaces.

In an ECU developed based upon e.g. release R4.0.3 - the standardized Applications Interfaces shall be updated to the new standardized Application Interfaces of e.g. release R4.1.1

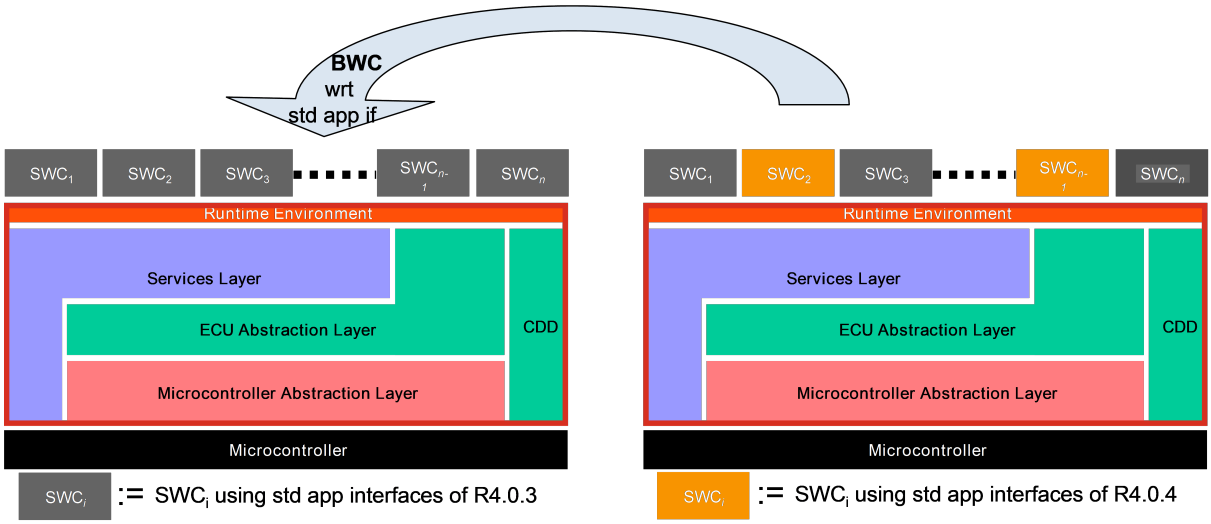


Figure 7.2: Example representation of BWC w.r.t. Application Interfaces

Preconditions:

- The standardized Application Interfaces are updated to latest releases; everything else stays the same
- Configurations are semantically equivalent
- The SWC is recompiled

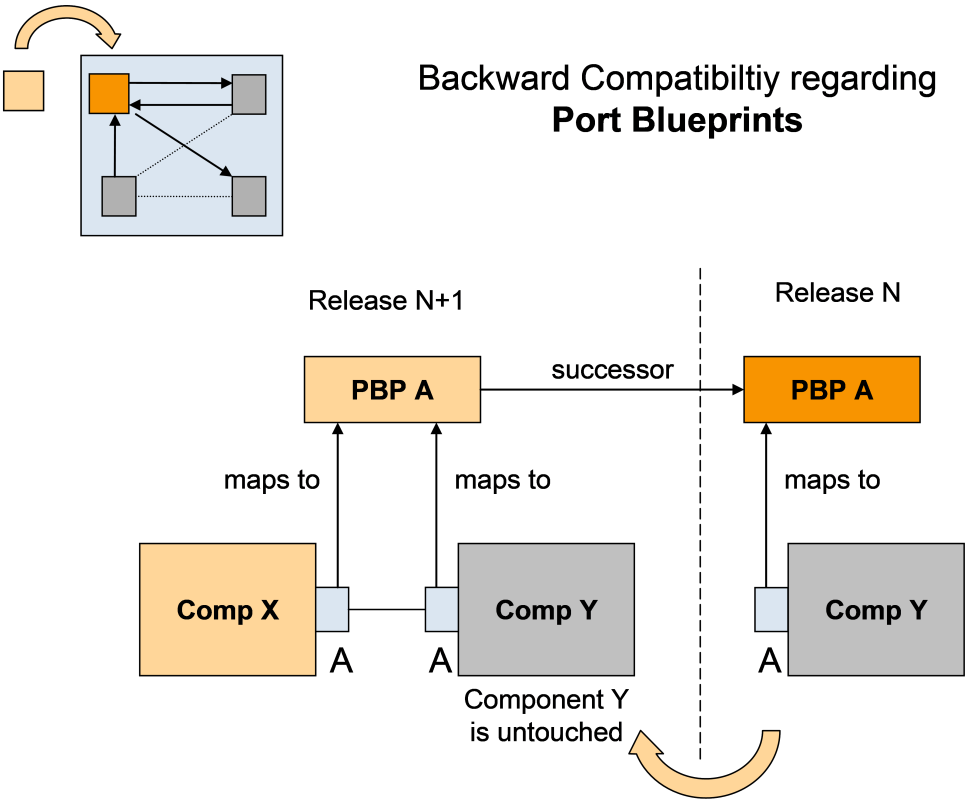


Figure 7.3: BWC with reference to Blueprints

Elements affecting Backward Compatibility; if these elements are changed, backward compatibility is affected (Refer SW-C and System Modeling Guide [6], Future extensions (ch. 5.4))

- Short name
- Enumeration data type
  - Enumeration value, enumeration value name
- Continuous data type
  - Resolution, physical limits, offset, unit
- Array data type
  - Number of elements, type of elements
- Record data type
  - Number of elements, elements name, elements type
- Sender-receiver interface
  - Number of data elements, name of data elements, type of data elements
- Client-server interface

  - Operation name, number of arguments, argument names, argument data types, argument in/out property

7.3 Summary

For any change in a **port blueprint** and its referenced elements (PortInterfaces, ApplicationDatatypes and Units) a new version of all affected elements shall be created. Exception: changes to descriptive elements (unless the meaning of original element is not modified).

New elements use same defined sequence numbering as currently used for interfaces (see SW-C and System Modeling Guide [6], Chapter 5.4). Modification of descriptive elements does not necessarily lead to a new version. This is applied to the following elements:

- Description
- LongName
- Introduction

Note: In case a descriptive element of a blueprint changes in a way that it's meaning with regard to the original blueprint is also changed, a new version of all affected elements shall be created.

8 Life Cycles and Workflow

8.1 Introduction

In order to support and document evolution and backward compatibility of the standardized model elements like PortPrototypeBlueprints, PortInterfaces, KeywordSets and other STANDARD or BLUEPRINT elements, AUTOSAR introduced the information about support life cycle states and milestones. This chapter informs about the representation of Life Cycles and Milestones in the AI-Tool and about the rules for how to use them to document the development of elements defined by Application Interfaces.

8.2 Life-Cycle States

Definition of "Life Cycle": The course of development or evolutionary stages of a model element during its life-time.

A life-cycle consists of a set of life cycle-states. A life-cycle state can be attached to an element in parallel to its version information.[3]

For the collection of Application Interface definitions, we usually do not apply versions because we defined clear rules for the naming of the elements. If an implementation relevant modification or a profound architectural modification would be needed, a new element shall be defined. The decision, and if necessary the replacement, of an element shall be documented in the Meta Data information of the element.

A typical life cycle is, e.g. DRAFT - VALID - OBSOLETE - REMOVED. In AI-Tool, we apply this in the following way:

- State DRAFT: A new element shall be introduced and the development of its formal description starts but the definition is not yet finally agreed
- State VALID: All corrections are discussed and accepted
- State OBSOLETE: The definition of the element is not up-to-date anymore and it could be substituted later by a new one, but for Backward compatibility reasons, it might be kept in the standard
- State REMOVED: The element is not available anymore, no backward compatibility

The first 3 states are seen as backward compatible modifications because the element is still available, even if may not be up-to-date. If the element will be removed (REMOVED), then this will be a modification which is not backward compatible because the element cannot be referenced by other AUTOSAR elements anymore.

Life-Cycle information traces the journey of an element into the standard and also into its obsolescence.

The topic of Life-Cycle states is further described in Chapter 11 of the GenericStructureTemplate [8] and in the AUTOSAR Metamodel [15].

8.2.1 Representation in Metamodel and .arxml

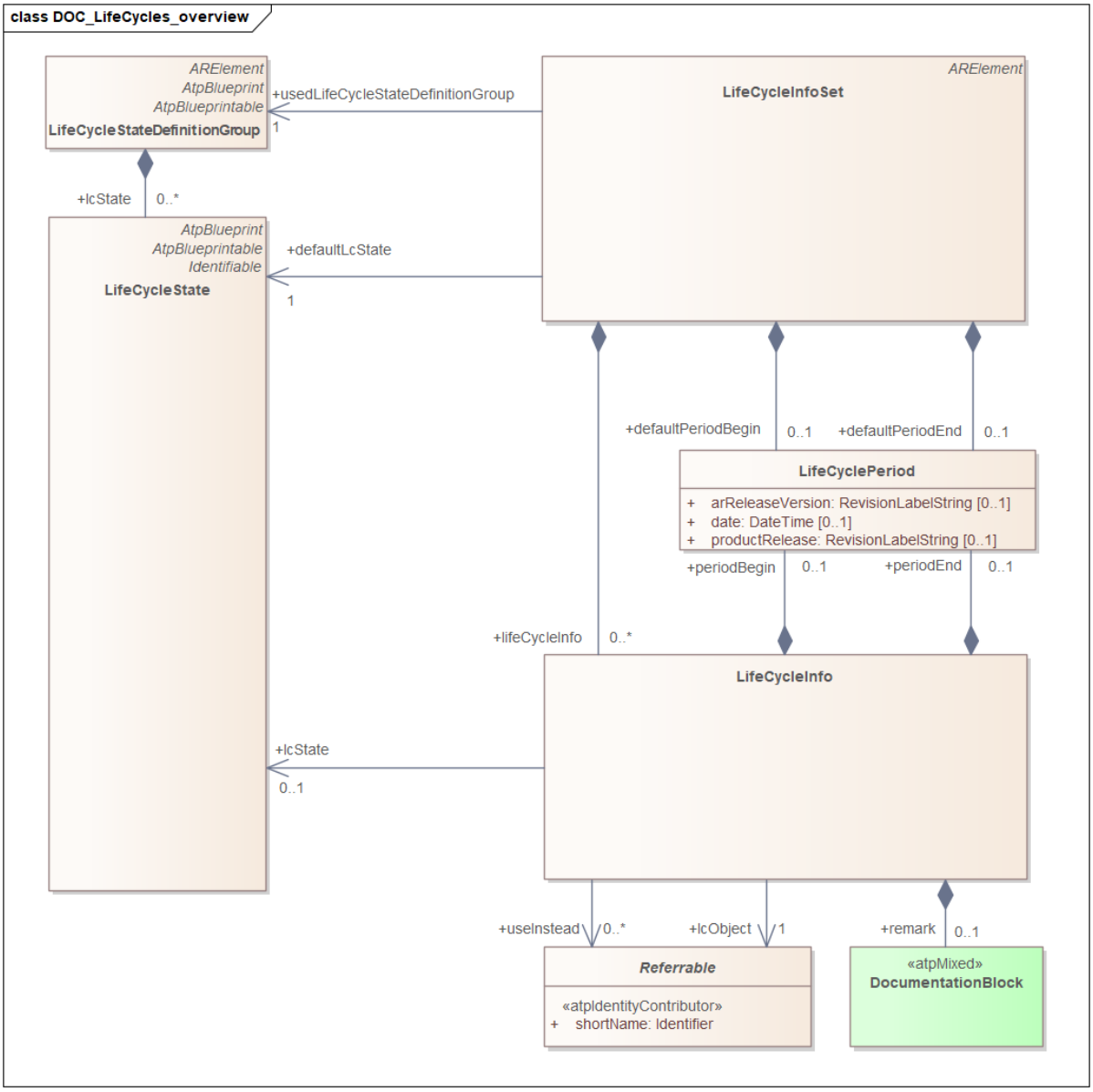


Figure 8.1: Overview of the Definition of Life-Cycles

For each AUTOSAR element of type "Referrable" a LifeCycleInfoSet can be given which references LifeCycleInfo elements for each LifeCycleState modification combined with a LifeCyclePeriod information. This can be used to document the history of an AUTOSAR element.

Example: In the Life-Cycle Information Editor of the AI-Tool (Figure 8.2), it is documented that the Keyword Basd is set to obsolete. Referenced for Use Instead are the existing Keywords Bas or Basc.

The Life Cycle information of elements is available in the .arxml files generated at:

- AUTOSAR_MOD_AISpecification_PhysicalDimension_LifeCycle_Standard.arxml: Application of the life-cycle concept to Physical Dimension definitions which are maintained within the AI-Tool but published in AUTOSAR_MOD_GeneralDefinitions.zip folder. This file is part of the AI deliverables inside the AUTOSAR_MOD_AISpecification.zip folder.
- AUTOSAR_MOD_AISpecification_Unit_LifeCycle_Standard.arxml: Application of the life-cycle concept to Physical Unit definitions which are maintained within the AI-Tool but published in AUTOSAR_MOD_GeneralDefinitions.zip folder. This file is part of the AI deliverables inside the AUTOSAR_MOD_AISpecification.zip folder.
- AUTOSAR_MOD_AISpecification_<Element>_LifeCycle_Standard.arxml: Application of the life-cycle concept to elements defined within the AI-Tool and published in the AUTOSAR_MOD_AISpecification. There is one file per model element and which are part of the AI deliverables inside the AUTOSAR_MOD_AISpecification.zip folder.
- AUTOSAR_MOD_AISpecificationExamples_LifeCycle.arxml: Application of the life-cycle concept to elements defined within the AI-Tool There is one file for all examples which is part of the AI deliverables inside the AUTOSAR_MOD_AISpecificationExamples.zip folder.
- AUTOSAR_MOD_SWDesignPatternExamples_LifeCycle.arxml: Application of the life-cycle concept to the SW Design Pattern Examples maintained within the AI-Tool and published in the AUTOSAR_MOD_AISpecificationExamples. There is one file for all Design Patterns which is part of the AI deliverables inside the AUTOSAR_MOD_AISpecificationExamples.zip folder.

Representation in .arxml:

```
<LIFE-CYCLE-INFO>
  <LC-OBJECT-REF BASE="KeywordSets" DEST="KEYWORD">KeywordList/Sec</LC-
      OBJECT-REF>
  <LC-STATE-REF DEST="LIFE-CYCLE-STATE">
    /AUTOSAR/GenDef/LifeCycleStateDefinitionGroups/AutosarLifeCycleStates/
        obsolete
  </LC-STATE-REF>
  <PERIOD-BEGIN>
    <AR-RELEASE-VERSION>4.2.1</AR-RELEASE-VERSION>
  </PERIOD-BEGIN>
  <USE-INSTEAD-REFS>
    <USE-INSTEAD-REF BASE="KeywordSets" DEST="KEYWORD">KeywordList/Idx2nd</
        USE-INSTEAD-REF>
  </USE-INSTEAD-REFS>
</LIFE-CYCLE-INFO>
```

The AUTOSAR_MOD_AISpecification_<Element>_LifeCycle_Standard.arxml contains only those elements that are marked as Obsolete (since default value is Valid) and the corresponding element to be used instead, i.e. the substitute of the obsolete element. The expiry version defines the begin of the period of the element being obsolete, i.e. the first AUTOSAR release where this element was set from valid to obsolete, in this case "R.4.2.1".

8.2.2 Representation in AI-Tool

In order to support the introduction of LifeCycle states and the Milestones in the AI-Tool, special fields are provided in the Meta Data view of each element editor.

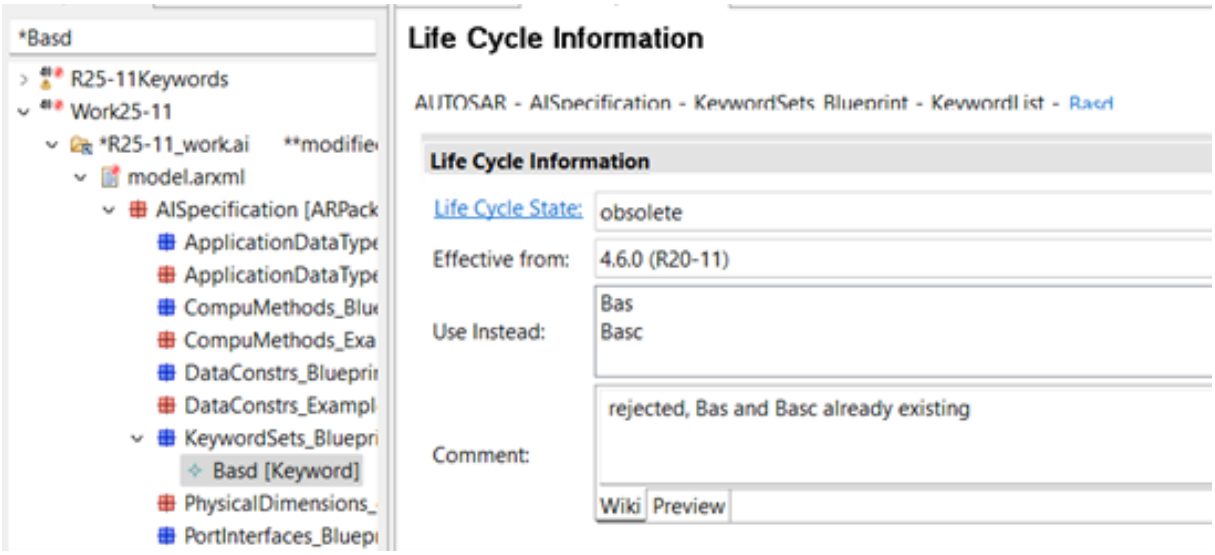


Figure 8.2: LifeCycle representation in the AI-Tool

The description of the fields is indicated below.

Field "Life Cycle State:" Currently there are 3 LifeCycle states defined for usage in AI-Tool:

- draft or empty (default state when introducing a new element)
- valid (to be set after the final review and agreement)
- obsolete (with a reasoning for its obsolescence and an alternative provided)

Field "Effective from:" Describe here the starting time of the current Life Cycle State. Any modification of the Life Cycle State leads to a new LifeCycleInfo with a new starting time. The selectable values, here, are the names of the official AUTOSAR releases.

Field "Use Instead:" This field can be used to give an recommendation for the replacement of the element in case the element would become obsolete (Figure 8.2).

Field "Comment:" Here, it is possible to give a short explanation of the reasons why the new status was decided (Figure 8.2).

8.3 Milestones

There is another way available to structure and document the process of Application Interfaces definition called Milestones. This principle was originally introduced to structure the internal AUTOSAR project development process. In the Working Group Application Interfaces, we apply the Milestone principles only in the AI-Tool to document the maturity of the definitions we collect there.

In Application Interfaces, we use the Milestone information to control the automatic support functions of the AI-Tool. We interpret Milestones as follows:

Milestone	Function
Milestone 0 MS0	• Element is not accepted • Architecture decision: element is not published into .arxml-files but kept as repository in the AI-Tool • Element cannot be modified • Element not published into .arxml-files
Milestone 1 MS1	• Initial state, represents a request to introduce a new element • Architecture decision: not finished, request for architecture discussion • Element can be modified • Element not published into .arxml-files
Milestone 2 MS2	• Refined draft version, agreed and refined during Architecture discussion • Architecture decision: Proposal for final review • Element can be modified • Element not published into .arxml-files
Milestone 3 MS3	• Reviewed draft version, agreed and refined during Architecture discussion • Architecture decision: Review successful, accepted for upcoming release • Element can be modified • Element not published into .arxml-files
Milestone 4 MS4	• Released version, agreed and refined during Architecture discussion • Architecture decision: Finally reviewed and accepted • Element cannot be modified • Element published into .arxml-files at release

Table 8.1: Interpretation and Control Functions of Milestone Concept in AI-Tool

As described in the table above, the different Milestones are used to control the functionality of the AI-Tool to support the work-flow for the definitions of Application Interface describing elements.

Modification to support and document the architecture decisions is only possible in MS1 - MS3. With these Milestones, we accompany the architecture discussion and the refinement and the review of the element descriptions

In MS0 and MS4 the architecture discussion is considered as finished. The description of the element is considered either as not accepted (MS0) or as finally reviewed and to be released with the next AUTOSAR Release Build (MS4).

To re-initiate the discussion of formerly rejected element definitions (MS0), it will be necessary to reset the Milestone to MS1 to allow modification and documentation of the architecture decisions.

The AI-Tool functionality does not allow any modifications on released elements beside resetting the Milestone of the element back to MS3.

For re-discussing released elements (MS4), which have already been described in a released version of the .arxml-files, we apply a similar approach. Here, we cannot modify the contents directly in the AI-Tool because each modification of the content would be published with next build as builds are started after each SVN-Commit of a modified content (.ai-file). Otherwise, we would modify released .arxml-files. To avoid modification of released .arxml-files, we document the intended modifications in the "Proposed Solution" of the Issue or in the of the Implementation Task. But we doe not apply the modifications in the AI-Tool before the last the corresponding Issue reaches phase "Implementation".

At the end of a development cycle, with each AUTOSAR release, only the elements which are in LifeCycleState "valid" and are marked with Milestone MS4 are used to generate the AUTOSAR_CP_MOD_AISpecification, the AUTOSAR_CP_MOD_AISpecificationExamples and the AUTOSAR_FO_MOD_GeneralBlueprints .arxml collections.

Elements in Milestone 4 cannot be modified anymore. The AI-Tool functionality does not allow any modifications beside resetting the Milestone of the element to MS3.

8.4 Work-Flow and Milestone propagation

8.4.1 Application Interfaces Work-Flow

To accompany and document the development of Application Interface descriptions we introduced a work-flow which allows working on different topics in the same .ai-file which is stored on SVN in the folder:
svn.autosar.org/repos/work/26_Standards/01_Sources/CP_MOD_AISpecification_443/AI-T-Workspace.

An AUTOSAR-WIKI page describes the CM process for modification of AI-Tool contents:
wiki.autosar.org/doku.php?id=tools:documenttooling:toolslandingpage:ait:process/_for/_aitool.

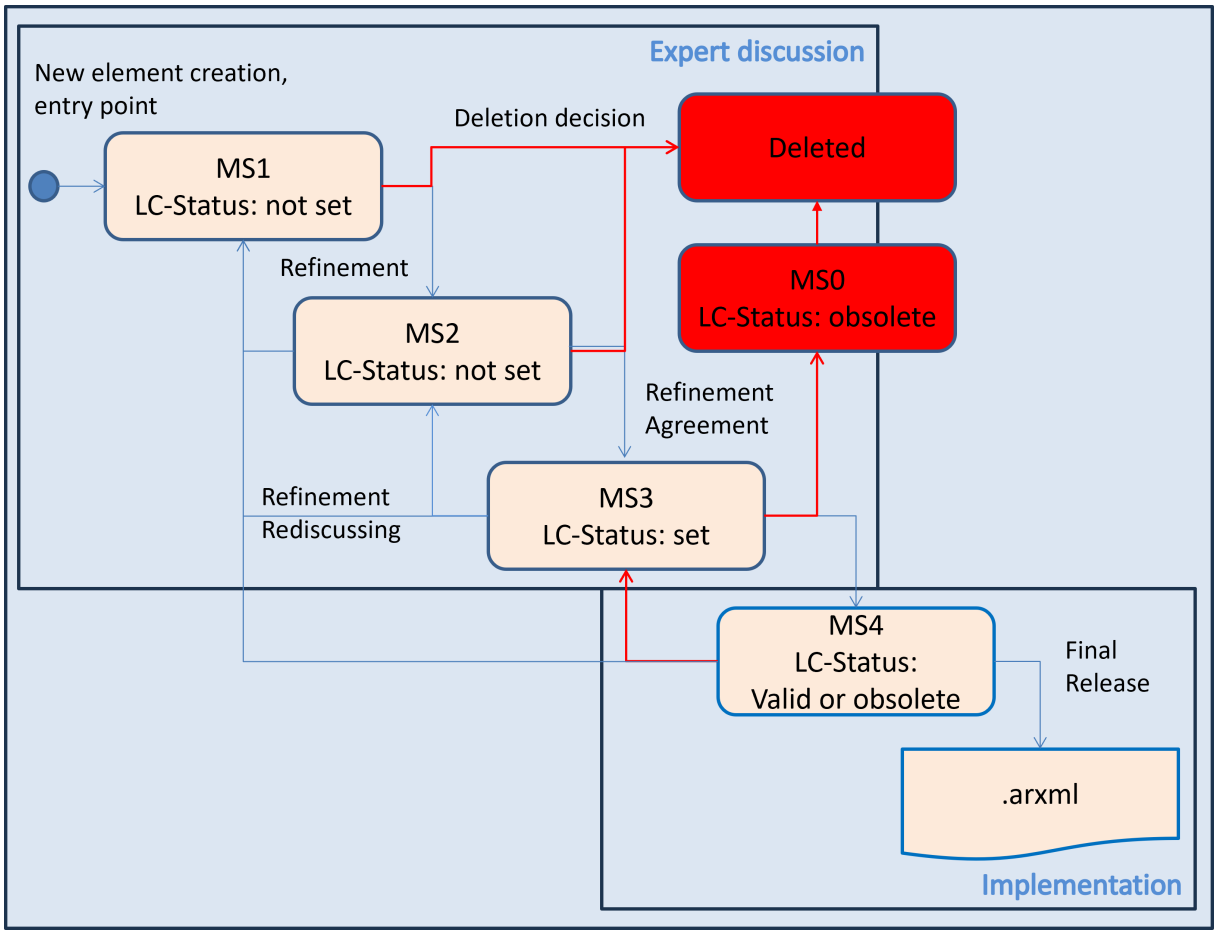


Figure 8.3: Workflow of Application Interfaces definition tracked by Life-Cycles and Milestones

8.4.1.1 Start Working for new Release

At the beginning of a new development cycle, we initiate an new Issue for each new topic. New topics might be, e.g.:

- Introduction of new elements
- Re-discussion/correction of a released element (already in MS4)
- Re-discussion of an element in MS0
- Removing a formerly released element

We agreed to have for each Release only one Issue where we collect the information about introduction, modification or deletion of all new elements of a type, like, e.g. one Issue for all Keyword discussions.

8.4.1.2 Working during Expert Discussion

During Expert Discussion phase, we directly implement some of the modifications in the .ai-file in the Sources folder.

The definitions of elements to be introduced newly can be edited and their Milestones can be increased until MS3. The same applies to elements which have been formerly set to MS0 and which are required to be re-discussed.

In AI-Tool, we only modify elements which are not yet part of a released AUTOSAR_CP_MOD_AISpecification. For released elements in MS4, we only document the needed modifications or the need for deletion, but we do not apply the modification in the AI-Tool before the final Release build.

All decisions taken are documented in the Proposed Solution of the Issue.

For the Commit on SVN, we use the Tag "#editorial" to avoid unintended modifications of the .arxml-files. With this, we keep the Change Management (CM) informed that modifications on the .ai-file have been done, but no modification of .arxml files, so no review of the .arxml-files is necessary.

8.4.1.3 Working after Expert Discussion Finished

When the Issue reaches the phase "Expert Discussion Finished", the work can be continued documenting further modifications in the section "Deviation from Agreed Solution" of the respective Implementation Task.

From now on, we can set all elements which shall be modified or introduced newly to MS4 in the AI-Tool, so that the modifications will be also published into the .arxml-files.

Formerly released elements (MS4) which shall be modified can be reset to MS3, now, to enable the modification of their definition or their deletion in the AI-Tool. Modifications should be documented in the life-cycle comment of the element. Afterwards these elements can be set to MS0 or can be reset to MS4 to finish the implementation.

8.4.2 Milestone Propagation

The development of standardized interface element definitions requires consistent definitions of elements. Some of these definitions, like,e.g. units or data types have a life-cycle which is independent of the life-cycles of elements using (referencing) them, like e.g. interfaces or ports. So the quality or maturity of such referencing elements depends on the quality of the definitions of the referenced elements. Therefore, the maturity (represented by the Milestone) of a referencing element cannot be higher than the Milestone of any of the elements which are referenced.

To ensure the consistency and correctness of the element definitions, the AI-Tool provides support via the means of milestone propagation.

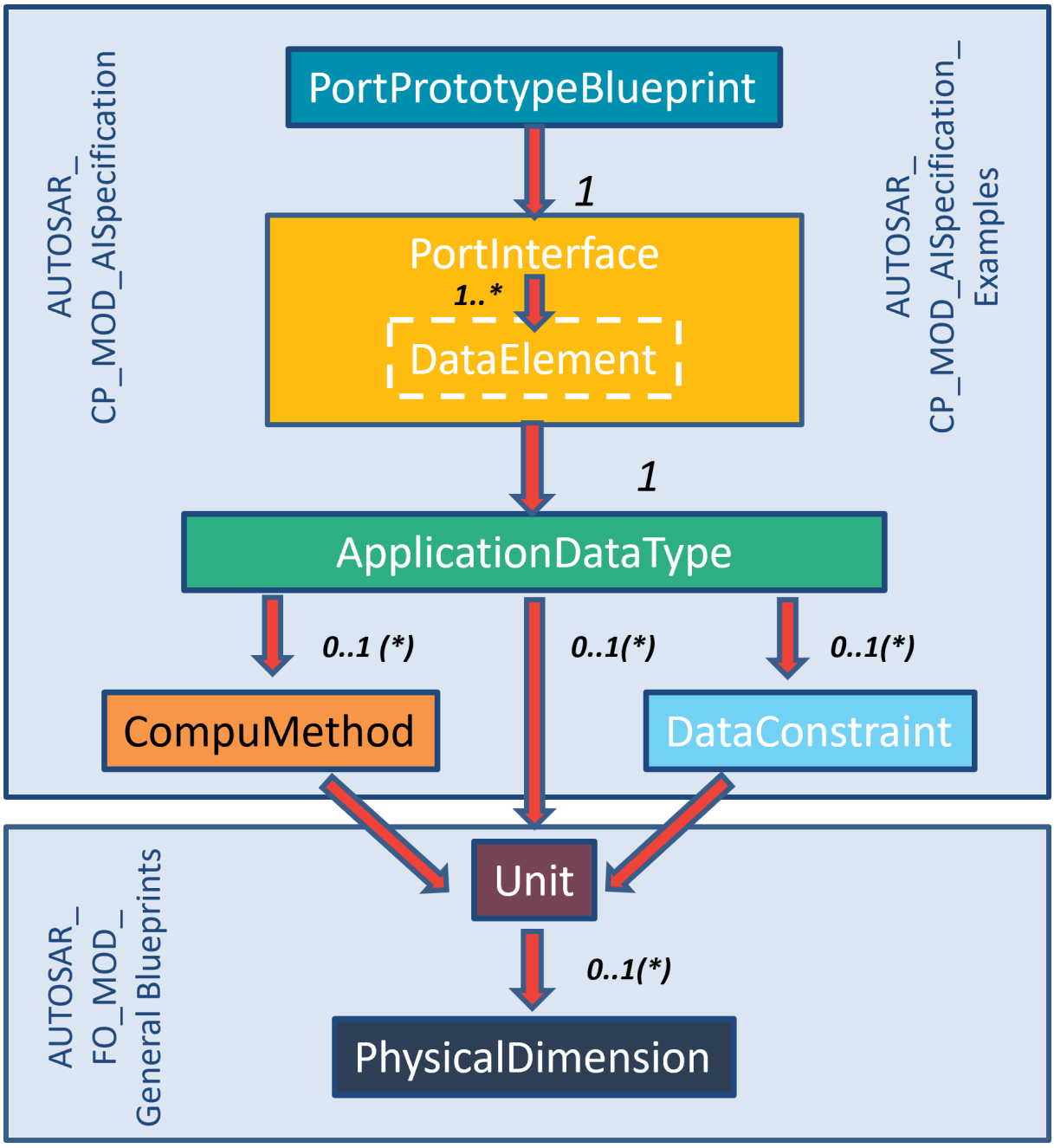


Figure 8.4: Relationships of Element Definitions as reflected in the AI-Tool

The relations described in Figure 8.4 lead to the necessity to restart the review process in case a referenced element would be modified. So if anyone would reset the Milestone of a released element from MS4 to Ms3 - to be able to rework the definition of the element, there will be show up a so called "Milestone Propagation" Window. There the user will be asked if the change of the Milestone of a referenced element shall be propagated to all elements in the reference chain. If so, the milestone of all referencing objects are reset to, e.g. MS3. The Milestone information of these elements needs to be changed manually back to MS4 when all necessary modifications and the reviews of all elements are done.

Example: The user wants to modify/rework the definition of a Physical Dimension "Length" (Len1). To be able to implement that change, the Milestone (MS4) must be reset to MS3. If so, it will be necessary to check, if the definitions of the referencing elements, in this case all Physical Units of type "Length" will still be correct, or if any corrections of these Unit definitions would be necessary. The same is applied to whole chain of referencing elements, like, e.g.:

- all CompuMethods referencing the concerned Units
- all DataConstraints referencing the concerned Units
- all ApplicationData Types referencing concerned Units, CompuMethods, and DataConstraints
- all DataElements referencing the concerned ApplicationDataTypes
- all Interfaces referencing the concerned DataElements
- all Ports referencing the concerned Interfaces
- all SwComponents referencing the concerned Ports
- all All CompositionSwComponents referencing the concerned SwComponents
- this applies to both Blueprint and Example elements

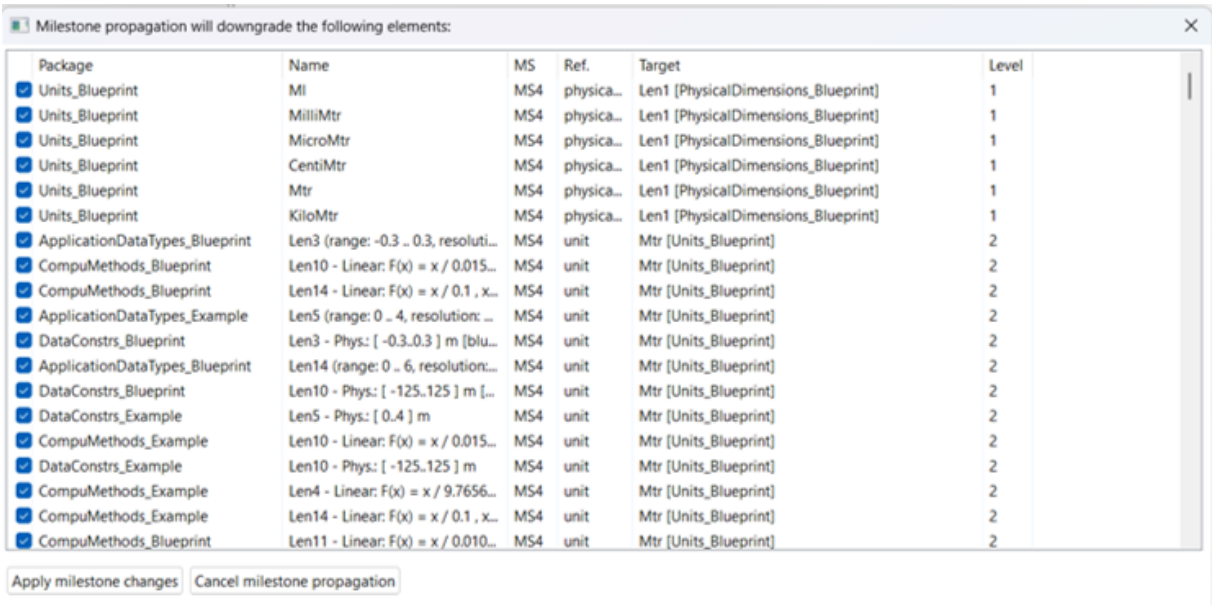


Figure 8.5: Milestone Propagation Window for the Physical Dimension "Length" in AI-Tool

With this approach, it might happen that a high effort will be caused to reset the Milestones of all elements in the reference chain. To avoid unnecessary effort in case these definitions would not be effected, there is also a possibility to skip the automatic Milestone propagation. But with that, the developer takes over responsibility and must carefully check for the consistency of the data definitions of the referncing elements.

9 View Concept in Application Interfaces (Variant Handling)

9.1 Introduction

A variant handling concept will help to implement different architectures in the vehicle using Application Interfaces. With the port prototype blueprint concept the final end-to-end communication of an interface will be performed by the configuration of a system. This will also allow different configurations of physical vehicles such as compact car, premium car, truck to be involved in the standardization of Application Interfaces.

As AUTOSAR only standardizes blueprints, there is no necessity to implement variant handling in the AI Specification. It is easily possible to add new blueprint elements for every variant needed. To differentiate between different variants, different views are introduced. A view allows filtering for blueprint elements for a specific variant or use case. For example, if a view for "trucks" is introduced, it would be possible to filter for the "truck" view, i.e. to filter for all blueprint elements that are relevant for the use case "trucks".

9.2 Implementation in Application Interfaces and Meta Model Representation

The views defined by Application Interfaces Working Group are in a package of Category BLUEPRINT. The reason for this is, that in a company environment, additional elements may be added.

For the first introduction of views, only views for each domain are introduced, i.e. the views "Body" (Body), "Powertrain" (Pt), "Chassis" (Chassis), "Occupant and Pedestrian Safety" (OccptPedSfty) and "Multimedia, Telematics and HMI" (MmedTelmHmi). More views might be added for future releases. Currently, these views can be specified for the following elements:

- ApplicationDataTypes_Blueprint
- CompuMethods_Blueprint
- DataConstrs_Blueprint
- PortInterfaces_Blueprint
- PortPrototypeBlueprints_Blueprint

in the AITool. The elements belonging to a specific view are marked by an activated Checkbox in the Views section of the Meta data editor of the element (Figure 9.1).

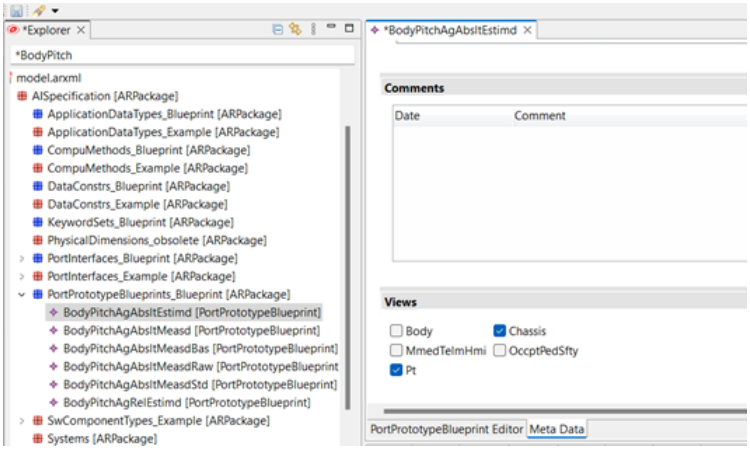


Figure 9.1: Representation of Views in AI-Tool

Additionally, it is also possible that 'multiple views' can be assigned to a single model element. For instance, in above Figure we see that a single model element (BodyPitchAgAbsItEstimd) has two views assigned to it (Chassis and Pt).

For some use cases it is necessary to establish a collection of elements. Such collections are orthogonal to packages. Therefore, a collection resides in a package but is established by associations to the collected elements as seen in the Figure 9.2. For the Application Interfaces however, only a subset of this methodology will be used and further details can be seen in [8].

The different views will be specified in form of collections with the CATEGORY SET and the ELEMENT-ROLE PART_OF_SUBSET.

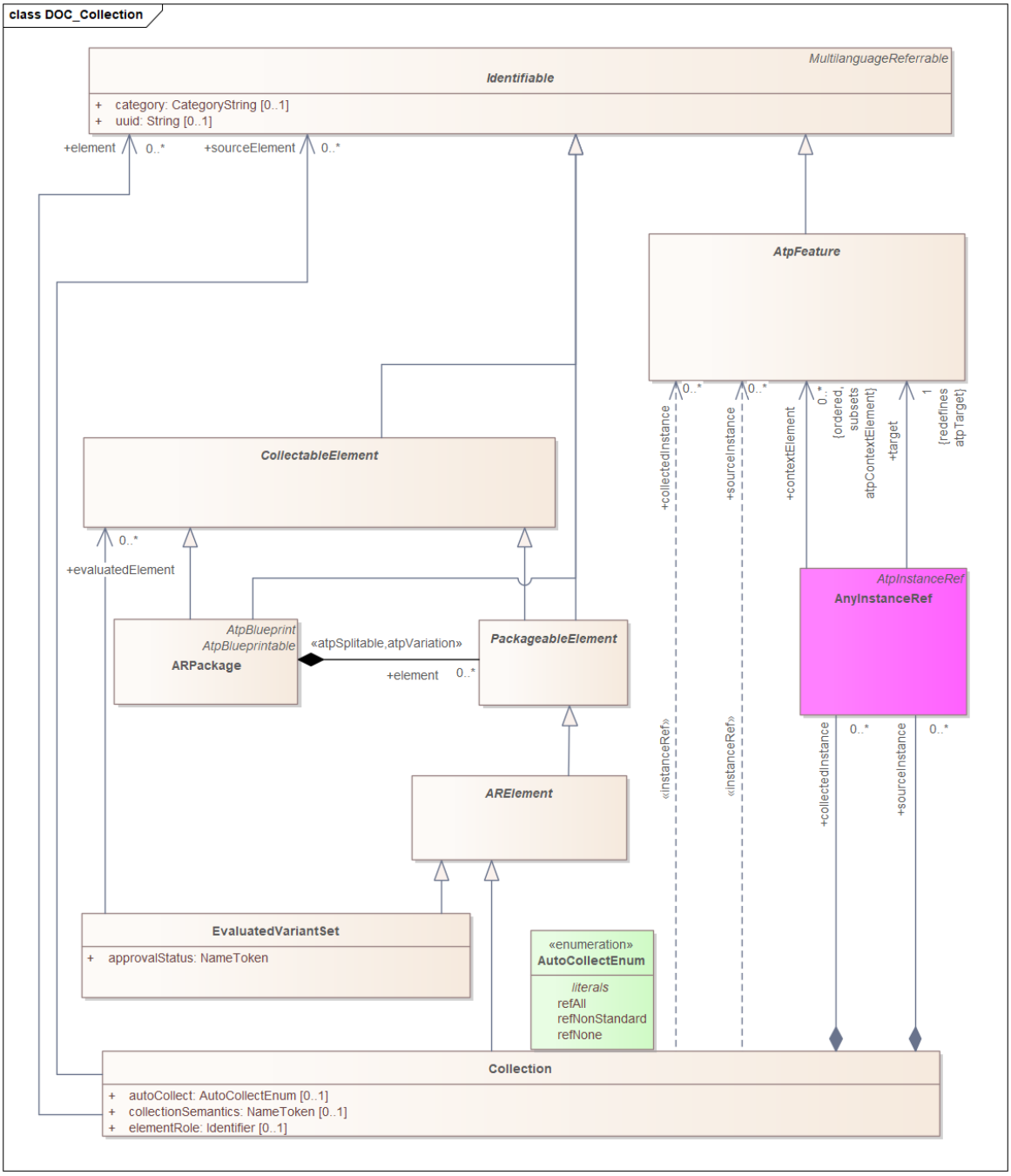


Figure 9.2: Representation of Collections in Meta Model

Two collections will be specified to define the views, one with AUTO-COLLECT=REF-ALL and the other one with AUTO-COLLECT=REF-NONE.

REF-ALL means that this collection automatically will include all referenced model elements. These elements are not listed within the collection.

REF-NONE means that this collection does not include any referenced model elements. That means that only the listed elements of the collection will belong to it.

For the generated XML the two generated collections will define the respective content.

With reference to xml extract below:

The collection with AUTO-COLLECT=REF-ALL contains a PortPrototypeBlueprint (AbsCtrlIntvg). That means that the referenced model elements like PortInterface, ApplicationDataType, CompuMethod and Unit also belong to this collection.

The collection with AUTO-COLLECT=REF-NONE contains a PortPrototypeBlueprint (AbsCtrlIntvg) and the referenced model elements like PortInterface, ApplicationDataType, CompuMethod and Unit within the collection.

For the specification of the views within the AI-Tool it is sufficient to mark a PortPrototypeBlueprint, e.g. the provider. That means it will belong to the view even if it is not marked on the receiver side. Also, it is possible to have two different views specified on different sheets. The XML generation will consider all the specified views.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<AUTOSAR xmlns="http://autosar.org/schema/r4.0" xmlns:xsi="http://www.w3.
    org/2001/XMLSchema-instance" xsi:schemaLocation="http://autosar.org/
    schema/r4.0 AUTOSAR_00053.xsd">
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>AUTOSAR</SHORT-NAME>
      <LONG-NAME>
        <L-4 L="EN">AUTOSAR</L-4>
      </LONG-NAME>
      <AR-PACKAGES>
        <AR-PACKAGE>
          <SHORT-NAME>AISpecification</SHORT-NAME>
          <LONG-NAME>
            <L-4 L="EN">AISpecification</L-4>
          </LONG-NAME>
          <AR-PACKAGES>
            <AR-PACKAGE>
              <SHORT-NAME>Collections_Blueprint</SHORT-NAME>
              <CATEGORY>BLUEPRINT</CATEGORY>
              <REFERENCE-BASES>
                <REFERENCE-BASE>
                  <SHORT-LABEL>PortPrototypeBlueprints</SHORT-LABEL>
                  <PACKAGE-REF DEST="AR-PACKAGE">/AUTOSAR/AISpecification/
                      PortPrototypeBlueprints_Blueprint</PACKAGE-REF>
                </REFERENCE-BASE>
                <REFERENCE-BASE>
                  <SHORT-LABEL>ApplicationDataTypes</SHORT-LABEL>
                  <PACKAGE-REF DEST="AR-PACKAGE">/AUTOSAR/AISpecification/
                      ApplicationDataTypes_Blueprint</PACKAGE-REF>
                </REFERENCE-BASE>
                <REFERENCE-BASE>
                  <SHORT-LABEL>ARKeywordSet</SHORT-LABEL>
                  <PACKAGE-REF DEST="AR-PACKAGE">/AUTOSAR/AISpecification/
                      KeywordSets_Blueprint</PACKAGE-REF>
                </REFERENCE-BASE>
                <REFERENCE-BASE>
                  <SHORT-LABEL>Collections_Blueprint</SHORT-LABEL>
                  <PACKAGE-REF DEST="AR-PACKAGE">/AUTOSAR/AISpecification/
                      Collections_Blueprint</PACKAGE-REF>
```

```
    </REFERENCE-BASE>
    <REFERENCE-BASE>
      <SHORT-LABEL>SwComponentTypes_Example</SHORT-LABEL>
      <PACKAGE-REF DEST="AR-PACKAGE">/AUTOSAR/AISpecification/
          SwComponentTypes_Example</PACKAGE-REF>
    </REFERENCE-BASE>
    <REFERENCE-BASE>
      <SHORT-LABEL>PortInterfaces_Blueprint</SHORT-LABEL>
      <PACKAGE-REF DEST="AR-PACKAGE">/AUTOSAR/AISpecification/
          PortInterfaces_Blueprint</PACKAGE-REF>
    </REFERENCE-BASE>
  </REFERENCE-BASES>
  <ELEMENTS>
    <COLLECTION>
      <SHORT-NAME>ChassisRefAll</SHORT-NAME>
      <CATEGORY>SET</CATEGORY>
      <AUTO-COLLECT>REF-ALL</AUTO-COLLECT>
      <ELEMENT-ROLE>PART_OF_SUBSET</ELEMENT-ROLE>
      <ELEMENT-REFS>
        <ELEMENT-REF BASE="PortPrototypeBlueprints" DEST="PORT-
            PROTOTYPE-BLUEPRINT">AbsCtrlIntvg</ELEMENT-REF>
        ...
        <ELEMENT-REF BASE="PortPrototypeBlueprints" DEST="PORT-
            PROTOTYPE-BLUEPRINT">AccrPedlMeasd</ELEMENT-REF>
      </ELEMENT-REFS>
    </COLLECTION>
      ...
    <COLLECTION>
      <SHORT-NAME>Chassis</SHORT-NAME>
      <CATEGORY>SET</CATEGORY>
      <AUTO-COLLECT>REF-NONE</AUTO-COLLECT>
      <ELEMENT-ROLE>PART_OF_SUBSET</ELEMENT-ROLE>
      <ELEMENT-REFS>
        <ELEMENT-REF BASE="PortPrototypeBlueprints" DEST="PORT-
            PROTOTYPE-BLUEPRINT">AArbnRes</ELEMENT-REF>
        ...
        <ELEMENT-REF BASE="PortPrototypeBlueprints" DEST="PORT-
            PROTOTYPE-BLUEPRINT">AbsCtrlIntvg</ELEMENT-REF>
        <ELEMENT-REF BASE="SwComponentTypes_Example" DEST="P-
            PORT-PROTOTYPE">Chassis/AbsCtrlIntvg</ELEMENT-REF>
        <ELEMENT-REF BASE="SwComponentTypes_Example" DEST="R-
            PORT-PROTOTYPE">CrsCtrlAndAcc/AbsCtrlIntvg</ELEMENT-
            REF>
        <ELEMENT-REF BASE="SwComponentTypes_Example" DEST="P-
            PORT-PROTOTYPE">Esc/AbsCtrlIntvg</ELEMENT-REF>
        <ELEMENT-REF BASE="SwComponentTypes_Example" DEST="R-
            PORT-PROTOTYPE">Rsc/AbsCtrlIntvg</ELEMENT-REF>
        <ELEMENT-REF BASE="SwComponentTypes_Example" DEST="R-
            PORT-PROTOTYPE">Ssm/AbsCtrlIntvg</ELEMENT-REF>
        <ELEMENT-REF BASE="SwComponentTypes_Example" DEST="R-
            PORT-PROTOTYPE">Steer/AbsCtrlIntvg</ELEMENT-REF>
        <ELEMENT-REF BASE="SwComponentTypes_Example" DEST="R-
            PORT-PROTOTYPE">SteerVehStabyCtrl/AbsCtrlIntvg</
            ELEMENT-REF>
        <ELEMENT-REF BASE="SwComponentTypes_Example" DEST="R-
            PORT-PROTOTYPE">Vlc/AbsCtrlIntvg</ELEMENT-REF>
```

```
                    <ELEMENT-REF BASE="PortInterfaces_Blueprint" DEST="
                        SENDER-RECEIVER-INTERFACE">AbsCtrlIntvg1</ELEMENT-
                        REF>
                    ...
                    <ELEMENT-REF BASE="SwComponentTypes_Example" DEST="R-
                        PORT-PROTOTYPE">Susp/YawStabyInfoByEsc</ELEMENT-REF>
                  </ELEMENT-REFS>
                </COLLECTION>
                <COLLECTION>
                  <SHORT-NAME>ChassisRefAllToChassis</SHORT-NAME>
                  <CATEGORY>RELATION</CATEGORY>
                  <ELEMENT-ROLE>AUTO_COLLECTED_FROM</ELEMENT-ROLE>
                  <ELEMENT-REFS>
                    <ELEMENT-REF BASE="Collections_Blueprint" DEST="
                        COLLECTION">Chassis</ELEMENT-REF>
                  </ELEMENT-REFS>
                  <SOURCE-ELEMENT-REFS>
                    <SOURCE-ELEMENT-REF BASE="Collections_Blueprint" DEST="
                        COLLECTION">ChassisRefAll</SOURCE-ELEMENT-REF>
                  </SOURCE-ELEMENT-REFS>
                </COLLECTION>
              </ELEMENTS>
            </AR-PACKAGE>
          </AR-PACKAGES>
        </AR-PACKAGE>
      </AR-PACKAGES>
    </AR-PACKAGE>
  </AR-PACKAGES>
</AUTOSAR>
```

10 Data Structure of Application Interface Definitions in the AI-Tool

The following sections show the representation of data used to define application interfaces on VfB level in the AI-Tool. The content maintained in the AI-Tool is collected in one large file called ApplicationInterfaces.ai. During a new AUTOSAR-Release, the Blueprint definitions are generated to the following .arxml collections

- Blueprint definitions beside the physical units: AUTOSAR_CP_MOD_AISpecification.zip [1]
- Example definitions: AUTOSAR_CP_MOD_AISpecificationExamples.zip [2]
- Physical Unit and Dimension definitions: AUTOSAR_FO_MOD_GeneralBlueprints.zip [16]

Note: All the figures and screenshots shown in the following sections are only snapshots and may not exactly match the views in the AI-Tool.

10.1 Main Views of the AI-Tool

The AI-Tool GUI provides several views to represent the contents of the .ai file.

10.1.1 Explorer View

There is an Explorer View, where all elements are listed. The Explorer View displays the content of the .ai-file structured according to the main package structure as defined in the AUTOSAR_CP_MOD_AIspecification and in the AUTOSAR_FO_MOD_GeneralBlueprints. There are the following packages available:

The AISpecification Package contains the following collections of definitions:

- ApplicationDataTypes_Blueprint [ARPackage]
- ApplicationDataTypes_Example [ARPackage]
- CompuMethods_Blueprint [ARPackage]
- CompuMethods_Example [ARPackage]
- DataConstrs_Blueprint [ARPackage]
- DataConstrs_Example [ARPackage]
- KeywordSets_Blueprint [ARPackage]
- PhysicalDimensions_obsolete [ARPackage]
- PortInterfaces_Blueprint [ARPackage]

- PortInterfaces_Example [ARPackage]
- PortPrototypeBlueprints_Blueprint [ARPackage]
- SwComponentTypes_Example [ARPackage]
- Systems [ARPackage]
- Units_obsolete [ARPackage]

The PhysicalUnits Package contains the following collections of definitions:

- PhysicalDimensions_Blueprint [ARPackage]
- Units_Blueprint

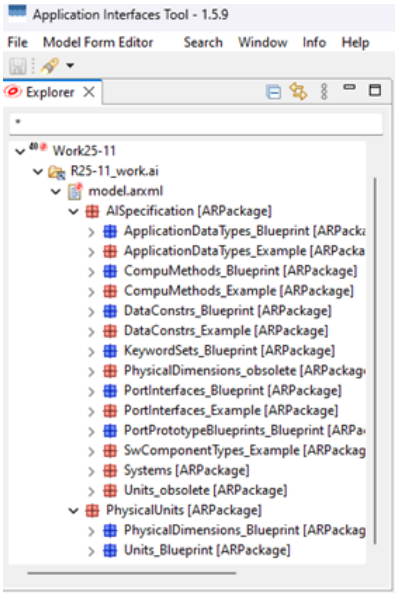


Figure 10.1: Explorer View in the AI-Tool GUI

10.1.2 Editor View

For each element type which can be found in the Explorer View, there is a specific editor available. This view allows the maintenance of all necessary information specific for the definition of the elements of this type. Compared with option to maintain all contents directly in .arxml, this provides a very convenient HMI which abstracts from the .arxml syntax rules.

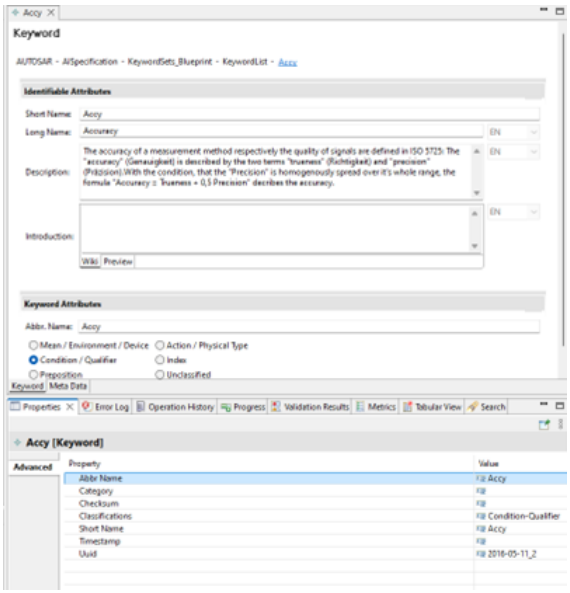


Figure 10.2: Editor View in the AI-Tool GUI, example for Keyword definition

10.1.3 Working Panel View

There is a Working Panel View where several working modes are supported. Here, you find for any special working mode a separate tab, where the information needed for the working mode is made available, see also, e.g. in Figure 10.2. The different working modes provided are:

Tab	Use Case
Properties	Editing Properties of an Element
Error Log Operation History Progress	Debugging of the tool
Validation Results	Validation of the content against AI rules, like, e.g. Naming Checks
Tabular View	Compare, search, filter element definitions per element type
Search	Search for element definitions in the whole .ai-file

Table 10.1: Tabs in the Editor View of the AI-File

In the following sections, the views of the packages are explained in detail. Some views in the tool abstract the detailed structure of the .arxml files but are collections of relevant information for the different use cases or element definitions.

The sections are ordered according to the ordering of the tables in the Tabular View.

Note: The figures and screen shots shown in the following sections are only snapshots and they may not match exactly to the current AI-Tool views.

10.2 Overview: Table AI Specification in Tabular View

The table AI Specification in the Tabular View of the AI-Tool gives an overview of all relevant information to describe the properties of the information exchanged between SW Components via ports. This is a high-level collection of port-properties allowing discussion of system interfaces abstracted from implementation details. It bases on the definitions in the ARPackages:

- ApplicationDataTypes_Blueprint
- CompuMethods_Blueprint
- DataConstrs_Blueprint
- PortInterfaces_Blueprint
- PortPrototypeBlueprints_Blueprint
- PhysicalDimensions_Blueprint
- Units_Blueprint

The elements of the PortPrototypeBlueprint definitions collected in this table are:

- Shortname
- Longname
- Description
- Interface Shortname
- Application Datatype Shortname
- Datatype SW intended Resolution
- Data Constraints Lower Limit
- Data Constraints Upper Limit
- Unit Displayname
- Physical Dimension Shortname
- Physical Dimension Longname
- Unit Displayname

Additionally, there is Milestone, Life Cycle State and Life Cycle and UseInstead information available for each PortPrototypeBlueprint definition.

Port Prototype information: The columns Shortname, Longname and Description are generated from the definitions in the ARPackage PortPrototypeBlueprints_Blueprint.

Port Interface information: The column Interface Shortname is generated from the definitions in the ARPackage PortInterfaces_Blueprint.

Application Datatype information: The column DATATYPE SHORTNAME is generated from the definitions in the ARPackage ApplicationDataTypes_Blueprint.

Computation Method information: The column Datatype SW Intended Resolution is generated from the definitions in the ARPackage CompuMethods_Blueprint.

Data Constraints information: The column Data Constraints Lower Limit and Data Constraints Upper Limit are generated from the definitions in the ARPackage DataConstrs_Blueprint.

Unit information: The column Unit Displayname is generated from the definitions in the ARPackage Units_Blueprint.

Physical Dimension information: The columns Physical Dimension Shortname and Physical Dimension Longname are generated from the definitions in the ARPackage PhysicalDimensions_Blueprint.

10.3 Application Array Data Types

This ARPackage contains definitions of ArrayDataTypes defined to be utilized in any of the PortInterfaces and in other complex DataTypes.

There are 2 ARPackages available in the AI-Tool Explorer for ApplicationDataType definitions:

- AISpecification_ApplicationDataType_Blueprint
- AISpecification_ApplicationDataType_Example

According to the AUTOSAR Software Component Template [4], the elements of the ApplicationArrayDataType definition are:

- Common Elements
  - Shortname
  - Longname
  - Description
  - Introduction
  - Sw Data Def Props: Calibration Access
  - Shortname of the Array Elements
  - Longname of the Array Elements
  - Description of the Array Elements
  - Introduction of the Array Elements
  - Max. Number of Elements

  - Array Size Semantics
  - Type (ApplicationDatatype of the Array Elements)
- Blueprint specific Elements
  - Blueprints: Name Pattern
  - Blueprints: Derived Elements
- Example specific Elements
  - Derived from

Additionally, there is Milestone, Life Cycle State and Life Cycle and UseInstead information available for each Application Array Data Type definition.

Shortname:
The Shortname of an array datatype shall reflect the functional/architectural content of the array elements. Usually Shortnames are derived from architecture descriptions in the Longnames or the descriptions of the AR Elements.

The Shortname of an array data type is defined as a referrable identifier and must therefore respect rules for identifiers. It is built by concatenating keywords and an index as defined by AUTOSAR_MOD_AISpecification_KeywordSet_Blueprint.arxml. The naming rules are defined at AUTOSAR Software Component Modeling Guide [6].

Longname:
The Longname of an array datatype shall reflect the functional/architectural content of the array elements. Usually, Longname and Description which are short architectural summaries of the content of the AR Elements, are the basis for the derivation of Shortnames.

The Longname of an array data type is the spelled-out wording of the Shortname, including the index. It is a plain text which shall also not contain any special characters. The naming rules are defined at AUTOSAR Software Component Modeling Guide [6].

Description:
The Description of an array data type shall contain a very short explanation of the cases, where the array elements shall be used. Usually, Longname and Description which are short architectural summaries of the content of the AR Elements are the basis for the derivation of Shortnames.

Introduction:
The Introduction of an array data type can be used to document the intended assignment of the array elements to physical components of the system, e.g. assignment of the four wheels in a car to the element indices 1, 2, 3 and 4. If necessary, additional information concerning use cases or application rules can be given, here.

Sw Data Def Props: SW Calibration Access:
The SW Calibration Access property of an data type describes, if and how the memory

cell, where the values of this type are stored, can be accessed. Possible entries, here, are:

- notAccessible (e.g. for only internal intermediate value storage)
- readOnly (e.g. for variables which shall be watched, but not modified)
- readWrite (e.g. for calibration data which can be modified in a garage)

Array Element Shortname:
The Shortname of the array data type elements shall reflect the functional/architectural content of the array elements. Usually, these Shortnames are derived from the Shortname of the array data type.

The Shortname of an array data type element is defined as a referrable identifier and must therefore respect rules for identifiers. It is built by concatenating keywords and an index as defined by AUTOSAR_MOD_AISpecification_KeywordSet_Blueprint.arxml. The naming rules are defined at AUTOSAR Software Component Modeling Guide [6].

Array Element Longname:
The Longname of the array data type elements shall reflect the functional/architectural content of the array elements. Usually, this Longname is derived from the Longname of the array data type but in most of the practical cases this is left empty.

Array Element Description:
The Description of the array data type elements shall contain a very short explanation of the cases, where the array elements shall be used. This Description can be derived from the Description of the array data type, but in most of the practical cases this is left empty.

Array Element Introduction:
This Introduction can be derived from the Introduction of the array data type, but in most of the practical cases this is left empty.

Max. Number of Elements:
Here, the maximum possible number of array elements is defined.

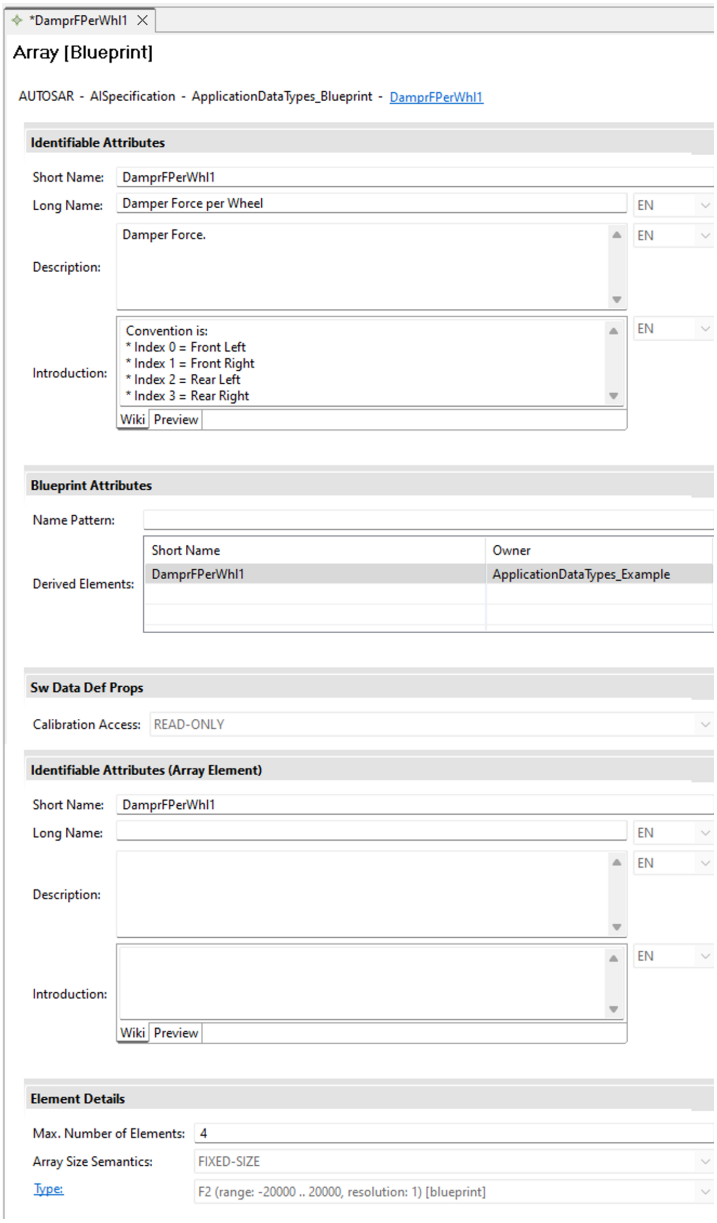


Figure 10.3: Editor View of Application Array Data Type Definitions

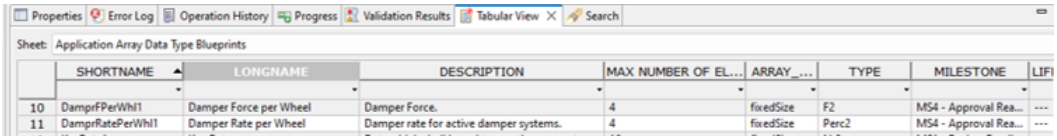


Figure 10.4: Tabular View of Application Array Data Type Definitions

Example for the definition of an ApplicationArrayDatatype in AUTOSAR_MOD_AISpecification_ApplicationDataType_Blueprint.arxml:

```
...
    <APPLICATION-ARRAY-DATA-TYPE>
      <SHORT-NAME>DamprFPerWhl1</SHORT-NAME>
      <LONG-NAME>
        <L-4 L="EN">Damper Force per Wheel</L-4>
```

```
      </LONG-NAME>
      <DESC>
        <L-2 L="EN">Damper Force.</L-2>
      </DESC>
      <CATEGORY>ARRAY</CATEGORY>
      <INTRODUCTION>
        <P>
          <L-1 L="EN">Convention is: </L-1>
        </P>
          <LIST TYPE="UNNUMBER">
            <ITEM>
              <P>
                <L-1 L="EN">Index 0 = Front Left </L-1>
              </P>
            </ITEM>
            <ITEM>
              <P>
                <L-1 L="EN">Index 1 = Front Right </L-1>
              </P>
            </ITEM>
            <ITEM>
              <P>
                <L-1 L="EN">Index 2 = Rear Left </L-1>
              </P>
            </ITEM>
            <ITEM>
              <P>
                <L-1 L="EN">Index 3 = Rear Right</L-1>
              </P>
            </ITEM>
          </LIST>
      </INTRODUCTION>
      <SW-DATA-DEF-PROPS>
        <SW-DATA-DEF-PROPS-VARIANTS>
          <SW-DATA-DEF-PROPS-CONDITIONAL>
            <SW-CALIBRATION-ACCESS>READ-ONLY</SW-CALIBRATION-ACCESS>
          </SW-DATA-DEF-PROPS-CONDITIONAL>
        </SW-DATA-DEF-PROPS-VARIANTS>
      </SW-DATA-DEF-PROPS>
      <ELEMENT>
        <SHORT-NAME>DamprFPerWhl1</SHORT-NAME>
        <TYPE-TREF BASE="ApplicationDataTypes" DEST="APPLICATION-PRIMITIVE-
            DATA-TYPE">F2</TYPE-TREF>
        <ARRAY-SIZE-SEMANTICS>FIXED-SIZE</ARRAY-SIZE-SEMANTICS>
        <MAX-NUMBER-OF-ELEMENTS>4</MAX-NUMBER-OF-ELEMENTS>
      </ELEMENT>
    </APPLICATION-ARRAY-DATA-TYPE>
...
```

10.4 Application Primitive Continuous Data Types

Application DataTypes define the data attributes which are needed from the application point of view, in order to exchange data between software components or between a

software component and a measurement and calibration tool. Application Interfaces does not standardize implementation DataTypes.

ApplicationPrimitiveDataTypes are defined to standardize reusable conversions of internal hexadecimal values into an interpretation in a physical environment. These primitive data types can be utilized in any of the PortInterfaces or in complex DataTypes like arrays, records or structures.

In case of continuous representation, the internal values are scaled to rational numbers. There are several computation methods available to describe these conversions.

There are 2 ARPackages available in the AI-Tool Explorer for ApplicationDataType definitions:

- AISpecification_ApplicationDataType_Blueprint
- AISpecification_ApplicationDataType_Example

According to the AUTOSAR Software Component Template [4], the elements of the ApplicationPrimitiveContinuousDataType definition are:

- Common Elements
  - Shortname
  - Longname
  - Description
  - Introduction
  - Resolution
  - Unit
  - Sw Data Def Props: Calibration Access
  - Data Constraints
  - Computation Method
- Blueprint specific Elements
  - Blueprints: Name Pattern
  - Blueprints: Derived Elements
- Example specific Elements
  - Derived from

Additionally, there is Milestone, Life Cycle State and Life Cycle and UseInstead information available for each Application Continuous Data Type definition.

Shortname:
The Shortname of a continuous data type shall reflect the physical interpretation of

the datatype. Usually Shortnames are derived from architecture descriptions in the Longnames or the Descriptions of the AR Elements.

The Shortname of a continuous data type is defined as a referrable identifier and must therefore respect Rules for identifiers. It is built by concatenating the keyword for the corresponding physical quantity and an index as defined by AUTOSAR_MOD_AISpecification_KeywordSet_Blueprint.arxml. The naming rules are defined at AUTOSAR Software Component Modeling Guide [6].

Longname:
The Shortname of a continuous data type shall reflect the physical interpretation of the datatype. Usually, Longname and Description which are short architectural summaries of the content of the AR Elements, are the basis for the derivation of Shortnames.

The Longname of a continuous data type is the spelled-out wording of the Shortname, including the index. It is a plain text which shall also not contain any special characters. The naming rules are defined at AUTOSAR Software Component Modeling Guide [6].

Description:
The Description of a continuous data type shall contain a very short explanation of the cases, where the data type shall be used or how the values can be interpreted in a physical world. Usually, Longname and Description which are short architectural summaries of the content of the AR Elements, are the basis for the derivation of Shortnames.

Introduction:
The Introduction of a continuous data type is currently not used. If necessary, additional information concerning use cases or application rules can be given, here.

Resolution:
The Resolution of a continuous data type describes, how an internally calculated hexadecimal value shall be displayed, e.g. in a calibration tool or in a dashboard.
Attention: The resolution specified, here, has to be synchronized with the definition for the related Computation Method. Unfortunately, the AUTOSAR Metamodel allows both the definition for a resolution related to an Application Datatype and the definition of a resolution related to a Computation Method (10.7). In the development phase, the specification of contradicting values for the resolution on DataType level and on CompuMethod level must be avoided.

Unit:
The Unit referenced by a continuous data type describes, how an internally calculated hexadecimal value shall be interpreted for a physical world. The reference is represented by the Shortname of the physical unit which allows to display all other definition elements of units, including the properties of the related physical dimension.See also chapter Units 10.14.

Sw Data Def Props: SW Calibration Access:
The SW Calibration Access property of a primitive data type describes, if and how the memory cell, where the values of this type are stored, can be accessed. Possible entries, here, are:

- notAccessible (e.g. for only internal intermediate value storage)
- readOnly (e.g. for variables which shall be watched, but not modified)
- readWrite (e.g. for calibration data which can be modified in a garage)

Data Constraints:
Here, a range of valid values can be described. For this a separate ARPackage is available and the reference to these definitions is represented by the Shortname of the DataConstraint. See also chapter Data Constraints 10.9.

Computation Method:
Here, the conversion of hexadecimal values into physically interpretable values or expressions is described. Also for this a separate ARPackage is available and the reference to these definitions is represented by the Shortname of the CompuMethod. Application primitive continuous Datatypes must have a CompuMethod, which allows the conversion of either hexadecimal values into integer values via a fix resolution or via floating point interpretation. There is an additional option to describe conversions which are a combination of an integer interpretation of a part of the value range and a text-table conversion for other values. See also chapter Compumethods 10.7.

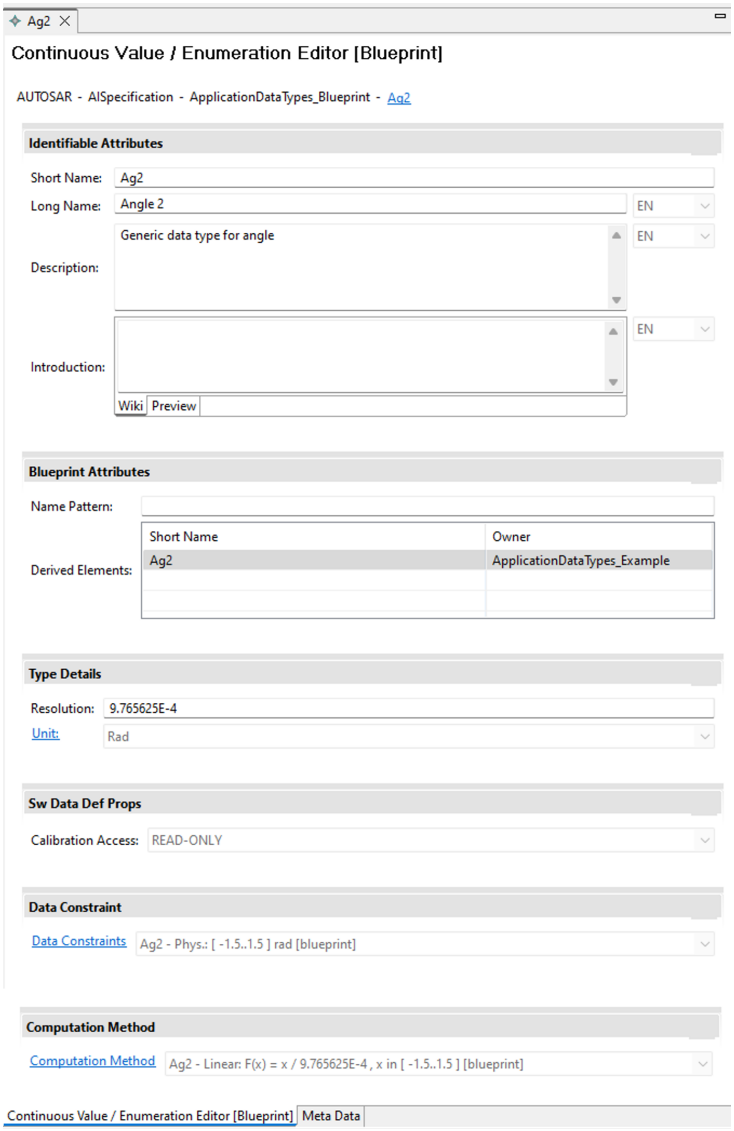


Figure 10.5: Editor view of Application Continuous Data Type Definitions

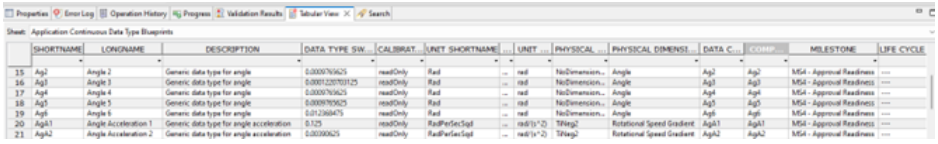


Figure 10.6: Tabular view of Application Continuous Data Type Definitions

Example for the definition of an ApplicationPrimitiveContinuousDatatype in AUTOSAR_MOD_AISpecification_ApplicationDataType_Blueprint.arxml:

```
  <ELEMENTS>
...
    <APPLICATION-PRIMITIVE-DATA-TYPE>
      <SHORT-NAME>Ag2</SHORT-NAME>
      <LONG-NAME>
        <L-4 L="EN">Angle 2</L-4>
      </LONG-NAME>
```

```
      <DESC>
        <L-2 L="EN">Generic data type for angle</L-2>
      </DESC>
      <CATEGORY>VALUE</CATEGORY>
      <SW-DATA-DEF-PROPS>
        <SW-DATA-DEF-PROPS-VARIANTS>
          <SW-DATA-DEF-PROPS-CONDITIONAL>
            <SW-CALIBRATION-ACCESS>READ-ONLY</SW-CALIBRATION-ACCESS>
            <COMPU-METHOD-REF BASE="CompuMethods" DEST="COMPU-METHOD">Ag2</
                COMPU-METHOD-REF>
            <DATA-CONSTR-REF BASE="DataConstrs" DEST="DATA-CONSTR">Ag2</
                DATA-CONSTR-REF>
            <SW-INTENDED-RESOLUTION>0.0009765625</SW-INTENDED-RESOLUTION>
            <UNIT-REF BASE="Units" DEST="UNIT">Rad</UNIT-REF>
          </SW-DATA-DEF-PROPS-CONDITIONAL>
        </SW-DATA-DEF-PROPS-VARIANTS>
      </SW-DATA-DEF-PROPS>
    </APPLICATION-PRIMITIVE-DATA-TYPE>
...
  </ELEMENTS>
```

10.5 Application Primitive Enum Data Types

This ARPackage contains the definitions of primitive enumeration DataTypes which can be utilized in any of the PortInterfaces and in complex DataTypes.

Enumeration data types represent a value range and the interpretation of the single values of this range as a kind of state of the system. Each of the values can be displayed in a dashboard or in a calibration tool as text. That's why they are also often called text-table conversions. Usually, enumeration data types are used to represent the results (States) of State Machines or Boolean operations. There are several computation methods available to describe these conversions.

There are 2 ARPackages available in the AI-Tool Explorer for ApplicationPrimitiveDataType definitions:

- AISpecification_ApplicationDataType_Blueprint
- AISpecification_ApplicationDataType_Example

According to the AUTOSAR Software Component Template [4], the elements of the ApplicationPrimitiveEnumDataType definition are:

- Common Elements
  - Shortname
  - Longname
  - Description
  - Introduction

  - Unit
  - Sw Data Def Props: Calibration Access
  - Data Constraints
  - Computation Method
- Blueprint specific Elements
  - Blueprints: Name Pattern
  - Blueprints: Derived Elements
- Example specific Elements
  - Derived from

Additionally, there is Milestone, Life Cycle State and Life Cycle and UseInstead information available for each Application Continuous Data Type definition.

Shortname:
The Shortname of an enumeration data type shall describe the functional content of a State Machine. Usually Shortnames are derived from architecture descriptions in the Longnames or the Descriptions of the AR Elements.

The Shortname of an enumeration data type is built by concatenating Keywords and an index as defined by AUTOSAR_MOD_AISpecification_KeywordSet_Blueprint.arxml. The naming rules are defined at AUTOSAR Software Component Modeling Guide [6].

Longname:
The Longname of an enumeration data type shall describe the functional content of a State Machine. Usually, Longname and Description which are short architectural summaries of the content of the AR Elements are the basis for the derivation of Shortnames.

The Longname of an enumeration data type is the spelled-out wording of the Shortname, including the index. It is a plain text and must not contain any special characters. The naming rules are defined at AUTOSAR Software Component Modeling Guide [6].

Resolution:
The Resolution of an enum data type is usually not used and therefore not available in the Editor View for enumeration data types in the AI-Tool. It is - if at all - inherently defined as resolution=1 because each hexadecimal value shall be interpreted as a single state of a State Machine. There should not be any floating point numbers interpretation.

Unit of Enumeration Data Types:
The Unit which could be referenced by an enumeration data type, is called NoUnit, but this can also be left empty.

Computation Method of Continuous Data Types:
The Computation Method of an enumeration data type must be a text-table conversion. See also chapter Compumethods 10.7.

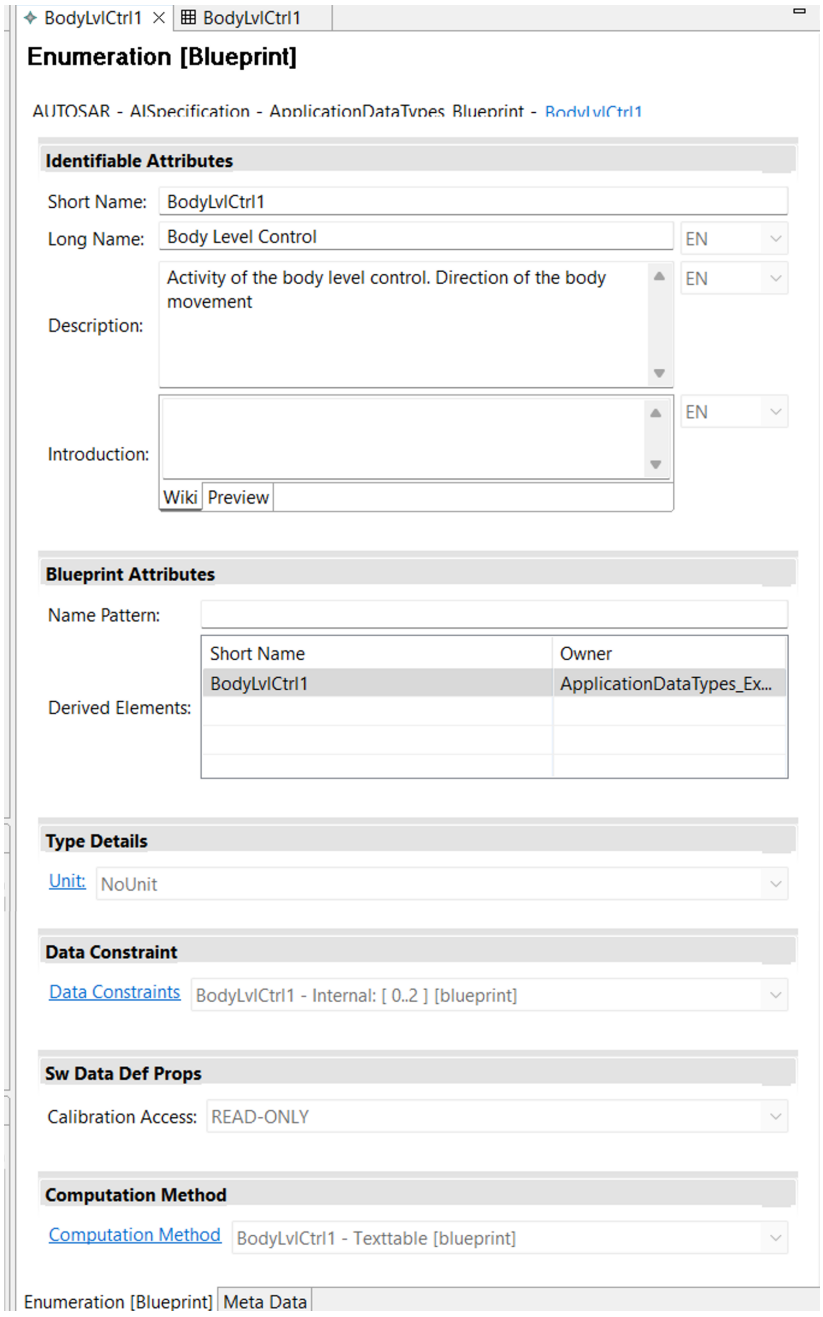


Figure 10.7: Editor View of Application Enumeration Data Type Definitions

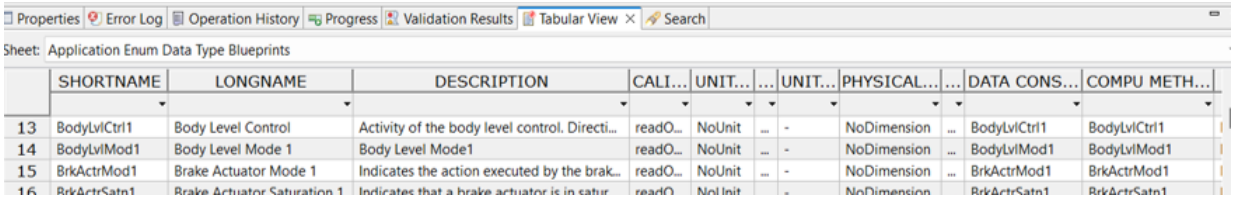


Figure 10.8: Tabular View of Application Enumeration Data Type Definitions

Example for the definition of an ApplicationPrimitiveEnumDatatype in AUTOSAR_MOD_AISpecification_ApplicationDataType_Blueprint.arxml:

```
  <ELEMENTS>
...
    <APPLICATION-PRIMITIVE-DATA-TYPE>
      <SHORT-NAME>BodyLvlCtrl1</SHORT-NAME>
      <LONG-NAME>
        <L-4 L="EN">Body Level Control</L-4>
      </LONG-NAME>
      <DESC>
        <L-2 L="EN">Activity of the body level control. Direction of the
            body movement</L-2>
      </DESC>
      <CATEGORY>VALUE</CATEGORY>
      <SW-DATA-DEF-PROPS>
        <SW-DATA-DEF-PROPS-VARIANTS>
          <SW-DATA-DEF-PROPS-CONDITIONAL>
            <SW-CALIBRATION-ACCESS>READ-ONLY</SW-CALIBRATION-ACCESS>
            <COMPU-METHOD-REF BASE="CompuMethods" DEST="COMPU-METHOD">
               BodyLvlCtrl1</COMPU-METHOD-REF>
            <DATA-CONSTR-REF BASE="DataConstrs" DEST="DATA-CONSTR">
               BodyLvlCtrl1</DATA-CONSTR-REF>
            <UNIT-REF BASE="Units" DEST="UNIT">NoUnit</UNIT-REF>
          </SW-DATA-DEF-PROPS-CONDITIONAL>
        </SW-DATA-DEF-PROPS-VARIANTS>
      </SW-DATA-DEF-PROPS>
    </APPLICATION-PRIMITIVE-DATA-TYPE>
  ...
```

10.6 Application Record Data Types

This ARPackage contains definitions of complex ApplicationRecordDataTypes defined to be utilized in any of the PortInterfaces and in other complex DataTypes.

Record data types are utilized to store the values of several predefined data types. Each record data type can have one or more record elements, each record element will have a different Shortname and may have different (sub) DataTypes than the other elements. These are similar data type definitions like the "C" language structure types.

There are 2 ARPackages available in the AI-Tool Explorer for ApplicationDataType definitions:

- AISpecification_ApplicationDataType_Blueprint
- AISpecification_ApplicationDataType_Example

According to the AUTOSAR Software Component Template [4], the elements of the ApplicationRecordDataType definition are:

- Common Elements
  - Shortname

  - Longname
  - Description
  - Introduction
  - Sw Data Def Props: Calibration Access
  - Shortname of the Record Elements
  - Type of the Record Elements (ApplicationDatatype of the Record Elements)
- Blueprint specific Elements
  - Blueprints: Name Pattern
  - Blueprints: Derived Elements
- Example specific Elements
  - Derived from

Additionally, there is Milestone, Life Cycle State and Life Cycle and UseInstead information available for each Application Array Data Type definition.

Shortname:
The Shortname of a record data type shall reflect the functional/architectural content of the whole Record. Usually Shortnames are derived from architecture descriptions in the Longnames or the descriptions of the AR Elements.

The Shortname of a record data type is defined as a referrable identifier and must therefore respect rules for identifiers. It is built by concatenating keywords and an index as defined by AUTOSAR_MOD_AISpecification_KeywordSet_Blueprint.arxml. The naming rules are defined at AUTOSAR Software Component Modeling Guide [6].

Longname:
The Longname of a record data type shall reflect the functional/architectural content of the record. Usually, Longname and Description which are short architectural summaries of the content of the AR Elements, are the basis for the derivation of Shortnames.

The Longname of a record data type is the spelled-out wording of the Shortname. It is a plain text which shall also not contain any special characters. The naming rules are defined at AUTOSAR Software Component Modeling Guide [6].

Description:
The Description of a record data type shall contain a very short explanation of the cases, where the record shall be used. Usually, Longname and Description which are short architectural summaries of the content of the AR Elements are the basis for the derivation of Shortnames.

Introduction:
The Introduction of a record data type can be used to give additional information concerning use cases or application rules.

Sw Data Def Props: SW Calibration Access:
The SW Calibration Access property of a data type describes, if and how the memory cell, where the values of this type are stored, can be accessed. Possible entries, here, are:

- notAccessible (e.g. for only internal intermediate value storage)
- readOnly (e.g. for variables which shall be watched, but not modified)
- readWrite (e.g. for calibration data which can be modified in a garage)

Record Element Shortname:
The Shortname of the record data type elements shall describe the content of the record element.

The Shortname of a record data type element is defined as a referrable identifier and must therefore respect rules for identifiers. It is built by concatenating keywords as defined by AUTOSAR_MOD_AISpecification_KeywordSet_Blueprint.arxml. The naming rules are defined at AUTOSAR Software Component Modeling Guide [6].

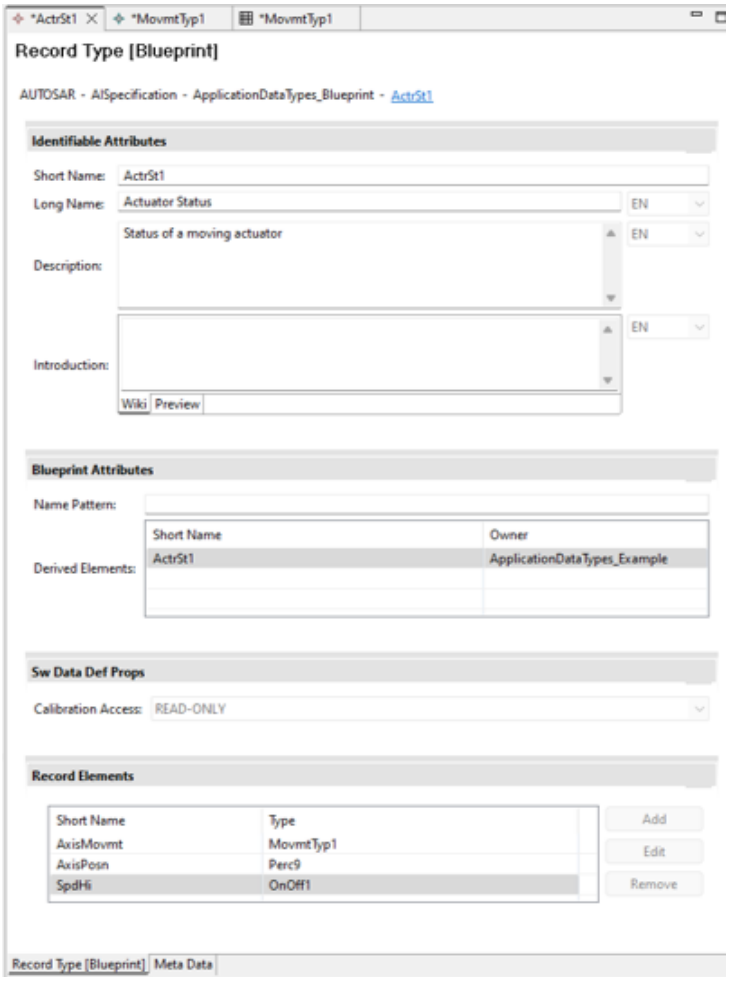


Figure 10.9: Editor View of Application Record Data Type Definitions

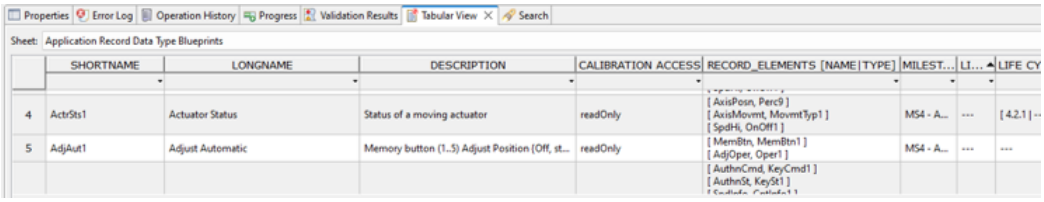


Figure 10.10: Tabular View of Application Record Data Type Definitions

Example for the definition of an ApplicationRecordDatatype in AUTOSAR_MOD_AISpecification_ApplicationDataType_Blueprint.arxml:

```
...
  <APPLICATION-RECORD-DATA-TYPE>
    <SHORT-NAME>ActrSt1</SHORT-NAME>
    <LONG-NAME>
      <L-4 L="EN">Actuator Status</L-4>
    </LONG-NAME>
    <DESC>
      <L-2 L="EN">Status of a moving actuator</L-2>
    </DESC>
    <CATEGORY>STRUCTURE</CATEGORY>
    <SW-DATA-DEF-PROPS>
      <SW-DATA-DEF-PROPS-VARIANTS>
        <SW-DATA-DEF-PROPS-CONDITIONAL>
          <SW-CALIBRATION-ACCESS>READ-ONLY</SW-CALIBRATION-ACCESS>
        </SW-DATA-DEF-PROPS-CONDITIONAL>
      </SW-DATA-DEF-PROPS-VARIANTS>
    </SW-DATA-DEF-PROPS>
    <ELEMENTS>
      <APPLICATION-RECORD-ELEMENT>
        <SHORT-NAME>AxisPosn</SHORT-NAME>
        <DESC>
          <L-2 L="EN">Current position of the actuator</L-2>
        </DESC>
        <TYPE-TREF BASE="ApplicationDataTypes" DEST="APPLICATION-PRIMITIVE-
            DATA-TYPE">Perc9</TYPE-TREF>
      </APPLICATION-RECORD-ELEMENT>
      <APPLICATION-RECORD-ELEMENT>
        <SHORT-NAME>AxisMovmt</SHORT-NAME>
        <DESC>
          <L-2 L="EN">Current movement type of the actuator</L-2>
        </DESC>
        <TYPE-TREF BASE="ApplicationDataTypes" DEST="APPLICATION-PRIMITIVE-
            DATA-TYPE">MovmtTyp1</TYPE-TREF>
      </APPLICATION-RECORD-ELEMENT>
      <APPLICATION-RECORD-ELEMENT>
        <SHORT-NAME>SpdHi</SHORT-NAME>
        <DESC>
          <L-2 L="EN">Current speed of the actuator</L-2>
        </DESC>
        <TYPE-TREF BASE="ApplicationDataTypes" DEST="APPLICATION-PRIMITIVE-
            DATA-TYPE">OnOff1</TYPE-TREF>
      </APPLICATION-RECORD-ELEMENT>
    </ELEMENTS>
  </APPLICATION-RECORD-DATA-TYPE>
```

. . .

10.7 Computation Methods

Here, we collect information to describe the relationship between a physical value and its internal mathematical representation and vice versa.

For the specification of Application Interfaces, we standardize only the direction from the internal mathematical representation to the representation in the physical world, even if the ComputationMethod can be used for both directions.

AUTOSAR_CP_TPS_SoftwareComponent Template [4], Chapter 'Computation Methods': For simple functions like identical (1:1 conversion) or linear functions this is sufficient because the inverse function can be derived quite easily from the defined function. In this case also the limits for the reverse direction can be gained by applying the forward function to the forward limits. For more complex functions (e.g. rational functions) it is usually not possible to compute the inverse function automatically. More seriously, the inversion yields ambiguous results if the function is not monotonic. To deal with such possible ambiguities directly, an inverse value can be provided explicitly for the function or for each of its parts respectively.

There are 2 ARPackages available in the AI-Tool Explorer for Computation Method definitions:

- AISpecification_CompuMethods_Blueprint
- AISpecification_CompuMethods_Example

According to the AUTOSAR Software Component Template [4], the elements of the ComputationMethod definition are:

- Common Elements
  - Category
  - Shortname
  - Longname
  - Description
  - Introduction
  - Unit
  - CompuScale
- Blueprint specific Elements
  - Blueprints: Name Pattern
  - Blueprints: Derived Elements

- Example specific Elements
  - Derived from

Additionally, there is Milestone, Life Cycle State and Life Cycle, and UseInstead information available for each Application Continuous Data Type definition.

Shortname:
The Shortname of a computation method is defined as a referrable identifier and must therefore respect rules for identifiers. It is built by concatenating the keyword for the corresponding physical quantity and an index as defined by AUTOSAR_MOD_AISpecification_KeywordSet_Blueprint.arxml. The naming rules are defined at AUTOSAR Software Component Modeling Guide [6].

Longname:
Usually, Longname and Description which are short architectural summaries of the content of the AR Elements, are the basis for the derivation of Shortnames. The Longname of a computation method is the spelled-out wording of the Shortname, including the index. It is a plain text which shall also not contain any special characters. The naming rules are defined at AUTOSAR Software Component Modeling Guide [6].

Description:
The Description of a computation method shall contain a very short explanation of the cases, where the data type shall be used or how the values can be interpreted in a physical world. Usually, Longname and Description which are short architectural summaries of the content of the AR Elements, are the basis for the derivation of Shortnames.

Introduction:
The Introduction of a computation method is currently not used. If necessary, additional information concerning use cases or application rules can be given, here.

Unit:
The Unit referenced by a computation method describes, how an internally calculated hexadecimal value shall be interpreted for a physical world. The reference is represented by the Shortname of the physical unit which allows to display all other definition elements of units, including the properties of the related physical dimension. See also chapter Units [10.14].

CompuScale:
One computation method definition can contain more than one CompuScales. These CompuScales include the description of the value ranges (intervals), their limits and properties and the value conversion description (e.g. Formula, Texttable, ...) for the values in the interval. An interval can be of the following types:

- CLOSED - The interval is limited by the value. The value is included.
- OPEN - The interval is limited by the value. The value is not included.

Category of Computation Methods
The Category of a computation method defines its basic structure. The following computation method categories are available:

- IDENTICAL
- LINEAR
- TEXTTABLE
- BITFIELD_TEXTTABLE
- RAT_FUNC
- SCALE_LINEAR
- SCALE_LINEAR_AND_TEXTTABLE
- SCALE_RAT_FUNC
- SCALE_RAT_FUNC_AND_TEXTTABLE
- TAB_NOINTP

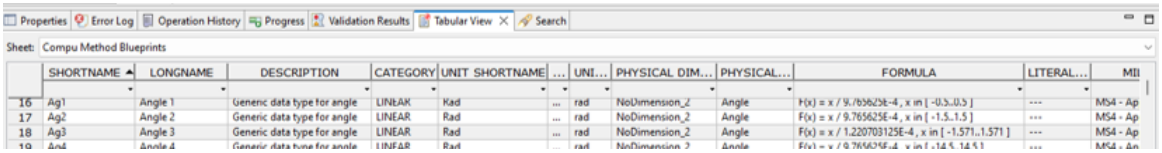


Figure 10.11: Tabular View of Computation Method Definitions Definitions

10.7.1 Computation Methods of type IDENTICAL

Computation Methods of type IDENTICAL are used to describe floating point representation of values according to the IEEE Standard 754 Floating Point Numbers. This means, that physical values are stored in the same number representation as they are measured or calculated. No conversion between physical numbers and internal number representation is necessary.

Shortname:
The Shortname of an IDENTICAL computation method reflects the relationship to the referenced physical Unit. It is built by concatenation of the Shortname of the referenced physical unit and the keyword Identcl. There shall be only one IDENTICAL computation method per physical unit.

Longname:
The Longname of a LINEAR computation method contains the Longname of the related physical unit and the word Identical.

Unit:
An IDENTICAL computation method can reference any of the defined physical units.

As IDENTICAL CompuMethods are a standardized representation of numbers, this can be used for all physical quantities.

CompuScale: Identical:
The Category information IDENTICAL is here sufficient to document that the implementation of these values shall be done in Floating Point format.

10.7.1.1 Rules and Recommendations for the usage of Float data

Here are few rule and recommendatios how float data can be used for Port Prototypes:

- Always use 1:1 Scaling: e.g. Internal representation = 10.1, use Physical Value 10.1Pa
- Only single precision calculations shall be done
- In case target ECU is known: Do not use float if the RAM/Stack resources are more critical than CPU load
- Float should always be used together with SI Unit as physical representation.
- Float is recommended if for one and the same signal either large range and low precision or small range and high precision is required. Examples:
  - Float is strictly recommended for Pressure ([Pa])
  - Float is strictly recommended for Injection Quantity ([kg])
- Float should not be used whenever integer precision is sufficient (e.g. Temperature ([K]))

While using float in Flat Instance Descriptors (SW Signals), few Rules/Recommendations are

- The compatibility rules of AUTOSAR meta model have to be fulfilled
- Any physical display representation can be used.

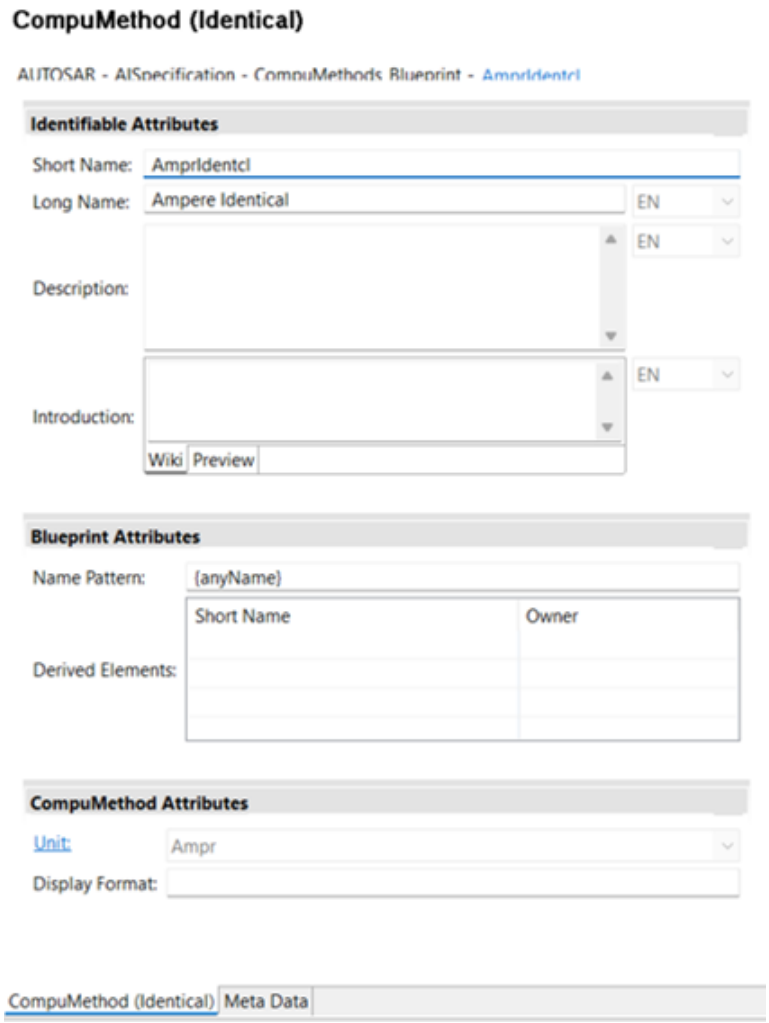


Figure 10.12: Editor View of IDENTICAL Computation Method Definitions

Example for the definition of a Computation Method of type IDENTICAL in AUTOSAR_MOD_AISpecification_CompuMethod_Blueprint.arxml:

```
  <ELEMENTS>
...
  <COMPU-METHOD>
    <SHORT-NAME NAME-PATTERN="{anyName}">AmprIdentcl</SHORT-NAME>
    <LONG-NAME> <L-4 L="EN">Ampere Identical</L-4></LONG-NAME>
    <CATEGORY>IDENTICAL</CATEGORY>
    <UNIT-REF BASE="Units" DEST="UNIT">Ampr</UNIT-REF>
  </COMPU-METHOD>
...
  </ELEMENTS>
```

10.7.2 Computation Methods of type LINEAR

Shortname:
The Shortname of a LINEAR computation method reflects the relationship to the physical dimension of the referenced Unit.

Longname:
The Longname of a LINEAR computation method shall describe the physical quantity of the related unit.

Unit:
A LINEAR computation method can reference any of the defined physical units. As LINEAR CompuMethods represent a fix factor and offset to convert physical values into internal values, this can be used for all physical quantities.

CompuScale: Formula:
The Formula section in the AI-Tool LINEAR computation method editor allows for exactly one CompuScale, with two values in compuNumerator and one value in compuDenominator for the coefficients (COMPU-RATIONAL-COEFFS) of a mathematical function description:
$F(x) = (a * x2 + b * x + c)/(d * x2 + e * x + f)$.

For LINEAR conversions the compuNumerator is usually set to 1 and thecoefficient for $x2$ d is for linear concersions set to 0 which reduces the formula to:
$F(x) = 1/(e * x + f)$.

In .arxml the order of the coefficients is exchanged to ascending order of exponents for x:
or
$F(x) = 1/(V1 + V2 * x)$,
where $e = V2$ and $f = V1$)

Example:
$F(x)[C] = 1/((40)[C] + (1)[C] * x)$

V1 and V2, here, can be called: V1 = Offset V2 = Factor (Resolution)

Example for the definition of a LINEAR computation method in AUTOSAR_MOD_AISpecification_CompuMethod_Blueprint.arxml:

```
  <ELEMENTS>
...
  <COMPU-METHOD>
    <SHORT-NAME>T4</SHORT-NAME>
    <LONG-NAME><L-4 L="EN">Temperature 4</L-4></LONG-NAME>
    <DESC>
      <L-2 L="EN">Generic data type for temperature</L-2>
    </DESC>
    <CATEGORY>LINEAR</CATEGORY>
    <UNIT-REF BASE="Units" DEST="UNIT">DegCgrd</UNIT-REF>
    <COMPU-PHYS-TO-INTERNAL>
      <COMPU-SCALES>
        <COMPU-SCALE>
```

```
          <LOWER-LIMIT INTERVAL-TYPE="CLOSED">-40</LOWER-LIMIT>
          <UPPER-LIMIT INTERVAL-TYPE="CLOSED">213</UPPER-LIMIT>
          <COMPU-RATIONAL-COEFFS>
            <COMPU-NUMERATOR>
              <V>40</V>
              <V>1</V>
            </COMPU-NUMERATOR>
            <COMPU-DENOMINATOR>
              <V>1</V>
            </COMPU-DENOMINATOR>
          </COMPU-RATIONAL-COEFFS>
        </COMPU-SCALE>
      </COMPU-SCALES>
    </COMPU-PHYS-TO-INTERNAL>
  </COMPU-METHOD>
...
  </ELEMENTS>
```

Limits:
The Limits section of a LINEAR computation method contains only one interval. The interval shall be defined as CLOSED.

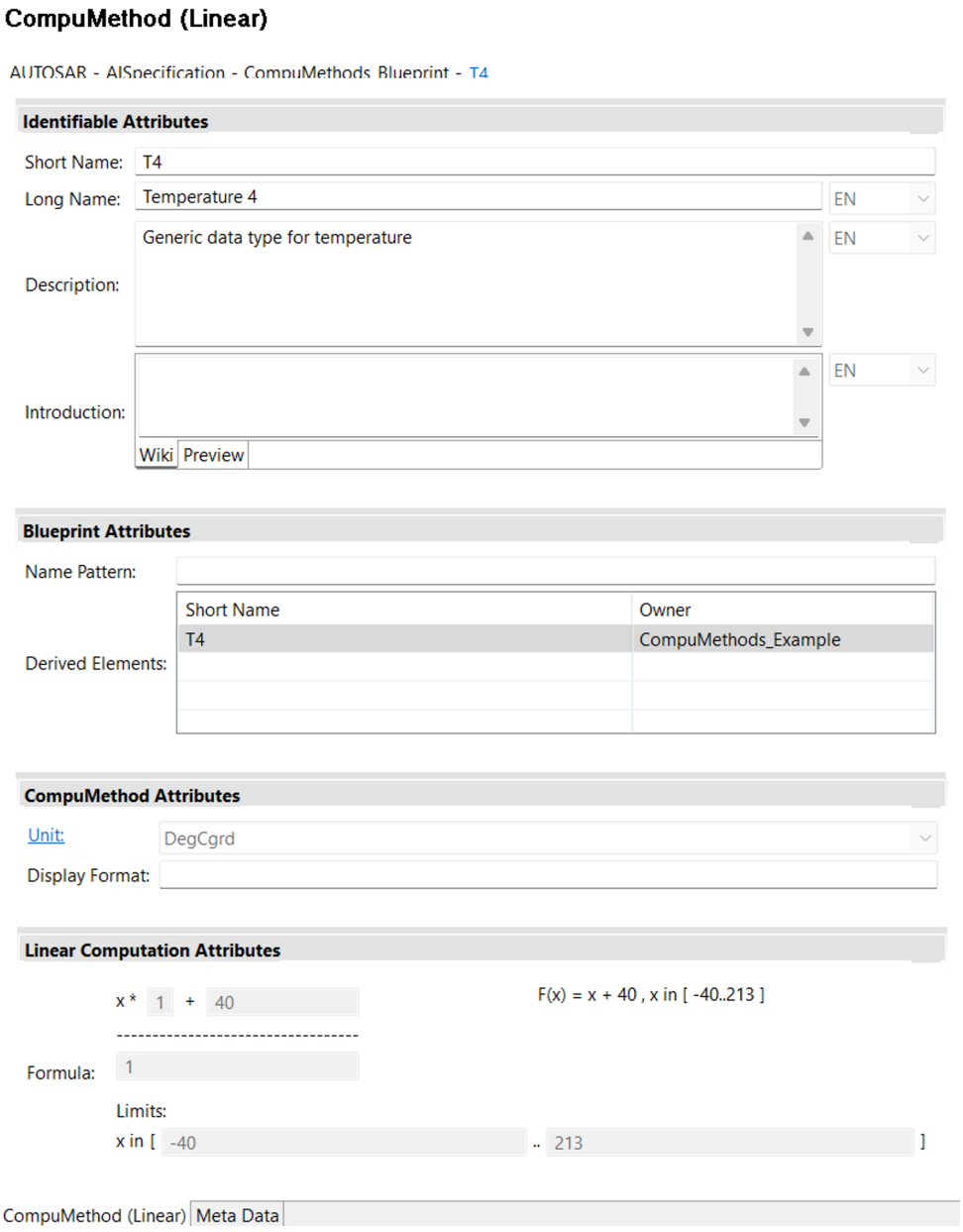


Figure 10.13: Editor View of LINEAR Computation Method Definitions

10.7.3 Computation Methods of type TEXTTABLE

Computation Methods of type TEXTTABLE are used to describe a textual representation of values in a calibration tool or a dashboard. This is usually utilized to implement enum data types, logical information or fields of logical information. Often, such representations are used to display the results of state machines (states) or logical calculations. Currently, there is no possibility to define so called Boolean data types explicitly. That's why the TEXTTABLE conversion is applied, here too.

Shortname:
The Shortname of a TEXTTABLE computation method reflects a relationship to the

behavior of the state machine or of the logical decision. It is built by concatenation of Keywords to describe the use-case of the state machine. There must not be any relationship to any physical unit.

Longname:
The Longname of a TEXTTABLE computation method is the spelled-out wording of the Shortname. It reflects a relationship to the behavior of the state machine or of the logical decision.

Unit:
A TEXTTABLE computation method can reference the physical unit NoUnit, only. There is no physical interpretation of any of the values in the interval possible.

CompuScale: Texttable Conversion:
The literals of the enumeration type are encoded in one COMPU-SCALE for each literal. Descriptions of the literals are not parsed into multiple elements. Each CompuScale contains the definition of the value range (limits), a description, and the literal (text) which shall be displayed instead of the value. The literals are built by concatenation of keywords. The description of the literal shall explain meaning of the text.

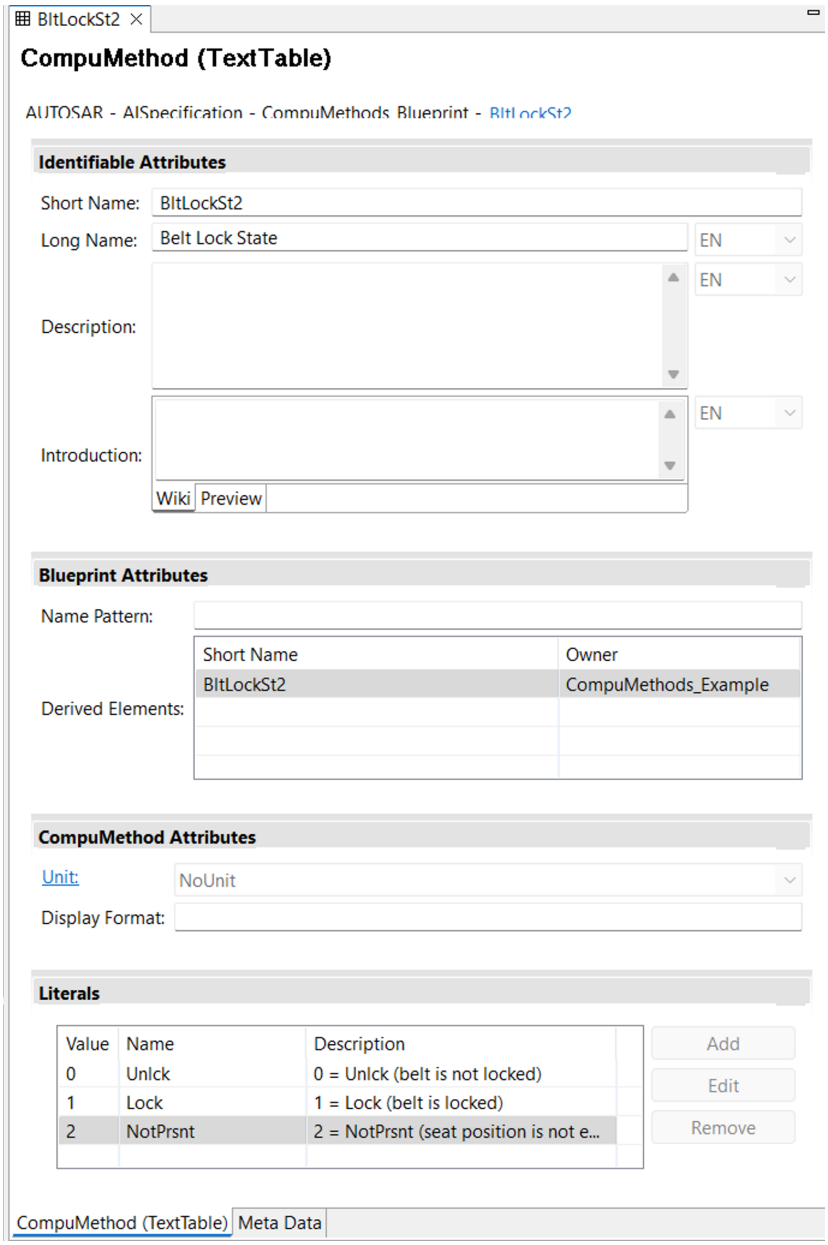


Figure 10.14: Editor View of Computation Method Type TEXTTABLE Definitions

Example for the definition of a Computation Method of type TEXTTABLE in AUTOSAR_MOD_AISpecification_CompuMethod_Blueprint.arxml:

```
  <ELEMENTS>
...
  <COMPU-METHOD>
   <SHORT-NAME>BltLockSt2</SHORT-NAME>
    <LONG-NAME>
      <L-4 L="EN">Belt Lock State</L-4>
    </LONG-NAME>
    <CATEGORY>TEXTTABLE</CATEGORY>
    <UNIT-REF BASE="Units" DEST="UNIT">NoUnit</UNIT-REF>
    <COMPU-INTERNAL-TO-PHYS>
      <COMPU-SCALES>
```

```
        <COMPU-SCALE>
          <DESC>
            <L-2 L="EN">0 = Unlck (belt is not locked)</L-2>
          </DESC>
          <LOWER-LIMIT INTERVAL-TYPE="CLOSED">0</LOWER-LIMIT>
          <UPPER-LIMIT INTERVAL-TYPE="CLOSED">0</UPPER-LIMIT>
          <COMPU-CONST>
            <VT>Unlck</VT>
          </COMPU-CONST>
        </COMPU-SCALE>
        <COMPU-SCALE>
          <DESC>
            <L-2 L="EN">1 = Lock (belt is locked)</L-2>
          </DESC>
          <LOWER-LIMIT INTERVAL-TYPE="CLOSED">1</LOWER-LIMIT>
          <UPPER-LIMIT INTERVAL-TYPE="CLOSED">1</UPPER-LIMIT>
          <COMPU-CONST>
            <VT>Lock</VT>
          </COMPU-CONST>
        </COMPU-SCALE>
        <COMPU-SCALE>
          <DESC>
            <L-2 L="EN">2 = NotPrsnt (seat position is not equipped or not
                fixed)</L-2>
          </DESC>
          <LOWER-LIMIT INTERVAL-TYPE="CLOSED">2</LOWER-LIMIT>
          <UPPER-LIMIT INTERVAL-TYPE="CLOSED">2</UPPER-LIMIT>
          <COMPU-CONST>
            <VT>NotPrsnt</VT>
          </COMPU-CONST>
        </COMPU-SCALE>
      </COMPU-SCALES>
    </COMPU-INTERNAL-TO-PHYS>
  </COMPU-METHOD>
...
  </ELEMENTS>
```

10.7.4 Computation Methods of type BITFIELD_TEXTTABLE

According to the AUTOSAR Software Component Template [4]: Similar to TEXTTABLE but for bit fields. BITFIELD_TEXTTABLE is derived from TEXTTABLE. The main difference is that TEXTTABLE results to a single value while BITFIELD_TEXTTABLE results to a concatenated value set. In difference to all other computational methods every CompuScale will be applied including the bit mask specified in mask. Therefore, it is allowed for this type of CompuMethod, that CompuScales overlap. To calculate the string reverse to a value, the string has to be split and the according value for each substring has to be summed up. The sum is finally transmitted. The processing has to be done in order of the CompuScale elements.

Currently, the AI-Tool does not provide an ability to define Computation Methods of type BITFIELD_TEXTTABLE. In case such Computation Methods should be maintained via

AI-Tool, the structure of the .ai-File would have to be enhanced and the tool has to be adapted.

10.7.5 Computation Methods of type RAT_FUNC

According to the AUTOSAR Software Component Template [4]: The rational function type is similar to the linear type without the restrictions for the compuNumerators and compuDenominators. It can have as many v elements as needed for the rational function. The sequence of the values v carries the information for the exponents, that means the first v is the coefficient for x0, the second v is the coefficient for x1, etc. With this sequence the values of the exponents can be entirely represented. A rational function is only applicable for conversions in the direction that it is defined for, i.e. the automatic calculation of the inverse function is not supported by the MCD system.

Currently, the AI-Tool does not provide an ability to define Computation Methods of type RAT_FUNC. In case such Computation Methods should be maintained via AI-Tool, the structure of the .ai-File would have to be enhanced and the tool has to be adapted.

10.7.6 Computation Methods of type SCALE_LINEAR

According to the AUTOSAR Software Component Template [4]: (This shall be) Used for a piecewise linear conversion. Properties depend on the used scale function. More than one compuScale can be defined. The scales shall each provide lowerLimit and upperLimit definitions. For details see definition of LINEAR [10.7.2].

Currently, the AI-Tool does not provide an ability to define Computation Methods of type SCALE_LINEAR. In case such Computation Methods should be maintained via AI-Tool, the structure of the .ai-File would have to be enhanced and the tool has to be adapted.

10.7.7 Computation Methods of type SCALE_LINEAR_AND_TEXTTABLE

According to the AUTOSAR Software Component Template [4]: Used for piecewise definition of one or more linear and several texttable scales. Properties depend on the used scale function. For details see definition of SCALE_LINEAR and TEXTTABLE. The scales shall each provide lowerLimit and upperLimit definitions.

Currently, the AI-Tool does not provide an ability to define Computation Methods of type SCALE_RAT_FUNC. In case such Computation Methods should be maintained via AI-Tool, the structure of the .ai-File would have to be enhanced and the tool has to be adapted.

10.7.8 Computation Methods of type SCALE_RAT_FUNC

According to the AUTOSAR Software Component Template [4]: The rational function type is similar to the linear type without the restrictions for the compuNumerators and compuDenominators. It can have as many v elements as needed for the rational function. The sequence of the values carries the information for the exponents, that means the first value is the coefficient for x0, the second v is the coefficient for x1, etc. With this sequence the values of the exponents can be entirely represented. A rational function is only applicable for conversions in the direction that it is defined for, i.e. the automatic calculation of the inverse function is not supported by the MCD system.

Currently, the AI-Tool does not provide an ability to define Computation Methods of type SCALE_RAT_FUNC. In case such Computation Methods should be maintained via AI-Tool, the structure of the .ai-File would have to be enhanced and the tool has to be adapted.

10.7.9 Computation Methods of type SCALE_RATIONAL_AND_TEXTTABLE

According to the AUTOSAR Software Component Template [4]: Used for piecewise definition of one rational and several texttable scales. Properties depend on the used scale function. For details see definition of RAT_FUNC and TEXTTABLE. The scales shall each provide lowerLimit and upperLimit definitions.

Currently, the AI-Tool does not provide an ability to define Computation Methods of type SCALE_RATIONAL_AND_TEXTTABLE. In case such Computation Methods should be maintained via AI-Tool, the structure of the .ai-File would have to be enhanced and the tool has to be adapted.

10.7.10 Computation Methods of type TAB_NOINTP

According to the AUTOSAR Software Component Template [4]: Similar to TEXTTABLE, but for numerical values. The values per scale are defined in CompuConst.

Currently, the AI-Tool does not provide an ability to define Computation Methods of type TAB_NOINTP. In case such Computation Methods should be maintained via AI-Tool, the structure of the .ai-File would have to be enhanced and the tool has to be adapted.

10.8 Port Interfaces

Interface definitions collect information to specify the properties of Ports. The interface definition typify the Ports referencing them. Interfaces can be referenced by any Ports of any SW Component. So, the same interface can be used to typify different Ports of different SW Components to ensure the compatibility of these Ports when they are connected.

Since PortInterfaces are designed to support reusability, it is recommended to reuse already defined PortInterfaces for PortPrototypes with the same kind of information to be transported or the same function to be used.

There are 2 ARPackages available in the AI-Tool Explorer for definition of interfaces:

- AISpecification_PortInterfaces_Blueprint
- AISpecification_PortInterfaces_Example

According to the AUTOSAR Software Component Template [4], the elements of a PortInterface definition are:

- Common Elements
  - Shortname
  - Longname
  - Description
  - Introduction
- Client-Server Interface specific Elements
  - Operations
  - Arguments
- Sender-Receiver Interface specific Elements
  - Data Elements
- Blueprint specific Elements
  - Blueprints: Name Pattern
  - Blueprints: Derived Elements
- Example specific Elements
  - Derived from

Additionally, there is Milestone, Life Cycle State and Life Cycle and UseInstead information available for each Interface definition.

10.8.1 Client-Server Interfaces

Client-server interfaces represent procedural interfaces of a SW Component. Ports typified by such interfaces provide access (API) to executable entities which can be used by other SW Components.

Please find additional information in the AUTOSAR Software Component Template [4], Chapter Client server Communication.

Shortname:
The Shortname of a Client-Server interface shall reflect the physical interpretation of the concerned data or the operational content of the operations. Usually Shortnames are derived from architecture descriptions in the Longnames or the descriptions of the AR Elements.

The Shortname of an interface is defined as a referrable identifier and must therefore respect Rules for identifiers. It is built by concatenating keywords and an index as defined by AUTOSAR_MOD_AISpecification_KeywordSet_Blueprint.arxml. The naming rules are defined at AUTOSAR Software Component Modeling Guide [6].

Longname:
The Longname of a client-server interface shall reflect the physical interpretation of the concerned data or the operational content of the operations. Usually, Longname and Description which are short architectural summaries of the content of the AR Elements, are the basis for the derivation of Shortnames.

The Longname of a client-server interface is the spelled-out wording of the Shortname, including the index. It is a plain text which shall also not contain any special characters. The naming rules are defined at AUTOSAR Software Component Modeling Guide [6].

Description:
The Description of a client-server interface shall contain a very short explanation of the cases, where the operations shall be used or how the values can be interpreted in a physical world.

Usually, Longname and Description which are short architectural summaries of the content of the AR Elements, are the basis for the derivation of Shortnames.

Introduction:
The Introduction of a client-server interface can give additional information about usage or restriction of usage of the interface. If necessary, additional information concerning use cases or application rules can be given, here.

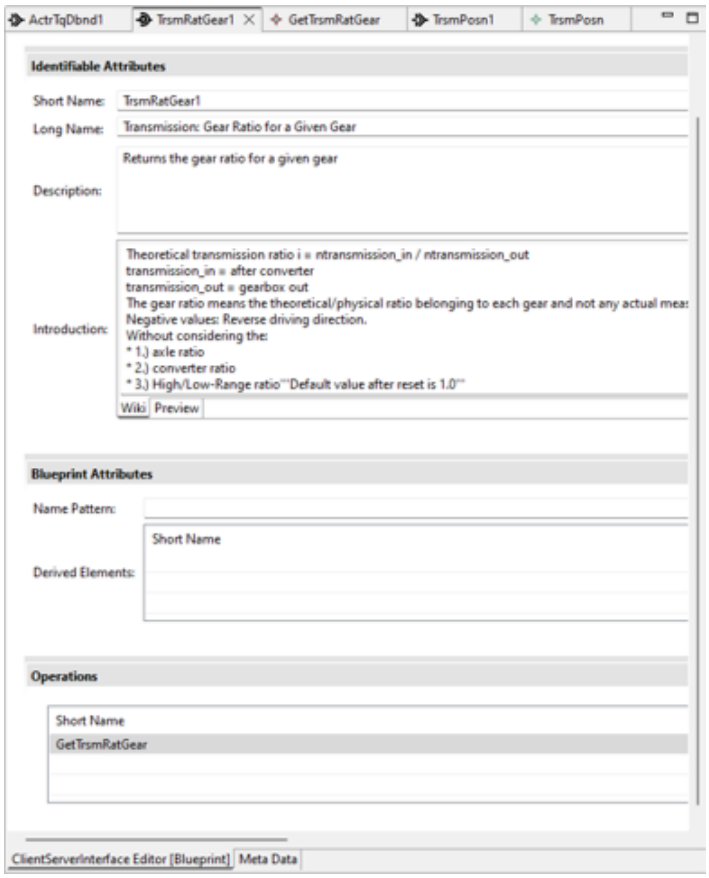


Figure 10.15: Editor view of Client-Server Interface Definition

Operations:
The Operations of a client-server interface describe, how values shall be modified when the client-server Port is called.

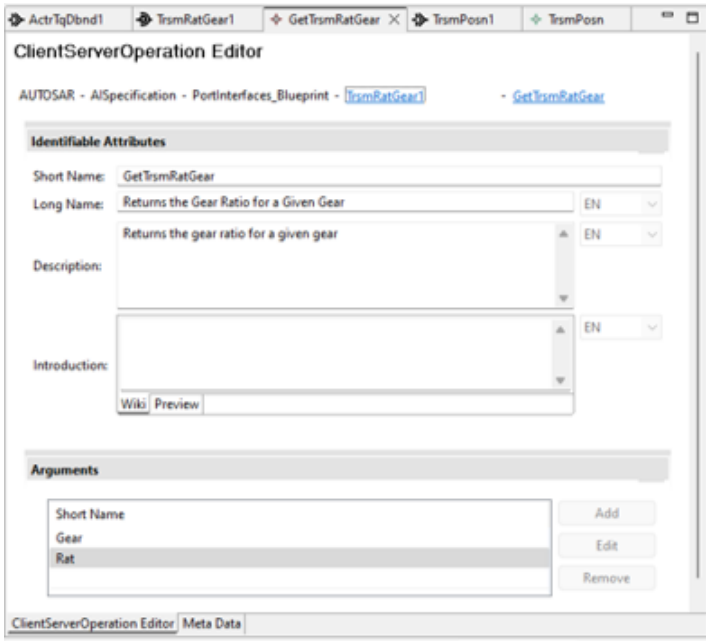


Figure 10.16: Editor view of Client-Server Interface Operation Definition

As depicted in the Figures (10.15, 10.17), a ClientServerInterface is composed of ClientServerOperations. The same ClientServerOperation cannot be reused in the context of different ClientServerInterfaces. A ClientServerOperation may contain 0 to many ArgumentDataPrototypes. The latter may be

- passed to the operation
- passed to and returned from the operation
- Returned from the operation.

Example: In the above screen shot, a client/server interface "TrsmRatGear1" is defined to return the gear ratio for a given gear, the client requests the operation 'GetTrsmRatGear' with input argument 'Gear'. The function call returns the output argument 'Rat'.

In AUTOSAR CP (and AP), client-server operations are defined as:
Operations declared within the scope of a client/server interface.

For AUTOSAR FO, the following definition is applied:
A remote procedure call declared within the scope of the current interface.

Arguments:
The Arguments of a client-server interface operation can be seen as own interfaces of the operation. When the operation of the interface is called the arguments transport the concerned information which need to be exchanged between the Client (caller) and the Server (callee) SW Component. Arguments are typified by data types and have names and a direction. The Direction defines if the Argument is an input or an output or input and output of the operation.

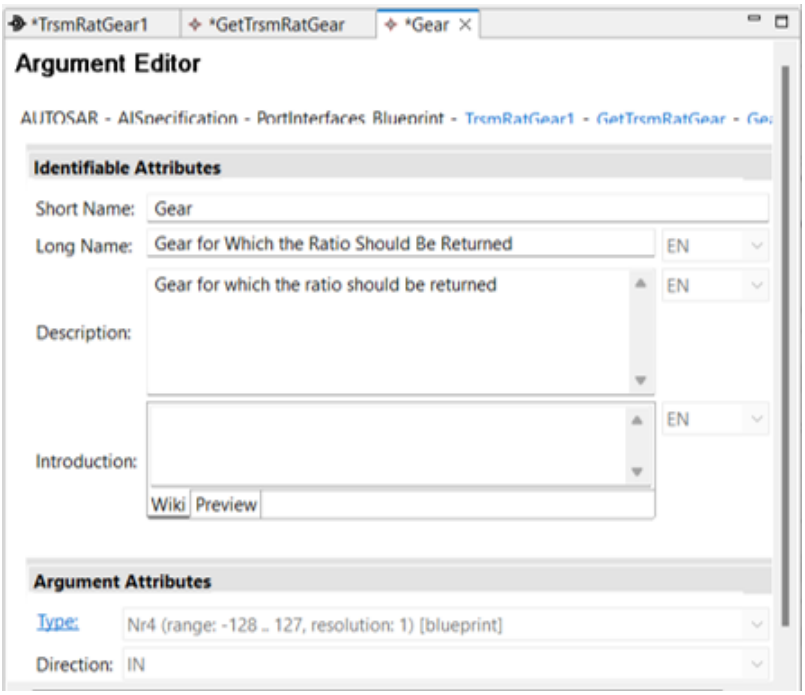


Figure 10.17: Editor view of Client-Server Operation Argument Definition

Example for the definition of a Client-Server Interface in AUTOSAR_MOD_AISpecification_PortInterfaces_Blueprint.arxml:

```
  <ELEMENTS>
...
  <CLIENT-SERVER-INTERFACE>
    <SHORT-NAME>TrsmRatGear1</SHORT-NAME>
    <LONG-NAME>
      <L-4 L="EN">Transmission: Gear Ratio for a Given Gear</L-4>
    </LONG-NAME>
    <DESC>
      <L-2 L="EN">Returns the gear ratio for a given gear</L-2>
    </DESC>
    <INTRODUCTION>
      <P>
        <L-1 L="EN">Theoretical transmission ratio i = ntransmission_in
         / ntransmission_out transmission_in = after converter
         transmission_out = gearbox out
         The gear ratio means the theoretical/physical ratio
         belonging to each gear and not any actual measured
         value (proposal for Continuously Variable Transmission
         (CVT): if there is a wide range for gear states, this
         value could deliver a theoretical value).
         Negative values: Reverse driving direction.
         Without considering the:
        </L-1>
       </P>
          <LIST TYPE="UNNUMBER">
            <ITEM><P><L-1 L="EN">1.) axle ratio </L-1></P></ITEM>
            <ITEM><P><L-1 L="EN">2.) converter ratio </L-1></P></ITEM>
            <ITEM><P><L-1 L="EN">3.) High/Low-Range ratio'Default␣value␣
               after␣reset␣is␣1.0'</L-1></P>
```

```
            </ITEM>
          </LIST>
    </INTRODUCTION>
    <IS-SERVICE>false</IS-SERVICE>
    <OPERATIONS>
      <CLIENT-SERVER-OPERATION>
        <SHORT-NAME>GetTrsmRatGear</SHORT-NAME>
        <LONG-NAME>
           <L-4 L="EN">Returns the Gear Ratio for a Given Gear</L-4>
        </LONG-NAME>
        <DESC>
          <L-2 L="EN">Returns the gear ratio for a given gear</L-2>
        </DESC>
        <ARGUMENTS>
          <ARGUMENT-DATA-PROTOTYPE>
            <SHORT-NAME>Gear</SHORT-NAME>
            <LONG-NAME>
              <L-4 L="EN">Gear for Which the Ratio Should Be Returned</L-4>
            </LONG-NAME>
            <DESC>
              <L-2 L="EN">Gear for which the ratio should be returned</L-2>
            </DESC>
            <TYPE-TREF BASE="ApplicationDataTypes"
              DEST="APPLICATION-PRIMITIVE-DATA-TYPE">Nr4
            </TYPE-TREF>
            <DIRECTION>IN</DIRECTION>
          </ARGUMENT-DATA-PROTOTYPE>
          <ARGUMENT-DATA-PROTOTYPE>
            <SHORT-NAME>Rat</SHORT-NAME>
            <LONG-NAME>
              <L-4 L="EN">Gear Ratio of Given Gear</L-4>
            </LONG-NAME>
            <DESC>
              <L-2 L="EN">Gear ratio of given gear</L-2>
            </DESC>
            <TYPE-TREF BASE="ApplicationDataTypes"
              DEST="APPLICATION-PRIMITIVE-DATA-TYPE">Fac1
            </TYPE-TREF>
            <DIRECTION>OUT</DIRECTION>
          </ARGUMENT-DATA-PROTOTYPE>
        </ARGUMENTS>
      </CLIENT-SERVER-OPERATION>
    </OPERATIONS>
  </CLIENT-SERVER-INTERFACE>

...
  </ELEMENTS>
```

10.8.2 Sender-Receiver Interfaces

Shortname:
The Shortname of a sender-receiver interface shall reflect the physical interpretation

of the concerned data. Sender-Receiver interfaces are used to define signals or signal collections which are exchanged between Ports of different SW Components.

Usually Shortnames are derived from architecture descriptions in the Longnames or the descriptions of the AR Elements.

The Shortname of an interface is defined as a referrable identifier and must therefore respect Rules for identifiers. It is built by concatenating keywords and an index as defined by AUTOSAR_MOD_AISpecification_KeywordSet_Blueprint.arxml. The naming rules are defined at AUTOSAR Software Component Modeling Guide [6].

Longname:
The Longname of a sender-receiver interface shall reflect the physical interpretation of the data exchanged via the data elements. Usually, Longname and Description which are short architectural summaries of the content of the AR Elements, are the basis for the derivation of Shortnames.

The Longname of a sender-receiver interface is the spelled-out wording of the Shortname, including the index. It is a plain text which shall also not contain any special characters. The naming rules are defined at AUTOSAR Software Component Modeling Guide [6].

Description:
The Description of a sender-receiver interface shall contain a very short explanation of the information exchanged via the data elements. Usually, Longname and Description which are short architectural summaries of the content of the AR Elements, are the basis for the derivation of Shortnames.

Introduction:
The Introduction of a sender-receiver interface can give additional information about usage or restriction of usage of the interface. If necessary, additional information concerning use cases or application rules can be given, here.

Data Elements:
As one interface can contain more than one elements (signals), it is necessary to define the inner structure of the interface. This is done via the list of contained signals, called Data Elements [10.8.2.1].

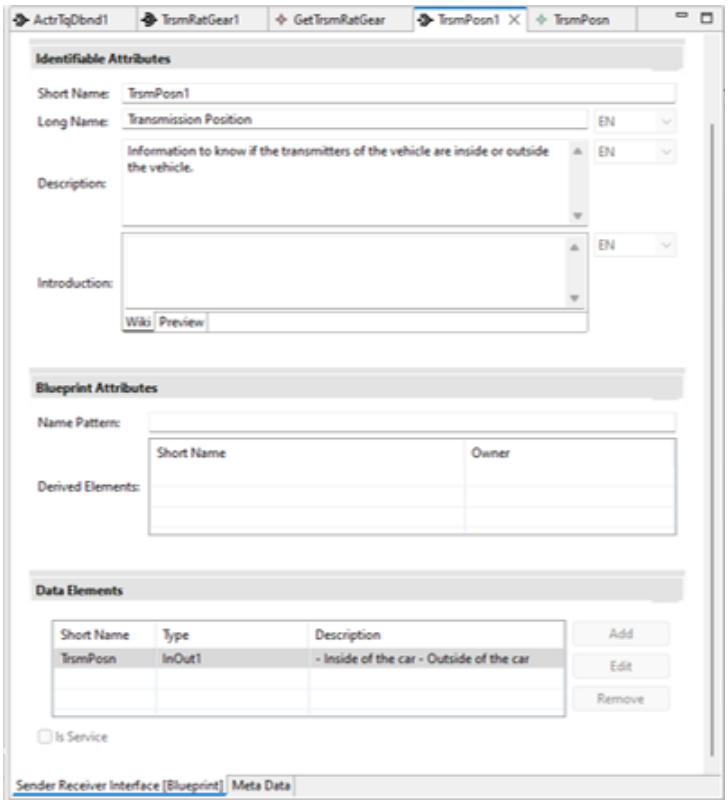
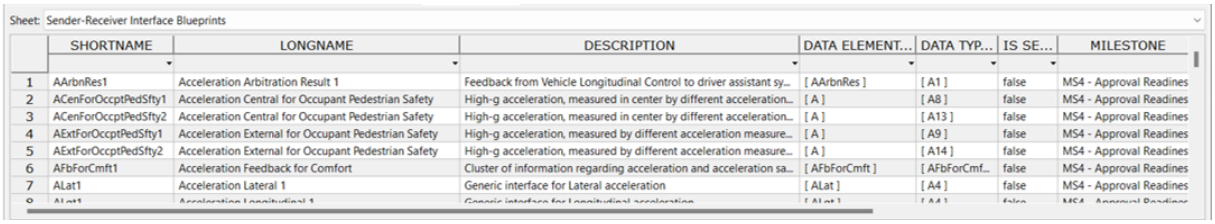


Figure 10.18: Editor view of Sender-Receiver Interface Definition



	SHORTNAME	LONGNAME	DESCRIPTION	DATA ELEMENT...	DATA TYP...	IS SE...	MILESTONE
1	AArbnRes1	Acceleration Arbitration Result 1	Feedback from Vehicle Longitudinal Control to driver assistant sy...	[AArbnRes]	[A1]	false	MS4 - Approval Readines
2	ACenForOccptPedSfty1	Acceleration Central for Occupant Pedestrian Safety	High-g acceleration, measured in center by different acceleration...	[A]	[A8]	false	MS4 - Approval Readines
3	ACenForOccptPedSfty2	Acceleration Central for Occupant Pedestrian Safety	High-g acceleration, measured in center by different acceleration...	[A]	[A13]	false	MS4 - Approval Readines
4	AExtForOccptPedSfty1	Acceleration External for Occupant Pedestrian Safety	High-g acceleration, measured by different acceleration measure...	[A]	[A9]	false	MS4 - Approval Readines
5	AExtForOccptPedSfty2	Acceleration External for Occupant Pedestrian Safety	High-g acceleration, measured by different acceleration measure...	[A]	[A14]	false	MS4 - Approval Readines
6	AFbForCmft1	Acceleration Feedback for Comfort	Cluster of information regarding acceleration and acceleration sa...	[AFbForCmft]	[AFbForCmf...	false	MS4 - Approval Readines
7	ALat1	Acceleration Lateral 1	Generic interface for Lateral acceleration	[ALat]	[A4]	false	MS4 - Approval Readines

Figure 10.19: Sender-Receiver Interfaces in Tabular View

10.8.2.1 Data Elements

A data element represents the properties of one element of a sender-receiver interface. One sender-receiver interface can contain more than one data elements. For each of them the definition contains again the elements Shortname, Longname, Description, and Introduction. Additionally, for each single data element, a Data Type shall be specified.

(Data) Type:
The data type of a data element of an Interface is defined via a reference to an existing ApplicationDataType [10.3],[10.4], [10.5], [10.6] . The reference is represented by the Shortname of the ApplicationDataType as defined in:

- AUTOSAR_MOD_AISpecification_ApplicationDataType_Blueprint.arxml

- AUTOSAR_MOD_AISpecificationExamples.arxml
- in AI-Tool: ApplicationDataTypes_Blueprint-Package
- in AI-Tool: ApplicationDataTypes_Example-Package.

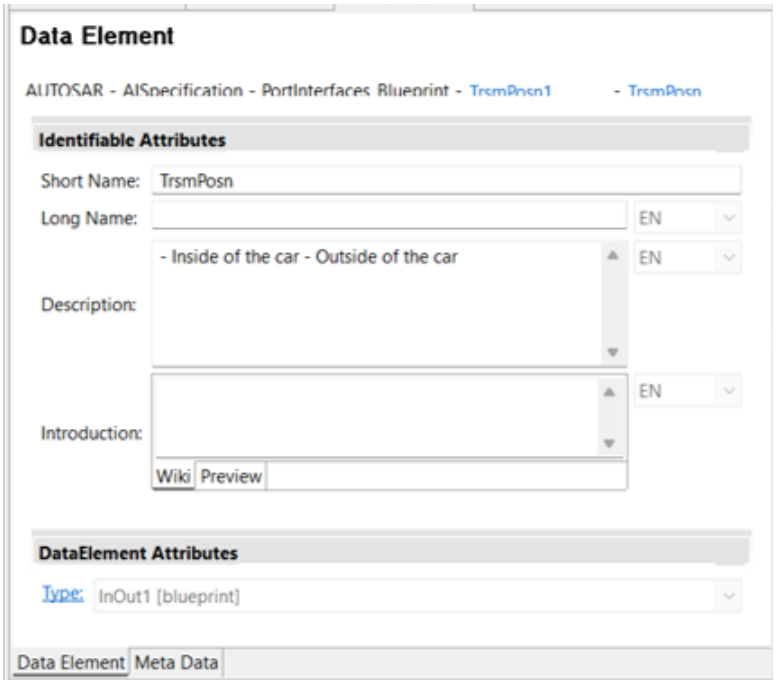


Figure 10.20: Editor view of Data Element Definitions in Sender-Receiver Interfaces

10.9 Data Constraints

The idea of Application Interfaces is to standardize SW Component interfaces on a very abstract level and completely independent of SW implementation. For the standardization on this very abstract level, the physical properties of the signals which are exchanged between different components are more important than implementation details. Therefore, the standardized interfaces are to be seen more on system level than on software and implementation level. However, a system architectural view requires also restrictions to useful physical value ranges. For this, the definition of Data Constraints is used.

There are 2 ARPackages available in the AI-Tool Explorer for the definition of Data Constraints:

- AISpecification_DataConstrs_Blueprint
- AISpecification_DataConstrs_Example

According to the AUTOSAR Software Component Template [4], the elements of the DataConstrs definition are:

- Common Elements

  - Shortname
  - Longname
  - Description
  - Introduction
  - Data Constraints
  - Physical Data Constraints
  - Internal Data Constraints
- Blueprint specific Elements
  - Blueprints: Name Pattern
  - Blueprints: Derived Elements
- Example specific Elements
  - Derived from

Additionally, there is Milestone, Life Cycle State and Life Cycle, and UseInstead information available for each Application Continuous Data Type definition.

Shortname:
The Shortname of a data constraint definition is defined as a referrable identifier and must therefore respect rules for identifiers. It is built by concatenating the keyword for the corresponding physical quantity and an index as defined by AUTOSAR_MOD_AISpecification_KeywordSet_Blueprint.arxml. Usually this Shortname is inherited from the name of the corresponding ApplicationDataType. The naming rules are defined at AUTOSAR Software Component Modeling Guide [6].

Longname:
Usually, Longname and Description which are short architectural summaries of the content of the AR Elements, are the basis for the derivation of Shortnames. The Longname of a data constraints definition is the spelled-out wording of the Shortname, including the index. It is a plain text which shall also not contain any special characters. The naming rules are defined at AUTOSAR Software Component Modeling Guide [6].

Description:
The Description of a data constraints definition shall contain a very short explanation of the cases, where the data type shall be used or how the values can be interpreted in a physical world. Usually, Longname and Description which are short architectural summaries of the content of the AR Elements, are the basis for the derivation of Shortnames.

Introduction:
The Introduction of a data constraints definition is currently not used. If necessary, additional information concerning use cases or application rules can be given, here.

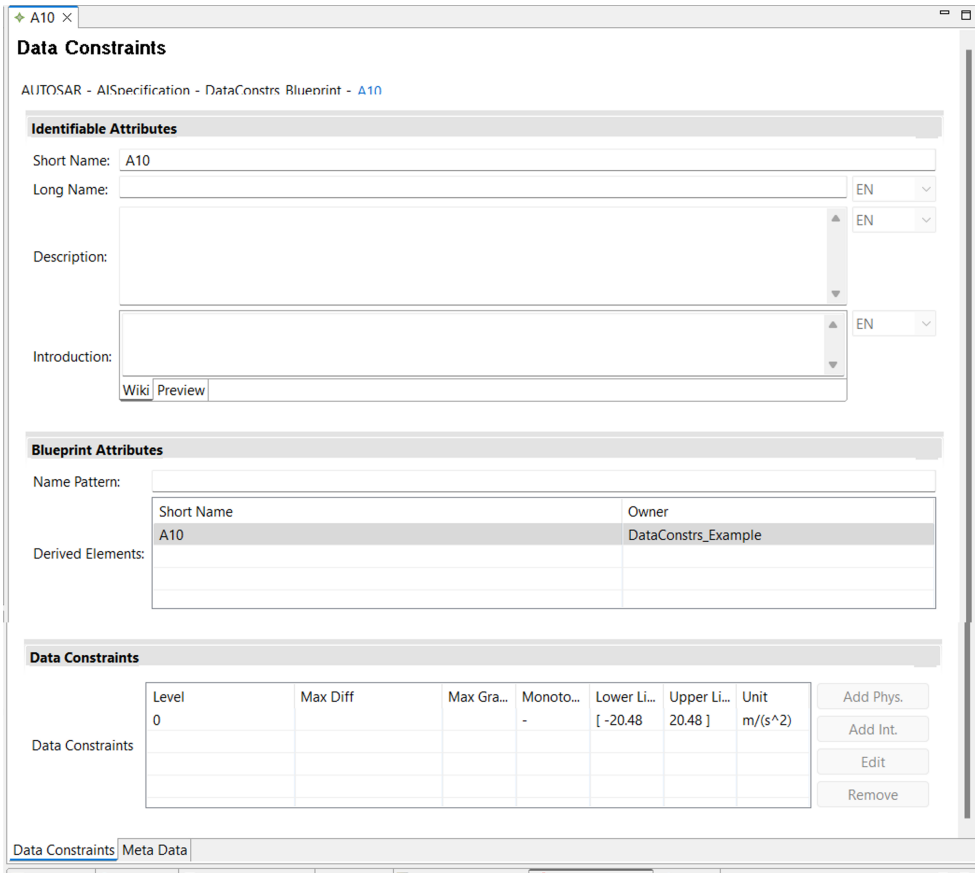


Figure 10.21: Editor view of Data Constraint Definitions

Data Constraints:
In this section all necessary data constraint rule definitions are collected. If more than one data constraint rule of the same type is available the definitions are rated by their Level information. According to AUTOSAR Software Component Template [4] the Levels are defined as:

- 0: This represents so called hard limits. They shall always be specified.
- 1: This represents so called soft limits. Soft limits may be violated after confirmation by the user of an MCD-System.

Other values may exist, but the semantics is outside the AUTOSAR scope.

To avoid too complex dependencies between several limitations, for the definition of Application Interfaces, we restrict the number of data constraint definitions to one per data type.

Data constraints can be defined as Internal or Physical value representation. For the definition of physical data constraint rules, a reference to a physical unit is mandatory. Internal value representation cannot be standardized on VFB level because this depends on the implementation method. This is not in the focus of Application Interface definitions.

Unit:
The reference to the physical unit definition is represented by the Shortname of the physical unit. The AI-Tool provides a selection field to select the referenced unit.

Max Diff:
Maximum difference that is permitted between two consecutive values if the constraint is applied to an axis. ([4])

Max Gradient:
This element specifies the maximum slope that may be used in curves and maps. ([4])

Monotony:
This specifies the monotony constraints on the data object. Note that this applies only to curves and maps.

This element specifies the monotony characteristics of the current internal or physical limits. If the element has no contents or if it is omitted, "no Monotony" is the default content. ([4]) In the AI-Tool, the monotony properties can be selected from a selection list.

Lower Limit:
This specifies the lower limit of the constraint. ([4]) Enter here a value which is a useful limitation of the physical value range.

Upper Limit:
This specifies the upper limit of the constraint. ([4]) Enter here a value which is a useful limitation of the physical value range.

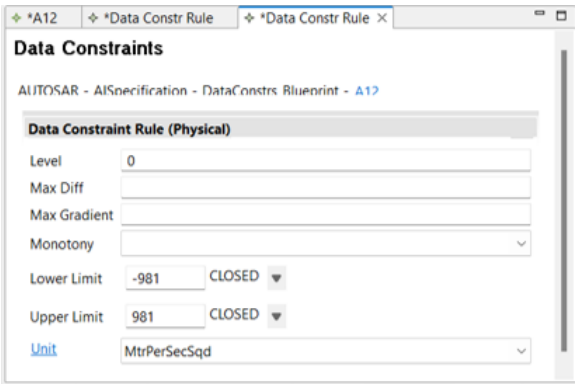


Figure 10.22: Editor view of Physical Data Constraint Definitions

	SHORTN...	...	...	DATA CO...	DATA CO...	DATA CON...	UNIT SHO...	UNIT LONG...	UNIT DISPL...	PHYSICAL ...	PHYSICAL...	MILESTONE	LIFE
1	A1			[-18]	[18]	[[0 | | | ---]]	[MtrPerSecSq...	[Meter Per Squar...	[m/(s^2)]	[Len1TiNeg2]	[Acceleration]	MS4 - Approval R...	---
2	A10			[-20.48]	[20.48]	[[0 | | | ---]]	[MtrPerSecSq...	[Meter Per Squar...	[m/(s^2)]	[Len1TiNeg2]	[Acceleration]	MS4 - Approval R...	---
3	A11			[-99]	[99]	[[0 | | | ---]]	[MtrPerSecSq...	[Meter Per Squar...	[m/(s^2)]	[Len1TiNeg2]	[Acceleration]	MS4 - Approval R...	---
4	A12			[-981]	[981]	[[0 | | | ---]]	[MtrPerSecSq...	[Meter Per Squar...	[m/(s^2)]	[Len1TiNeg2]	[Acceleration]	MS4 - Approval R...	---
5	A13			[-1962]	[1962]	[[0 | | | ---]]	[MtrPerSecSq...	[Meter Per Squar...	[m/(s^2)]	[Len1TiNeg2]	[Acceleration]	MS4 - Approval R...	---

Figure 10.23: Tabular view of Data Constraint Definitions

Example for the definition of a DataConstrs definition in AUTOSAR_MOD_AISpecification_DataConstr_Blueprint.arxml:

```
  <ELEMENTS>
...
  <DATA-CONSTR>
    <SHORT-NAME>A12</SHORT-NAME>
    <DATA-CONSTR-RULES>
      <DATA-CONSTR-RULE>
        <PHYS-CONSTRS>
          <LOWER-LIMIT INTERVAL-TYPE="CLOSED">-981</LOWER-LIMIT>
          <UPPER-LIMIT INTERVAL-TYPE="CLOSED">981</UPPER-LIMIT>
          <UNIT-REF BASE="Units" DEST="UNIT">MtrPerSecSqd</UNIT-REF>
        </PHYS-CONSTRS>
      </DATA-CONSTR-RULE>
    </DATA-CONSTR-RULES>
  </DATA-CONSTR>
...
  </ELEMENTS>
```

For Float data types, the generic DataConstrs for [-INF,+INF] is defined

```
<DATA-CONSTR>
  <SHORT-NAME NAME-PATTERN="{anyName}">FloatDataRange</SHORT-NAME> <!-- the
      name will be chosen based on naming rules -->
  <DATA-CONSTR-RULES>
    <DATA-CONSTR-RULE>
      <PHYS-CONSTRS>
        <LOWER-LIMIT INTERVAL-TYPE="CLOSED">-INF</LOWER-LIMIT>
        <UPPER-LIMIT INTERVAL-TYPE="CLOSED">+INF</UPPER-LIMIT>
        <!-- <UNIT-REF BASE="Units" DEST="UNIT">MtrPerSecSqd</UNIT-REF> (no
            unit required) -->
      </PHYS-CONSTRS>
    </DATA-CONSTR-RULE>
  </DATA-CONSTR-RULES>
</DATA-CONSTR>
```

10.10 Keywords

As basis for the definition of Shortnames for Application Interface definition elements there is a collection of so called Keywords available. These definitions are commonly agreed in AUTOSAR and only defined as Blueprints because there are no derived Example elements necessary.

Keywords are collected and maintained in the AI-Tool Explorer in the ARPackage:

- AISpecification_KeywordSet_Blueprint

According to the AUTOSAR Software Component Template [4], the elements of the Keyword definition are:

- Shortname
- Longname
- Description

- Introduction
- Abbreviation
- Classification

Additionally, there is Milestone, Life Cycle State and Life Cycle, and UseInstead information available for each Keyword definition.

Shortname:
The Shortname of a Keyword is defined as a referrable identifier and must therefore respect rules for identifiers. It is built according to the keyword rules as defined in the AUTOSAR_CP_TR_SWCModelingGuide ([6]). This means, according to these rules that the Shortname of a keyword must be unique. For some use cases it was necessary to have the same abbreviation for different meanings. For these use cases the Shortname can differ from the intended Abbreviation.

Longname:
The Longname of a Keyword is the spelled-out wording of the Keyword. It is a plain text which shall also not contain any special characters. Usually the Longname exists first and is the basis for the derivation of the Shortname. Even if often used, it is not recommended to use abbreviations as Longnames (e.g.,bad example at table 10.2).

Description:
The Description of a Keyword shall contain a very short explanation of the information which shall be expressed, when the Keyword is used to build the names of an element of Application Interface definitions. Interpretation which is not intended by the usage of the Keyword can also be given, here. Usually, Longname, Description, and Introduction are the basis for the derivation of the Shortname.

Introduction:
The Introduction of a Keyword can be used to describe further use cases, interpretation or restrictions of these. Currently, the Introduction field of Keyword definitions is not often used.

Abbreviation:
The Abbreviation of a Keyword is the term which shall be used to build Shortnames of elements of Application Interface definition. The Abbreviations are not strictly unique because the same Abbreviation can be interpreted differently for different use cases.

Example:

Shortname	Abbreviation	Longname	Description	Phys. Dim. Longname
Ac	Ac	AC	Alternating Current / Voltage	ConditionQualifier
Ac1	Ac	AC	Air Conditioner or Air Conditioning (A/C)	MeanEnvironmentDevice

Table 10.2: Example for Keyword Definition

Classification:
Keywords are classified to enable the formulation of rules for the building of names. These rules are defined in the chapter Naming Convention for AUTOSAR Model Elements in the AUTOSAR_CP_TR_SWCModelingGuide ([6]). The Classification of a Keyword defines the sequence of the Keyword Abbreviations in a Shortname. The usecases of the Classifications of Keywords can be found in the table below (see also: [6])):

Sequence	Semantic Field Name	Description	Rules and Examples
1	Mean-Environment-Device	Physical mean, environment. Define the element subject of Action-PhysicalType.	It shall be a noun. It can be also a compound definition. Abbreviation or acronym cannot end with a digit. Examples for Mean: Fuel Examples for Environment: Air Examples for Device: Accelerator, Accelerator pedal, Engine
2	*Action - Physical Type*	Action or physical type conditioning or modifying the Mean-Environment-Device.	The Action shall be a verb. The Physical Type shall be a noun. It can be also a compound. Abbreviation or acronym cannot end with a digit. Examples for Action: Move, Pull, Release, Lock, Open, Close, Shift Examples for Physical Type: Temperature, Speed
3	Condition- Qualifier	Qualifies the Mean-Environment-Device or Action-Physical-Type in terms of data flow, event issuing or expresses a particular condition of the signal in terms of numeric treatment, time validity, precision quality, location.	It shall be a noun or an adjective. It can be also a compound definition. Abbreviation or acronym cannot end with a digit. Examples for Condition: Absolute, Old, New, Absolute, Estimated, Examples for Qualifier: Request, Command, Status
4	*Index*	Identifies the signal as part of a logically structured information. Can be used to identify elements multiply instanced (index) or section of information	It shall be a number, a single character, or an adjective describing the part When used, it is always the last keyword in the sequence: Examples for Index: BrakeSwitch1
5	*Preposition*	Used for joining/separating complex naming patterns made by several semantic fields	Example: EngNAndPosn CoolgReqFromSteer TqActAtClu

Table 10.3: Usage of Keyword Classifications

The colors in this table, are used to explain the keyword sequence in the AUTOSAR_CP_TR_SWCModelingGuide ([6]).

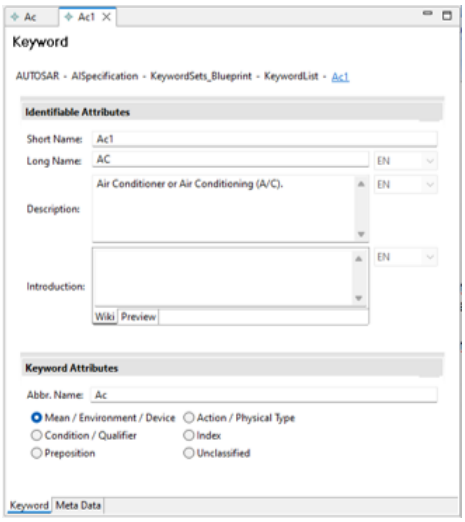


Figure 10.24: Editor view of Keyword Definitions

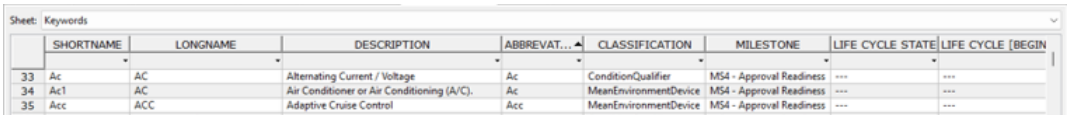


Figure 10.25: Tabular view of Keyword Definitions

Example for the definition of a Keyword definition in AUTOSAR_MOD_AISpecification_KeywordSet_Blueprint.arxml:

```
  <ELEMENTS>
...
  <KEYWORD>
    <SHORT-NAME>Ac1</SHORT-NAME>
    <LONG-NAME>
      <L-4 L="EN">AC</L-4>
    </LONG-NAME>
    <DESC>
      <L-2 L="EN">Air Conditioner or Air Conditioning (A/C).</L-2>
    </DESC>
    <ABBR-NAME>Ac</ABBR-NAME>
    <CLASSIFICATIONS>
      <CLASSIFICATION>Mean-Environment-Device</CLASSIFICATION>
    </CLASSIFICATIONS>
  </KEYWORD>
...
  </ELEMENTS>
```

10.11 Port Prototypes

PortPrototype definitions collect all information to describe the properties of the information exchanged between SW Components via ports. Also here, the information is abstracted from implementation details and collects requirements to Port implementation for SW Components. These definitions allow discussion of interface properties on

System Architecture level, collecting the physical properties of these ports and their architectural description.

There are 2 ARPackages available in the AI-Tool Explorer for the definition of PortPrototypes:

- AISpecification_PortPrototypeBlueprints_Blueprint
- AISpecification_SwComponentTypes_Example

There is no own collection of PortPrototypeBlueprints_Example available, because the derivation of a PortPrototype from a PortPrototype_Blueprint which shall be referenced by a SwComponentType requires a concrete decision for some of the properties, e.g. the direction. For a PortPrototype of a SwComponentType it must be defined, if the Port is provided, required or both. Therefore, examples of PortPrototypes can only exist as child element of a SwComponentType_Example.

According to the AUTOSAR Software Component Template [4], the elements of the PortPrototype definition are:

- Shortname
- Longname
- Description
- Introduction
- Interface
- Provided ComSpecs
- Required ComSpecs
- Blueprint specific Elements
  - Blueprints: Name Pattern
  - Blueprints: Derived Elements
- Example specific Elements
  - Derived from

Additionally, there is Milestone, Life Cycle State and Life Cycle and UseInstead information available for each PortPrototypeBlueprint definition.

Shortname:
The Shortname of a PortPrototype shall reflect the architectural interpretation of the element. Usually Shortnames are derived from architecture descriptions in the Longnames or the descriptions of the AR Elements.

The Shortname of a PortPrototype is defined as an referrable identifier and must therefore respect Rules for identifiers. It is built by concatenating keywords as defined by

AUTOSAR_MOD_AISpecification_KeywordSet_Blueprint.arxml. The naming rules are defined at AUTOSAR Software Component Modeling Guide [6].

Longname:
The Shortname of a PortPrototype shall reflect the architectural interpretation of the element. It shall describe how the information which is exchanged via the port shall be interpreted and used in the physical world of the automotive environment. Usually, Longname and Description which are short architectural summaries of the content of the AR Elements, are the basis for the derivation of Shortnames.

The Longname of a PortPrototype is the spelled-out wording of the Shortname. It is a plain text which shall also not contain any special characters. The naming rules are defined at AUTOSAR Software Component Modeling Guide [6].

Description:
The Description of a PortPrototype shall reflect the architectural interpretation of the element. It shall describe how the information which is exchanged via the port shall be interpreted and used in the physical world of the automotive environment.

Usually, Longname and Description which are short architectural summaries of the content of the AR Elements, are the basis for the derivation of Shortnames. Usually, Longname and Description which are short architectural summaries of the content of the AR Elements, are the basis for the derivation of Shortnames.

Introduction:
The Introduction of a PortPrototype shall reflect the architectural interpretation of the element. It shall describe how the information which is exchanged via the port shall be interpreted and used in the physical world of the automotive environment. The Introduction can be used as source for the generation of parts of explanatory documents, like, e.g. AUTOSAR_CP_EXP_AIPowertrain ([10]).

For PortPrototypes this field can contain larger text passages describing the architectural content and physical interpretation of the Port.

Example:

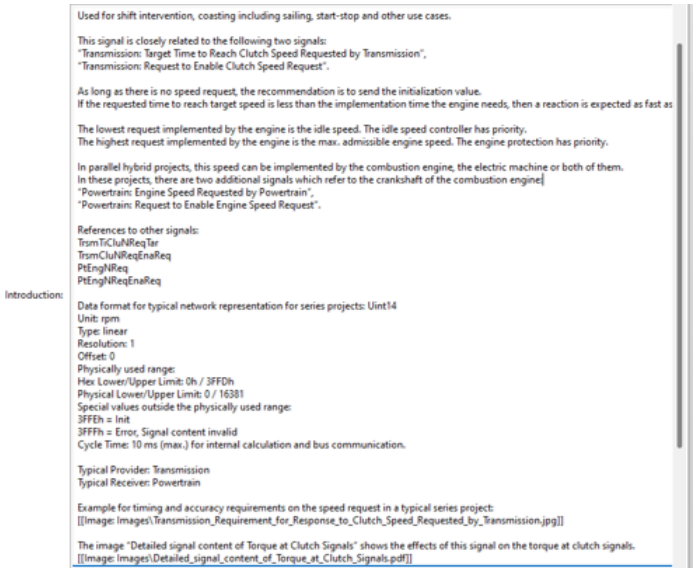


Figure 10.26: Example of an informative Introduction for a PortPrototype

Interface:
The Interface field of a PortPrototype definition contains the reference to the interface definition (10.8)which describes the properties of the information which is exchanged by the port. In the selection list of the field all defined interfaces are collected and can be referenced.

The interface typifies the properties of the ports content.

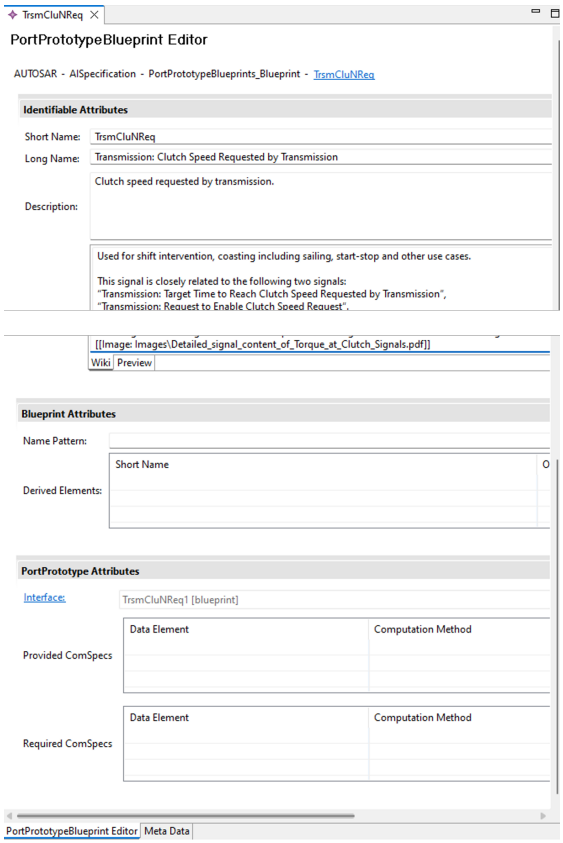


Figure 10.27: Editor view of PortPrototype Definitions

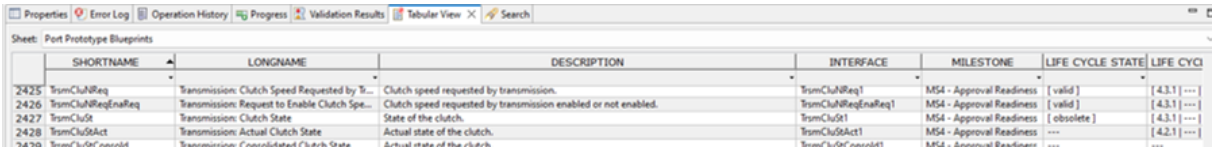


Figure 10.28: Tabular view of PortPrototype Definitions

Example for the definition of a Keyword definition in AUTOSAR_MOD_AISpecification_PortPrototypeBlueprints_Blueprint.arxml:

```
  <ELEMENTS>
...
  <PORT-PROTOTYPE-BLUEPRINT>
    <SHORT-NAME>TrsmCluNReq</SHORT-NAME>
    <LONG-NAME>
      <L-4 L="EN">Transmission: Clutch Speed Requested by Transmission</L-4
          >
    </LONG-NAME>
    <DESC>
      <L-2 L="EN">Clutch speed requested by transmission.</L-2>
    </DESC>
    <INTRODUCTION>
      <P>
        <L-1 L="EN">Used for shift intervention, coasting including
                    sailing, start-stop and other use cases.</L-1>
        <L-1 L="EN">This signal is closely related to the following
```

```
                    two signals:Transmission: Target Time to Reach
                    Clutch Speed Requested by Transmission, Transmission:
                    Request to Enable Clutch Speed Request.</L-1>
        <L-1 L="EN">As long as there is no speed request, the ...</L-1>
      .
      .
      .
      </P>
      <FIGURE>
        <L-GRAPHIC L="EN">
          <GRAPHIC FILENAME="Images\Transmission_Requirement_for_...jpg"/>
        </L-GRAPHIC>
      </FIGURE>
        .
        .
        .
    </INTRODUCTION>
    <INTERFACE-REF BASE="PortInterfaces" DEST="SENDER-RECEIVER-INTERFACE">
                        TrsmCluNReq1
    </INTERFACE-REF>
  </PORT-PROTOTYPE-BLUEPRINT>
...
  </ELEMENTS>
```

Provided ComSpecs:
The usage of ComSpec allows for the specification of communication attributes for the PortPrototype. Currently the section Provided ComSpecs in the AI-Tool shall not be used.

10.12 SW Components

In order to standardize Application Interfaces a number of SW Components are described within the AI-Tool decomposed into bigger function groups (formerly called domains) and their main functions. In the AI-Tool we use this to describe a recommended high level architecture (recommended inner substructure) of the functionality.

Nevertheless, these components must be seen as examples only. They are necessary in order to specify the ports/port prototypes in a consistent way. Each Port/PortPrototype needs a connection to a Component to be specified properly. SwComponents can be reused like but are not a commonly agreed standardized SW Architecture.

SwComponentPrototypes are only used for implementing SwComponentTypes in a specific role, i.e. they are used to instantiate the SwComponentType.

Example: A SwComponentPrototype WiprRe fulfills the role of implementing the CompositionSwComponentType Wipr for the rear window wiper of a vehicle while the SwComponentPrototype WiprFrnt fulfills the role of the CompositionSwComponentType Wipr for the front window wiper. Both SwComponentPrototypes exist only as child element of the CompositionSwComponent WiprWshr (other example, see (Figure 6.4).

All standardized information about CompositionSwComponentType and SwComponentPrototype definition is collected in an ARPackage available in the AI-Tool Explorer:

- AISpecification_SwComponentTypes_Example

According to the AUTOSAR Software Component Template [4], the elements of the SwComponent definition are:

- Shortname
- Longname
- Description
- Introduction
- CompositionSwComponentType specific Elements
  - Components
  - Ports
  - Connectors
- SwComponentPrototype specific Elements
  - Type

Additionally, there is Milestone, Life Cycle State and Life Cycle and UseInstead information available for each PortPrototypeBlueprint definition.

Shortname:
The Shortname of a SwComponentType shall reflect the architectural content of the element. Usually Shortnames are derived from architecture descriptions in the Longnames or the descriptions of the AR Elements.

The Shortname of a SwComponentType is defined as a referrable identifier and must therefore respect Rules for identifiers. It is built by concatenating keywords as defined by AUTOSAR_MOD_AISpecification_KeywordSet_Blueprint.arxml. The naming rules are defined at AUTOSAR Software Component Modeling Guide [6].

Longname:
The Longname of a SwComponentType shall reflect the architectural content of the element.

Usually, Longname and Description which are short architectural summaries of the content of the AR Elements, are the basis for the derivation of Shortnames.

The Longname of a SwComponentType is the spelled-out wording of the Shortname. It is a plain text which shall also not contain any special characters. The naming rules are defined at AUTOSAR Software Component Modeling Guide [6].

Description:
The Description of a SwComponentType shall reflect the architectural content of the element as a short overview explanation.

Usually, Longname and Description which are short architectural summaries of the content of the AR Elements, are the basis for the derivation of Shortnames.

Introduction:
The Introduction of a SwComponentType shall reflect the architectural content of the element. It can be used to give more detailed information about the functional content of the component.

Components:
A CompositionSwComponentType can be used to define a recommended decomposition into several sub-components (logical or functional architecture representation). In the Components-section of the editor this decomposition can be described. It contains the definition of SwComponentPrototypes, elements which are defined child elements of the CompositionSwComponentType. These are instances derived from predefined CompositionSwComponentTypes. The SwComponentPrototypes only exist in the Components-definition of a CompositionSwComponentType and can have different properties than the CompositionSwComponentType they are derived from, like e.g., they can have a different Shortname.

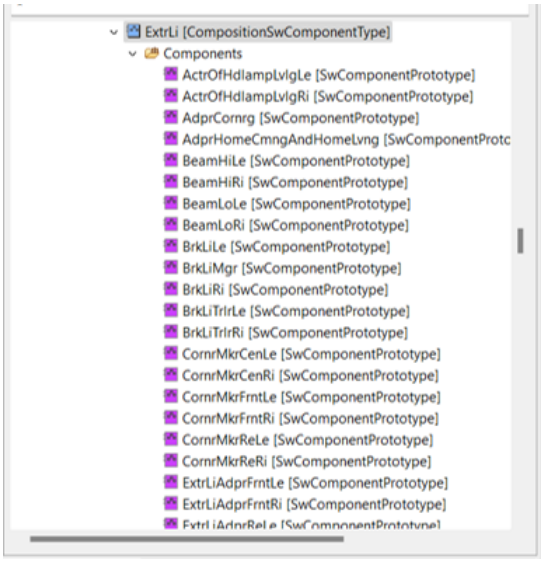


Figure 10.29: Decomposition of the CompositionSwComponent ExtrLi into SwComponentPrototypes

In the figure (Figure 10.29)above how the decomposition of a CompositionSwComponent into SwComponentPrototypes is shown, how the decomposition is displayed in the Ai-Tool. The definition of the decomposition is done in the editor of the CompositionSwComponent.

Connectors:
A CompositionSwComponentType owns the definition of all interconnections between its child SwComponents and the connections of internal interfaces to the outside of the SwComponent. The Connectors describe which and how Ports of the parent SwComponent and its child SwComponents are connected to others. With this mean the internal data flow and service flow of the CompositionSwComponent can be described.

Ports:
Each SwComponentType definition includes the description of Ports which represent an option to interact with other SwComponentTypes. These Ports are instances of predefined Port descriptions.

Only PortPrototypes which are instances of predefined PortPrototypeBlueprints, existing in the CompositionSwComponent description can have a direction. The direction information defines, if the exchanged information or service is an output delivered/provided by this SwComponent (provide Port, PPort) or if it is an input/use of data or services (require Port, RPort) to the functionality which is needed to produce correct outputs. It is also possible to have Ports which allow both directions (provide-require Port, PRPort), e.g. to represent feedback-loops, where, e.g. an output value which was calculated one calculation cycle before, is feed back as an input for the next cycle.

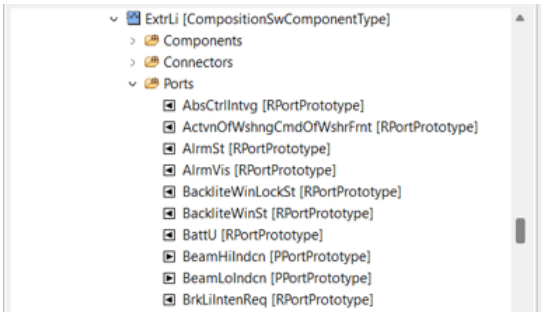


Figure 10.30: Definition of the Direction of the Ports of a SwComponent

In the figure (Figure 10.30)above, is shown, how the direction of the information flow via a Port is specified. There are different types of Portprototype descriptions which can be used to define the direction, here. The icons for RPortPrototypes and PPortPrototypes are different.

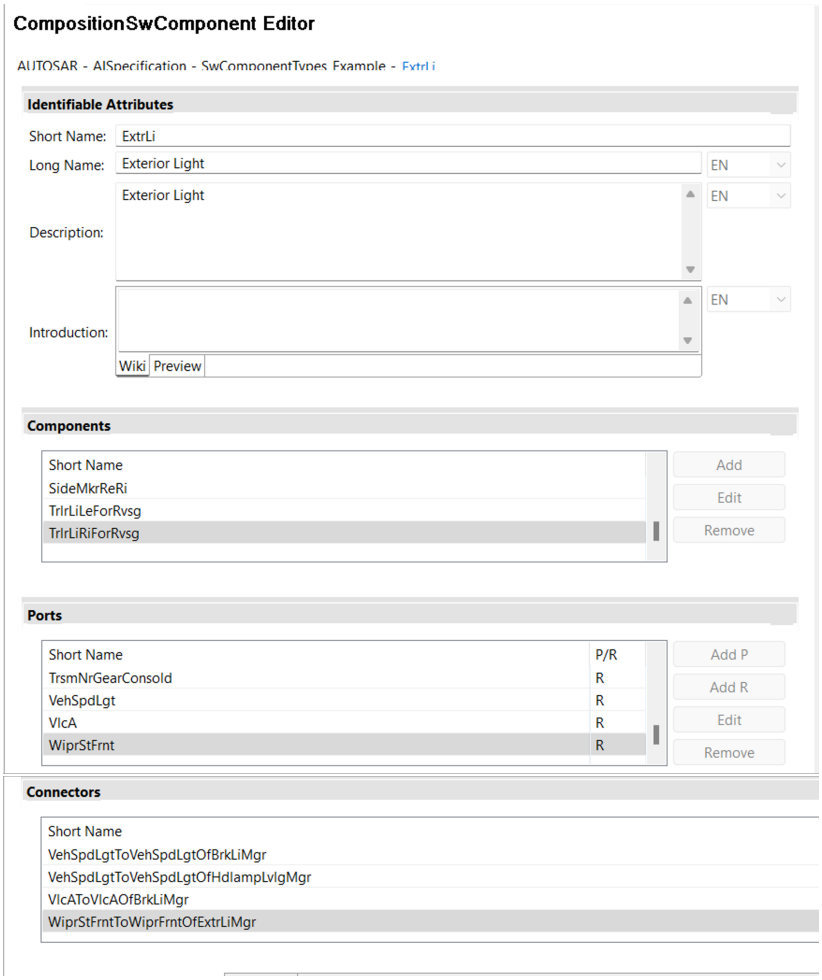


Figure 10.31: Editor view of CompositionSwComponents

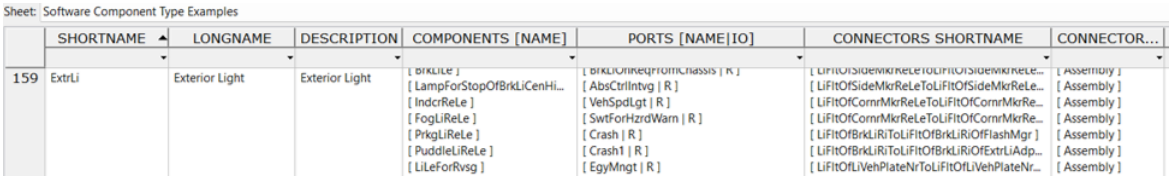


Figure 10.32: Tabular view of CompositionSwComponents

Example for the definition of a CompositionSwComponent in AUTOSAR_MOD_AISpecificationExamples.arxml:

```
  <ELEMENTS>
...
  <COMPOSITION-SW-COMPONENT-TYPE>
    <SHORT-NAME>Snsr</SHORT-NAME>
    <LONG-NAME>
      <L-4 L="EN">Sensor</L-4>
    </LONG-NAME>
    <DESC>
      <L-2 L="EN">Design pattern of components for a typical  device</L-2>
    </DESC>
    <PORTS>
```

```
    <P-PORT-PROTOTYPE>
      <SHORT-NAME>Consold</SHORT-NAME>
      <LONG-NAME>
        <L-4 L="EN">Consolidated Physical Value</L-4>
      </LONG-NAME>
      <DESC>
        <L-2 L="EN">Consolidated physical value, either a measured value
            or a modeled value</L-2>
      </DESC>
      <INTRODUCTION>
        <P>
          <L-1 L="EN">Consolidated physical value, either a measured
              value (Measd) or a modeled value (Estimd). Final filtered
              and offset corrected consolidated actuator value/physical
              sensor value. Virtual physical sensor value/fused sensor
              value that comes as close as possible to the technical
              signal. In case of inability to provide a physical sensor
              value (e.g. failure, implausibility or other reasons) a
              substitute value/default value or a frozen value is provided
              .</L-1>
        </P>
      </INTRODUCTION>
      <PROVIDED-INTERFACE-TREF DEST="SENDER-RECEIVER-INTERFACE">/AUTOSAR/
          AISpecification/SwComponentTypes_Example/SwDesignPatterns/
          SnsrActrAbstraction/PortInterfaces_Example/Consold</PROVIDED-
          INTERFACE-TREF>
    </P-PORT-PROTOTYPE>
    <R-PORT-PROTOTYPE>
      .
      .
      .
    </R-PORT-PROTOTYPE>
  </PORTS>
  <COMPONENTS>
    <SW-COMPONENT-PROTOTYPE>
      <SHORT-NAME>DevSnsrVirt</SHORT-NAME>
      <LONG-NAME>
        <L-4 L="EN">Virtual Sensor Device Driver</L-4>
      </LONG-NAME>
      <TYPE-TREF DEST="COMPOSITION-SW-COMPONENT-TYPE">/AUTOSAR/
          AISpecification/SwComponentTypes_Example/SwDesignPatterns/
          SnsrActrAbstraction/DevSnsrVirt</TYPE-TREF>
    </SW-COMPONENT-PROTOTYPE>
    <SW-COMPONENT-PROTOTYPE>
      .
      .
      .
    </SW-COMPONENT-PROTOTYPE>
  </COMPONENTS>
  <CONNECTORS>
    <DELEGATION-SW-CONNECTOR>
      <SHORT-NAME>EstimdToEstimdOfDevSnsrVirt</SHORT-NAME>
      <INNER-PORT-IREF>
        <R-PORT-IN-COMPOSITION-INSTANCE-REF>
          <CONTEXT-COMPONENT-REF DEST="SW-COMPONENT-PROTOTYPE">/AUTOSAR
              /AISpecification/SwComponentTypes_Example/SwDesignPatterns
```

```
                /SnsrActrAbstraction/Snsr/DevSnsrVirt</CONTEXT-COMPONENT-
                REF>
              <TARGET-R-PORT-REF DEST="R-PORT-PROTOTYPE">/AUTOSAR/
                AISpecification/SwComponentTypes_Example/SwDesignPatterns/
                SnsrActrAbstraction/DevSnsrVirt/Estimd</TARGET-R-PORT-REF>
            </R-PORT-IN-COMPOSITION-INSTANCE-REF>
          </INNER-PORT-IREF>
          <OUTER-PORT-REF DEST="R-PORT-PROTOTYPE">/AUTOSAR/AISpecification/
            SwComponentTypes_Example/SwDesignPatterns/SnsrActrAbstraction/
            Snsr/Estimd</OUTER-PORT-REF>
        </DELEGATION-SW-CONNECTOR>
        <DELEGATION-SW-CONNECTOR>
          .
          .
          .
        </DELEGATION-SW-CONNECTOR>
        <ASSEMBLY-SW-CONNECTOR>
          <SHORT-NAME>MeasdOfDevDrvrSnsrToMeasdOfDevSnsrVirt</SHORT-NAME>
          <PROVIDER-IREF>
            <CONTEXT-COMPONENT-REF DEST="SW-COMPONENT-PROTOTYPE">/AUTOSAR/
              AISpecification/SwComponentTypes_Example/SwDesignPatterns/
              SnsrActrAbstraction/Snsr/DevDrvrSnsr</CONTEXT-COMPONENT-REF>
            <TARGET-P-PORT-REF DEST="P-PORT-PROTOTYPE">/AUTOSAR/
              AISpecification/SwComponentTypes_Example/SwDesignPatterns/
              SnsrActrAbstraction/DevDrvrSnsr/Measd</TARGET-P-PORT-REF>
          </PROVIDER-IREF>
          <REQUESTER-IREF>
            <CONTEXT-COMPONENT-REF DEST="SW-COMPONENT-PROTOTYPE">/AUTOSAR/
              AISpecification/SwComponentTypes_Example/SwDesignPatterns/
              SnsrActrAbstraction/Snsr/DevSnsrVirt</CONTEXT-COMPONENT-REF>
            <TARGET-R-PORT-REF DEST="R-PORT-PROTOTYPE">/AUTOSAR/
              AISpecification/SwComponentTypes_Example/SwDesignPatterns/
              SnsrActrAbstraction/DevSnsrVirt/Measd</TARGET-R-PORT-REF>
          </REQUESTER-IREF>
        </ASSEMBLY-SW-CONNECTOR>
        <ASSEMBLY-SW-CONNECTOR>
          .
          .
          .
        </ASSEMBLY-SW-CONNECTOR>
      </CONNECTORS>
    </COMPOSITION-SW-COMPONENT-TYPE>
...
    </ELEMENTS>
```

10.13 Connectors

Connectors represent the interaction between SoftwareComponents via their Ports. If Ports of SoftwareComponents are connected via such a connector this means that the SoftwareComponents exchange data, client-server actions, or modes. Connectors are only defined as substructure of SoftwareComponents because they exist only in and are owned by a CompositionSwComponentType which contains other SwComponent-

Prototypes. The CompositionSwComponentType and the SwComponentPrototypes interact with each other.

There are 2 different types of connectors:

- Delegation Connector
- Assembly Connector

10.13.1 DelegationSwConnector

A DelegationSwConnector represents the interaction between the "outer" SwComponentPrototype and an "inner" SwComponentPrototype or vice versa. Here, ports of the same direction are connected, provided ports are connected to inner or outer provided ports and the same applies to requested ports.

The representation of a DelegationSwConnector definition consists of the following elements:

- Shortname
- Longname
- Description
- Introduction
- Inner Port Component
- Inner Port
- Outer Port

Additionally, there is Milestone, Life Cycle State and Life Cycle and UseInstead information available for each connector definition.

Shortname:

The Shortname of a DelegationSwConnector is built automatically by the AI-Tool DelegationSwConnector-editor when selecting the port of the inner SwComponentPrototype and the port of the outer CompositionSwComponentType. Shortnames are defined as a referrable identifiers and must therefore respect C-Coding Rules. They are build by concatenating keywords as defined by AUTOSAR_MOD_AISpecification_KeywordSet_Blueprint.arxml.

Longname:
Description:
Introduction:
Longname, Description and Introduction for SwConnectors are currently not used because their use case and application is intrinsically related to the type of the connector.

Connectors cannot be standardized as Blueprint because they only exist as part of a CompositionSwComponentType definition.

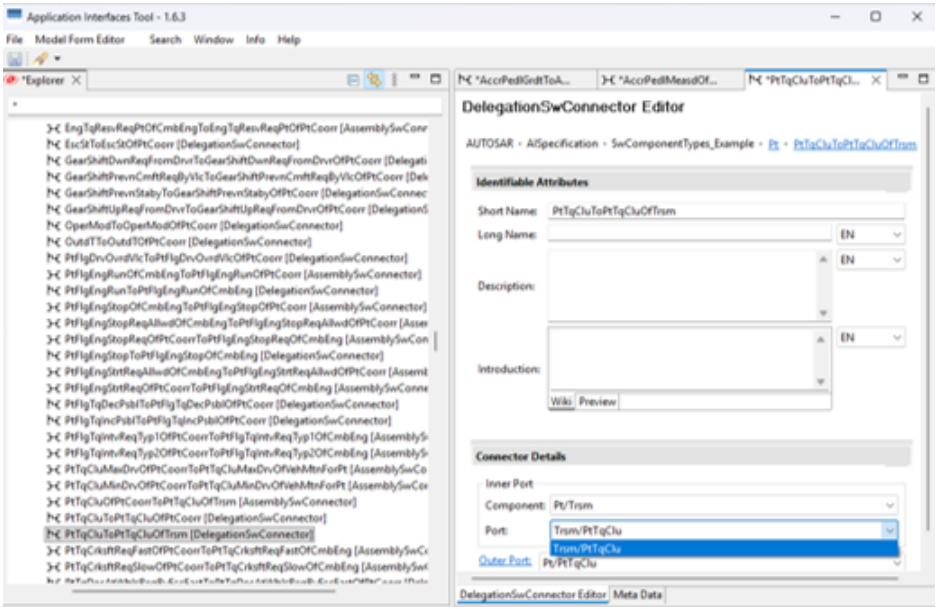


Figure 10.33: Editor view of Delegation Connector Definitions

10.13.2 AssemblySwConnector

An AssemblySwConnector represents the interaction between 2 "inner" SwComponentPrototypes of a CompositionSwComponentType via their ports. Here, In each case, a provider port is connected to a requested port.

The representation of an AssemblySwConnector definition consists of the following elements:

- Shortname
- Longname
- Description
- Introduction
- Provider Component
- Provider Port
- Requester Component
- Requester Port

Additionally, there is Milestone, Life Cycle State and Life Cycle and UseInstead information available for each connector definition.

Shortname:

The Shortname of an AssemblySwConnector is built automatically by the AI-Tool AssemblySwConnector-editor when selecting the P-Port of the providing SwComponentPrototype and the R-Port of the requesting SwComponentPrototype. Shortnames are defined as referrable identifiers and must therefore respect C-Coding Rules. They are build by concatenating keywords as defined by AUTOSAR_MOD_AISpecification_KeywordSet_Blueprint.arxml.

Longname:
Description:
Introduction:
Longname, Description and Introduction for SwConnectors are currently not used because their use case and application is intrinsically related to the type of the connector.

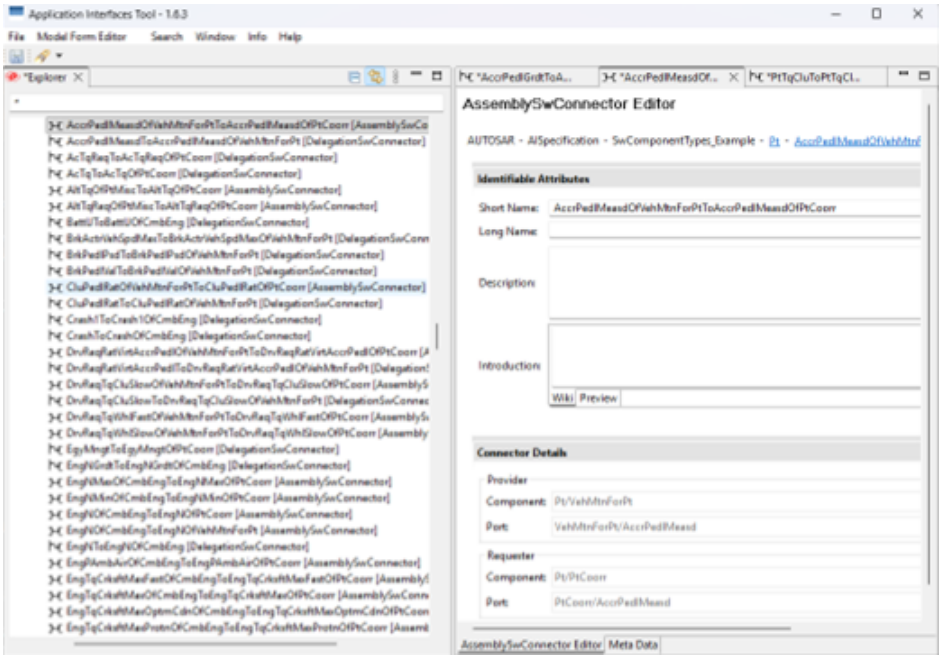


Figure 10.34: Editor view of Assembly Connector Definitions

Example for .arxml structure for the arxml description of a connector as defined in AUTOSAR_MOD_AISpecificationExamples.arxml:

```
<CONNECTORS>
  <DELEGATION-SW-CONNECTOR>
    <SHORT-NAME>
     ActvtOfActrAtHoodForPedProtnToActvtOfActrAtHoodForPedProtnOfHoodPedProtn

    </SHORT-NAME>
    <INNER-PORT-IREF>
      <R-PORT-IN-COMPOSITION-INSTANCE-REF>
        <CONTEXT-COMPONENT-REF BASE="SwComponentTypes"
         DEST="SW-COMPONENT-PROTOTYPE">ActrPoolOccptPedSfty1/HoodPedProtn
        </CONTEXT-COMPONENT-REF>
        <TARGET-R-PORT-REF BASE="SwComponentTypes"
```

```
          DEST="R-PORT-PROTOTYPE">HoodPedProtn/ActvtOfActrAtHoodForPedProtn
        </TARGET-R-PORT-REF>
      </R-PORT-IN-COMPOSITION-INSTANCE-REF>
    </INNER-PORT-IREF>
    <OUTER-PORT-REF BASE="SwComponentTypes"
       DEST="R-PORT-PROTOTYPE">ActrPoolOccptPedSfty1/
          ActvtOfActrAtHoodForPedProtn
    </OUTER-PORT-REF>
  </DELEGATION-SW-CONNECTOR>
...
  <ASSEMBLY-SW-CONNECTOR>
    <SHORT-NAME>
      SnsrInclnOutOfDrvrSnsrInclnToSnsrInclnOutOfSnsrAdprCmplxAlrm
    </SHORT-NAME>
    <PROVIDER-IREF>
      <CONTEXT-COMPONENT-REF BASE="SwComponentTypes"
       DEST="SW-COMPONENT-PROTOTYPE">Antithft/DrvrSnsrIncln
      </CONTEXT-COMPONENT-REF>
      <TARGET-P-PORT-REF BASE="SwComponentTypes"
       DEST="P-PORT-PROTOTYPE">SnsrIncln/SnsrInclnOut
      </TARGET-P-PORT-REF>
    </PROVIDER-IREF>
    <REQUESTER-IREF>
      <CONTEXT-COMPONENT-REF BASE="SwComponentTypes"
       DEST="SW-COMPONENT-PROTOTYPE">Antithft/SnsrAdprCmplxAlrm
      </CONTEXT-COMPONENT-REF>
      <TARGET-R-PORT-REF BASE="SwComponentTypes"
       DEST="R-PORT-PROTOTYPE">SnsrAdprCmplxAlrm/SnsrInclnOut
      </TARGET-R-PORT-REF>
    </REQUESTER-IREF>
  </ASSEMBLY-SW-CONNECTOR>
  ...
  </CONNECTORS>
```

10.14 Physical Units

The commonly agreed definition of physical units is a basis for the development of reusable and exchangeable SW components. Our representation of physical units definition consists of the following elements:

- Shortname
- Longname
- Description
- Introduction
- Displayname
- Physical Dimension Shortname
- FactorSiToUnit

- OffsetSiToUnit

Additionally, there is Milestone, Life Cycle State and Life Cycle and UseInstead information available for each physical dimension definition.

Shortname:
The Shortname of a physical unit is defined as a referrable identifier and must therefore respect C-Coding Rules. It is built by concatenating keywords as defined by AUTOSAR_MOD_AISpecification_KeywordSet_Blueprint.arxml. Special characters which are usually used by the SI Unit names are replaced by keywords, like, e.g. the Greek letter µ is represented as Micro.

Longname:
The Longname (see table 10.4, column Longname) of a physical unit is the spelled-out wording of the short and the display name. It is a plain text which shall also not contain any special characters.

Description:
Descriptions of Units are currently not used.

Introduction:
If necessary, additional information concerning use cases or application rules can be given, here. But currently it is not used in any unit definition, by now.

Displayname:
Usually physical units are displayed in documentation or application tooling with their names as defined by the SI standard. Unfortunately, such names can contain special characters like , e.g. µ or °. But for documentation and application purpose these names are the officially defined and commonly known names. So, the Displayname of a physical unit specifies how the unit shall be displayed in documentation or other human interface channels.

Shortname	Longname	Displayname	Phys. Dim. Shortname	Phys. Dim. Longname	Factor SI to Unit	Offset SI to Unit
MegaJou	Megajoule	MJ	Len2M1TiNeg2_1	Energy	1.0E-6	0.0
KiloJou	Kilojoule	kJ	Len2M1TiNeg2_1	Energy	0.001	0.0

Table 10.4: Example for Physical Unit Definition

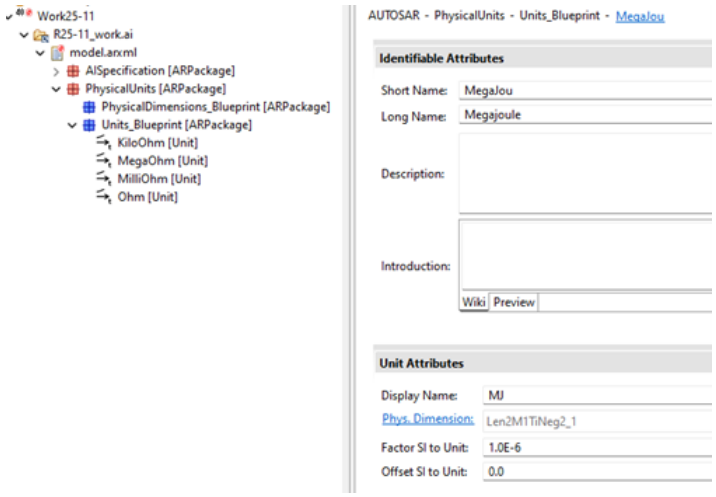


Figure 10.35: Editor view of Unit Definitions

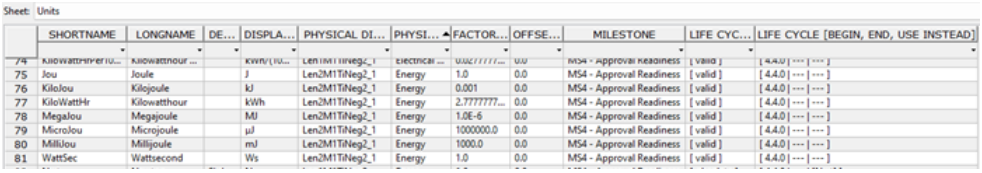


Figure 10.36: Unit Definitions in Tabular View

Example for .arxml structure for physical units as defined in AUTOSAR_FO_MOD_GeneralBlueprints_AUTOSAR_FO_MOD_PhysicalUnits_Blueprint:

```
<UNIT>
    <SHORT-NAME>NwtMtr</SHORT-NAME>
    <LONG-NAME>
      <L-4 L="EN">Newtonmeter</L-4>
    </LONG-NAME>
    <DISPLAY-NAME>Nm</DISPLAY-NAME>
    <FACTOR-SI-TO-UNIT>1.0</FACTOR-SI-TO-UNIT>
    <OFFSET-SI-TO-UNIT>0.0</OFFSET-SI-TO-UNIT>
    <PHYSICAL-DIMENSION-REF BASE="PhysicalDimensions" DEST="PHYSICAL-
       DIMENSION">
      Len2M1TiNeg2
    </PHYSICAL-DIMENSION-REF>
 </UNIT>
```

10.15 Physical Dimensions

Here, the physical dimensions used to specify physical units are defined. The physical dimensions are referenced by the unit definition via their Shortnames.

The elements of the physical dimension definition are:

- Shortname
- Longname

- Description
- Length Exponent
- Mass Exponent
- Time Exponent
- Temperature Exponent
- Electric Current Exponent
- Luminous Intensity Exponent
- Amount of Substance Exponent

Additionally, there is Milestone, Life Cycle State and Life Cycle and UseInstead information available for each physical dimension definition.

Attention: As the physical dimensions and the units got reworked, some years ago, there are two packages of physical dimensions and units visible in the Explorer view of the AI-Tool:

- AISpecification [ARPackage]_PhysicalDimensions_obsolete [ARPackage]
- PhysicalUnits [ARPackage]_PhysicalDimensions [ARPackage]

The package PhysicalDimensions_obsolete [ARPackage] contains outdated definitions which need to be kept for backward compatibility reasons. New unit definitions shall not reference these obsolete physical dimensions anymore.

Shortname:
The Shortname (see 10.5, column Shortname) of a physical dimension is defined as a referrable identifier and must therefore respect C-Coding Rules. It is built of the keyword abbreviations of the 7 SI-Base Physical Quantities as defined by AUTOSAR_MOD_AISpecification_KeywordSet_Blueprint.arxml.:

- Length (Len)
- Mass (M)
- Time (Ti)
- Temperature (T)
- Electric Current (I)
- Luminous Intensity (Ilmn)
- Amount of Substance (Amnt)

combined with:

- the exponents for the Base Quantities
- if necessary, the Keyword ‘Neg’ for negative exponents representing fractions

- if necessary, the Term NoDimension for physical dimension having all exponents set to 0, like, e.g. angle, ratio, numbers, etc..
- if necessary to add an index, there shall be an underscore
- if necessary, an index for physical dimensions having the same combination of exponents but representing different physical quantities like, e.g. Energy (SI Derived Base Unit Joule) and Torque (SI Derived Base Unit Newtonmeter).

Longname:
The Longname (see 10.5, column Longname) of a physical dimension is the name of the represented physical quantity, e.g.:

Description:
The Description field for physical dimensions is currently not used, because the Longname and the combination of the exponents contains the essential information, here.

Exponents for the 7 SI-Base Quantities
Each physical unit - and therefore - the corresponding physical quantity can be represented by a product of exponents of the 7 SI-Base Units (and quantities). Therefore, the combination of these exponents defines a physical quantity (see 10.5, columns kg, m, s, K, A, mol, cd). Unfortunately, this representation is- from mathematical point of view - not biunique. There are physical dimensions, having the same combination of exponents which are not compatible for transformation into each other, like, e.g.:

Shortname	Longname	kg	m	s	K	A	mol	cd
Len2M1TiNeg2	Torque	1	2	-2	0	0	0	0
Len2M1TiNeg2_1	Energy	1	2	-2	0	0	0	0

Table 10.5: Example for Physical Dimension Definition

From mathematical point of view, energy is a scalar product whereas torque is defined as a cross-product. Therefore, it is necessary to have separate Shortnames for these physical dimensions to clearly distinguish between the two of them and to avoid not intended transformation between the referencing units, like e.g. Nm and J.

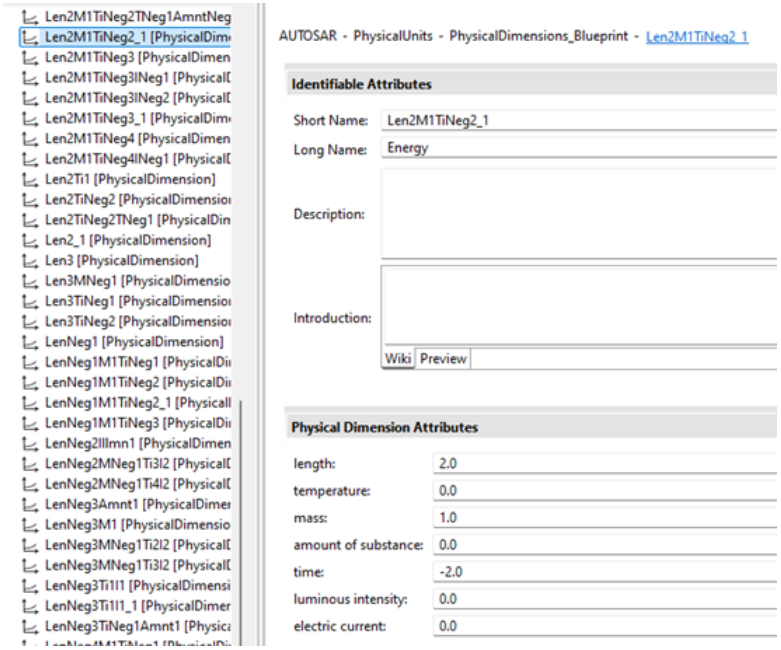
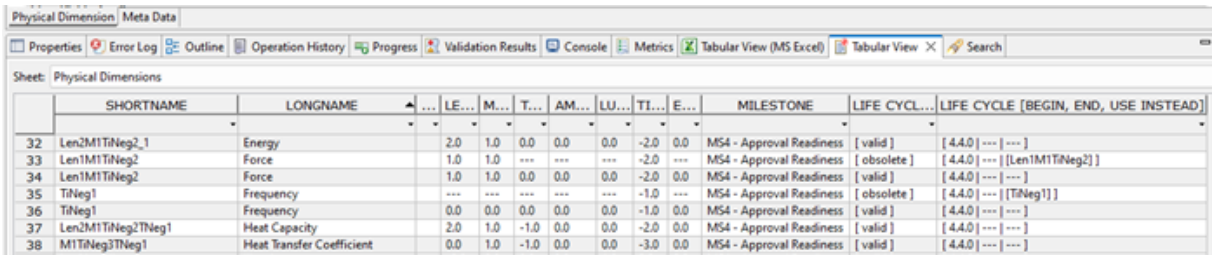


Figure 10.37: Editor view of Physical Dimension Definitions



	SHORTNAME	LONGNAME	...	LE...	M...	T...	AM...	LU...	TI...	E...	MILESTONE	LIFE CYCL...	LIFE CYCLE [BEGIN, END, USE INSTEAD]
32	Len2M1TiNeg2_1	Energy		2.0	1.0	0.0	0.0	0.0	-2.0	0.0	MS4 - Approval Readiness	[valid]	[4.4.0 | --- | ---]
33	Len1M1TiNeg2	Force		1.0	1.0	---	---	---	-2.0	---	MS4 - Approval Readiness	[obsolete]	[4.4.0 | --- | [Len1M1TiNeg2]]
34	Len1M1TiNeg2	Force		1.0	1.0	0.0	0.0	0.0	-2.0	0.0	MS4 - Approval Readiness	[valid]	[4.4.0 | --- | ---]
35	TiNeg1	Frequency		---	---	---	---	---	-1.0	---	MS4 - Approval Readiness	[obsolete]	[4.4.0 | --- | [TiNeg1]]
36	TiNeg1	Frequency		0.0	0.0	0.0	0.0	0.0	-1.0	0.0	MS4 - Approval Readiness	[valid]	[4.4.0 | --- | ---]
37	Len2M1TiNeg2TNeg1	Heat Capacity		2.0	1.0	-1.0	0.0	0.0	-2.0	0.0	MS4 - Approval Readiness	[valid]	[4.4.0 | --- | ---]
38	M1TiNeg3TNeg1	Heat Transfer Coefficient		0.0	1.0	-1.0	0.0	0.0	-3.0	0.0	MS4 - Approval Readiness	[valid]	[4.4.0 | --- | ---]

Figure 10.38: Physical Dimension Definitions in Tabular View

Example for the .arxml structure for physical dimensions as defined in AUTOSAR_FO_MOD_GeneralBlueprints:
AUTOSAR_FO_MOD_PhysicalDimensions_Blueprint:

```
<?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xml="
    http://www.w3.org/XML/1998/namespace" xmlns="http://autosar.org/schema/
    r4.0" xsi:schemaLocation="http://autosar.org/schema/r4.0 AUTOSAR_00053.
    xsd">
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>AUTOSAR</SHORT-NAME>
      <LONG-NAME>
        <L-4 L="EN">AUTOSAR</L-4>
      </LONG-NAME>
      <AR-PACKAGES>
        <AR-PACKAGE>
          <SHORT-NAME>PhysicalUnits</SHORT-NAME>
          <AR-PACKAGES>
            <AR-PACKAGE>
              <SHORT-NAME>PhysicalDimensions_Blueprint</SHORT-NAME>
              <CATEGORY>BLUEPRINT</CATEGORY>
```

```
<ELEMENTS>
  <PHYSICAL-DIMENSION>
    <SHORT-NAME>Amnt1</SHORT-NAME>
    <LONG-NAME>
      <L-4 L="EN">Quantity of substance</L-4>
    </LONG-NAME>
    <LENGTH-EXP>0</LENGTH-EXP>
    <MASS-EXP>0</MASS-EXP>
    <TIME-EXP>0</TIME-EXP>
    <CURRENT-EXP>0</CURRENT-EXP>
    <TEMPERATURE-EXP>0</TEMPERATURE-EXP>
    <MOLAR-AMOUNT-EXP>1</MOLAR-AMOUNT-EXP>
    <LUMINOUS-INTENSITY-EXP>0</LUMINOUS-INTENSITY-EXP>
  </PHYSICAL-DIMENSION>
```

11 Relationship between AI-Tool data and .arxml Output

The data maintained by the AI-Tool, reflects the structure defined in the AUTOSAR Meta-Model and is used to generate XML descriptions of the AUTOSAR Application Interfaces. These .arxml descriptions shall adhere to the AUTOSAR Schema [14] which is generated from the AUTOSAR Meta-Model [15]. The following chapters describe how contents of the AI-Tool are used to generate the AUTOSAR Schema elements in the relevant .arxml files.

11.1 Overview

11.1.1 Dependencies of XML Generation

Figure 11.1 illustrates the dependencies in the XML generation process. Currently, the AI-Tool reflects the structure for multiple AUTOSAR Metamodel releases in one database, e.g. for R24-11 and R25-11. This common database is then used by the AI-Tool XML Generation, to generate XML descriptions for each supported release. This approach implies that not all data from the AI-Tool will be reflected in all generated XML files as only the valid definitions are taken into account. Data which was set not set to VALID or are in LifeCycle states MS0, MS1, MS2 and MS3 will not be available in the next generated XML files.

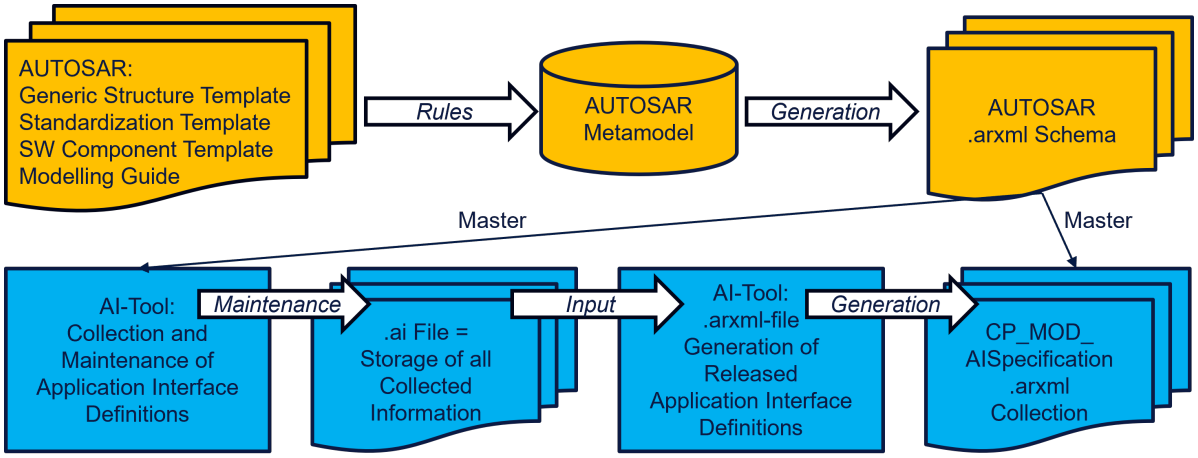


Figure 11.1: Dependencies in the XML generation process for application interfaces

11.1.2 Contents of Generated XML

11.1.2.1 File Distribution

From R19-11 onwards, the content produced by the Application Interfaces Working Group is delivered in the SVN folder:

//svn.autosar.org/repos/work/26_Standards/02_Releases/10_CP/Rxxxxx/ApplicationInterfaces

This folder contains the following files:

- AUTOSAR_CP_EXP_AIADASAndVMC.pdf
- AUTOSAR_CP_EXP_AIBodyAndComfort.pdf
- AUTOSAR_CP_EXP_AIChassis.pdf
- AUTOSAR_CP_EXP_AIHMIMultimediaAndTelematics.pdf
- AUTOSAR_CP_EXP_AIOccupantAndPedestrianSafety.pdf
- AUTOSAR_CP_EXP_PowerTrain.pdf
- AUTOSAR_CP_EXP_AIUserGuide.pdf
- AUTOSAR_MOD_AISpecification.zip archive
- AUTOSAR_MOD_AISpecificationExamples.zip archive
- AUTOSAR_CP_RS_SWCModeling.pdf
- AUTOSAR_CP_TR_AIDesignPatternsCatalogue.pdf
- AUTOSAR_CP_TR_AIMeasurementCalibrationDiagnostics.pdf
- AUTOSAR_CP_TR_SWCModelingGuide.pdf

The contents maintained via the AI-Tool are exported into the .arxml file collections as listed below.

11.1.2.2 Contents of the AUTOSAR_CP_MOD_AISpecification.zip

File	File Type	Content/Use Case
AUTOSAR_CC_AISpecification.xml	.xml file	Collection of all collections contained in the AUTOSAR_MOD_AISpecification
AUTOSAR_MOD_AISpecification_ ApplicationDataType_Blueprint.arxml	.arxml file	Collection of all released Application Data Type Blueprint definitions
AUTOSAR_MOD_AISpecification_ ApplicationDataType_LifeCycle_Standard.arxml	.arxml file	Collection of Life Cycle information for the Application Data Type Blueprint definitions
AUTOSAR_MOD_AISpecification_ Collection_Body_Blueprint.arxml	.arxml file	Collection of all Elements defined by the Body Domain
AUTOSAR_MOD_AISpecification_ Collection_Chassis_Blueprint.arxml	.arxml file	Collection of all Elements defined by the Chassis Domain
AUTOSAR_MOD_AISpecification_ Collection_MmedTelmHmi_Blueprint.arxml	.arxml file	Collection of all Elements defined by the HMI, Multimedia and Telematics Domain
AUTOSAR_MOD_AISpecification_ Collection_OccptPedSfty_Blueprint.arxml	.arxml file	Collection of all Elements defined by the Occupant and Pedestrian Safety Systems Domain
AUTOSAR_MOD_AISpecification_ Collection_Pt_Blueprint.arxml	.arxml file	Collection of all Elements defined by the Powertrain Domain

▽

△

File	File Type	Content/Use Case
AUTOSAR_MOD_AISpecification_ CompuMethod_Blueprint.arxml	.arxml file	Collection of Computation Method Blueprint Definitions
AUTOSAR_MOD_AISpecification_ CompuMethod_LifeCycle_Standard.arxml	.arxml file	Collection of Life Cycle information for the Computation Method definitions
AUTOSAR_MOD_AISpecification_ DataConstr_Blueprint.arxml	.arxml file	Collection of all Data Constraint Blueprint definitions
AUTOSAR_MOD_AISpecification_ DataConstr_LifeCycle_Standard.arxml	.arxml file	Collection of Life Cycle information for the Data Constraint definitions
AUTOSAR_MOD_AISpecification_ PortInterface_Blueprint.arxml	.arxml file	Collection of all Port Interface Blueprint definitions
AUTOSAR_MOD_AISpecification_ PortInterface_LifeCycle_Standard.arxml	.arxml file	Collection of Life Cycle information for Port Interface Definitions
AUTOSAR_MOD_AISpecification_ PortPrototypeBlueprint_Blueprint.arxml	.arxml file	Collection of all Port Prototype Blueprint definitions
AUTOSAR_MOD_AISpecification_ PortPrototypeBlueprint_LifeCycle_ Standard.arxml	.arxml file	Collection of Life Cycle information for Port Prototype Definitions
AUTOSAR_MOD_AISpecification_ KeywordSet_Blueprint.arxml	.arxml file	Collection of all Keyword Set Blueprint definitions which can be used to build Application Interface names
AUTOSAR_MOD_AISpecification_ Keyword_LifeCycle_Standard.arxml	.arxml file	Collection of Life Cycle information for Keyword definitions
AUTOSAR_MOD_AISpecification_ PhysicalDimension_Standard.arxml	.arxml file	Collection of Physical Dimension definitions Attention: This is only there for Backward Compatibility Reasons. The definitions, here, must not be used for the definition of new elements
AUTOSAR_MOD_AISpecification_ PhysicalDimension_LifeCycle_ Standard.arxml	.arxml file	Collection of Life Cycle information for Physical Dimension definitions
AUTOSAR_MOD_AISpecification_ Unit_Standard.arxml	.arxml file	Collection of Physical Unit definitions Attention: This is only there for Backward Compatibility Reasons. The definitions, here, must not be used for the definition of new elements
_disclaimer.txt	Textfile	Collection of legal regulations.
_readme.txt	Textfile	Overview of contents and additional legal regulations.
Accuracy_requirements_torque_signals.jpg	.jpg-file	Picture explaining the accuracy requirements for the implementation of Torque signals.
catalog_V3_0_0.ml.xsd	.xsd	Description of the XML-Schema for the .arxml export.
Detailed_signal_content_of_ Torque_at_Clutch_Signals.pdf	.pdf-file	Detailed description of the definition of Torque at Clutch signals.
State_of_Power_Transmission_ of_Gearbox.pdf	.pdf-file	Description of the states of Power Transmission of Gearbox including Drive-off Element and Gearbox Core for all Gearbox Types.
Transmission_Accuracy_Timing_ Requirements_Torque_Signals.jpg	.jpg-file	Picture explaining the Transmission Accuracy requirements to Torque signals.
Transmission_Requirement_for_ Response_to_Clutch_Speed_ Requested_by_Transmission.jpg	.jpg-file	Picture explaining the Transmission requirements to response time to Clutch Speed requested by Transmission.
Transmis- sion_Requirement_for_Response_to_ Minimum_Clutch_Speed_Request.jpg	.jpg-file	Picture explaining the Transmission requirements to Minimum Clutch Speed signal response time.
Transmission_Requirement_for_ Response_to_TrsmTqCluFast_ Torque_Decrease_Request.jpg	.jpg-file	Picture explaining the Transmission requirements to the response time to fast Torque at Clutch decrease request.

▽

△

File	File Type	Content/Use Case
Transmission_Requirement_for_ Response_to_TrsmTqCluFast_ Torque_Increase_Request.jpg	.jpg-file	Picture explaining the Transmission requirements to the response time to fast Torque at Clutch increase request.
Transmission_Requirement_for_ Response_to_TrsmTqCluMaxProtn_ Gearbox_Protection.jpg	.jpg-file	Picture explaining the Transmission requirements to the Torque at Clutch Maximum for Gearbox Protection.
Transmission_Requirement_for_ Response_to_TrsmTqCluResv_ Torque_Reserve_ Torque_Decrease_Request.jpg	.jpg-file	Picture explaining the Transmission requirements to the response time to Torque at Clutch Reserve Decrease request.
Transmission_Requirement_for_ Response_to_TrsmTqCluResv_ Torque_Reserve_ Torque_Increase_Request.jpg	.jpg-file	Picture explaining the Transmission requirements to the response time to Torque at Clutch Reserve Increase request.
xml.xsd	.xsd-file	XML-Style Sheet describing the properties of the elements used by the .arxml files.

Table 11.1: Structure of AUTOSAR_CP_MOD_AISpecification

11.1.2.3 Contents of the AUTOSAR_CP_MOD_AISpecification_Examples.zip

File	File Type	Content Use Case
AUTOSAR_CC_ AISpecificationExamples.xml	.xml-file	Collection of all collections contained in the AUTOSAR_MOD_AISpecification_Examples; includes all information needed for the definition of Example elements
AUTOSAR_MOD_ AISpecificationExamples.arxml	.arxml-file	Collection of all definitions for all Example elements including some copies of the BluePrint elements which are used for the definition of the Example elements
AUTOSAR_MOD_ AISpecificationExamples_ LifeCycle_Standard.arxml	.arxml-file	Collection of Life Cycle information for the Example elements
AUTOSAR_MOD_ SwDesignPatternExamples.arxml	.arxml-file	Collection of Examples for the application of Naming Patterns as described in AUTOSAR_CP_TR_AIDesignPatternsCatalogue.pdf
AUTOSAR_MOD_ SwDesignPatternExamples_ LifeCycle.arxml	.arxml-file	Collection of Life Cycle information for the Naming Pattern Examples
_disclaimer.txt	.txt-file	Collection of legal regulations
_readme.txt	.txt-file	Overview of contents and additional legal regulations
catalog_V3_0_0.ml.xsd	.xsd-file	Description of the XML-Schema for the .arxml export
xml.xsd	.xsd-file	XML-Style Sheet describing the properties of the elements used by the .arxml files

Table 11.2: Structure of AUTOSAR_CP_MOD_AISpecification_Examples.zip

For more details please refer to the readme.txt file under the Application Interfaces deliverables. The AUTOSAR_CC_AISpecification.xml catalog file is used to help resolve references in case needed by specific tools.

11.1.2.4 Definitions of Physical Dimensions and Units

Since release R19-11 the official definitions of Units and Physical Dimensions are maintained by the AI-Tool but they are generated into AUTOSAR_MOD_GeneralBlueprints.zip. Since then the MethodologyAndTemplate section of the official Foundation Platform (FO) Release on SVN repository (//svn.autosar.org/repos/work/26_Standards/02_Releases/00_FO/R25-11/MethodologyAndTemplates) provides besides others the AUTOSAR_MOD_GeneralBlueprints.zip archive which contains according to the General Blueprints Supplement document ([17]) the follwing elements:

File	File Type	Content Use Case
AUTOSAR_MOD_ PhysicalDimensions_Blueprint.arxml	.arxml-file	Collection of Blueprint definitions of Physical Dimensions used for the definition of Physical Units
AUTOSAR_MOD_ Units_Blueprint.arxml	.arxml-file	Collection of Blueprint definitions of Physical Units
AUTOSAR_MOD_ BswDataTypes_Blueprint.arxml	.arxml-file	Collection of Data Type definitions used for the Implementation of BSW Modules
AUTOSAR_MOD_ BswModuleEntrys_Blueprint.arxml	.arxml-file	Collection of standardized API entries (C-functionprototypes) into the BSW module or cluster
AUTOSAR_MOD_ BswServiceDataTypes_ Blueprint.arxml	.arxml-file	Collection of Data Type definitions used for the Implementation of BSW Services
AUTOSAR_MOD_ BswServiceInterfaceMappings_ Blueprint.arxml	.arxml-file	Collection of blueprints of the mapping per client-server-interface
AUTOSAR_MOD_ BswServiceInterfaces_ Blueprint.arxml	.arxml-file	Collection of blueprinted specifications of standardized AUTOSAR Interfaces for BSW Modules
AUTOSAR_MOD_ CommonDataTypes_ Blueprint.arxml	.arxml-file	Collection of blueprints of the Base Types, CompuMethods and ImplementationDataTypes for Platform, Standard and General Definitions
AUTOSAR_MOD_ Cube_RecordLayouts_ Blueprint.arxml	.arxml-file	Collection of blueprints of SwRecordLayouts for cuboids
AUTOSAR_MOD_ IFL_RecordLayouts_ Blueprint.arxml	.arxml-file	Collection of blueprints of the floating point InterpolationRoutineMappingSets and SwRecordLayouts
AUTOSAR_MOD_ IFX_RecordLayouts_ Blueprint.arxml	.arxml-file	Collection of blueprints of the fix point InterpolationRoutineMappingSets and SwRecordLayouts
AUTOSAR_MOD_ MemoryMapping_ SwAddrMethods_ Blueprint.arxml	.arxml-file	Collection of blueprints of the SwAddrMethods
AUTOSAR_MOD_ SWCServiceRelatedInterfaces_ Blueprint.arxml	.arxml-file	Collection of blueprints of the ClientServerInterfaces derived from the Standardized AUTOSAR Interfaces of the NVRAM Manager
AUTOSAR_MOD_ ValBlk_SwRecordLayouts_ Blueprint.arxml	.arxml-file	Collection of blueprints of SwRecordLayouts for Value and Valueblocks
AUTOSAR_TP_ FormulaLanguage_TestCases_ Blueprint.arxml	.arxml-file	Collection of various predefined test cases to validate the formula language expressions
AUTOSAR_TR_ PredefinedNames_ Blueprint.arxml and FO_TR_PredefinedNames	.arxml-file	Were removed in R25-11
xml.xsd	.xsd-file	XML-Style Sheet describing the properties of the elements used by the .arxml files

Table 11.3: Structure of AUTOSAR_FO_MOD_GeneralBlueprints.zip

Please note that the files AUTOSAR_MOD_PhysicalDimensions_Blueprint.arxml, and AUTOSAR_MOD_Units_Blueprint.arxml will be released by the AUTOSAR Foundation releases but they are part of Application Interfaces delivery.

11.1.3 Schema Structure

In order to understand the XML generation, it is necessary to understand the relation between meta-model and schema. Generally, the schema contains an xsd:group for each class of the meta-model. The group contains all attributes of the class, including aggregations and references as sequence of xsd:element. Concrete classes (in contrast to abstract classes) also have a corresponding xsd:complexType. These are sequences of all inherited groups from parent elements. The general concept behind the structure of the schema will be described according to the example depicted in the following diagram, which shows the structure of the Unit element in the meta-model, including its inheritance hierarchy.

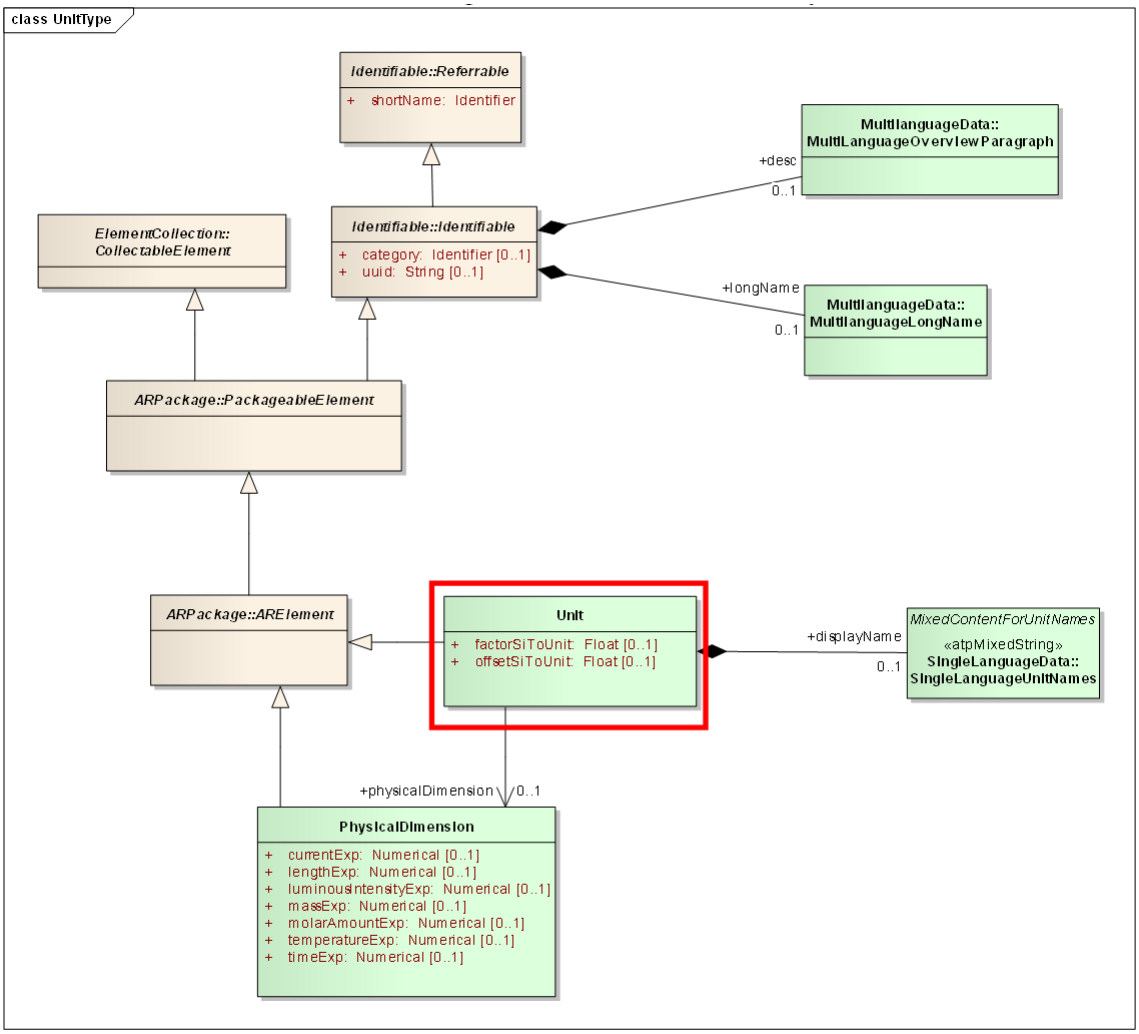


Figure 11.2: Meta-model defining the structure of the element Unit

This structure can be found in the following elements in the AUTOSAR schema:

```
<xsd:group name="UNIT">
  <xsd:annotation>...</xsd:annotation>
  <xsd:sequence>
    <xsd:element
      name="DISPLAY-NAME"
      type="AR:SINGLE-LANGUAGE-UNIT-NAMES"
      minOccurs="0"/>
    <xsd:element
      name="FACTOR-SI-TO-UNIT"
      type="xsd:double" minOccurs="0"/>
    <xsd:element
      name="OFFSET-SI-TO-UNIT"
      type="xsd:double" minOccurs="0"/>
    <xsd:element
      name="PHYSICAL-DIMENSION-REF"
      minOccurs="0">
    ...
    </xsd:element>
  </xsd:sequence>
</xsd:group>
<!-- complex type for class Units::Unit -->
<xsd:complexType name="UNIT" abstract="false" mixed="false">
  <xsd:sequence>
    <xsd:group ref="AR:AR-OBJECT"/>
    <xsd:group ref="AR:REFERRABLE"/>
    <xsd:group ref="AR:IDENTIFIABLE"/>
    <xsd:group ref="AR:COLLECTABLE-ELEMENT"/>
    <xsd:group ref="AR:PACKAGEABLE-ELEMENT"/>
    <xsd:group ref="AR:AR-ELEMENT"/>
    <xsd:group ref="AR:UNIT"/>
  </xsd:sequence>
  ...
</xsd:complexType>
```

The structuring into groups (for all classes including abstract ones) and complex types (for concrete classes only) leads to the particularity, that only concrete classes can be used on XML instance level, and that the inheritance hierarchy is not visible on instance level. The following example shows a unit on instance level, having only attributes from Identifiable:

```
<UNIT>
  <SHORT-NAME>Gr</SHORT-NAME>
  <LONG-NAME>
    <L-4 L="EN">Gram</L-4>
  </LONG-NAME>
  <DISPLAY-NAME>g</DISPLAY-NAME>
  <FACTOR-SI-TO-UNIT>1000.0</FACTOR-SI-TO-UNIT>
  <OFFSET-SI-TO-UNIT>0.0</OFFSET-SI-TO-UNIT>
  <PHYSICAL-DIMENSION-REF BASE="PhysicalDimensions" DEST="PHYSICAL-
     DIMENSION">M1</PHYSICAL-DIMENSION-REF>
</UNIT>
```

Details on the relation of meta-model and AUTOSAR Schema can be found in the Model Persistence Rules for XML [18].

11.2 PortPrototypeBlueprints

It is not in the scope of the AUTOSAR Application Interfaces to define complete system compositions. All software component composition types are defined in a package with category EXAMPLE and meant only as illustration of usage of the standardized elements. However, it is in the scope of the Application Interfaces to describe the roles that interfaces can play in compositions. This can be done using PortPrototypeBlueprints, which define potential ports of a component type and can carry more attributes to pre-define values for usages of the blueprint, e.g. an initial value. For details on PortPrototypeBlueprints see the Standardization Template [5].

The PortPrototypeBlueprints are collected within a single package /AUTOSAR/AISpecification/PortPrototypeBlueprints_Blueprint:

```
<AR-PACKAGE>
  <SHORT-NAME>PortPrototypeBlueprints_Blueprint</SHORT-NAME>
  <CATEGORY>BLUEPRINT</CATEGORY>
  <REFERENCE-BASES>...</REFERENCE-BASES>
  <ELEMENTS>
    <PORT-PROTOTYPE-BLUEPRINT>
    <SHORT-NAME NAME-PATTERN="{anyName}"> AbsCtrlIntvg</SHORT-NAME>
    <LONG-NAME><L-4 L="EN">ABS Control Intervening</L-4></LONG-NAME>
    <DESC>
      <L-2 L="EN">Antilock Braking Systems (ABS) control is active (at
         least one wheel)</L-2>
    </DESC>
    <INTERFACE-REF DEST="SENDER-RECEIVER-INTERFACE" BASE="PortInterfaces">
       AbsCtrlIntvg1</INTERFACE-REF>
  </PORT-PROTOTYPE-BLUEPRINT>
  <PORT-PROTOTYPE-BLUEPRINT>
    <SHORT-NAME NAME-PATTERN="{anyName}">AbsFlgActv</SHORT-NAME>
    <LONG-NAME>
      <L-4 L="EN">AbsControlActive</L-4>
    </LONG-NAME>
    <DESC>
      <L-2 L="EN">Anti Blocking Systems (ABS) control is active (at least
         one wheel)</L-2>
    </DESC>
    <INTERFACE-REF DEST="SENDER-RECEIVER-INTERFACE" BASE="PortInterfaces">
       AbsCtrlIntvg1</INTERFACE-REF>
  </PORT-PROTOTYPE-BLUEPRINT>
  ...
  </ELEMENTS>
</AR-PACKAGE>
```

11.3 PortInterfaces

Each provided or required PortPrototype references a PortInterface defined in the AI Tool.

Both sender-receiver-interface definitions and client-server-interface definitions are maintained in the package /AUTOSAR/AISpecification/PortInterfaces_Blueprint. They

are also part of /AUTOSAR/AISpecification/PortInterfaces_Example, but only as a copy of the blueprint so that they could be used for PortPrototypes.

11.3.1 Sender-Receiver-Interface

Figure 11.3 shows an sender-receiver-interface description which defines short-name, long-name, description, the contained data elements, and the derived EXAMPLE-Element. The direction of the dataflow is defined by the port.

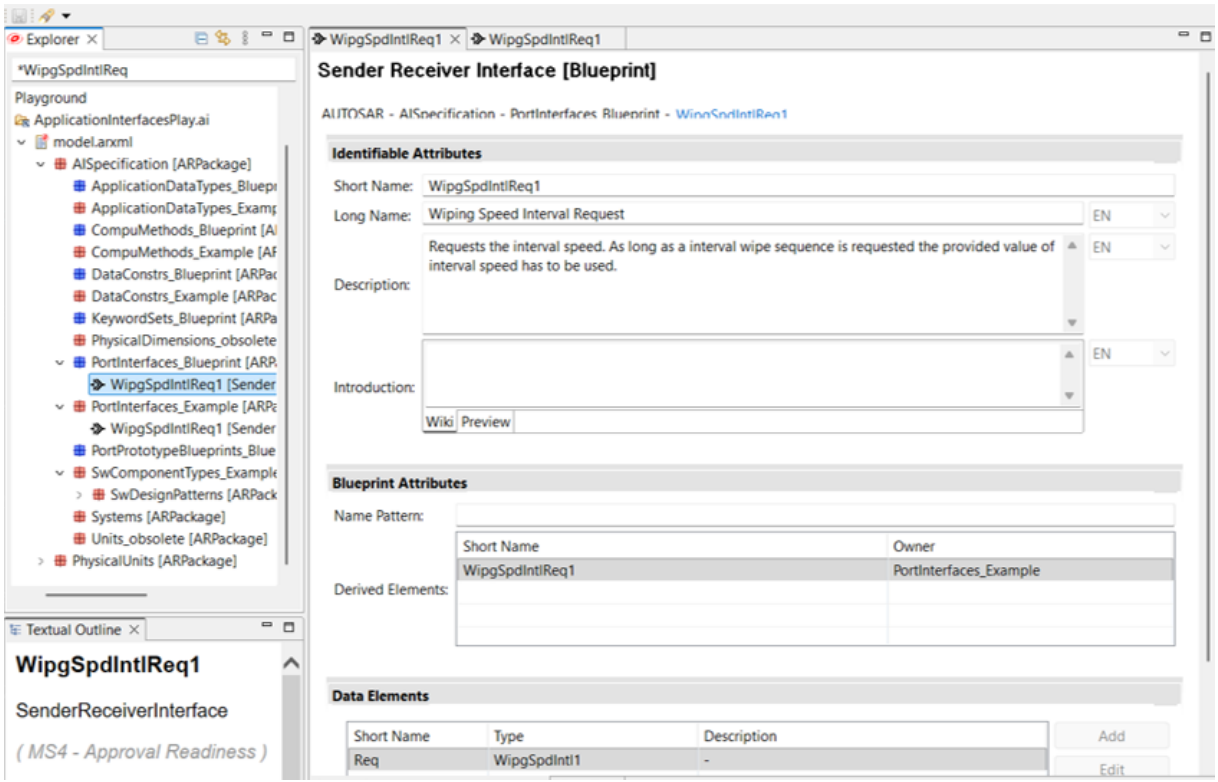


Figure 11.3: Structure of the Interface definitions

The XML generator generates the following output for the above Interface definitions:

```
<SENDER-RECEIVER-INTERFACE>
  <SHORT-NAME NAME-PATTERN="{anyName}">WipgSpdIntlReq1</SHORT-NAME>
  <LONG-NAME>
    <L-4 L="EN">Wiping Speed Interval Request</L-4>
  </LONG-NAME>
  <DESC>
    <L-2 L="EN">Requests the interval speed. As long as a interval wipe
       sequence is requested the provided value of interval speed has to be
        used.</L-2>
  </DESC>
  <IS-SERVICE>false</IS-SERVICE>
  <DATA-ELEMENTS>
    <VARIABLE-DATA-PROTOTYPE>
      <SHORT-NAME NAME-PATTERN="{anyName}">Req</SHORT-NAME>
      <TYPE-TREF DEST="APPLICATION-PRIMITIVE-DATA-TYPE" BASE="
          ApplicationDataTypes">WipgSpdIntl1</TYPE-TREF>
```

```
    </VARIABLE-DATA-PROTOTYPE>
  </DATA-ELEMENTS>
</SENDER-RECEIVER-INTERFACE>
```

The corresponding BluprintMapping for the above interface is:

```
<BLUEPRINT-MAPPING-SET>
  <SHORT-NAME>PortInterfaceBlueprintMappings</SHORT-NAME>
  <BLUEPRINT-MAPPING>
    <BLUEPRINT-REF DEST="SENDER-RECEIVER-INTERFACE" BASE="
       PortInterfaces_Blueprint">WipgSpdIntlReq1</BLUEPRINT-REF>
    <DERIVED-OBJECT-REF DEST="SENDER-RECEIVER-INTERFACE" BASE="
       PortInterfaces">WipgSpdIntlReq1</DERIVED-OBJECT-REF>
  </BLUEPRINT-MAPPING>
```

The Blueprint and Derived element is represented in the above XML extract. This mapping shows that the package /AUTOSAR/AISpecification/PortInterfaces_Blueprint provide the blueprint interfaces which are derived within the EXAMPLE categorized package /AUTOSAR/AISpecification/PortInterfaces_Example.

The information on queuing and signal qualifiers is currently not used for XML generation. The referenced type for each data element must be defined in the DataType sheets.

11.4 Life Cycle State

The Life Cycle information is collected in the View "Metadata" of the element editors in the AI-Tool as shown below alongwith the alternative to be used and expiry date.

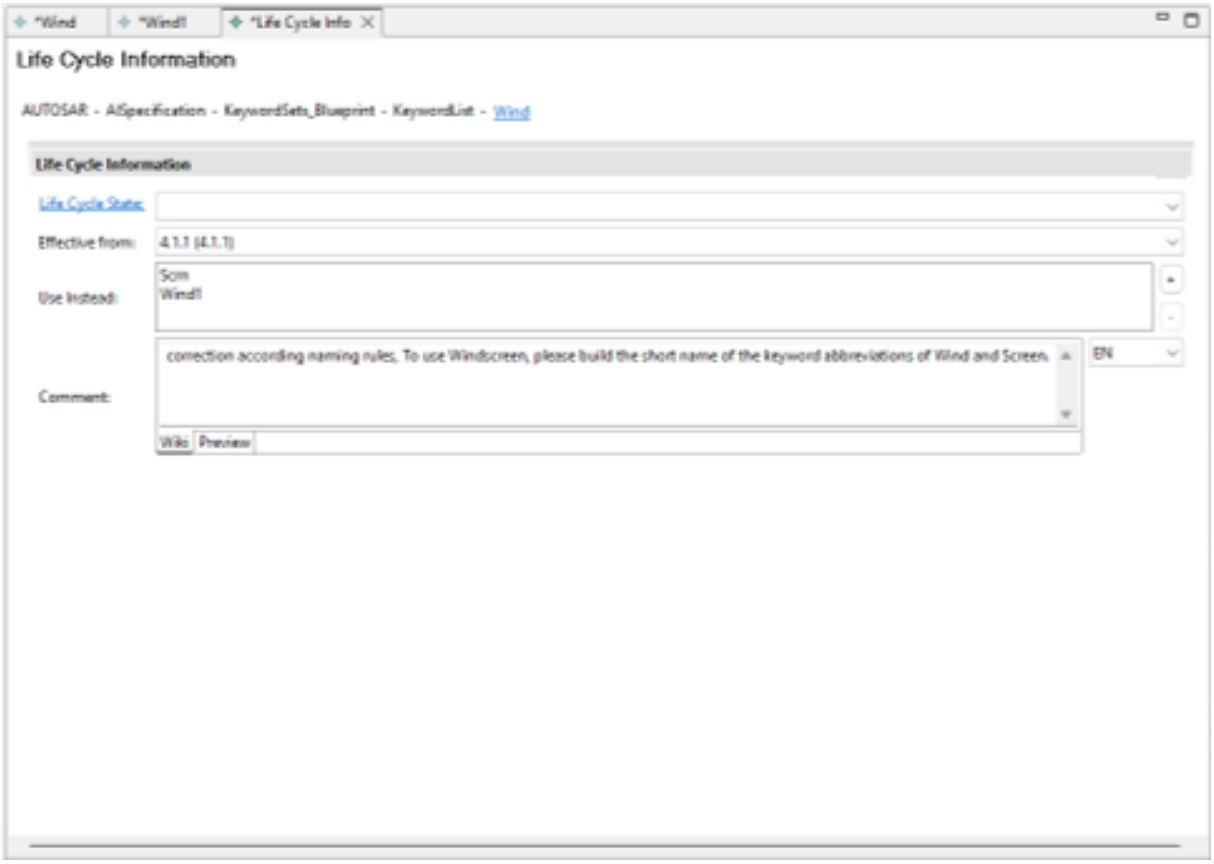


Figure 11.4: Life Cycle State definition in AI-Tool

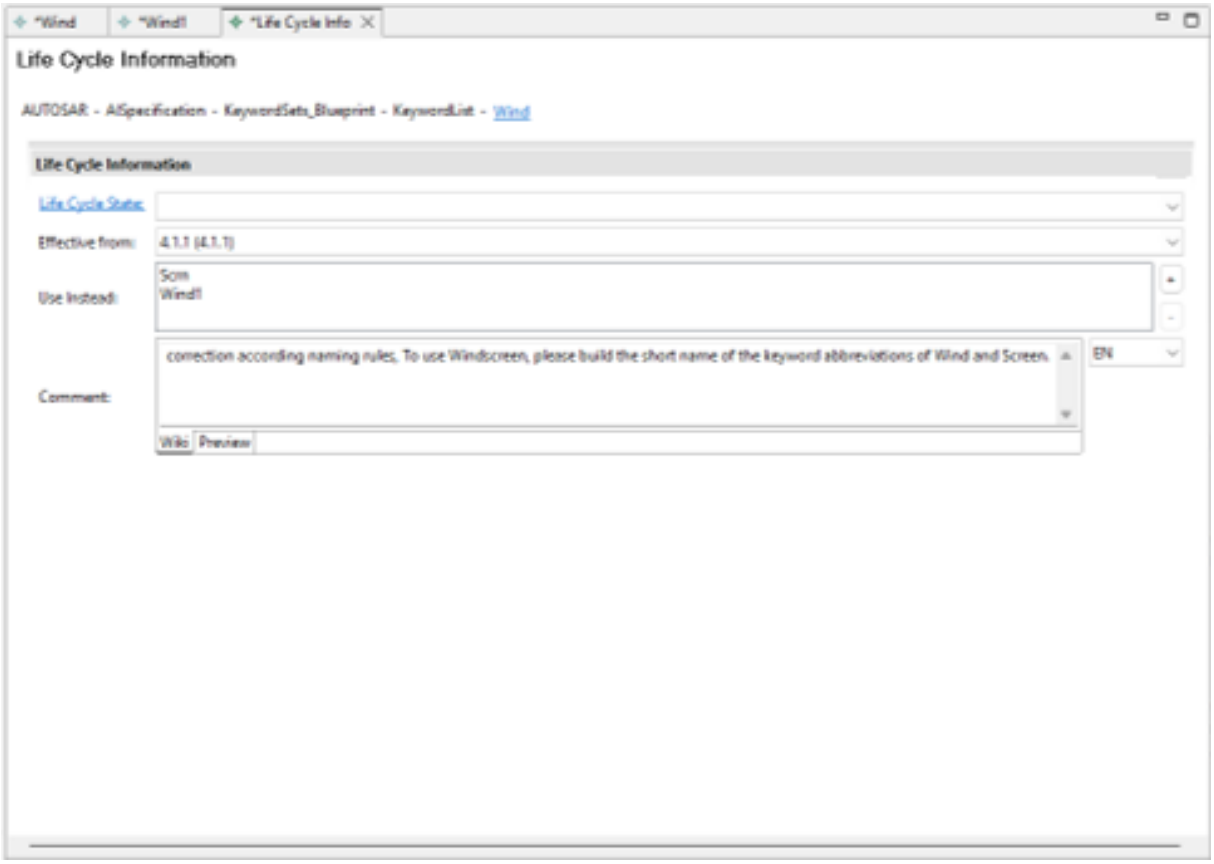


Figure 11.5: Life Cycle State definition in AI-Tool - Tabular View

The Life Cycle State of these elements in AI-Tool is output in the .arxml file AUTOSAR_MOD_AISpecification_XXElementTypeXX_Standard_LifeCycle.arxml

There is one Life Cycle info set per model element grouped under the respective categories under the LifeCycleInfoSets . The Obsolete elements under each category is labeled <category>Obslt (e.g KeywordObslt, ApplDataTypObslt, DataConstrObslt, CompuMethodObslt, PortIfObslt, PortPrototypeBlueprintObslt). Both the spellings "obsolete" and "Obsolete" are recognized by the XML generator and the DEFAULT-LC-STATE-REF points to "obsolete" as seen below.

For the above example, the Keyword element "Wind" is set to obsolete under the BASE "KeywordSets" as seen in the .arxml extract below.

The PERIOD-BEGIN is used describe the expiry date of the element (R4.1.1 in the above case) i.e. the fist AUTOSAR release for which the respective element was set to "obsolete".

The Comment and Use Instead columns translates to the REMARK and USE-INSTEAD sections respectively in the XML description.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<AUTOSAR xmlns="http://autosar.org/schema/r4.0" xmlns:xsi="http://www.w3.
    org/2001/XMLSchema-instance" xsi:schemaLocation="http://autosar.org/
    schema/r4.0 AUTOSAR_00054.xsd">
  <AR-PACKAGES>
```

```
<AR-PACKAGE>
  <SHORT-NAME>AUTOSAR</SHORT-NAME>
  <LONG-NAME>
    <L-4 L="EN">AUTOSAR</L-4>
  </LONG-NAME>
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>AISpecification</SHORT-NAME>
      <LONG-NAME>
        <L-4 L="EN">AISpecification</L-4>
      </LONG-NAME>
      <AR-PACKAGES>
        <AR-PACKAGE>
          <SHORT-NAME>LifeCycleInfoSets</SHORT-NAME>
          <CATEGORY>STANDARD</CATEGORY>
          <REFERENCE-BASES>
            <REFERENCE-BASE>
              <SHORT-LABEL>KeywordSets</SHORT-LABEL>
              <PACKAGE-REF DEST="AR-PACKAGE">
                /AUTOSAR/AISpecification/KeywordSets_Blueprint
              </PACKAGE-REF>
            </REFERENCE-BASE>
          </REFERENCE-BASES>
          <ELEMENTS>
            <LIFE-CYCLE-INFO-SET>
              <SHORT-NAME>KeywordObslt</SHORT-NAME>
              <DEFAULT-LC-STATE-REF DEST="LIFE-CYCLE-STATE">
                /AUTOSAR/GenDef/LifeCycleStateDefinitionGroups/
                   AutosarLifeCycleStates/obsolete
              </DEFAULT-LC-STATE-REF>
              <LIFE-CYCLE-INFOS>
                <LIFE-CYCLE-INFO>
                  <LC-OBJECT-REF BASE="KeywordSets" DEST="KEYWORD">
                    KeywordList/Wind
                  </LC-OBJECT-REF>
                  <PERIOD-BEGIN>
                    <AR-RELEASE-VERSION>4.1.1</AR-RELEASE-VERSION>
                  </PERIOD-BEGIN>
                  <REMARK>
                    <P>
                     <L-1 L="EN">correction according naming rules, To
                        use Windscreen,
                                 please build the short name of the
                                    keyword abbreviations
                                 of Wind and Screen.
                     </L-1>
                    </P>
                  </REMARK>
                  <USE-INSTEAD-REFS>
                    <USE-INSTEAD-REF BASE="KeywordSets" DEST="KEYWORD">
                      KeywordList/Wind1
                    </USE-INSTEAD-REF>
                    <USE-INSTEAD-REF BASE="KeywordSets" DEST="KEYWORD">
                      KeywordList/Scrn
                    </USE-INSTEAD-REF>
                  </USE-INSTEAD-REFS>
```

```
                    </LIFE-CYCLE-INFO>
                    <LIFE-CYCLE-INFO>
                      ...
                    </LIFE-CYCLE-INFO>
                  </LIFE-CYCLE-INFO-SET>
                </ELEMENTS>
              </REFERENCE-BASES>
            </AR-PACKAGE>
          </AR-PACKAGES>
        </AR-PACKAGE>
      </AR-PACKAGES>
    </AR-PACKAGE>
  </AR-PACKAGES>
</AUTOSAR>
```

Similarly the Life Cycle state of other elements (e.g PortInterfaces, PortPrototypes,etc..) are also generated in the XML and the corresponding BASE is referenced. The elements marked obsolete will not appear in the Examples package or BlueprintMappings.

In certain cases, there may be multiple entries under the attribute "Use Instead" (see Figure 11.4)

For such situations, it is possible to enter more than one line of "Use instead" information. In the above example the Keyword Short Name"Wind" is rendered obsolete for "Windscreen" and instead needs to be constructed from the abbreviations for "Wind" and "Screen" (Scrn) respectively.

11.5 Blueprint Mapping Sets

The blueprint mechanism is meant as an aid for the creation of PortPrototypes the AUTOSAR XML also provides a mapping mechanism which describes the relation between a blueprint and a prototype. This mechanism allows decoupling of the PortPrototype from the blueprint without interfering with architectural requirements.

BlueprintMappingSets are used to establish a connection between a Blueprint and elements derived from this Blueprint. This helps to trace back the respective Blueprint that was used to create this element. Blueprint Mapping Sets are used for different elements including PortPrototypeBlueprints, PortInterfaces, Application DataTypes, etc. They are defined in the package /AUTOSAR/AISpecification/BlueprintMappingSets_Example. The following .arxml snippet provides an example for the Blueprint Mapping of a PortInterface and is equivalent for other elements.

```
<BLUEPRINT-MAPPING-SET>
  <SHORT-NAME>PortInterfaceBlueprintMappings</SHORT-NAME>
  <BLUEPRINT-MAPS>
    <BLUEPRINT-MAPPING>
      <BLUEPRINT-REF DEST="CLIENT-SERVER-INTERFACE" BASE="
          PortInterfaces_Blueprint">
        TrsmRatGear1
      </BLUEPRINT-REF>
```

```
      <DERIVED-OBJECT-REF DEST="CLIENT-SERVER-INTERFACE" BASE="
         PortInterfaces">
        TrsmRatGear1
      </DERIVED-OBJECT-REF>
    </BLUEPRINT-MAPPING>
    <BLUEPRINT-MAPPING>
      <BLUEPRINT-REF DEST="SENDER-RECEIVER-INTERFACE" BASE="
         PortInterfaces_Blueprint">
        ALgt2
      </BLUEPRINT-REF>
      <DERIVED-OBJECT-REF DEST="SENDER-RECEIVER-INTERFACE" BASE="
         PortInterfaces">
        ALgt2
      </DERIVED-OBJECT-REF>
    </BLUEPRINT-MAPPING>
    ...
  </BLUEPRINT-MAPS>
</BLUEPRINT-MAPPING-SET>
```

This mapping shows that the package /AUTOSAR/AISpecification/PortInterfaces_Blueprint provides the blueprint interfaces which are derived within the EXAMPLE categorized package /AUTOSAR/AISpecification/PortInterfaces_Example.

Similarly for the DataConstraints the Blueprint Mapping is as under:

```
<BLUEPRINT-MAPPING-SET>
  <SHORT-NAME>DataConstrBlueprintMappings</SHORT-NAME>
  <BLUEPRINT-MAPS>
    <BLUEPRINT-MAPPING>
      <BLUEPRINT-REF DEST="DATA-CONSTR" BASE="DataConstrs_Blueprint">
        TrsmTyp1</BLUEPRINT-REF>
      <DERIVED-OBJECT-REF DEST="DATA-CONSTR" BASE="DataConstrs">TrsmTyp1</
        DERIVED-OBJECT-REF>
    </BLUEPRINT-MAPPING>
    ...
  </BLUEPRINT-MAPS>
</BLUEPRINT-MAPPING-SET>
```