Document Title	Modeling Show Cases Report
Document Category	Technical report (TR)
Document UID	789
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	• No content changes
2025-11-27	R25-11	AUTOSAR Release Management	• Added Integration Requirements description
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	• Add Show case Structured Requirement • Changed Document Status from Final to published
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	• 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 Introduction 8

2 Overview 9

3 Measurement and Calibration 10

3.1 Introductory Show Case . . . 11
3.1.1 Physical System . . . 11
3.1.1.1 Components Overview . . . 12
3.1.1.2 The Environment . . . 13
3.1.1.3 The Plant . . . 14
3.1.1.4 The Controller . . . 15
3.1.2 AUTOSAR Modeling . . . 16
3.1.3 RTE Generation, Measurement and Calibration . . . 17
3.1.3.1 FlatMap . . . 18
3.1.3.2 ECU Documentation, Measurement and Calibration . . . 18
3.1.4 A2L File . . . 19
3.1.5 Implementation in C . . . 19
3.1.6 A walk with T_Plant through the Show Case . . . 22
3.1.6.1 Physical System . . . 22
3.1.6.1.1 Components . . . 22
3.1.6.1.2 Equations . . . 23
3.1.6.2 AUTOSAR Modeling . . . 23
3.1.6.2.1 Physical Dimension and Unit . . . 23
3.1.6.2.2 Application Data Type . . . 25
3.1.6.2.3 Port Interface . . . 28
3.1.6.2.4 Software Components . . . 29
3.1.6.3 System . . . 34
3.1.6.4 ECU Configuration . . . 36
3.1.6.5 RTE Generation . . . 38
3.1.6.6 Implementation in C . . . 39
3.1.6.7 A2L File . . . 40
3.1.6.8 Measurement and Calibration Tool . . . 41
3.1.7 Show cases in the Example . . . 42
3.1.7.1 CompositionSwComponentTypes . . . 42
3.1.7.2 ParameterSwComponentTypes . . . 43
3.1.7.3 ApplicationSwComponentTypes . . . 44
3.1.7.4 ParameterInterfaces . . . 50
3.1.7.5 SenderReceiverInterfaces . . . 53
3.1.7.6 ApplicationDataTypes, Category VALUE . . . 54
3.1.7.7 Units . . . 57
3.1.7.8 PhysicalDimensions . . . 59
3.1.7.9 SwAddrMethods . . . 62

3.2 Advanced Show Case . . . 63
3.2.1 General Objectives of the Model Structure . . . 63
3.2.1.1 The Ecu Description . . . 63
3.2.1.2 The Ecu Extract . . . 63
3.2.1.2.1 The ECU Flat Map . . . 63
3.2.1.3 Data Types and Data Objects . . . 64
3.2.1.4 Axis, Curves and Maps . . . 65
3.2.1.5 Axis, Curves and Maps on ApplicationDataType level . . . 65
3.2.1.6 Axis, Curves and Maps on DataPrototype and SwComponent-Prototype level . . . 67
3.2.1.6.1 Instantiation of Axis, Curves and Maps . . . 67
3.2.1.6.2 Linking map and curve instances to its axes instances . . . 68
3.2.1.6.3 Linking axes instances to its working point instances . . . 69
3.2.1.6.4 Axis, Curves and Maps in the ECU Flat Map . . . 71
3.2.1.7 Arrays of Maps and Axes . . . 72
3.2.1.7.1 Arrays of Maps and Axes in the ECU Flat Map . . . 72
3.2.1.8 Measurement of Modes . . . 73
3.2.1.8.1 Enabling Measurement of Modes . . . 73
3.2.1.8.2 Modes in the ECU Flat Map . . . 73
3.2.2 Show cases in the Example . . . 75
3.2.2.1 CompositionSwComponentTypes . . . 75
3.2.2.2 ParameterSwComponentTypes . . . 80
3.2.2.3 ApplicationSwComponentTypes . . . 82
3.2.2.4 ParameterInterfaces . . . 85
3.2.2.5 ModeSwitchInterfaces . . . 87
3.2.2.6 SenderReceiverInterfaces . . . 88
3.2.2.7 ApplicationDataTypes, Category BOOLEAN . . . 90
3.2.2.8 ApplicationDataTypes, Category VALUE . . . 90
3.2.2.9 ApplicationDataTypes, Category COM_AXIS . . . 94
3.2.2.10 ApplicationDataTypes, Category CURVE . . . 95
3.2.2.11 ApplicationDataTypes, Category MAP . . . 96
3.2.2.12 ApplicationArrayDataTypes . . . 96
3.2.2.13 ApplicationRecordDataTypes . . . 97
3.2.2.14 ModeDeclarationGroups . . . 98
3.2.2.15 Units . . . 98
3.2.2.16 PhysicalDimensions . . . 100
3.2.2.17 SwAddrMethods . . . 101
4 Structured Requirements 103
4.1 Specification items as requirements . . . 103
4.2 Diagnostic requirements . . . 104
4.3 Decomposition of requirements . . . 105

5 Integration Requirements 108
5.1 Defining Integration Requirements using StructuredReq 109
5.1.1 StructuredReq.shortName 109
5.1.2 StructuredReq.category 110
5.1.3 StructuredReq.importance 110
5.1.4 StructuredReq.rationale 110
5.1.5 StructuredReq.useCase 110
5.1.6 StructuredReq.description 110
5.1.6.1 DefinitionReference 111
5.1.6.2 Description 112
5.1.6.3 ARVersion 112
5.1.6.4 ModuleVersion 112
5.1.6.5 SetValue 112
5.1.6.6 SupportedLowValue 113
5.1.6.7 SupportedHighValue 113
5.1.6.8 SupportedValue 113
5.2 Applying Integration Requirements 114
5.2.1 Single configuration value in a single container 114
5.2.2 Single configuration value in multiple containers 118
5.2.3 Multiple configuration values in a container 121
5.2.4 Tracing configuration changes 121
5.3 Ranges in Integration Requirements 122
5.4 Possible Conflicts caused by applied Integration Requirements 122
5.4.1 Merge of Integration Requirements 123
5.4.1.1 No merge of fixed values 123
5.4.1.2 Merge of ranges 123
5.4.1.3 Order dependency between suggested and mandatory integration requirements 125
5.5 Reporting 125
A Mentioned Class Tables 127

References

[1] Methodology for Classic Platform
AUTOSAR_CP_TR_Methodology

[2] Modeling Show Cases Examples
AUTOSAR_CP_EXP_ModelingShowCases

[3] Software Component Template
AUTOSAR_CP_TPS_SoftwareComponentTemplate

[4] Standardization Template
AUTOSAR_FO_TPS_StandardizationTemplate

[5] Specification of Platform Types for Classic Platform
AUTOSAR_CP_SWS_PlatformTypes

1 Introduction

The objective of this report is the illustration and execution of AUTOSAR modeling and the AUTOSAR methodology (see [1]) for selected show cases.

Each show case focuses on a few specific topics and gives an overview of their basic usage and their application in the field. Where appropriate, the show cases are based on real world applications of the AUTOSAR standard.

It contains

- explanatory background on the functional use case for which the specific part of the AUTOSAR modeling is applied.
- illustration of the AUTOSAR model content in form of interlinked tables
- explanation of the processing results of these AUTOSAR models (e.g. C code, A2L files, ...)
- snippets of the full-blown examples. The complete examples are provided in the archive `AUTOSAR_EXP_ModelingShowCases.zip` [2].

2 Overview

The report is organized in chapters according to the main focus of the contained show cases. Each chapter contains a topic specific overview and at least one show case. Each chapter is self contained and understandable without reading any other chapter.

The technical report on the AUTOSAR Methodology [1] deserves a special mentioning as accompanying document for going through the show cases.

In the first version of the technical report, the show cases are targeting the topic of measurement and calibration, involving the creation of A2L files based on AUTOSAR models. For these show cases, also the specification of the `SoftwareComponentTemplate` [3] is a good accompanying document.

This updated version is extended by an additional show case targeting the topic of Structured Requirements. For these show cases, also the specification of the `StandardizationTemplate` [4] is a good accompanying document.

3 Measurement and Calibration

Measurements and Calibration (short: MC) is a major step in the development of electronic control units (ECUs). Measurement and Calibration systems (MC systems), involving software tools (MC tools) as well as the hardware to access an ECU (not in focus here), enable the developer to measure variables and to adapt calibration parameters (or "characteristics") during the run-time of the ECU.

For instance, the following tasks are regularly done by "Measurement and Calibration"

- Adaptation to real hardware (e.g. inserting the electrical characteristics of a sensor)
- Calibration of controllers (e.g. adjusting the parameters of a closed loop controller)
- Tuning of ECU internal environment models (e.g. for "virtual sensors")
- Validation of ECU functions
- Tracking of development errors
- Collecting data for automated optimization of parameters

The "Introductory Show Case" (see 3.1), illustrates all basic artifacts on the way from a physical system that is to be controlled by an ECU until measuring and calibrating with a MC system.

As didactic simplification only a few data types were used, e.g. neither CURVEs[1] nor MAPs[2] were chosen, nor any ApplicationCompositeDataType.

However, those advanced topics, their modeling in AUTOSAR as well as their transfer to a MC tool, is of particular interest: they are regularly needed and used the field. Therefore, the "Advanced Show Case" in chapter 3.2 especially highlights these topics. This show case is directly derived from the real world modeling and structuring approach of a major Tier 1 in the powertrain domain. So it also illustrates "good practices" in the field for designing AUTOSAR systems which are to be measured and calibrated later on in their development.

[1] CURVEs are two dimensional functions defined via axis points and the corresponding function values. Interpolation or extrapolation is used to calculate function values that are not directly defined.

[2] MAPs are similar to CURVEs but three dimensional

3.1 Introductory Show Case

As introduction to measurement and calibration with AUTOSAR a simple, artificial closed-loop control system was chosen. This allows interesting feedback of the system when using a MC tool. At the same time, the model, the source code and generated files are still comprehensible.

A drawback is, that not all typical "real world" data types are featured, for instance. Such topics are covered in the "Advanced Show Case", chapter 3.2.

3.1.1 Physical System

This section contains a description of the physical system setup. It can safely be skipped, if only the AUTOSAR modeling itself is of interest to the reader.

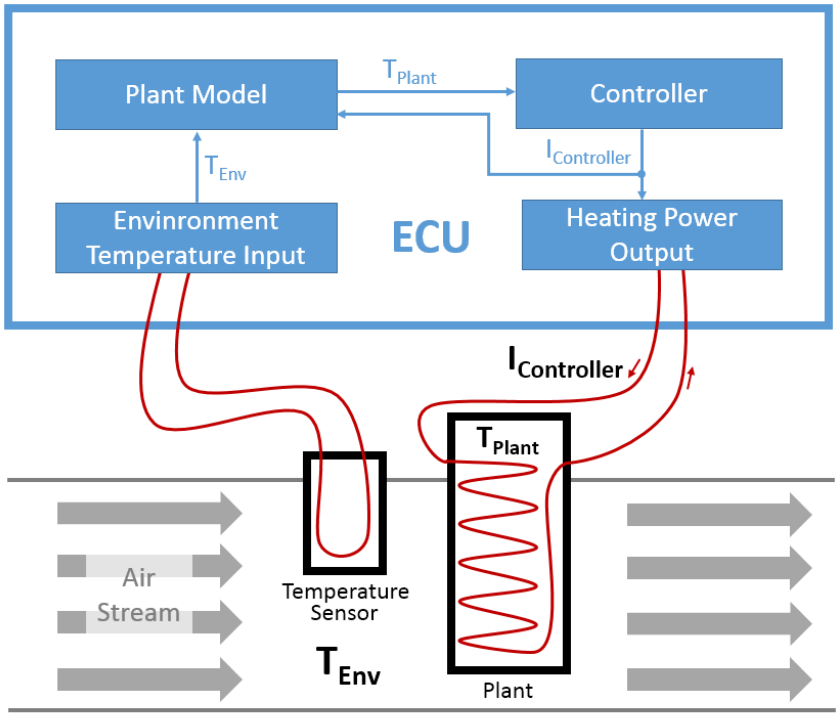


Figure 3.1: Physical Overview

Figure 3.1 shows the major physical values and entities of our system. The control task is the following: The plant is a sensor in an airstream that requires heating. The temperature T_{Plant} is to be controlled by the ECU. However, there is no direct way to measure T_{Plant}.

For the estimation of T_{Plant} the following properties of the system are used:

- T_{Plant} depends on the temperature of the environment T_{Env}, i.e. the temperature of the air stream. T_{Env} can be measured directly.

- T_{Plant} depends on the current $I_{Controller}$ which is output by the ECU and controlled by the controller and therefore known.
- The plant itself acts as a thermal energy storage. So T_{Plant} also depends on the heat quantity that is currently stored within the plant.
- All other influences on T_{Plant} are considered to be insignificant. So they can safely be ignored for this control task.

An estimation of T_{Plant} can be calculated by a plant model, which uses T_{Env} and $I_{Controller}$ as inputs and has the stored heat quantity as internal state.

3.1.1.1 Components Overview

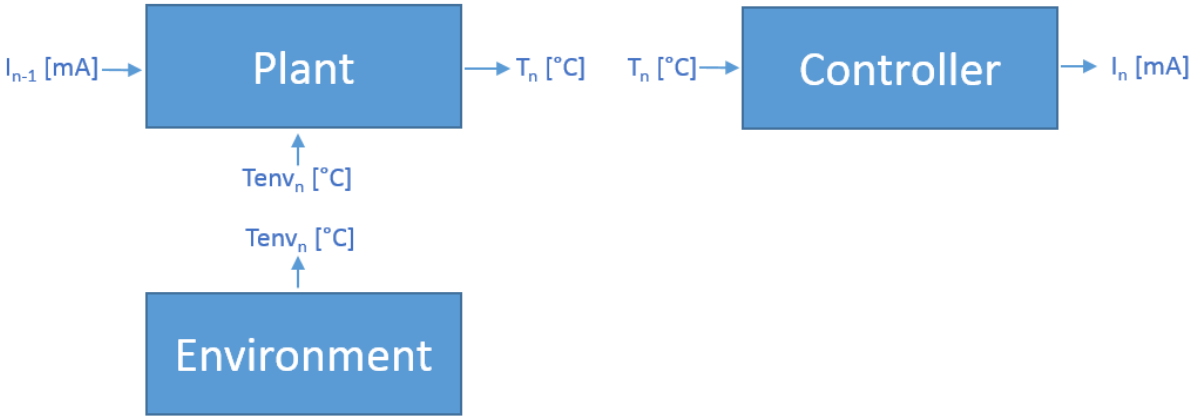


Figure 3.2: Component Overview

For this show case, the interaction with a real physical environment is completely left out, i.e. there is no `Heating Power Output` component and the profile of T_{Env} is randomly generated inside the component `Environment`. This cuts off a lot of complexity from the example, and allows to run the software system on a PC without complex environment simulations.

For completeness: The plant model is calculated inside the component `Plant` and the controller inside the component `Controller`.

As typical for ECUs the calculations happen in a time-discrete manner, i.e. the calculations in the components are executed periodically at discrete in time steps. In the following, the index $n \in \{1, 2, ...\}$ denotes the current time step. The previous time step is denoted by the index $n - 1$. The index 0 denotes the initialization value. This also means that time step 1 is the first, that is actually calculated by the ECU.

Furthermore Δt denotes the time in seconds, that elapsed between the calculation of the previous time step and the current time step. In case of time step 1, Δt denotes the time that elapsed between initialization of the system and time step 1. For setting

the actual value of Δt the frequency bandwidth of the physical properties in the system has to be taken into account. Decreasing the value of Δt usually increases the quality of the sampling of physical signals up to a certain point where the costs of further decreasing the value of Δt outweighs the benefit gained in terms of signal quality.

3.1.1.2 The Environment

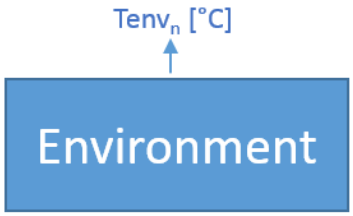


Figure 3.3: Environment

The modeling and implementation of the environment is not in the focus of this show case. The temperature $Tenv_n[°C]$ is generated (pseudo) randomly. This is done in order to see the controller and the plant model "in action" during run-time of the system.

The generated profile is a random walk limited by an upper and a lower boundary, with saturation at these boundaries.

The random walk is configurable via $\mathtt{T}_{\mathtt{LowLimit}}$ [°C] and $\mathtt{T}_{\mathtt{HighLimit}}$ [°C], for the boundaries, and $\mathtt{T}_{\mathtt{StepSize}}$ [K], for the change of the temperature during one time step.

Assuming $rand_n$ [-] $\in \{-1, 0, 1\}$ and $n \in \{1, 2, 3, ...\}$ then $Tenv_n$ is characterized by this equation (with $Tenv_0$ [°C] $= -273.15$ [°C]):

$$Tenv_n\,[°C] \;=\; Tenv_{n-1}\,[°C] + \mathtt{T}_{\mathtt{StepSize}}\,[K] \cdot rand_n\,[-]$$

if and only if $Tenv_n$ would be inside the boundaries, i.e.

$$\mathtt{T}_{\mathtt{LowLimit}} \;<\; Tenv_n \;<\; \mathtt{T}_{\mathtt{HighLimit}}$$

If $Tenv_n$ would be outside one of the boundaries, it is set to the value of that boundary.

3.1.1.3 The Plant

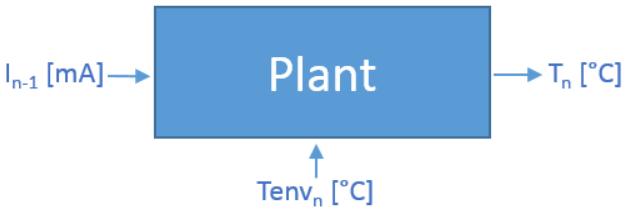


Figure 3.4: Plant

The plant is an electrically heated mass that is exposed to the air flow in the environment. The heat quantity $Qplant$ that is stored inside the plant is considered to always be directly proportional to the temperature T with constant proportionality factor. Neither the mass of the plant, nor the specific heat capacity changes during the run-time of our system.

For simplicity, this proportionality factor is considered to be $1\,[\frac{\mathtt{J}}{\mathtt{K}}]$. For the calculations inside the `Plant` component, we are always using $[\mathtt{K}]$ as unit for temperatures, so the conversion from and to $[\mathtt{°C}]$ only happens at the interface of the component.

With this, we have the following:

$$
\begin{aligned}
Qplant_n\,[\mathtt{J}] &= T_n\,[\mathtt{K}] \cdot 1\,[\tfrac{\mathtt{J}}{\mathtt{K}}] \\
T_n\,[\mathtt{K}] &= \frac{Qplant_n\,[\mathtt{J}]}{1\,[\tfrac{\mathtt{J}}{\mathtt{K}}]}
\end{aligned}
$$

This also means, that $Qplant_n\,[\mathtt{J}] = 0\,[\mathtt{J}]$ corresponds $T_n\,[\mathtt{K}] = 0\,[\mathtt{K}]$, i.e. absolute zero. So $Qplant_n\,[\mathtt{J}] \geq 0\,[\mathtt{J}]$ shall always be true.

In each time step, there are two heat flows: One from the electrical heater to the plant and one from the plant to the environment. A negative heat flow means that heat energy is flowing away from the plant. Respectively, a positive heat flow means that heat energy is stored in the plant.

The heat flow $Qheater_n\,[\mathtt{J}]$ from the electrical heater to the plant in one time step is considered to be proportional to the current $I_n\,[\mathtt{mA}]$ through the plant during this time step. The proportionality factor is $\mathtt{h_{Heater}}\,[\frac{\mathtt{J}}{\mathtt{mA\,s}}]$. Of course, the plant can only be heated up by the electrical heater, i.e. a "negative" current I_n would not cool down the plant, but causes the same heat up as $-I_n$. So we have

$$
Qheater_n\,[\mathtt{J}] = \mid I_n \mid\,[\mathtt{mA}] \cdot \mathtt{h_{Heater}}\,[\tfrac{\mathtt{J}}{\mathtt{mA\,s}}] \cdot \Delta t\,[\mathtt{s}]
$$

The cool down of the plant can only happen via the second heat flow, i.e. the heat flow $Qenv_n\,[\mathtt{J}]$ from the plant to the environment. The flow in one time step is considered

to be proportional to the difference between the temperature of the plant (calculated from the stored heat quantity during the last time step) and the temperature of the environment (received in this time step, but actually "measured" during the last time step). With the proportionality factor $\mathtt{h_{Env}}\left[\frac{\mathtt{J}}{\mathtt{K}}\right]$, we have:

$$Qenv_n\left[\mathtt{J}\right] \;=\; (Tenv_n\left[\mathtt{K}\right] - T_{n-1}\left[\mathtt{K}\right]) \cdot \mathtt{h_{Env}}\left[\frac{\mathtt{J}}{\mathtt{K}}\right] \cdot \Delta t\left[\mathtt{s}\right]$$

The heat quantity that was stored in the plant in last time step $Qplant_{n-1}$ is now modified by these two heat flows. This results in the stored heat quantity in the current time step. With $Qplant_0[\mathtt{J}] = 0[\mathtt{J}]$, we have

$$Qplant_n\left[\mathtt{J}\right] \;=\; Qplant_{n-1}\left[\mathtt{J}\right] + Qheater_n\left[\mathtt{J}\right] + Qenv_n\left[\mathtt{J}\right]$$

3.1.1.4 The Controller



Figure 3.5: Controller

For the closed loop control an I controller (by and large) was chosen for component `Controller`. This means that the amplification of the input signal is proportional to the integral of the errors, i.e. the deviation between measured variable and setpoint. Because the controller cannot actively cool down the temperature of the plant, the output $I_n >= 0$ for all n.

Again, all temperatures are converted to and from $[\mathtt{°C}]$ at the interface of the component. All internal calculation are done in $[\mathtt{K}]$.

The error during the current time step is the difference between $\mathtt{T_{SetPoint}}\left[\mathtt{K}\right]$ and the measured variable $T_n\left[\mathtt{K}\right]$:

$$e_n\left[\mathtt{K}\right] \;=\; \mathtt{T_{SetPoint}}\left[\mathtt{K}\right] - T_n\left[\mathtt{K}\right]$$

The integral part of the controller is calculated via summing up all errors from the previous steps. With $eSum_n\left[\mathtt{Ks}\right] = 0\left[\mathtt{Ks}\right]$ we have:

$$eSum_n\left[\mathtt{Ks}\right] \;=\; eSum_{n-1}\left[\mathtt{Ks}\right] + e_n\left[\mathtt{K}\right] \cdot \Delta t$$

A further design decision for the controller was, to limit the integral and to saturate at the limits. This has the benefit that it limits the current I_n that is output by the controller. Furthermore, it enables the controller to react faster after long deviations.

The lower limit is $0\,[\mathtt{Ks}]$. So if $eSum_n$ would fall below zero in time step n, we set $eSum_n[\mathtt{Ks}] = 0\,[\mathtt{Ks}]$. The upper limit is $\mathtt{L_{MaxESum}}\,[\mathtt{Ks}]$. If $eSum_n$ would exceed $L_{MaxESum}$ in time step n we set $eSum_n[\mathtt{Ks}] = \mathtt{L_{MaxESum}}\,[\mathtt{Ks}]$.

The integral state $eSum_n$ of the controller is then amplified by $\mathtt{k}\left[\frac{\mathtt{mA}}{\mathtt{Ks}}\right]$ to calculate the current $I_n\,[\mathtt{mA}]$, i.e. the output of the controller:

$$I_n\,[\mathtt{mA}] = eSum_n\,[\mathtt{Ks}] \cdot \mathtt{k}\left[\frac{\mathtt{mA}}{\mathtt{Ks}}\right]$$

So the limitations of the $eSum_n$ guarantee, that

$$0\,[\mathtt{mA}] \leq I_n\,[\mathtt{mA}] \leq \mathtt{L_{MaxESum}}\,[\mathtt{Ks}] \cdot \mathtt{k}\left[\frac{\mathtt{mA}}{\mathtt{Ks}}\right]$$

3.1.2 AUTOSAR Modeling

This section gives a brief overview of the AUTOSAR modeling. More insight can be gained by browsing through the hyper-linked tables in section 3.1.7. These tables are generated from the AUTOSAR model of this show case. If this is still not sufficient, the complete model is available in `.arxml` format in `AUTOSAR_EXP_ModelingShowCases.zip` [2].

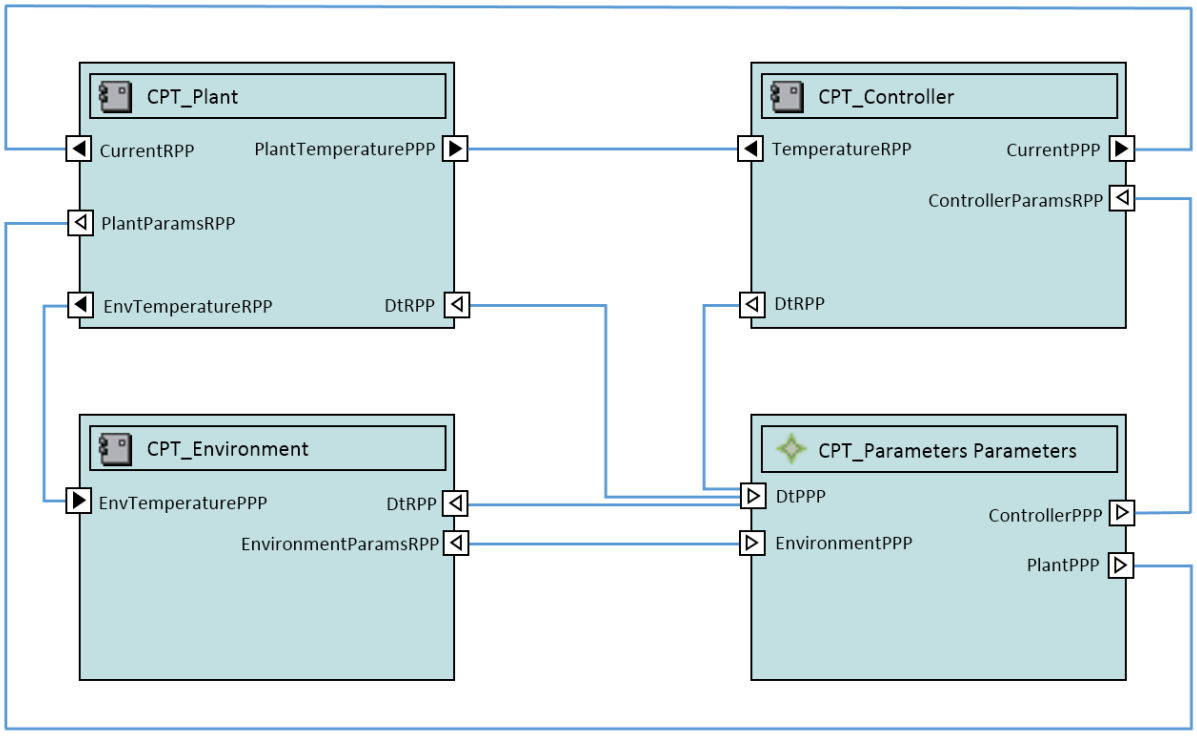


Figure 3.6: Example `Composition`

In this show case the components specified in section 3.1.1.1 are modeled as `ApplicationSwComponentType`s.

- Environment
- Plant
- Controller

To keep the example simple, no SwcImplementations were modeled. For some tasks, like generation of a MemMap for an embedded controller, this would be needed.

The in- and outputs of the ApplicationSwComponentTypes are modeled as SenderReceiverInterface. The internal state is realized as implicitInterRunnableVariables. Besides the illustrative aspect, the rationale for this design decision was that the internal state is likely to be used by more than one runnable in ApplicationSwComponentTypes (at least "outside" of an introductory show case).

For variables that should just be available for measurement in a MC tool, arTypedPerInstanceMemorys are used. For this use case, no synchronization of access to the variable needs to be implemented, so the way with the least overhead was chosen.

All parameters in the specification of the components were put in a fourth SwComponentType, in the ParameterSwComponentType "Parameters".

A distinct ParameterInterface was defined for the parameters of each of the three ApplicationSwComponentTypes. The respective PPortPrototypes of the ParameterSwComponentType hold the initValue for each ParameterDataPrototype in the PortPrototypes. Each value is specified in a ValueSpecification aggregated by a ParameterProvideComSpec.

The component types are instantiated in the CompositionSwComponentType "Composition".

This Composition is the type of the rootSoftwareComposition of the ECU_Extract. This also implies that all SwComponentPrototypes of Composition are mapped to one EcuInstance.

Some information on the FlatMap can be found in section 3.1.3.1.

3.1.3 RTE Generation, Measurement and Calibration

The McSupport is an interface between the RTE generator and the A2L generator. A RTE generator provides a McDataInstance for each calibrateable or measurable object. From logical view the generation of McSupport could be seen in two steps:

1. Provide unique names for all parameters, measurements, component prototypes which are instantiated one or multiple times. This is done by the used AUTOSAR Authoring Tool.

2. Generate the McSupport itself. This is usually done by the RTE generator. A2L supports only one global namespace, while AUTOSAR defines own namespaces within each ARPackage. This means, that on the one hand unique names are needed for all objects which are to be accessible during measurement and calibration (parameters,

measurements, component prototypes). But on the other hand, unique names are needed for all other things that will appear in A2L, e.g. CompuMethods, Units. For them the RTE generator will create unique names.

AUTOSAR specifies additionally an AliasNameSet to override names which is not used here.

See AUTOSAR_EXP_ModelingShowCases.zip [2] for the generated Rte_McSupportData.arxml file.

3.1.3.1 FlatMap

In this show case, the FlatMap gives unique names to the

- dataElements
- implicitInterRunnableVariables
- arTypedPerInstanceMemorys

The RTE-Generator uses this information for the generation of the McSupport file as well as for generation of the .c and .h files.

The FlatMap consists of a FlatInstanceDescriptor for each instance of these VariableDataPrototypes.

The concrete modeling of the FlatMap for this use case can be found in AUTOSAR_EXP_ModelingShowCases.zip [2].

3.1.3.2 ECU Documentation, Measurement and Calibration

When developing an ECU one usual requirement is, that objects described in A2L can be easily found in the documentation of the ECU. This is a challenge since documentation is on the level of SwComponentTypes while A2L is defined on the level of a System of category "ECU_EXTRACT".

- The names of SwComponentPrototypes are potentially different to the names of SwComponentTypes
- The names of McDataInstances are potentially different to the names of DataPrototypes

The challenge gets bigger, if types are instantiated multiple times. This issue needs to be solved by proper architecture, modeling conventions and clever generation of the FlatMap.

In this show case, this topic is only slightly touched by instantiating TemperatureSRIF two times, for the interface transporting T_{Env} as well as for the interface transporting T_{Plant}.

It is demonstrated that the FlatMap can be used to solve the issue. However, we manually crafted our FlatMap, which is usually not possible in the field. FlatMaps are usually automatically generated by customizable, "clever", not standardized tools.

3.1.4 A2L File

With the information in the McSupport file an A2L file is generated. However, for this generation the memory addresses for the variables and characteristics are needed. They are usually extracted from the map file that is output by the linker of the ECU executable. The exact process as well as the tool for the A2L file generation is not standardized.

An example A2L file is provided in AUTOSAR_EXP_ModelingShowCases.zip.

3.1.5 Implementation in C

The implementation in C is a straight forward realization of the physical specification within the AUTOSAR modeling (see section 3.1.1.1 and 3.1.2). Therefore, the listings are presented without further explanation besides the comments in the source code.

A remark on the (pseudo) random numbers generated in line 22 of Environment.c (Listing Listing 3.1): The numbers don't have good "pseudo randomness" properties but are sufficient for this show case, nevertheless. This way of generation was only chosen, because it fits in one line of C code without introducing a dependency to a library.

```
1  #include "Rte_Environment.h"
2
3  #define envRE_START_SEC_CODE
4  #include "Environment_MemMap.h"
5
6  FUNC (void, Environment_CODE) envRE_func (void)
7  {
8     /* read parameters for simulation of the temperature profile   */
9     float32 lLowLimit   = Rte_Prm_EnvParamsRPP_env_TLowLimit();
10    float32 lStepSize   = Rte_Prm_EnvParamsRPP_env_TStepSize();
11
12    /* retrieve internal state                                     */
13    uint32  lSeed     = Rte_IrvIRead_envRE_Seed();
14    float32 lTEnv     = Rte_IrvIRead_envRE_TEnv();
15    float32 direction = (float32)(lSeed % 3) - 1.0;
16
17    /* calc high limit with parameter, store for measurement       */
18    *Rte_Pim_THighLimit()
19         = lLowLimit + Rte_Prm_EnvParamsRPP_env_THighLimitDistance();
20
21    /*  update state for pseudo random number generation           */
22    lSeed = (8253729 * lSeed + 2396403);
23
24    /* calculate environment temperature                           */
```

```
25      lTEnv += lStepSize * direction;
26
27      /* saturating environment temperature at the bounds             */
28      if( lTEnv < lLowLimit)  { lTEnv = lLowLimit; }
29      if( lTEnv > *Rte_Pim_THighLimit())
30                              { lTEnv = *Rte_Pim_THighLimit(); }
31
32      /* Store internal state                                         */
33      Rte_IrvIWrite_envRE_Seed(lSeed);
34      Rte_IrvIWrite_envRE_TEnv(lTEnv);
35
36      /* write output                                                 */
37      Rte_IWrite_envRE_EnvTemperaturePPP_T(lTEnv);
38  }
39  #define envRE_STOP_SEC_CODE
40  #include "Environment_MemMap.h"
```

Listing 3.1: Environment.c

```
1   #include "Rte_Plant.h"
2
3   #define plantRE_START_SEC_CODE
4   #include "Plant_MemMap.h"
5
6   FUNC (void, Plant_CODE) plantRE_func (void)
7   {
8       /* read input                                                   */
9       float32 lTenv    = Rte_IRead_plantRE_EnvTemperatureRPP_T();
10      float32 lI       = Rte_IRead_plantRE_CurrentRPP_I();
11
12      /* retrieve internal state                                      */
13      float32 lQPlant  = Rte_IrvIRead_plantRE_QPlant();
14
15      /* read parameters                                              */
16      float32 lDt      = Rte_Prm_DtRPP_Dt();
17      float32 lEFactor = Rte_Prm_PlantParamsRPP_plnt_EnvFactor();
18      float32 lHFactor = Rte_Prm_PlantParamsRPP_plnt_HeaterFactor();
19
20      /* heat capacity of 1 assumed                                   */
21      float32 lTPlant     = lQPlant;
22
23       /* calculate heat flows, store in PIM to make them measurable */
24      *Rte_Pim_QEnv()     = (lTenv - lTPlant) * lEFactor * lDt;
25      *Rte_Pim_QHeater()  = lI * lHFactor * lDt;
26
27      /* update heat quantity in plant                                */
28      lQPlant  = lQPlant + *Rte_Pim_QHeater() + *Rte_Pim_QEnv();
29
30      /* limit heat quantity to absolute zero                         */
31      lQPlant  = lQPlant < 0 ? 0 : lQPlant;
32
33      /* heat capacity of 1 assumed                                   */
34      lTPlant  = lQPlant;
35
36      /* store internal state of plant: stored heat quantity          */
37      Rte_IrvIWrite_plantRE_QPlant(lQPlant);
```

```
38
39     /* Write output of plant: temerature of plant                    */
40     Rte_IWrite_plantRE_PlantTemperaturePPP_T(lTPlant);
41  }
42  #define plantRE_STOP_SEC_CODE
43  #include "Plant_MemMap.h"
```

Listing 3.2: Plant.c

```
1   #include "Rte_Controller.h"
2
3   #define ControllerRE_START_SEC_CODE
4   #include "Controller_MemMap.h"
5
6   FUNC (void, Controller_CODE) controllerRE_func (void)
7   {
8      /* read input, define output variable                           */
9      float32 lT        = Rte_IRead_ControllerRE_TemperatureRPP_T();
10     float32 lI;
11
12     /* retrieve internal state: Sum of errors until last time step */
13     float32 lESum     = Rte_IrvIRead_ControllerRE_ESum();
14
15     /* read parameters                                              */
16     float32 lDt       = Rte_Prm_DtRPP_Dt();
17     float32 lSetPoint = Rte_Prm_ControllerParamsRPP_ctrl_SetPoint();
18     float32 lK        = Rte_Prm_ControllerParamsRPP_ctrl_K();
19     float32 lMaxESum  = Rte_Prm_ControllerParamsRPP_ctrl_MaxESum();
20
21      /* store current error in PIM to make it measurable           */
22     *Rte_Pim_E() = lSetPoint - lT;
23
24      /* update eSum                                                */
25     lESum += *Rte_Pim_E() * lDt;
26
27     /* limit   eSum                                                */
28     if(lESum > lMaxESum) { lESum = lMaxESum; }
29     if(lESum < 0)        { lESum = 0;       }
30
31     /* Controller equation: Calculation of manipulated variable    */
32     lI = lESum * lK;
33
34     /* Store internal state                                        */
35     Rte_IrvIWrite_ControllerRE_ESum(lESum);
36
37     /* Write output of controller                                  */
38     Rte_IWrite_ControllerRE_CurrentPPP_I(lI);
39  }
40  #define ControllerRE_STOP_SEC_CODE
41  #include "Controller_MemMap.h"
```

Listing 3.3: Controller.c

3.1.6 A walk with T_Plant through the Show Case

This section revisits the complete show case, but focuses on one physical value: T_{Plant}. It visits all artifacts and highlights all places that relate to T_{Plant} to illustrate the dependencies between all artifacts.

3.1.6.1 Physical System

Our journey begins at the physical system, where the value of the physical system outside of the `ECU` is identified with a software value inside the `ECU`.

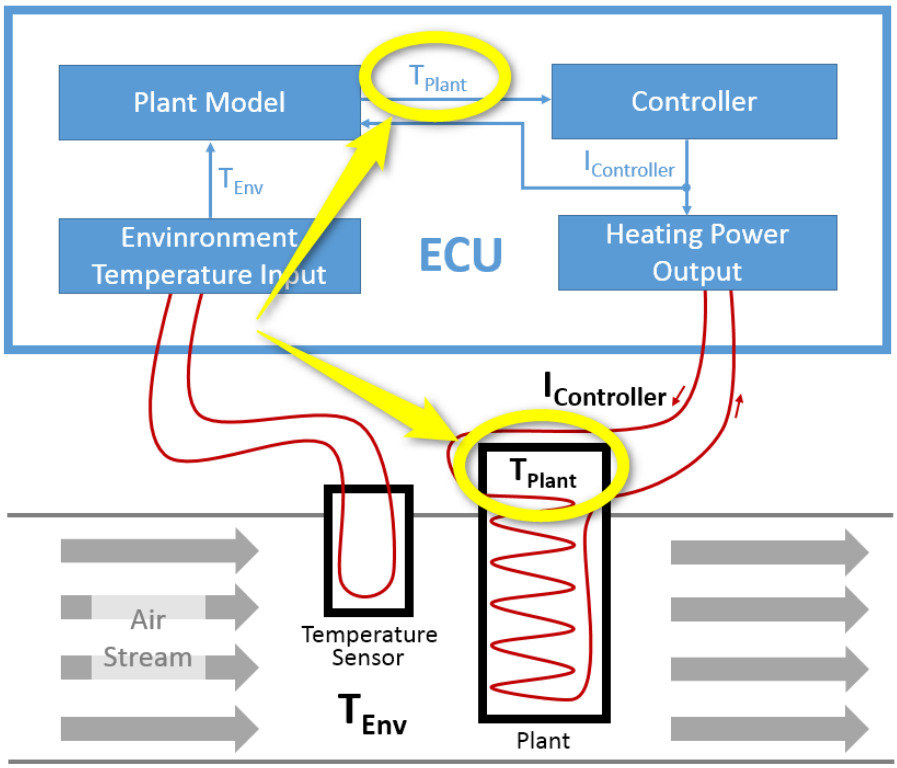


Figure 3.7: Physical Overview

3.1.6.1.1 Components

It was located at the interface between two architectural components, sent by the `Plant` and received by the `Controller`. Furthermore a sequencing was introduced[3], i.e. in one time step the `Plant` is calculated before the `Controller`.

[3]Please note that this sequencing is a design decision. As there is also a data flow from the `Plant` to the `Controller` one could also argue for another calculation sequence.

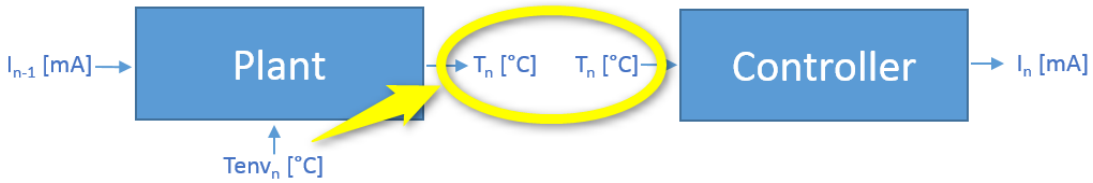


Figure 3.8: Component Overview

3.1.6.1.2 Equations

The functional behavior is defined by the equations for the `Plant`

$$Qplant_n\,[\mathrm{J}] = T_n\,[\mathrm{K}]\;1\left[\frac{\mathrm{J}}{\mathrm{K}}\right]$$

$$T_n\,[\mathrm{K}] = \frac{Qplant_n\,[\mathrm{J}]}{1\left[\frac{\mathrm{J}}{\mathrm{K}}\right]}$$

Figure 3.9: Dependency between Q_{Plant} and T_{Plant}

$$Qenv_n\,[\mathrm{J}] = (Tenv_n\,[\mathrm{K}] - T_{n-1}\,[\mathrm{K}]) \cdot \mathrm{h_{Env}}\left[\frac{\mathrm{J}}{\mathrm{K}}\right] \cdot \Delta t\,[\mathrm{s}]$$

Figure 3.10: Heat flow from the `Plant` to the `Environment`

T_{Plant} is also used by the physical equations in the component `Controller`:

$$e_n\,[\mathrm{K}] = \mathrm{T_{SetPoint}}\,[\mathrm{K}] - T_n\,[\mathrm{K}]$$

Figure 3.11: Calculation of the control error in the `Controller`

Furthermore, calculations inside the components are done in Kelvin [K]. The conversion from and to [°C] happens at the interface level.

3.1.6.2 AUTOSAR Modeling

This architecture, i.e. the layout of the physical system, is modeled in AUTOSAR. The functional behavior defined by the equations will be implemented in C Code later on.

3.1.6.2.1 Physical Dimension and Unit

A `PhysicalDimension` is defined: T_{Plant} is a temperature.

Common PhysicalDimension attributes	
shortName	Temperature
currentExp	0
lengthExp	0
luminousIntensity-Exp	0
massExp	0
molarAmountExp	0
temperatureExp	1
timeExp	0

Table 3.1: PhysicalDimension Temparature

The corresponding ARXML description is:

```
<PHYSICAL-DIMENSION>
  <SHORT-NAME>Temperature</SHORT-NAME>
  <LENGTH-EXP>0</LENGTH-EXP>
  <MASS-EXP>0</MASS-EXP>
  <TIME-EXP>0</TIME-EXP>
  <CURRENT-EXP>0</CURRENT-EXP>
  <TEMPERATURE-EXP>1</TEMPERATURE-EXP>
  <MOLAR-AMOUNT-EXP>0</MOLAR-AMOUNT-EXP>
  <LUMINOUS-INTENSITY-EXP>0</LUMINOUS-INTENSITY-EXP>
</PHYSICAL-DIMENSION>
```

Listing 3.4: Physical Dimension of Temperature

T_{Plant} shall have the Unit DegreeCelsius:

Common Unit attributes	
shortName	DegreeCelsius
displayName	℃
offsetSiToUnit	-273.15
factorSiToUnit	1.0
physicalDimension	Temperature

Table 3.2: Unit DegreeCelsius

The corresponding ARXML description is:

```
<UNIT>
  <SHORT-NAME>DegreeCelsius</SHORT-NAME>
  <DISPLAY-NAME>°C</DISPLAY-NAME>
  <FACTOR-SI-TO-UNIT>1.0</FACTOR-SI-TO-UNIT>
  <OFFSET-SI-TO-UNIT>-273.15</OFFSET-SI-TO-UNIT>
  <PHYSICAL-DIMENSION-REF DEST="PHYSICAL-DIMENSION">/McInt/
      PhysicalDimensions/Temperature</PHYSICAL-DIMENSION-REF>
</UNIT>
```

Listing 3.5: Unit Degree Celsius

The following is presented for completeness, although not directly needed for T_{Plant}. It is possible to link more than one unit to a physical dimension. So in the model, there is also a definition for the unit Kelvin:

Common Unit attributes	
shortName	Kelvin
displayName	K
offsetSiToUnit	0.0
factorSiToUnit	1.0
physicalDimension	Temperature

Table 3.3: Unit Kelvin

The corresponding ARXML Code is:

```
<UNIT>
  <SHORT-NAME>Kelvin</SHORT-NAME>
  <DISPLAY-NAME>K</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 DEST="PHYSICAL-DIMENSION">/McInt/
     PhysicalDimensions/Temperature</PHYSICAL-DIMENSION-REF>
</UNIT>
```

Listing 3.6: Unit Kelvin

3.1.6.2.2 Application Data Type

A new ApplicationDataType is defined for temperatures in degree Celsius:

Common ApplicationDataType attributes			
shortName	Temperature_C		
category	VALUE		
desc	Type for a temperature in [℃]		
swCalibrationAccess	readOnly		
unit	DegreeCelsius		
Range			
Conversion			
category	LINEAR		
direction	compuInternalToPhys		
desc	lowerLimit	upperLimit	compuNumerator / compuDenominator
-	-	-	$Phys = \frac{-273.15 + 1 * Internal}{1}$

Table 3.4: ApplicationDataType Temperature_C

The corresponding ARXML Code is split between the definition of the ApplicationDataType:

```
<APPLICATION-PRIMITIVE-DATA-TYPE>
  <SHORT-NAME>Temperature_C</SHORT-NAME>
  <DESC>
    <L-2 L="EN">Type for a temperature in [°C]</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 DEST="COMPU-METHOD">/McInt/CompuMethods/
            Temperature_C</COMPU-METHOD-REF>
        <UNIT-REF DEST="UNIT">/McInt/Units/DegreeCelsius</UNIT-REF>
      </SW-DATA-DEF-PROPS-CONDITIONAL>
    </SW-DATA-DEF-PROPS-VARIANTS>
  </SW-DATA-DEF-PROPS>
</APPLICATION-PRIMITIVE-DATA-TYPE>
```

Listing 3.7: Application Data Type

and the CompuMethod, which is referenced by the ApplicationDataType:

```
<COMPU-METHOD>
  <SHORT-NAME>Temperature_C</SHORT-NAME>
  <DESC>
    <L-2 L="EN">Conversion from [°C] to [K]</L-2>
  </DESC>
  <CATEGORY>LINEAR</CATEGORY>
  <DISPLAY-FORMAT>%.1f</DISPLAY-FORMAT>
  <UNIT-REF DEST="UNIT">/McInt/Units/DegreeCelsius</UNIT-REF>
  <COMPU-INTERNAL-TO-PHYS>
    <COMPU-SCALES>
      <COMPU-SCALE>
        <COMPU-RATIONAL-COEFFS>
          <COMPU-NUMERATOR>
            <V>-273.15</V>
            <V>1</V>
          </COMPU-NUMERATOR>
          <COMPU-DENOMINATOR>
            <V>1</V>
          </COMPU-DENOMINATOR>
        </COMPU-RATIONAL-COEFFS>
      </COMPU-SCALE>
    </COMPU-SCALES>
  </COMPU-INTERNAL-TO-PHYS>
</COMPU-METHOD>
```

Listing 3.8: Conversion

This ApplicationDataType is mapped to the ImplementationDataType float32. The DataTypeMappingSet that contains this DataTypeMap is referenced inside the SwcInternalBehaviors of the ApplicationSwComponentTypes presented later on.

```
<DATA-TYPE-MAPPING-SET>
  <SHORT-NAME>DataTypeMappingSet</SHORT-NAME>
  <DATA-TYPE-MAPS>
    <DATA-TYPE-MAP>
      <APPLICATION-DATA-TYPE-REF DEST="APPLICATION-PRIMITIVE-DATA-TYPE">/
          McInt/ApplicationDataTypes/Temperature_C</APPLICATION-DATA-TYPE-
          REF>
      <IMPLEMENTATION-DATA-TYPE-REF DEST="IMPLEMENTATION-DATA-TYPE">/
          AUTOSAR_PlatformTypes/ImplementationDataTypes/float32</
          IMPLEMENTATION-DATA-TYPE-REF>
    </DATA-TYPE-MAP>
  </DATA-TYPE-MAPS>
</DATA-TYPE-MAPPING-SET>
```

Listing 3.9: Type Mapping

For completeness, also the ARXML containing the definition of `float32` is inserted here:

```
<AR-PACKAGE>
  <SHORT-NAME>AUTOSAR_PlatformTypes</SHORT-NAME>
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>ImplementationDataTypes</SHORT-NAME>
      <ELEMENTS>
        <IMPLEMENTATION-DATA-TYPE>
          <SHORT-NAME>float32</SHORT-NAME>
          <CATEGORY>VALUE</CATEGORY>
          <SW-DATA-DEF-PROPS>
            <SW-DATA-DEF-PROPS-VARIANTS>
              <SW-DATA-DEF-PROPS-CONDITIONAL>
                <BASE-TYPE-REF DEST="SW-BASE-TYPE">/AUTOSAR_PlatformTypes/
                    SwBaseTypes/float32</BASE-TYPE-REF>
              </SW-DATA-DEF-PROPS-CONDITIONAL>
            </SW-DATA-DEF-PROPS-VARIANTS>
          </SW-DATA-DEF-PROPS>
        </IMPLEMENTATION-DATA-TYPE>
      </ELEMENTS>
    </AR-PACKAGE>
    <AR-PACKAGE>
      <SHORT-NAME>SwBaseTypes</SHORT-NAME>
      <ELEMENTS>
        <SW-BASE-TYPE>
          <SHORT-NAME>float32</SHORT-NAME>
          <CATEGORY>FIXED_LENGTH</CATEGORY>
          <BASE-TYPE-SIZE>32</BASE-TYPE-SIZE>
          <BASE-TYPE-ENCODING>IEEE754</BASE-TYPE-ENCODING>
        </SW-BASE-TYPE>
      </ELEMENTS>
    </AR-PACKAGE>
  </AR-PACKAGES>
</AR-PACKAGE>
```

Listing 3.10: Implementation Type and Base Type

3.1.6.2.3 Port Interface

The Temperature_C is used to define the SenderReceiverInterface which is used to type the "transport" of a temperature in degree Celsius between SwComponentTypes. Please note that in the show case, this PortInterface is not only used to type the "transport" of T_{Plant}, but also to type the "transport" of T_{Env}.

Common SenderReceiverInterface attributes	
shortName	TemperatureSRIF
desc	Interface type for transferring temperatures in [°C]
properties of the dataElements	
properties of VariableDataPrototype	
shortName	T
type	Temperature_C
swImplPolicy	standard
swCalibrationAccess	readOnly
swAddrMethod	VAR

Table 3.5: SenderReceiverInterface TemperatureSRIF

In ARXML:

```
<SENDER-RECEIVER-INTERFACE>
  <SHORT-NAME>TemperatureSRIF</SHORT-NAME>
  <DESC>
    <L-2 L="EN">Interface type for transferring temperatures in [°C]</L-2>
  </DESC>
  <IS-SERVICE>false</IS-SERVICE>
  <DATA-ELEMENTS>
    <VARIABLE-DATA-PROTOTYPE>
      <SHORT-NAME>T</SHORT-NAME>
      <SW-DATA-DEF-PROPS>
        <SW-DATA-DEF-PROPS-VARIANTS>
          <SW-DATA-DEF-PROPS-CONDITIONAL>
            <SW-ADDR-METHOD-REF DEST="SW-ADDR-METHOD">/McInt/SwAddrMethods/
                VAR</SW-ADDR-METHOD-REF>
            <SW-CALIBRATION-ACCESS>READ-ONLY</SW-CALIBRATION-ACCESS>
            <SW-IMPL-POLICY>STANDARD</SW-IMPL-POLICY>
          </SW-DATA-DEF-PROPS-CONDITIONAL>
        </SW-DATA-DEF-PROPS-VARIANTS>
      </SW-DATA-DEF-PROPS>
      <TYPE-TREF DEST="APPLICATION-PRIMITIVE-DATA-TYPE">/McInt/
          ApplicationDataTypes/Temperature_C</TYPE-TREF>
    </VARIABLE-DATA-PROTOTYPE>
  </DATA-ELEMENTS>
</SENDER-RECEIVER-INTERFACE>
```

Listing 3.11: Port Interface

For completeness, also the referenced SwAddrMethod is described here:

Common SwAddrMethod attributes	
shortName	VAR
desc	Memory section for variables
sectionType	var
memoryAllocationKeywordPolicy	addrMethodShortName
sectionInitializationPolicy	-
option	safetyQM

Table 3.6: SwAddrMethod VAR

In ARXML:

```
<SW-ADDR-METHOD>
  <SHORT-NAME>VAR</SHORT-NAME>
  <DESC>
  <L-2 L="EN">Memory section for variables</L-2>
  </DESC>
  <OPTIONS>
  <OPTION>safetyQM</OPTION>
  </OPTIONS>
</SW-ADDR-METHOD>
```

Listing 3.12: Software Address Method

3.1.6.2.4 Software Components

The two ApplicationSwComponentTypes Controller and Plant are using T_{Plant}.

In Plant a PPortPrototype, typed by TemperatureSRIF, is defined for sending out T_{Plant}.

Furthermore dataWriteAccess is granted to the single RunnableEntity in this ApplicationSwComponentType. You also see the symbol, i.e. the name of the implementing C function, as well as the TimingEvent that triggers the execution of the RunnableEntity. These two are of further interest for tying together the system.

Common ApplicationSwComponentType attributes	
shortName	Plant
properties of the ports	
properties of PPortPrototype	
shortName	PlantTemperaturePPP
desc	Port for sending out the estimated temperature of the plant
providedInterface	TemperatureSRIF
[...]	
internalBehavior	PlantInternalBehavior
[...]	
properties of the runnables	
properties of RunnableEntity	
shortName	plantRE
symbol	plantRE_func
properties of the events	
properties of TimingEvent	
shortName	plant100ms
startOnEvent	plantRE
period	0.1

Table 3.7: ApplicationSwComponentType Plant

In ARXML:

```
<APPLICATION-SW-COMPONENT-TYPE>
  <SHORT-NAME>Plant</SHORT-NAME>
  <PORTS>
    <P-PORT-PROTOTYPE>
      <SHORT-NAME>PlantTemperaturePPP</SHORT-NAME>
      <DESC>
        <L-2 L="EN">Port for sending out the estimated temperature of the
            plant</L-2>
      </DESC>
      <PROVIDED-INTERFACE-TREF DEST="SENDER-RECEIVER-INTERFACE">/McInt/
          PortInterfaces/TemperatureSRIF</PROVIDED-INTERFACE-TREF>
    </P-PORT-PROTOTYPE>
  </PORTS>
  <INTERNAL-BEHAVIORS>
    <SWC-INTERNAL-BEHAVIOR>
      <SHORT-NAME>PlantInternalBehavior</SHORT-NAME>
      <DATA-TYPE-MAPPING-REFS>
        <DATA-TYPE-MAPPING-REF DEST="DATA-TYPE-MAPPING-SET">/McInt/
            DataTypeMappings/DataTypeMappingSet</DATA-TYPE-MAPPING-REF>
      </DATA-TYPE-MAPPING-REFS>
      <EVENTS>
        <TIMING-EVENT>
          <SHORT-NAME>plant100ms</SHORT-NAME>
          <START-ON-EVENT-REF DEST="RUNNABLE-ENTITY">/McInt/SwComponents/
              Plant/PlantInternalBehavior/plantRE</START-ON-EVENT-REF>
```

```
          <PERIOD>0.1</PERIOD>
        </TIMING-EVENT>
      </EVENTS>
      <RUNNABLES>
        <RUNNABLE-ENTITY>
          <SHORT-NAME>plantRE</SHORT-NAME>
          <DATA-WRITE-ACCESSS>
            <VARIABLE-ACCESS>
              <SHORT-NAME>DWA_PlantTemperature</SHORT-NAME>
              <ACCESSED-VARIABLE>
                <AUTOSAR-VARIABLE-IREF>
                  <PORT-PROTOTYPE-REF DEST="P-PORT-PROTOTYPE">/McInt/
                      SwComponents/Plant/PlantTemperaturePPP</PORT-PROTOTYPE
                      -REF>
                  <TARGET-DATA-PROTOTYPE-REF DEST="VARIABLE-DATA-PROTOTYPE"
                      >/McInt/PortInterfaces/TemperatureSRIF/T</TARGET-DATA-
                      PROTOTYPE-REF>
                </AUTOSAR-VARIABLE-IREF>
              </ACCESSED-VARIABLE>
            </VARIABLE-ACCESS>
          </DATA-WRITE-ACCESSS>
        </RUNNABLE-ENTITY>
      </RUNNABLES>
    </SWC-INTERNAL-BEHAVIOR>
  </INTERNAL-BEHAVIORS>
</APPLICATION-SW-COMPONENT-TYPE>
```

Listing 3.13: Plant

In `Controller` a `RPortPrototype`, typed by `TemperatureSRIF`, is defined for receiving T_{Plant}.

Furthermore `dataReadAccess` is granted to the single `RunnableEntity` in this `ApplicationSwComponentType`. You also see the `symbol`, i.e. the name of the implementing C function, as well as the `TimingEvent` that triggers the execution of the `RunnableEntity`. These two are of further interest for tying together the system.

Common ApplicationSwComponentType attributes	
shortName	Controller
properties of the ports	
properties of RPortPrototype	
shortName	TemperatureRPP
desc	Port to receive the temperature of the plant
requiredInterface	TemperatureSRIF
[...]	
internalBehavior	ControllerInternalBehavior
[...]	
properties of the runnables	
properties of RunnableEntity	
shortName	ControllerRE
symbol	controllerRE_func
properties of the events	
properties of TimingEvent	
shortName	controller100ms
startOnEvent	ControllerRE
period	0.1

Table 3.8: ApplicationSwComponentType Controller

In ARXML:

```
<APPLICATION-SW-COMPONENT-TYPE>
  <SHORT-NAME>Controller</SHORT-NAME>
  <PORTS>
    <R-PORT-PROTOTYPE>
      <SHORT-NAME>TemperatureRPP</SHORT-NAME>
      <DESC>
        <L-2 L="EN">Port to receive the temperature of the plant</L-2>
      </DESC>
      <REQUIRED-INTERFACE-TREF DEST="SENDER-RECEIVER-INTERFACE">/McInt/
          PortInterfaces/TemperatureSRIF</REQUIRED-INTERFACE-TREF>
    </R-PORT-PROTOTYPE>
  </PORTS>
  <INTERNAL-BEHAVIORS>
    <SWC-INTERNAL-BEHAVIOR>
      <SHORT-NAME>ControllerInternalBehavior</SHORT-NAME>
      <DATA-TYPE-MAPPING-REFS>
        <DATA-TYPE-MAPPING-REF DEST="DATA-TYPE-MAPPING-SET">/McInt/
            DataTypeMappings/DataTypeMappingSet</DATA-TYPE-MAPPING-REF>
      </DATA-TYPE-MAPPING-REFS>
      <EVENTS>
        <TIMING-EVENT>
          <SHORT-NAME>controller100ms</SHORT-NAME>
          <START-ON-EVENT-REF DEST="RUNNABLE-ENTITY">/McInt/SwComponents/
              Controller/ControllerInternalBehavior/ControllerRE</START-ON-
              EVENT-REF>
```

```
          <PERIOD>0.1</PERIOD>
        </TIMING-EVENT>
      </EVENTS>
      <RUNNABLES>
        <RUNNABLE-ENTITY>
          <SHORT-NAME>ControllerRE</SHORT-NAME>
          <DATA-READ-ACCESSS>
            <VARIABLE-ACCESS>
              <SHORT-NAME>DRA_temperature</SHORT-NAME>
              <ACCESSED-VARIABLE>
                <AUTOSAR-VARIABLE-IREF>
                  <PORT-PROTOTYPE-REF DEST="R-PORT-PROTOTYPE">/McInt/
                      SwComponents/Controller/TemperatureRPP</PORT-PROTOTYPE
                      -REF>
                  <TARGET-DATA-PROTOTYPE-REF DEST="VARIABLE-DATA-PROTOTYPE"
                      >/McInt/PortInterfaces/TemperatureSRIF/T</TARGET-DATA-
                      PROTOTYPE-REF>
                </AUTOSAR-VARIABLE-IREF>
              </ACCESSED-VARIABLE>
            </VARIABLE-ACCESS>
          </DATA-READ-ACCESSS>
        </RUNNABLE-ENTITY>
      </RUNNABLES>
    </SWC-INTERNAL-BEHAVIOR>
  </INTERNAL-BEHAVIORS>
</APPLICATION-SW-COMPONENT-TYPE>
```

Listing 3.14: Controller

The two ApplicationSwComponentTypes are then used to type SwComponentPrototypes in the Compositon. The PortPrototypes of the SwComponentPrototypes are connected by an AssemblySwConnector:

Common CompositionSwComponentType attributes	
shortName	Composition
properties of the components	
properties of SwComponentPrototype	
shortName	CPT_Controller
type	Controller
properties of SwComponentPrototype	
shortName	CPT_Plant
type	Plant
[...]	

Table 3.9: CompositionSwComponentType Composition

In ARXML:

```
<COMPOSITION-SW-COMPONENT-TYPE>
  <SHORT-NAME>Composition</SHORT-NAME>
  <COMPONENTS>
    <SW-COMPONENT-PROTOTYPE>
      <SHORT-NAME>CPT_Controller</SHORT-NAME>
```

```
      <TYPE-TREF DEST="APPLICATION-SW-COMPONENT-TYPE">/McInt/SwComponents/
          Controller</TYPE-TREF>
    </SW-COMPONENT-PROTOTYPE>
    <SW-COMPONENT-PROTOTYPE>
      <SHORT-NAME>CPT_Plant</SHORT-NAME>
      <TYPE-TREF DEST="APPLICATION-SW-COMPONENT-TYPE">/McInt/SwComponents/
          Plant</TYPE-TREF>
    </SW-COMPONENT-PROTOTYPE>
  </COMPONENTS>
  <CONNECTORS>
    <ASSEMBLY-SW-CONNECTOR>
      <SHORT-NAME>
          ASC_CPT_Plant_TemperaturePPP_CPT_Controller_TemperatureRPP</SHORT-
          NAME>
      <PROVIDER-IREF>
        <CONTEXT-COMPONENT-REF DEST="SW-COMPONENT-PROTOTYPE">/McInt/
            SwComponents/Composition/CPT_Plant</CONTEXT-COMPONENT-REF>
        <TARGET-P-PORT-REF DEST="P-PORT-PROTOTYPE">/McInt/SwComponents/
            Plant/PlantTemperaturePPP</TARGET-P-PORT-REF>
      </PROVIDER-IREF>
      <REQUESTER-IREF>
        <CONTEXT-COMPONENT-REF DEST="SW-COMPONENT-PROTOTYPE">/McInt/
            SwComponents/Composition/CPT_Controller</CONTEXT-COMPONENT-REF>
        <TARGET-R-PORT-REF DEST="R-PORT-PROTOTYPE">/McInt/SwComponents/
            Controller/TemperatureRPP</TARGET-R-PORT-REF>
      </REQUESTER-IREF>
    </ASSEMBLY-SW-CONNECTOR>
  </CONNECTORS>
</COMPOSITION-SW-COMPONENT-TYPE>
```

Listing 3.15: Composition

3.1.6.3 System

In the ECU_Extract, i.e. a System with category ECU_EXTRACT, the Composition is used to type the rootSoftwareComposition. All SwComponentPrototypes in Composition are mapped to the single EcuInstance in this show case.

```
<ECU-INSTANCE>
  <SHORT-NAME>EcuInstance</SHORT-NAME>
</ECU-INSTANCE>
<SYSTEM>
  <SHORT-NAME>EcuExtract</SHORT-NAME>
  <CATEGORY>ECU_EXTRACT</CATEGORY>
  <MAPPINGS>
  <SYSTEM-MAPPING>
    <SHORT-NAME>SystemMapping</SHORT-NAME>
    <SW-MAPPINGS>
    <SWC-TO-ECU-MAPPING>
      <SHORT-NAME>SwcToEcuMapping</SHORT-NAME>
      <COMPONENT-IREFS>
      <COMPONENT-IREF>
```

```
        <CONTEXT-COMPOSITION-REF DEST="ROOT-SW-COMPOSITION-PROTOTYPE">/
            McInt/System/EcuExtract/RootSwCompositionPrototype</CONTEXT-
            COMPOSITION-REF>
        <TARGET-COMPONENT-REF DEST="SW-COMPONENT-PROTOTYPE">/McInt/
            SwComponents/Composition/CPT_Controller</TARGET-COMPONENT-REF>
      </COMPONENT-IREF>
      <COMPONENT-IREF>
        <CONTEXT-COMPOSITION-REF DEST="ROOT-SW-COMPOSITION-PROTOTYPE">/
            McInt/System/EcuExtract/RootSwCompositionPrototype</CONTEXT-
            COMPOSITION-REF>
        <TARGET-COMPONENT-REF DEST="SW-COMPONENT-PROTOTYPE">/McInt/
            SwComponents/Composition/CPT_Plant</TARGET-COMPONENT-REF>
      </COMPONENT-IREF>
      </COMPONENT-IREFS>
      <ECU-INSTANCE-REF DEST="ECU-INSTANCE">/McInt/System/EcuInstance</ECU-
          INSTANCE-REF>
    </SWC-TO-ECU-MAPPING>
    </SW-MAPPINGS>
  </SYSTEM-MAPPING>
  </MAPPINGS>
  <ROOT-SOFTWARE-COMPOSITIONS>
  <ROOT-SW-COMPOSITION-PROTOTYPE>
    <SHORT-NAME>RootSwCompositionPrototype</SHORT-NAME>
    <FLAT-MAP-REF DEST="FLAT-MAP">/McInt/FlatMaps/FlatMap</FLAT-MAP-REF>
    <SOFTWARE-COMPOSITION-TREF DEST="COMPOSITION-SW-COMPONENT-TYPE">/McInt/
        SwComponents/Composition</SOFTWARE-COMPOSITION-TREF>
  </ROOT-SW-COMPOSITION-PROTOTYPE>
  </ROOT-SOFTWARE-COMPOSITIONS>
</SYSTEM>
```

Listing 3.16: System and EcuInstance

The FlatMap that is referenced in the ECU_Extract, gives the name TPlant to a dataElement (see ecuExtractReference below). The name TPlant is later on displayed in the MC Tool.

```
<FLAT-MAP>
  <SHORT-NAME>FlatMap</SHORT-NAME>
  <INSTANCES>
  <FLAT-INSTANCE-DESCRIPTOR>
    <SHORT-NAME>TPlant</SHORT-NAME>
    <ECU-EXTRACT-REFERENCE-IREF>
    <CONTEXT-ELEMENT-REF DEST="ROOT-SW-COMPOSITION-PROTOTYPE">/McInt/System
        /EcuExtract/RootSwCompositionPrototype</CONTEXT-ELEMENT-REF>
    <CONTEXT-ELEMENT-REF DEST="SW-COMPONENT-PROTOTYPE">/McInt/SwComponents/
        Composition/CPT_Plant</CONTEXT-ELEMENT-REF>
    <CONTEXT-ELEMENT-REF DEST="P-PORT-PROTOTYPE">/McInt/SwComponents/Plant/
        PlantTemperaturePPP</CONTEXT-ELEMENT-REF>
    <TARGET-REF DEST="VARIABLE-DATA-PROTOTYPE">/McInt/PortInterfaces/
        TemperatureSRIF/T</TARGET-REF>
    </ECU-EXTRACT-REFERENCE-IREF>
  </FLAT-INSTANCE-DESCRIPTOR>
  </INSTANCES>
</FLAT-MAP>
```

Listing 3.17: FlatMap

3.1.6.4 ECU Configuration

There are further things that need to be defined before the RTE and the OS can be generated. For instance, the order in which the RTEEvents for the RunnableEntitys are invoked and the assignment to an OsTask. This is done via EcucModuleConfigurationValues. The interesting parts of the RTE configuration are:

```
<ECUC-CONTAINER-VALUE>
  <SHORT-NAME>controller100ms</SHORT-NAME>
  <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/EcucDefs/
     Rte/RteSwComponentInstance/RteEventToTaskMapping</DEFINITION-REF>
  <PARAMETER-VALUES>
  <ECUC-NUMERICAL-PARAM-VALUE>
    <DEFINITION-REF DEST="ECUC-INTEGER-PARAM-DEF">/AUTOSAR/EcucDefs/Rte/
       RteSwComponentInstance/RteEventToTaskMapping/RtePositionInTask</
       DEFINITION-REF>
    <VALUE>3</VALUE>
  </ECUC-NUMERICAL-PARAM-VALUE>

  </PARAMETER-VALUES>
  <REFERENCE-VALUES>
  <ECUC-REFERENCE-VALUE>
    <DEFINITION-REF DEST="ECUC-REFERENCE-DEF">/AUTOSAR/EcucDefs/Rte/
       RteSwComponentInstance/RteEventToTaskMapping/RteMappedToTaskRef</
       DEFINITION-REF>
    <VALUE-REF DEST="ECUC-CONTAINER-VALUE">/McInt/OS/OS_CFG/task_100ms</
       VALUE-REF>
  </ECUC-REFERENCE-VALUE>
  <ECUC-REFERENCE-VALUE>
    <DEFINITION-REF DEST="ECUC-FOREIGN-REFERENCE-DEF">/AUTOSAR/EcucDefs/Rte
       /RteSwComponentInstance/RteEventToTaskMapping/RteEventRef</
       DEFINITION-REF>
    <VALUE-REF DEST="TIMING-EVENT">/McInt/SwComponents/Controller/
       ControllerInternalBehavior/controller100ms</VALUE-REF>
  </ECUC-REFERENCE-VALUE>
  </REFERENCE-VALUES>
</ECUC-CONTAINER-VALUE>

<ECUC-CONTAINER-VALUE>
  <SHORT-NAME>plant100ms</SHORT-NAME>
  <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/EcucDefs/
     Rte/RteSwComponentInstance/RteEventToTaskMapping</DEFINITION-REF>
  <PARAMETER-VALUES>
  <ECUC-NUMERICAL-PARAM-VALUE>
    <DEFINITION-REF DEST="ECUC-INTEGER-PARAM-DEF">/AUTOSAR/EcucDefs/Rte/
       RteSwComponentInstance/RteEventToTaskMapping/RtePositionInTask</
       DEFINITION-REF>
    <VALUE>2</VALUE>
  </ECUC-NUMERICAL-PARAM-VALUE>

  </PARAMETER-VALUES>
  <REFERENCE-VALUES>
  <ECUC-REFERENCE-VALUE>
    <DEFINITION-REF DEST="ECUC-REFERENCE-DEF">/AUTOSAR/EcucDefs/Rte/
       RteSwComponentInstance/RteEventToTaskMapping/RteMappedToTaskRef</
       DEFINITION-REF>
```

```
    <VALUE-REF DEST="ECUC-CONTAINER-VALUE">/McInt/OS/OS_CFG/task_100ms</
        VALUE-REF>
  </ECUC-REFERENCE-VALUE>
  <ECUC-REFERENCE-VALUE>
    <DEFINITION-REF DEST="ECUC-FOREIGN-REFERENCE-DEF">/AUTOSAR/EcucDefs/Rte
        /RteSwComponentInstance/RteEventToTaskMapping/RteEventRef</
        DEFINITION-REF>
    <VALUE-REF DEST="TIMING-EVENT">/McInt/SwComponents/Plant/
        PlantInternalBehavior/plant100ms</VALUE-REF>
  </ECUC-REFERENCE-VALUE>
  </REFERENCE-VALUES>
</ECUC-CONTAINER-VALUE>
```

Listing 3.18: RTE Config

This part of the OS configuration defines the name of the OSTask, that we see later on in the generated C code:

```
<ECUC-MODULE-CONFIGURATION-VALUES>
  <SHORT-NAME>OS_CFG</SHORT-NAME>
  <DEFINITION-REF DEST="ECUC-MODULE-DEF">/AUTOSAR/EcucDefs/Os</DEFINITION-
      REF>
  <CONTAINERS>
    <ECUC-CONTAINER-VALUE>
      <SHORT-NAME>task_100ms</SHORT-NAME>
      <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/
          EcucDefs/Os/OsTask</DEFINITION-REF>
    </ECUC-CONTAINER-VALUE>
  </CONTAINERS>
</ECUC-MODULE-CONFIGURATION-VALUES>
```

Listing 3.19: OsConfig

These configurations are tied to the ECU_Extract by an EcucValueCollection:

```
<ECUC-VALUE-COLLECTION>
  <SHORT-NAME>EcucValueCollection</SHORT-NAME>
  <ECU-EXTRACT-REF DEST="SYSTEM">/McInt/System/EcuExtract</ECU-EXTRACT-REF>
  <ECUC-VALUES>
  <ECUC-MODULE-CONFIGURATION-VALUES-REF-CONDITIONAL>
    <ECUC-MODULE-CONFIGURATION-VALUES-REF DEST="ECUC-MODULE-CONFIGURATION-
        VALUES">/McInt/RTE/RTE_CFG</ECUC-MODULE-CONFIGURATION-VALUES-REF>
  </ECUC-MODULE-CONFIGURATION-VALUES-REF-CONDITIONAL>
  <ECUC-MODULE-CONFIGURATION-VALUES-REF-CONDITIONAL>
    <ECUC-MODULE-CONFIGURATION-VALUES-REF DEST="ECUC-MODULE-CONFIGURATION-
        VALUES">/McInt/OS/OS_CFG</ECUC-MODULE-CONFIGURATION-VALUES-REF>
  </ECUC-MODULE-CONFIGURATION-VALUES-REF-CONDITIONAL>
  </ECUC-VALUES>
</ECUC-VALUE-COLLECTION>
```

Listing 3.20: EcuC Value Collection

This completes the presentation of the AUTOSAR modeling in our walk through.

3.1.6.5 RTE Generation

In the following, some snippets of the generated RTE are presented. However, they are examples only and may differ if different RTE generators are used.

Among other things, the OsTask is generated as defined in the ECU configuration above:

```
...
#define RTE_START_SEC_VAR
#include "MemMap.h" /*lint !e537 permit multiple inclusion */
...
VAR(float32, RTE_DATA) TPlant;
...
#define RTE_STOP_SEC_VAR
#include "MemMap.h" /*lint !e537 permit multiple inclusion */
...
TASK(task_100ms)
{
    ...
    Rte_ImplicitBufs.isa_1._task_100ms.sbuf1.value = TPlant;
    ...
    plantRE_func();
    ...
    controllerRE_func();
    ...
    TPlant = Rte_ImplicitBufs.isa_1._task_100ms.sbuf1.value;
    ...
} /* task_100ms */
...
```

Listing 3.21: Rte.c

Also a MACRO to write T_{Plant} in the Plant

```
...
#define Rte_IRead_plantRE_EnvTemperatureRPP_T() ((CONST(float32,
    RTE_DATA)) Rte_ImplicitBufs.isa_1._task_100ms.sbuf0.value )
...
```

Listing 3.22: Rte_Plant.h

and to read T_{Plant} in the Controller

```
...
#define Rte_IRead_ControllerRE_TemperatureRPP_T() ((CONST(float32,
    RTE_DATA)) Rte_ImplicitBufs.isa_1._task_100ms.sbuf1.value )
...
```

Listing 3.23: Rte_Controller.h

was generated. Furthermore, the McSupport file is generated as an interface between the "AUTOSAR world" and the "A2L world". As the reader can see, this is a compilation of necessary data from the AUTOSAR model presented before:

```
<BSW-IMPLEMENTATION>
  <SHORT-NAME>Rte</SHORT-NAME>
```

```
  <MC-SUPPORT>
    <MC-VARIABLE-INSTANCES>
      <MC-DATA-INSTANCE>
       <SHORT-NAME>TPlant</SHORT-NAME>
       <DESC>
         <L-2 L="EN">Type for a temperature in [°C]</L-2>
       </DESC>
       <CATEGORY>VALUE</CATEGORY>
       <FLAT-MAP-ENTRY-REF DEST="FLAT-INSTANCE-DESCRIPTOR">/McInt/FlatMaps/
          FlatMap/TPlant</FLAT-MAP-ENTRY-REF>
       <RESULTING-PROPERTIES>
         <SW-DATA-DEF-PROPS-VARIANTS>
           <SW-DATA-DEF-PROPS-CONDITIONAL>
             <BASE-TYPE-REF DEST="SW-BASE-TYPE">/AUTOSAR_PlatformTypes/
                SwBaseTypes/float32</BASE-TYPE-REF>
             <SW-CALIBRATION-ACCESS>READ-ONLY</SW-CALIBRATION-ACCESS>
             <COMPU-METHOD-REF DEST="COMPU-METHOD">/McInt/CompuMethods/
                Temperature_C</COMPU-METHOD-REF>
             <DISPLAY-FORMAT>%.1f</DISPLAY-FORMAT>
             <UNIT-REF DEST="UNIT">/McInt/Units/DegreeCelsius</UNIT-REF>
           </SW-DATA-DEF-PROPS-CONDITIONAL>
         </SW-DATA-DEF-PROPS-VARIANTS>
       </RESULTING-PROPERTIES>
       <SYMBOL>TPlant</SYMBOL>
       </MC-DATA-INSTANCE>
    </MC-VARIABLE-INSTANCES>
  </MC-SUPPORT>
</BSW-IMPLEMENTATION>
```

Listing 3.24: McSupportData

3.1.6.6 Implementation in C

The implementation in C-Code is a direct implementation of the physical equations. The `Plant` uses the `MACRO`, generated by the `RTE` generator, to write T_{Plant}:

```
1  #include "Rte_Plant.h"
2  ...
3  FUNC (void, Plant_CODE) plantRE_func (void)
4  {
5     ...
6     /* heat capacity of 1 assumed                                    */
7     float32 lTPlant     = lQPlant;
8      /* calculate heat flows, store in PIM to make them measurable */
9     *Rte_Pim_QEnv()     = (lTenv - lTPlant) * lEFactor * lDt;
10    ...
11    /* heat capacity of 1 assumed                                    */
12    lTPlant  = lQPlant;
13    ...
14    /* Write output of plant: temerature of plant                    */
15    Rte_IWrite_plantRE_PlantTemperaturePPP_T(lTPlant);
16 }
```

Listing 3.25: Plant

The Controller uses the MACRO, generated by the RTE generator, to read T_{Plant}:

```
#include "Rte_Controller.h"
...
FUNC (void, Controller_CODE) controllerRE_func (void)
{
   /* read input, define output variable                          */
   float32 lT        = Rte_IRead_ControllerRE_TemperatureRPP_T();
   ...

    /* store current error in PIM to make it measurable           */
   *Rte_Pim_E() = lSetPoint - lT;
   ...
}
```

Listing 3.26: Controller

3.1.6.7 A2L File

Using the McSupport file and the map file from the linker, an example A2L file was generated for this show case. The snippet below is an example only and could differ if a different A2L file generator is used:

```
...
/begin MEASUREMENT TPlant
       "TPlant"
       FLOAT32_IEEE
       McInt_CompuMethods_Temperature_C
       0
       0
       -1E+32
       1E+32
       DISPLAY_IDENTIFIER  "TPlant"
       ECU_ADDRESS 0xe000001c
       FORMAT  "%.1f"
       PHYS_UNIT "°C"
/end MEASUREMENT
...
/begin UNIT McInt_PhysicalDimensions_Temparature
       "McInt_PhysicalDimensions_Temparature"
       "McInt_PhysicalDimensions_Temparature"
       EXTENDED_SI
       SI_EXPONENTS 0 0 0 0 1 0 0
/end UNIT
/begin UNIT McInt_Units_DegreeCelsius
       "McInt_Units_DegreeCelsius"
       "°C"
       DERIVED
       REF_UNIT McInt_PhysicalDimensions_Temparature
       UNIT_CONVERSION 1 -273.15
/end UNIT
/begin COMPU_METHOD McInt_CompuMethods_Temperature_C
       "McInt_CompuMethods_Temperature_C"
       LINEAR
       "%f"
```

```
33          "°C"
34          COEFFS_LINEAR 1 -273.15
35          REF_UNIT McInt_Units_DegreeCelsius
36  /end COMPU_METHOD
37  ...
```

Listing 3.27: A2L File

3.1.6.8 Measurement and Calibration Tool

The A2L file is then used by a MC tool to measure T_{Plant}. Of course, in addition to the A2L file a suitable ECU access[4] must be available, to actually do measurement and calibration with the AUTOSAR system of this show case. However, the ECU access is not presented because this is not in the focus of this show case.

Below is a typical screen shot from a MC tool during an actual measurement and calibration task. You can see T_{Plant} measured and displayed in degree Celsius.

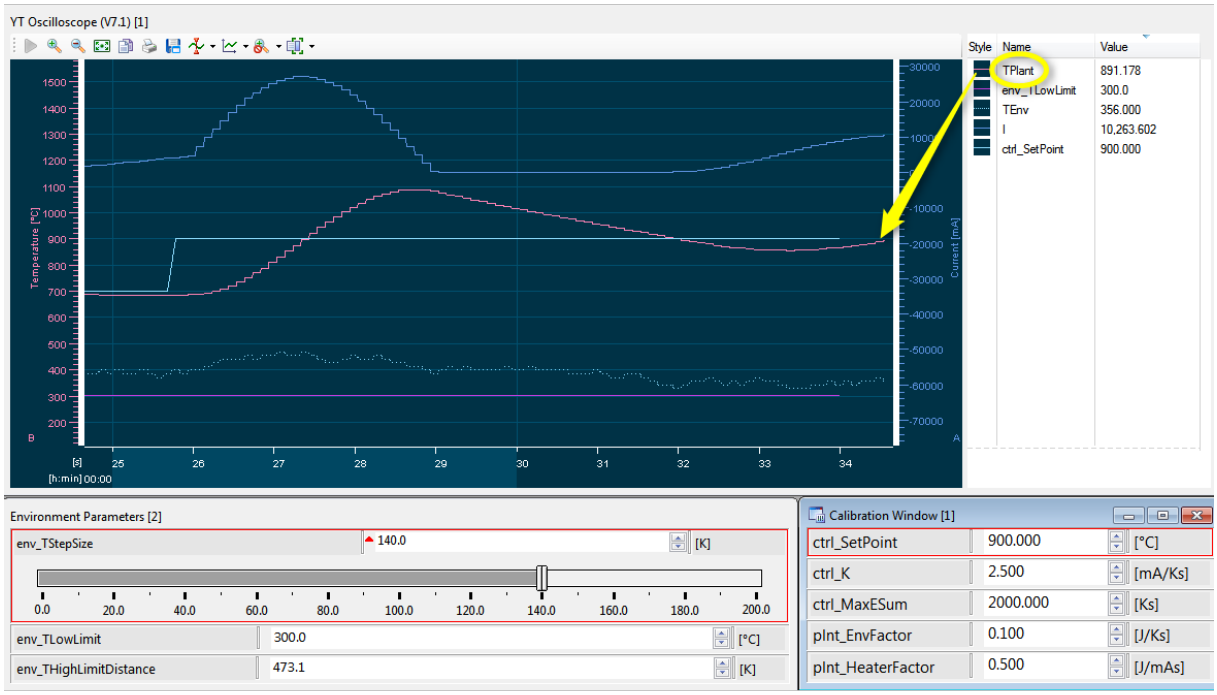


Figure 3.12: Screenshot of a MC Tool

[4] for instance, a measurement and calibration service like XCP or a hardware access to the memory of the micro controller

3.1.7 Show cases in the Example

3.1.7.1 CompositionSwComponentTypes

Common CompositionSwComponentType attributes	
shortName	Composition
properties of the components	
properties of SwComponentPrototype	
shortName	CPT_Controller
type	Controller
properties of SwComponentPrototype	
shortName	CPT_Parameters
type	Parameters
properties of SwComponentPrototype	
shortName	CPT_Plant
type	Plant
properties of SwComponentPrototype	
shortName	CPT_Environment
type	Environment

Table 3.10: CompositionSwComponentType Composition

3.1.7.2 ParameterSwComponentTypes

Common ParameterSwComponentType attributes	
shortName	Parameters
desc	Type for providing the parameters to the ApplicationSwCompoments
properties of the ports	
properties of PPortPrototype	
shortName	ControllerPPP
desc	Port for providing the parameters for the controller
providedInterface	ControllerPIF
properties of PPortPrototype	
shortName	PlantPPP
desc	Port for providing the parameters for the plant
providedInterface	PlantPIF
properties of PPortPrototype	
shortName	EnvironmentPPP
desc	Port for providing the parameters for the environment
providedInterface	EnvironmentPIF
properties of PPortPrototype	
shortName	DtPPP
desc	Time of one time step
providedInterface	DtPIF

Table 3.11: ParameterSwComponentType Parameters

3.1.7.3 ApplicationSwComponentTypes

Common ApplicationSwComponentType attributes	
shortName	Controller
properties of the ports	
properties of RPortPrototype	
shortName	TemperatureRPP
desc	Port to receive the temperature of the plant
requiredInterface	TemperatureSRIF
properties of PPortPrototype	
shortName	CurrentPPP
desc	Port for sending out the current output by this controller
providedInterface	CurrentSRIF
properties of RPortPrototype	
shortName	ControllerParamsRPP
desc	Port to get the parameters for the controller
requiredInterface	ControllerPIF
properties of RPortPrototype	
shortName	DtRPP
desc	Port to get delta t, i.e. time of one time step
requiredInterface	DtPIF
internalBehavior	ControllerInternalBehavior

Table 3.12: ApplicationSwComponentType Controller

Common SwcInternalBehavior attributes	
shortName	ControllerInternalBehavior
properties of implicitInterRunnableVariables / explicitInterRunnableVariables	
properties of VariableDataPrototype	
shortName	ESum
desc	Internal state of the controller: the sum of control errors
type	ESum
swImplPolicy	standard
swCalibrationAccess	readOnly
swAddrMethod	CODE
properties of the arTypedPerInstanceMemorys	
properties of VariableDataPrototype	
shortName	E
desc	Measurement point for the control error, the deviation between set point and acutal temperature of the plant, in the current time step
type	Temperature_K
swImplPolicy	standard
swCalibrationAccess	readOnly
swAddrMethod	CODE
properties of the runnables	
properties of RunnableEntity	
shortName	ControllerRE
symbol	controllerRE_func
properties of the events	
properties of TimingEvent	
shortName	controller100ms
startOnEvent	ControllerRE
period	0.1

Table 3.13: SwcInternalBehavior ControllerInternalBehavior

Common ApplicationSwComponentType attributes	
shortName	Plant
properties of the ports	
properties of RPortPrototype	
shortName	CurrentRPP
desc	Port to receive the current from the controller
requiredInterface	CurrentSRIF
properties of PPortPrototype	
shortName	PlantTemperaturePPP
desc	Port for sending out the estimated temperature of the plant
providedInterface	TemperatureSRIF
properties of RPortPrototype	
shortName	PlantParamsRPP
desc	Port to get the parameters for the plant
requiredInterface	PlantPIF
properties of RPortPrototype	
shortName	EnvTemperatureRPP
desc	Port to receive the tempertature of the environment
requiredInterface	TemperatureSRIF
properties of RPortPrototype	
shortName	DtRPP
desc	Port to get delta t, i.e. time of one time step
requiredInterface	DtPIF
internalBehavior	PlantInternalBehavior

Table 3.14: ApplicationSwComponentType Plant

Common SwcInternalBehavior attributes	
shortName	PlantInternalBehavior
properties of implicitInterRunnableVariables / explicitInterRunnableVariables	
properties of VariableDataPrototype	
shortName	QPlant
desc	Internal state of the plant: the stored eneregy quantity in the current time step
type	Energy
swImplPolicy	standard
swCalibrationAccess	readOnly
swAddrMethod	CODE
properties of the arTypedPerInstanceMemorys	
properties of VariableDataPrototype	
shortName	QHeater
desc	Measurement point for heat flow between the electrical heater and the plant in the current time step.
type	Energy
swImplPolicy	standard
swCalibrationAccess	readOnly
swAddrMethod	VAR
properties of VariableDataPrototype	
shortName	QEnv
desc	Measurement point for heat flow between the plant and the environment in the current time step.
type	Energy
swImplPolicy	standard
swCalibrationAccess	readOnly
swAddrMethod	CODE
properties of the runnables	
properties of RunnableEntity	
shortName	plantRE
symbol	plantRE_func
properties of the events	
properties of TimingEvent	
shortName	plant100ms
startOnEvent	plantRE
period	0.1

Table 3.15: SwcInternalBehavior PlantInternalBehavior

Common ApplicationSwComponentType attributes	
shortName	Environment
properties of the ports	
properties of PPortPrototype	
shortName	EnvTemperaturePPP
desc	Port to send out the temperature of the environment
providedInterface	TemperatureSRIF
properties of RPortPrototype	
shortName	EnvParamsRPP
desc	Port to get the parameters for the environment
requiredInterface	EnvironmentPIF
properties of RPortPrototype	
shortName	DtRPP
desc	Port to get delta t, i.e. time of one time step
requiredInterface	DtPIF
internalBehavior	EnvironmentInternalBehavior

Table 3.16: ApplicationSwComponentType Environment

Common SwcInternalBehavior attributes	
shortName	EnvironmentInternalBehavior
properties of implicitInterRunnableVariables / explicitInterRunnableVariables	
properties of VariableDataPrototype	
shortName	Seed
desc	Internal state of the environment: the current seed for the (pseudo) random number generation
type	uint32
swImplPolicy	standard
swCalibrationAccess	notAccessible
swAddrMethod	CODE
properties of VariableDataPrototype	
shortName	TEnv
desc	Internal state of the environment: the temperture of the environment
type	Temperature_C
swImplPolicy	standard
swCalibrationAccess	readOnly
swAddrMethod	CODE
properties of the arTypedPerInstanceMemorys	
properties of VariableDataPrototype	
shortName	THighLimit
desc	Measurement point for the upper limit of the generated temperature profile
type	Temperature_C
swImplPolicy	standard
swCalibrationAccess	readOnly
swAddrMethod	CODE
properties of the runnables	
properties of RunnableEntity	
shortName	envRE
symbol	envRE_func
properties of the events	
properties of TimingEvent	
shortName	env100ms
startOnEvent	envRE
period	0.1

Table 3.17: SwcInternalBehavior EnvironmentInternalBehavior

3.1.7.4 ParameterInterfaces

Common ParameterInterface attributes	
shortName	ControllerPIF
desc	Interface with all parameters for the controller
properties of the parameters	
properties of ParameterDataPrototype	
shortName	ctrl_SetPoint
desc	Set point for the temperature of the plant
type	Temperature_C
swImplPolicy	standard
swCalibrationAccess	readWrite
swAddrMethod	CALIB
properties of ParameterDataPrototype	
shortName	ctrl_K
desc	Amplification factor for the I-controller
type	Amplification
swImplPolicy	standard
swCalibrationAccess	readWrite
swAddrMethod	CALIB
properties of ParameterDataPrototype	
shortName	ctrl_MaxESum
desc	Upper limit of the integal part of the I-controller
type	ESum
swImplPolicy	standard
swCalibrationAccess	readWrite
swAddrMethod	CALIB

Table 3.18: ParameterInterface ControllerPIF

Common ParameterInterface attributes	
shortName	PlantPIF
desc	Interface with all parameters for the plant
properties of the parameters	
properties of ParameterDataPrototype	
shortName	pInt_EnvFactor
desc	Proportionality factor for the heat flow between plant and environment
type	EnvFactor
swImplPolicy	standard
swCalibrationAccess	readWrite
swAddrMethod	CALIB
properties of ParameterDataPrototype	
shortName	pInt_HeaterFactor
desc	Proportionality factor for the heat flow between plant and the electrical heater
type	HeaterFactor
swImplPolicy	standard
swCalibrationAccess	readWrite
swAddrMethod	CALIB

Table 3.19: ParameterInterface PlantPIF

Common ParameterInterface attributes	
shortName	EnvironmentPIF
desc	Interface with all parameters for the environment
properties of the parameters	
properties of ParameterDataPrototype	
shortName	env_TLowLimit
desc	Lower limit of the generated temeprature profile
type	Temperature_C
swImplPolicy	standard
swCalibrationAccess	readWrite
swAddrMethod	CALIB
properties of ParameterDataPrototype	
shortName	env_TStepSize
desc	The maximal temperature diffenrence of the environment in one time step
type	Temperature_K
swImplPolicy	standard
swCalibrationAccess	readWrite
swAddrMethod	CALIB
properties of ParameterDataPrototype	
shortName	env_THighLimitDistance
desc	Distance of the upper limit from the lower limit for the generated temeprature profile.
type	Temperature_K
swImplPolicy	standard
swCalibrationAccess	readWrite
swAddrMethod	CALIB

Table 3.20: ParameterInterface EnvironmentPIF

Common ParameterInterface attributes	
shortName	DtPIF
properties of the parameters	
properties of ParameterDataPrototype	
shortName	Dt
desc	Scheduling time of the components
type	Time
swImplPolicy	standard
swCalibrationAccess	readWrite
swAddrMethod	CALIB

Table 3.21: ParameterInterface DtPIF

3.1.7.5 SenderReceiverInterfaces

Common SenderReceiverInterface attributes	
shortName	TemperatureSRIF
desc	Interface type for transferring temperatures in [℃]
properties of the dataElements	
properties of VariableDataPrototype	
shortName	T
type	Temperature_C
swImplPolicy	standard
swCalibrationAccess	readOnly
swAddrMethod	VAR

Table 3.22: SenderReceiverInterface TemperatureSRIF

Common SenderReceiverInterface attributes	
shortName	CurrentSRIF
desc	Interface type for transferring a current in [mA]
properties of the dataElements	
properties of VariableDataPrototype	
shortName	I
type	Current
swImplPolicy	standard
swCalibrationAccess	readOnly
swAddrMethod	VAR

Table 3.23: SenderReceiverInterface CurrentSRIF

3.1.7.6 ApplicationDataTypes, Category VALUE

Common ApplicationDataType attributes				
shortName	Temperature_C			
category	VALUE			
desc	Type for a temperature in [℃]			
swCalibrationAccess	readOnly			
unit	DegreeCelsius			
Range				
Conversion				
	category	LINEAR		
	direction	compuInternalToPhys		
	desc	lowerLimit	upperLimit	compuNumerator / compuDenominator
	-	-	-	$Phys = \frac{-273.15 + 1 * Internal}{1}$

Table 3.24: ApplicationDataType Temperature_C

Common ApplicationDataType attributes				
shortName	Current			
category	VALUE			
desc	Type for the current in [mA]			
swCalibrationAccess	readOnly			
unit	MilliAmpere			
Range				
Conversion				
	category	LINEAR		
	direction	compuInternalToPhys		
	desc	lowerLimit	upperLimit	compuNumerator / compuDenominator
	-	-	-	$Phys = \frac{0 + 1000 * Internal}{1}$

Table 3.25: ApplicationDataType Current

Common ApplicationDataType attributes	
shortName	EnvFactor
category	VALUE
desc	Type for the environt factor in [J/Ks]
swCalibrationAccess	readOnly
unit	JoulePerKelvinSecond
Range	
Conversion	
category	IDENTICAL
direction	-

Table 3.26: ApplicationDataType EnvFactor

Common ApplicationDataType attributes	
shortName	Temperature_K
category	VALUE
desc	Type for a temperature in [K]
swCalibrationAccess	readOnly
unit	Kelvin
Range	
Conversion	
category	IDENTICAL
direction	-

Table 3.27: ApplicationDataType Temperature_K

Common ApplicationDataType attributes	
shortName	Amplification
category	VALUE
desc	Type for an amplification factor in a controller in [mA/Ks]
swCalibrationAccess	readOnly
unit	MilliAmperePerKelvinSecond
Range	
Conversion	
category	IDENTICAL
direction	-

Table 3.28: ApplicationDataType Amplification

Common ApplicationDataType attributes	
shortName	Energy
category	VALUE
desc	Type for energy [J]
swCalibrationAccess	readOnly
unit	Joule
Range	
Conversion	
category	IDENTICAL
direction	-

Table 3.29: ApplicationDataType Energy

Common ApplicationDataType attributes	
shortName	ESum
category	VALUE
desc	Type for the sum of control errors of an I controller in [Ks]
swCalibrationAccess	readOnly
unit	KelvinSecond
Range	
Conversion	
category	IDENTICAL
direction	-

Table 3.30: ApplicationDataType ESum

Common ApplicationDataType attributes	
shortName	HeaterFactor
category	VALUE
desc	Type of a proportionality factor for the heat flow from an electrical heater to a thermal energy storage in [J/mAs]
swCalibrationAccess	readOnly
unit	JoulePerMilliAmpereSecond
Range	
Conversion	
category	IDENTICAL
direction	-

Table 3.31: ApplicationDataType HeaterFactor

Common ApplicationDataType attributes	
shortName	Time
category	VALUE
desc	Type for time in [s]
swCalibrationAccess	readOnly
unit	Second
Range	
Conversion	
category	IDENTICAL
direction	-

Table 3.32: ApplicationDataType Time

3.1.7.7 Units

Common Unit attributes	
shortName	DegreeCelsius
displayName	℃
offsetSiToUnit	-273.15
factorSiToUnit	1.0
physicalDimension	Temperature

Table 3.33: Unit DegreeCelsius

Common Unit attributes	
shortName	Kelvin
displayName	K
offsetSiToUnit	0.0
factorSiToUnit	1.0
physicalDimension	Temperature

Table 3.34: Unit Kelvin

Common Unit attributes	
shortName	Joule
displayName	J
offsetSiToUnit	0.0
factorSiToUnit	1.0
physicalDimension	Energy

Table 3.35: Unit Joule

Common Unit attributes	
shortName	MilliAmpere
displayName	mA
offsetSiToUnit	0.0
factorSiToUnit	1000.0
physicalDimension	Current

Table 3.36: Unit MilliAmpere

Common Unit attributes	
shortName	KelvinSecond
displayName	Ks
offsetSiToUnit	0.0
factorSiToUnit	1.0
physicalDimension	TemperatureTime

Table 3.37: Unit KelvinSecond

Common Unit attributes	
shortName	JoulePerKelvinSecond
displayName	J/Ks
offsetSiToUnit	0.0
factorSiToUnit	1.0
physicalDimension	EnergyPerTemperatureTime

Table 3.38: Unit JoulePerKelvinSecond

Common Unit attributes	
shortName	JoulePerMilliAmpereSecond
displayName	J/mAs
offsetSiToUnit	0.0
factorSiToUnit	0.001
physicalDimension	EnergyPerCurrentTime

Table 3.39: Unit JoulePerMilliAmpereSecond

Common Unit attributes	
shortName	MilliAmperePerKelvinSecond
displayName	mA/Ks
offsetSiToUnit	0.0
factorSiToUnit	1000.0
physicalDimension	CurrentPerTemperatureTime

Table 3.40: Unit MilliAmperePerKelvinSecond

Common Unit attributes	
shortName	Second
displayName	s
offsetSiToUnit	0.0
factorSiToUnit	1.0
physicalDimension	Time

Table 3.41: Unit Second

3.1.7.8 PhysicalDimensions

Common PhysicalDimension attributes	
shortName	Energy
currentExp	0
lengthExp	2
luminousIntensity-Exp	0
massExp	1
molarAmountExp	0
temperatureExp	0
timeExp	-2

Table 3.42: PhysicalDimension Energy

Common PhysicalDimension attributes	
shortName	Current
currentExp	1
lengthExp	0
luminousIntensity-Exp	0
massExp	0
molarAmountExp	0
temperatureExp	0
timeExp	0

Table 3.43: PhysicalDimension Current

Common PhysicalDimension attributes	
shortName	CurrentPerTemperatureTime
currentExp	1
lengthExp	0
luminousIntensity-Exp	0
massExp	0
molarAmountExp	0
temperatureExp	-1
timeExp	-1

Table 3.44: PhysicalDimension CurrentPerTemperatureTime

Common PhysicalDimension attributes	
shortName	EnergyPerCurrentTime
currentExp	-1
lengthExp	2
luminousIntensity-Exp	0
massExp	1
molarAmountExp	0
temperatureExp	0
timeExp	-3

Table 3.45: PhysicalDimension EnergyPerCurrentTime

Common PhysicalDimension attributes	
shortName	EnergyPerTemperatureTime
currentExp	0
lengthExp	2
luminousIntensity-Exp	0
massExp	1
molarAmountExp	0
temperatureExp	-1
timeExp	-3

Table 3.46: PhysicalDimension EnergyPerTemperatureTime

Common PhysicalDimension attributes	
shortName	Time
currentExp	0
lengthExp	0
luminousIntensity-Exp	0
massExp	0
molarAmountExp	0
temperatureExp	0
timeExp	1

Table 3.47: PhysicalDimension Time

Common PhysicalDimension attributes	
shortName	Temperature
currentExp	0
lengthExp	0
luminousIntensity-Exp	0
massExp	0
molarAmountExp	0
temperatureExp	1
timeExp	0

Table 3.48: PhysicalDimension Temperature

Common PhysicalDimension attributes	
shortName	TemperatureTime
currentExp	0
lengthExp	0
luminousIntensity-Exp	0
massExp	0
molarAmountExp	0
temperatureExp	1
timeExp	1

Table 3.49: PhysicalDimension TemperatureTime

3.1.7.9 SwAddrMethods

Common SwAddrMethod attributes	
shortName	VAR
desc	Memory section for variables
sectionType	var
memoryAllocation-KeywordPolicy	addrMethodShortName
section-Initialization-Policy	-
option	safetyQM

Table 3.50: SwAddrMethod VAR

Common SwAddrMethod attributes	
shortName	CALIB
desc	Memory section for calibration parameters
sectionType	var
memoryAllocation-KeywordPolicy	addrMethodShortName
section-Initialization-Policy	-
option	safetyQM

Table 3.51: SwAddrMethod CALIB

Common SwAddrMethod attributes	
shortName	CODE
desc	Memory section for code
sectionType	var
memoryAllocation-KeywordPolicy	addrMethodShortName
section-Initialization-Policy	-
option	safetyQM

Table 3.52: SwAddrMethod CODE

3.2 Advanced Show Case

3.2.1 General Objectives of the Model Structure

3.2.1.1 The Ecu Description

Since the show case is focusing on measurement and calibration only a minimal system model is provided. Hereby the file `Pprj_EcuDescr_U_SystemNodeStub.arxml` defines the `System` `SystemU_EcuDescr` of `category` `ECU_SYSTEM_DESCRIPTION` which contains only the `RootSwCompositionPrototype`. The file `Pprj_EcuDescr_U.arxml` contains the according `CompositionSwComponentType` describing the hierarchical top-level-composition of software components shown in table `SystemURoot-Composition_EcuDescr`.

3.2.1.2 The Ecu Extract

The file `Pprj_EcuExtract_U_SystemNodeStub.arxml` defines the `System` `SystemU_System` of `category` `ECU_EXTRACT` which contains only the `RootSwCompositionPrototype` `SystemU` referencing the `ECU Flat Map` and the flat top-level-composition `SystemU_Root`. The file `Pprj_EcuExtract_U.arxml` contains the according `CompositionSwComponentType` describing the flat top-level-composition of software components shown in table `SystemU_Root`.

Please note that the flat top-level-composition uses the identical software component types as the hierarchical top-level-composition. Therefore an identification of component and data instances in the hierarchical software component structure or in the flat structure requires the correct iteration from the according `System` nodes.

3.2.1.2.1 The ECU Flat Map

The file `Pprj_EcuExtract_U_FlatMap.arxml` contains the `ECU Flat Map`.

The `ECU Flat Map` is utilized to assign unique and comprehensible names to all `DataPrototype`s representing measurements and characteristics. This is important for the calibration engineers[5]

The applied strategy for the creation of a `FlatInstanceDescriptor.shortName` is to shorten it to the `shortName` of the `DataPrototype` when only a single instance of the `DataPrototype` is used.

[5] Calibration engineers in this context means the engineers working with measurement and calibration tooling e.g. to determine the correct calibration parameter values in order to adopt functionality in the software components to the mechanical components in the vehicle.

3.2.1.3 Data Types and Data Objects

The components are designed top down coming from the physical function down to the implementation in the target programming language C. Hereby the interfaces of Software Components are typically typed with ApplicationDataTypes in order to describe the physical meaning of the DataPrototypes. The only exceptions are the interfaces to AUTOSAR Services which are typed by ImplementationDataTypes directly as those are standardized. ApplicationPrimitiveDataTypes are mainly of category

- BOOLEAN
- VALUE
- CURVE
- MAP
- COM_AXIS

and the most important CompuMethod categorys are

- LINEAR
- TEXTTABLE

In case of LINEAR conversions it is supported to differentiate the Unit used for the implemented calculations and an additional Unit used in the MCD system. This relationship of such Units are expressed with UnitGroups. The ARElements are structured in a way to support the common usage of elements relevant for the interface description up to the level of PortInterfaces by several Component Descriptions. Those elements are located under Tier1/ARPlatform1/DataDictionary/<KindPackage> in the file Pprj_DataDictionary.arxml.

The CompuMethods and DataConstrs are exclusively used by one ApplicationPrimitiveDataType. The possible reuse between ApplicationPrimitiveDataTypes supported by AUTOSAR is not used in this model structure. When such a ApplicationDataType is defined the intended mapping to the reasonable ImplementationDataType is already considered in order to get an optimal usage of the possible range of the ImplementationDataType. Nevertheless, the several physical meanings are **not** reflected by the definition of individual ImplementationDataType but only the standardized Platform Types [5] are used to describe primitives on implementation level. This has the effect that the RTE APIs are typed by the standardized Platform Types in cases of primitives and arrays of primitives. Only structure types are getting observable in the types of RTE APIs. This approach allows the direct usage of data read from RTE in mathematical or interpolation libraries without any type cast.

The memory allocation of the data objects is controlled by the usage of SwAddrMethods. Those are defined for ParameterDataPrototypes and VariableDataPrototypes on level of the PortInterfaces. A few examples are shown in the chapter 3.2.2.17 for the basic uses cases like calibration parameter, normal data and code.

3.2.1.4 Axis, Curves and Maps

The show case contains description for axis, curves and maps which are in AUTOSAR so called compound primitives. In order to understand the structure and the defined attributes in the example it is helpful to understand how such objects are described in AUTOSAR. For this it is necessary to look at the hierarchy of ApplicationDataTypes, DataPrototypes, PortPrototypes, SwComponentTypes and FlatMap.

3.2.1.5 Axis, Curves and Maps on ApplicationDataType level

Figure 3.13 is based on the example of the ApplicationPrimitiveDataType Map_Time_Lnr_s_uint16. It shows the relationships between the ApplicationPrimitiveDataTypes describing the

- MAP itself
- its axis being a group axis
- in turn the properties of a matching working point

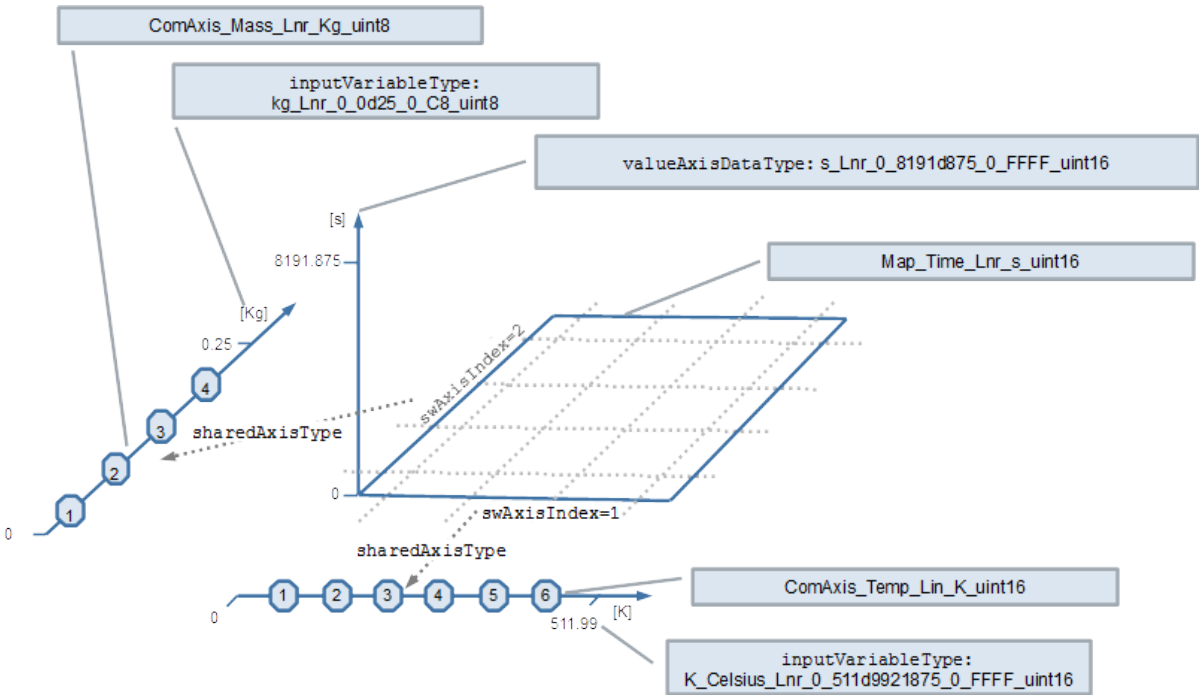


Figure 3.13: ApplicationPrimitiveDataType of category MAP and its group axes

The ApplicationPrimitiveDataType Map_Time_Lnr_s_uint16 defines a data type for a MAP with group axes. The physical meaning and range of the contained values is described with the ApplicationPrimitiveDataType s_Lnr_0_8191d875_0_FFFF_uint16. It is referenced with the valueAxisDataType attribute. This means it's a value in the range 0 .. 8191.875 [second] with the resolution of 0.125 [second].

The referenced ApplicationPrimitiveDataType in the role valueAxisDataType represents the primitive data type of the value axis within a compound primitive (e.g. CURVE, MAP). It supersedes CompuMethod, Unit, and BaseType. In the particular example, the valueAxisDataType provides the properties of the primitive elements of the CURVE or MAP via a valueAxisDataType reference to an ApplicationPrimitiveDataType. This in turn defines the attributes:

- dataConstr
- compuMethod
- displayFormat
- unit
- swCalibrationAccess

Thereby, despite being set, the value of swCalibrationAccess of the referenced ApplicationPrimitiveDataType is meaningless for the using CURVE and MAP. Note: The referenced data type needs to be a real primitive (typically of category VALUE. Category BOOLEAN is also supported).

The ApplicationPrimitiveDataType of the CURVE and MAP can additionally define SwDataDefProps which are relevant for the whole compound primitive. Currently the following attributes are used in the example:

- swCalprmAxisSet
- swRecordLayout
- swCalibrationAccess (but will be refined on DataPrototype level)

Further on, via the dataTypeMapping of the using software component, the properties of ImplementationDataType and SwBaseType are described.

As axes of the MAP two group axes are used. The properties of the group axes are described by two ApplicationPrimitiveDataTypes of category COM_AXIS. The attribute swAxisIndex indicates for which dimension the group axis applies (1 = X, 2 = Y). With the attribute sharedAxisType the reference to the ApplicationPrimitiveDataType describing the axis is defined.

In the example, the group axis ComAxis_Temp_Lin_K_uint16 defines the the applicable minimum and maximum number of axis points. Additionally the inputVariableType reference to the ApplicationPrimitiveDataType K_Celsius_Lnr_0_511d9921875_0_FFFF_uint16 defines the properties of the input value for

the axis. This in turn corresponds to the values stored as axis point. The same principle applies for the group axis ComAxis_Mass_Lnr_Kg_uint8.

Please note, the above mentioned properties are defined on the level of ApplicationDataTypes and so far not any data instance implementing such properties exists. This requires an instantiation of such ApplicationDataTypes.

3.2.1.6 Axis, Curves and Maps on DataPrototype and SwComponentPrototype level

3.2.1.6.1 Instantiation of Axis, Curves and Maps

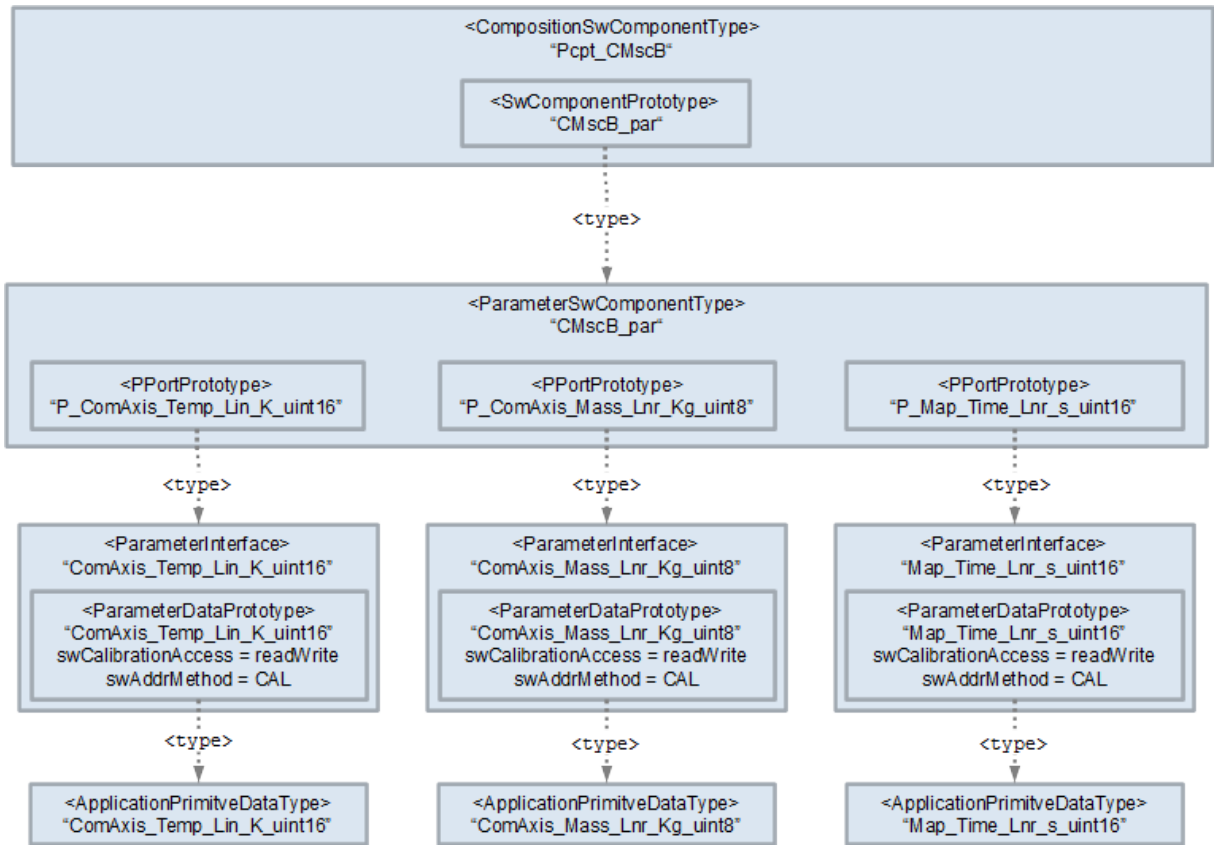


Figure 3.14: Instantiation of a MAP and its group axes

Figure 3.14 shows the instantiation of the ApplicationPrimitiveDataType ComAxis_Temp_Lin_K_uint16, ComAxis_Mass_Lnr_Kg_uint8, and Map_Time_Lnr_s_uint16 up to the level of the CompositionSwComponentType Pcpt_CMscB.

Thereby ParameterDataPrototypes are typed by the mentioned ApplicationPrimitiveDataTypes. Each ParameterDataPrototype is owned by an own ParameterInterface. This offers the most flexibility to instantiate the map and axes independently from each other. On the level of the ParameterDataPrototype additionally the swCalibrationAccess and the swAddrMethod is defined. Further on,

the ParameterSwComponentType CMscB_par defines three PPortPrototypes typed by the ParameterInterfaces.

Please note that a group axes of a curve or map are not necessarily provided by the same ParameterSwComponentType as the one providing the curve or map. This case is illustrated with the map Arr1dMap_Time_Lnr_s_uint16 using the group axes Arr1dComAxis_Temp_Lin_K_uint16 provided by CMscD_par and ComAxis_Mass_Lnr_Kg_uint8 provided by CMscB_par.

3.2.1.6.2 Linking map and curve instances to its axes instances

Consider a software component that uses curves and maps with group axes. It is than required to denote which instance of curve and map uses which instance of a group axis as axis of abscissae and, in case of a map, as axis of ordinate.

The AUTOSAR meta model provides hereby two possibilities:

- RunnableEntity.parameterAccess.swDataDefProps

  or

- SwcInternalBehavior.instantiationDataDefProps.swDataDefProps.

Inside one software component it's very unlikely, that the same curve or map is used with different axes by different RunnableEntitys (note that this cannot be expressed by ASAM MCD-2MC, also) . Therefore, in this show case the second ability is used. This avoids the risk of inconsistencies when several RunnableEntitys are defining parameterAccesses to the same curve or map instance.

The according instantiationDataDefProps.parameterInstance references the map instance in the scope of the SwComponentType and the swDataDefProps.swCalprmAxisSet.swCalprmAxis.swCalprmAxisTypeProps.swCalprmRef references the applied group axes with the according SwCalprmAxis.swAxisIndex

```
<INSTANTIATION-DATA-DEF-PROPS>
  <PARAMETER-INSTANCE>
    <AUTOSAR-PARAMETER-IREF>
      <PORT-PROTOTYPE-REF DEST="R-PORT-PROTOTYPE">/Tier1/ARPlatform1/
          Pcpt_CMscB/CMscB/R_Map_Time_Lnr_s_uint16</PORT-PROTOTYPE-REF>
      <TARGET-DATA-PROTOTYPE-REF DEST="PARAMETER-DATA-PROTOTYPE">/Tier1/
          ARPlatform1/DataDictionary/PortInterfaces/V1_0_0/
          Map_Time_Lnr_s_uint16/Map_Time_Lnr_s_uint16</TARGET-DATA-
          PROTOTYPE-REF>
    </AUTOSAR-PARAMETER-IREF>
  </PARAMETER-INSTANCE>
  <SW-DATA-DEF-PROPS>
    <SW-DATA-DEF-PROPS-VARIANTS>
      <SW-DATA-DEF-PROPS-CONDITIONAL>
        <SW-CALPRM-AXIS-SET>
          <SW-CALPRM-AXIS>
```

```
            <SW-AXIS-INDEX>1</SW-AXIS-INDEX>
            <SW-AXIS-GROUPED>
              <AR-PARAMETER>
                <AUTOSAR-PARAMETER-IREF>
                  <PORT-PROTOTYPE-REF DEST="R-PORT-PROTOTYPE">/Tier1/
                      ARPlatform1/Pcpt_CMscB/CMscB/
                      R_ComAxis_Temp_Lin_K_uint16</PORT-PROTOTYPE-REF>
                  <TARGET-DATA-PROTOTYPE-REF DEST="PARAMETER-DATA-
                      PROTOTYPE">/Tier1/ARPlatform1/DataDictionary/
                      PortInterfaces/V1_0_0/ComAxis_Temp_Lin_K_uint16/
                      ComAxis_Temp_Lin_K_uint16</TARGET-DATA-PROTOTYPE-
                      REF>
                </AUTOSAR-PARAMETER-IREF>
              </AR-PARAMETER>
            </SW-AXIS-GROUPED>
          </SW-CALPRM-AXIS>
          <SW-CALPRM-AXIS>
            <SW-AXIS-INDEX>2</SW-AXIS-INDEX>
            <SW-AXIS-GROUPED>
              <AR-PARAMETER>
                <AUTOSAR-PARAMETER-IREF>
                  <PORT-PROTOTYPE-REF DEST="R-PORT-PROTOTYPE">/Tier1/
                      ARPlatform1/Pcpt_CMscB/CMscB/
                      R_ComAxis_Mass_Lnr_Kg_uint8</PORT-PROTOTYPE-REF>
                  <TARGET-DATA-PROTOTYPE-REF DEST="PARAMETER-DATA-
                      PROTOTYPE">/Tier1/ARPlatform1/DataDictionary/
                      PortInterfaces/V1_0_0/ComAxis_Mass_Lnr_Kg_uint8/
                      ComAxis_Mass_Lnr_Kg_uint8</TARGET-DATA-PROTOTYPE-
                      REF>
                </AUTOSAR-PARAMETER-IREF>
              </AR-PARAMETER>
            </SW-AXIS-GROUPED>
          </SW-CALPRM-AXIS>
        </SW-CALPRM-AXIS-SET>
      </SW-DATA-DEF-PROPS-CONDITIONAL>
    </SW-DATA-DEF-PROPS-VARIANTS>
  </SW-DATA-DEF-PROPS>
</INSTANTIATION-DATA-DEF-PROPS>
```

Listing 3.28: Example of an `InstantiationDataDefProps` for a map

3.2.1.6.3 Linking axes instances to its working point instances

When a software component uses compound primitives containing axes (e.g. curves, maps, or group axes) it's beneficial to indicate which data is used as input for the according axis. This enables the measurement and calibration tool to display the current working point. Like explained in section 3.2.1.6.2, this information can be provided

- at the `ParameterAccess.swDataDefProps` of the compound primitives containing the axis **or**
- by means of `instantiationDataDefProps.swDataDefProps`.

In this show case the second ability is used for the same reasons as discussed in section 3.2.1.6.2.

The according instantiationDataDefProps.parameterInstance references the axes instance in the scope of the SwComponentType CMscB. The swDataDefProps.swCalprmAxisSet.swCalprmAxis.swCalprmAxisTypeProps.swVariableRef references the applied working point variable (in this case, a dataElement in a RPortPrototype) with the according SwCalprmAxis.swAxisIndex.

```
<INSTANTIATION-DATA-DEF-PROPS>
  <PARAMETER-INSTANCE>
    <AUTOSAR-PARAMETER-IREF>
      <PORT-PROTOTYPE-REF DEST="R-PORT-PROTOTYPE">/Tier1/ARPlatform1/
          Pcpt_CMscB/CMscB/R_ComAxis_Temp_Lin_K_uint16</PORT-PROTOTYPE-
          REF>
      <TARGET-DATA-PROTOTYPE-REF DEST="PARAMETER-DATA-PROTOTYPE">/Tier1/
          ARPlatform1/DataDictionary/PortInterfaces/V1_0_0/
          ComAxis_Temp_Lin_K_uint16/ComAxis_Temp_Lin_K_uint16</TARGET-
          DATA-PROTOTYPE-REF>
    </AUTOSAR-PARAMETER-IREF>
  </PARAMETER-INSTANCE>
  <SW-DATA-DEF-PROPS>
    <SW-DATA-DEF-PROPS-VARIANTS>
      <SW-DATA-DEF-PROPS-CONDITIONAL>
        <SW-CALPRM-AXIS-SET>
          <SW-CALPRM-AXIS>
            <SW-AXIS-INDEX>1</SW-AXIS-INDEX>
            <SW-AXIS-INDIVIDUAL>
              <SW-VARIABLE-REFS>
                <AUTOSAR-VARIABLE>
                  <AUTOSAR-VARIABLE-IREF>
                    <PORT-PROTOTYPE-REF DEST="R-PORT-PROTOTYPE">/Tier1/
                        ARPlatform1/Pcpt_CMscB/CMscB/
                        R_PrimData_Temperature_Lin_K_C_uint16</PORT-
                        PROTOTYPE-REF>
                    <TARGET-DATA-PROTOTYPE-REF DEST="VARIABLE-DATA-
                        PROTOTYPE">/Tier1/ARPlatform1/DataDictionary/
                        PortInterfaces/V1_0_0/
                        PrimData_Temperature_Lin_K_C_uint16/
                        PrimData_Temperature_Lin_K_C_uint16</TARGET-DATA-
                        PROTOTYPE-REF>
                  </AUTOSAR-VARIABLE-IREF>
                </AUTOSAR-VARIABLE>
              </SW-VARIABLE-REFS>
            </SW-AXIS-INDIVIDUAL>
          </SW-CALPRM-AXIS>
        </SW-CALPRM-AXIS-SET>
      </SW-DATA-DEF-PROPS-CONDITIONAL>
    </SW-DATA-DEF-PROPS-VARIANTS>
  </SW-DATA-DEF-PROPS>
</INSTANTIATION-DATA-DEF-PROPS>
```

Listing 3.29: Example of an InstantiationDataDefProps for an axis

3.2.1.6.4 Axis, Curves and Maps in the ECU Flat Map

The ECU Flat Map contains entries for all curves, maps, axes and working point variables. The used naming patterns are described in 3.2.1.2.1.

```
<FLAT-INSTANCE-DESCRIPTOR>
  <SHORT-NAME>Map_Time_Lnr_s_uint16</SHORT-NAME>
  <ECU-EXTRACT-REFERENCE-IREF>
    <CONTEXT-ELEMENT-REF DEST="ROOT-SW-COMPOSITION-PROTOTYPE">/Tier1/
       ARPlatform1/System/SystemU_System/SystemU</CONTEXT-ELEMENT-REF>
    <CONTEXT-ELEMENT-REF DEST="SW-COMPONENT-PROTOTYPE">/Tier1/ARPlatform1/
       System/CompositionSwComponentTypes/SystemU_Root/CMscB_par</CONTEXT-
       ELEMENT-REF>
    <CONTEXT-ELEMENT-REF DEST="P-PORT-PROTOTYPE">/Tier1/ARPlatform1/
       Pcpt_CMscB/CMscB_par/P_Map_Time_Lnr_s_uint16</CONTEXT-ELEMENT-REF>
    <TARGET-REF DEST="PARAMETER-DATA-PROTOTYPE">/Tier1/ARPlatform1/
       DataDictionary/PortInterfaces/V1_0_0/Map_Time_Lnr_s_uint16/
       Map_Time_Lnr_s_uint16</TARGET-REF>
  </ECU-EXTRACT-REFERENCE-IREF>
</FLAT-INSTANCE-DESCRIPTOR>
<FLAT-INSTANCE-DESCRIPTOR>
  <SHORT-NAME>ComAxis_Temp_Lin_K_uint16</SHORT-NAME>
  <ECU-EXTRACT-REFERENCE-IREF>
    <CONTEXT-ELEMENT-REF DEST="ROOT-SW-COMPOSITION-PROTOTYPE">/Tier1/
       ARPlatform1/System/SystemU_System/SystemU</CONTEXT-ELEMENT-REF>
    <CONTEXT-ELEMENT-REF DEST="SW-COMPONENT-PROTOTYPE">/Tier1/ARPlatform1/
       System/CompositionSwComponentTypes/SystemU_Root/CMscB_par</CONTEXT-
       ELEMENT-REF>
    <CONTEXT-ELEMENT-REF DEST="P-PORT-PROTOTYPE">/Tier1/ARPlatform1/
       Pcpt_CMscB/CMscB_par/P_ComAxis_Temp_Lin_K_uint16</CONTEXT-ELEMENT-
       REF>
    <TARGET-REF DEST="PARAMETER-DATA-PROTOTYPE">/Tier1/ARPlatform1/
       DataDictionary/PortInterfaces/V1_0_0/ComAxis_Temp_Lin_K_uint16/
       ComAxis_Temp_Lin_K_uint16</TARGET-REF>
  </ECU-EXTRACT-REFERENCE-IREF>
</FLAT-INSTANCE-DESCRIPTOR>
<FLAT-INSTANCE-DESCRIPTOR>
  <SHORT-NAME>PrimData_Temperature_Lin_K_C_uint16</SHORT-NAME>
  <ECU-EXTRACT-REFERENCE-IREF>
    <CONTEXT-ELEMENT-REF DEST="ROOT-SW-COMPOSITION-PROTOTYPE">/Tier1/
       ARPlatform1/System/SystemU_System/SystemU</CONTEXT-ELEMENT-REF>
    <CONTEXT-ELEMENT-REF DEST="SW-COMPONENT-PROTOTYPE">/Tier1/ARPlatform1/
       System/CompositionSwComponentTypes/SystemU_Root/CMscA</CONTEXT-
       ELEMENT-REF>
    <CONTEXT-ELEMENT-REF DEST="P-PORT-PROTOTYPE">/Tier1/ARPlatform1/
       Pcpt_CMscA/CMscA/P_PrimData_Temperature_Lin_K_C_uint16</CONTEXT-
       ELEMENT-REF>
    <TARGET-REF DEST="VARIABLE-DATA-PROTOTYPE">/Tier1/ARPlatform1/
       DataDictionary/PortInterfaces/V1_0_0/
       PrimData_Temperature_Lin_K_C_uint16/
       PrimData_Temperature_Lin_K_C_uint16</TARGET-REF>
  </ECU-EXTRACT-REFERENCE-IREF>
</FLAT-INSTANCE-DESCRIPTOR>
```

Listing 3.30: Example of a `FlatInstanceDescriptor` for map axis and working point variable

3.2.1.7 Arrays of Maps and Axes

The ability of curves, maps and cuboids is usually used to describe the physical dependency of a characteristic on other physical input values. Hereby each input value is described by an orthogonal axis. In contrast to this, arrays are used to group a set of values of the same nature which can be handled by the same algorithm. Typically, in this case the algorithm iterates over the array with an index. Nevertheless, each array element may represent a particular part of the vehicle, e.g. a specific cylinder or a specific sensor. It's possible to combine these design principles. This ends up in the need to describe arrays of curves, maps, cuboids and the according axes.

The show case illustrates the model of those objects by the following elements:

- Arr1dMap_Time_Lnr_s_uint16
- Arr1dComAxis_Temp_Lin_K_uint16
- Arr1dPrimData_Temperature_Lin_K_C_uint16

Hereby, the array of the map Arr1dMap_Time_Lnr_s_uint16 uses for the x-axis an array of group axes Arr1dComAxis_Temp_Lin_K_uint16 which in turn uses an array of primitive values as working points Arr1dPrimData_Temperature_Lin_K_C_uint16. In this case, the n'th map uses the n'th x-axes which uses the n'th value as working point. In contrast, the map uses one group axis ComAxis_Mass_Lnr_Kg_uint8 for the y-axis. In this case all maps in the array are using the same y-axis.

3.2.1.7.1 Arrays of Maps and Axes in the ECU Flat Map

In the ECU Flat Map the ability to reference ApplicationCompositeElement-DataPrototypes is used to express the specific meaning of each array element in the array of map and group axis.

For instance, each element in the array Arr1dMap_Time_Lnr_s_uint16 is named in a way to indicate the specific meaning:

- Arr1dMap_Time_Lnr_s_uint16_FrontLeft
- Arr1dMap_Time_Lnr_s_uint16_FrontRight
- Arr1dMap_Time_Lnr_s_uint16_RearLeft
- Arr1dMap_Time_Lnr_s_uint16_RearRight

The following listing shows the structure of such an FlatInstanceDescriptior on one example:

```
<FLAT-INSTANCE-DESCRIPTOR>
  <SHORT-NAME>Arr1dMap_Time_Lnr_s_uint16_FrontLeft</SHORT-NAME>
  <ECU-EXTRACT-REFERENCE-IREF>
    <CONTEXT-ELEMENT-REF DEST="ROOT-SW-COMPOSITION-PROTOTYPE">/Tier1/
       ARPlatform1/System/SystemU_System/SystemU</CONTEXT-ELEMENT-REF>
```

```
    <CONTEXT-ELEMENT-REF DEST="SW-COMPONENT-PROTOTYPE">/Tier1/ARPlatform1/
        System/CompositionSwComponentTypes/SystemU_Root/CMscD_par</CONTEXT-
        ELEMENT-REF>
    <CONTEXT-ELEMENT-REF DEST="P-PORT-PROTOTYPE">/Tier1/ARPlatform1/
        Pcpt_CMscD/CMscD_par/P_Arr1dMap_Time_Lnr_s_uint16</CONTEXT-ELEMENT-
        REF>
    <CONTEXT-ELEMENT-REF DEST="PARAMETER-DATA-PROTOTYPE">/Tier1/ARPlatform1
        /DataDictionary/PortInterfaces/V1_0_0/Arr1dMap_Time_Lnr_s_uint16/
        Arr1dMap_Time_Lnr_s_uint16</CONTEXT-ELEMENT-REF>
    <TARGET-REF DEST="APPLICATION-ARRAY-ELEMENT" INDEX="0">/Tier1/
        ARPlatform1/DataDictionary/ApplicationDataTypes/
        Map_Time_Lnr_s_uint16_ScNoOfWheels/
        Map_Time_Lnr_s_uint16_ScNoOfWheels</TARGET-REF>
  </ECU-EXTRACT-REFERENCE-IREF>
</FLAT-INSTANCE-DESCRIPTOR>
```

Listing 3.31: Example of a `FlatInstanceDescriptor` for an `ApplicationCompositeElementDataPrototype`

Please note the usage of the `index` attribute in the `target` reference.

3.2.1.8 Measurement of Modes

3.2.1.8.1 Enabling Measurement of Modes

The measurement of a mode is enabled in the software-component description by setting the `ModeDeclarationGroupPrototype.swCalibrationAccess` to `readOnly`. See `ModeDirection`.

3.2.1.8.2 Modes in the ECU Flat Map

`AUTOSAR` supports the measurement of the current mode, the previous mode and the next mode. Hereby the last two are useful when the mode is measured during a ongoing transition to identify the kind of transition. In this show case only the measurement of the current mode is illustrated. For this, the `FlatMap` contains a `FlatInstanceDescriptor` pointing to the `ModeDeclarationGroupPrototype` which is to be measured. The `role` attribute of the `FlatInstanceDescriptor` is set to `CURRENT_MODE`

```
<FLAT-INSTANCE-DESCRIPTOR>
  <SHORT-NAME>ModeDirection</SHORT-NAME>
  <ROLE>CURRENT_MODE</ROLE>
  <ECU-EXTRACT-REFERENCE-IREF>
    <CONTEXT-ELEMENT-REF DEST="ROOT-SW-COMPOSITION-PROTOTYPE">/Tier1/
        ARPlatform1/System/SystemU_System/SystemU</CONTEXT-ELEMENT-REF>
    <CONTEXT-ELEMENT-REF DEST="SW-COMPONENT-PROTOTYPE">/Tier1/ARPlatform1/
        System/CompositionSwComponentTypes/SystemU_Root/CMscA</CONTEXT-
        ELEMENT-REF>
    <CONTEXT-ELEMENT-REF DEST="P-PORT-PROTOTYPE">/Tier1/ARPlatform1/
        Pcpt_CMscA/CMscA/P_ModeDirection</CONTEXT-ELEMENT-REF>
```

```
    <TARGET-REF DEST="MODE-DECLARATION-GROUP-PROTOTYPE">/Tier1/ARPlatform1/
       DataDictionary/PortInterfaces/V1_0_0/ModeDirection/ModeDirection</
       TARGET-REF>
  </ECU-EXTRACT-REFERENCE-IREF>
</FLAT-INSTANCE-DESCRIPTOR>
```

Listing 3.32: Example of a `FlatInstanceDescriptor` for a `ModeDeclarationGroup-Prototype`

3.2.2 Show cases in the Example

3.2.2.1 CompositionSwComponentTypes

Common CompositionSwComponentType attributes	
shortName	Pcpt_CMscA
desc	Modeling show case for primitive measurement and calculation.
properties of the ports	
properties of PPortPrototype	
shortName	P_ModeDirection
desc	Mode to indicate a direction
providedInterface	ModeDirection
properties of PPortPrototype	
shortName	P_PrimCal_Mass_Lnr_Kg
desc	Primitive calibration parameter for minimum egg mass.
providedInterface	PrimCal_Mass_Lnr_Kg
properties of PPortPrototype	
shortName	P_PrimData_Mass_Lnr_Kg_uint8
desc	Mass in kilogram
providedInterface	PrimData_Mass_Lnr_Kg_uint8
properties of PPortPrototype	
shortName	P_PrimData_StepsSpeed_Txt_sint8
desc	Stepwise speed indication
providedInterface	PrimData_StepsSpeed_Txt_sint8
properties of PPortPrototype	
shortName	P_PrimData_Temperature_Lin_K_C_uint16
desc	Temperature 1 in Kelvin but displayed as degree Celsius
providedInterface	PrimData_Temperature_Lin_K_C_uint16
properties of RPortPrototype	
shortName	R_PrimData_StepsSpeed_Txt_sint8
desc	Stepwise speed indication
requiredInterface	PrimData_StepsSpeed_Txt_sint8
properties of the components	
properties of SwComponentPrototype	
shortName	CMscA
type	CMscA
properties of SwComponentPrototype	
shortName	CMscA_par
type	CMscA_par

▽

△

Table 3.53: CompositionSwComponentType Pcpt_CMscA

Common CompositionSwComponentType attributes	
shortName	SystemU_Root
properties of the components	
properties of SwComponentPrototype	
shortName	CMscA
type	CMscA
properties of SwComponentPrototype	
shortName	CMscA_par
type	CMscA_par
properties of SwComponentPrototype	
shortName	CMscB
type	CMscB
properties of SwComponentPrototype	
shortName	CMscB_par
type	CMscB_par
properties of SwComponentPrototype	
shortName	CMscC_nvm
type	CMscC_nvm
properties of SwComponentPrototype	
shortName	CMscD
type	CMscD
properties of SwComponentPrototype	
shortName	CMscD_par
type	CMscD_par

Table 3.54: CompositionSwComponentType SystemU_Root

Common CompositionSwComponentType attributes	
shortName	SystemURootComposition_EcuDescr
properties of the components	
properties of SwComponentPrototype	
shortName	CMscC
type	Pcpt_CMscC

Table 3.55: CompositionSwComponentType SystemURootComposition_EcuDescr

Common CompositionSwComponentType attributes	
shortName	Pcpt_CMscD
desc	Modeling show case for arrays of axes and mapes.
properties of the ports	
properties of RPortPrototype	
shortName	R_ComAxis_Mass_Lnr_Kg_uint8
desc	Shared axis for mass
requiredInterface	ComAxis_Mass_Lnr_Kg_uint8
properties of RPortPrototype	
shortName	R_PrimData_MassCorrected_Lnr_Kg_uint8
desc	Primitve data for the corrected mass in kg.
requiredInterface	PrimData_MassCorrected_Lnr_Kg_uint8
properties of the components	
properties of SwComponentPrototype	
shortName	CMscD
type	CMscD
properties of SwComponentPrototype	
shortName	CMscD_par
type	CMscD_par

Table 3.56: CompositionSwComponentType Pcpt_CMscD

Common CompositionSwComponentType attributes	
shortName	Pcpt_CMscC
desc	Composit of modeling show case C
properties of the ports	
properties of PPortPrototype	
shortName	P_PrimData_Time_Lnr_s_uint16
desc	Primitve data holding a time value.
providedInterface	PrimData_Time_Lnr_s_uint16
properties of PPortPrototype	
shortName	P_PrimData_ValidState_Txt_noUnit_boolean
desc	Boolean representing the data validity
providedInterface	PrimData_ValidState_Txt_noUnit_boolean
properties of the components	
properties of SwComponentPrototype	
shortName	CMscA
type	Pcpt_CMscA
properties of SwComponentPrototype	
shortName	CMscB
type	Pcpt_CMscB
properties of SwComponentPrototype	
shortName	CMscC_nvm
type	CMscC_nvm
properties of SwComponentPrototype	
shortName	CMscD
type	Pcpt_CMscD

Table 3.57: CompositionSwComponentType Pcpt_CMscC

Common CompositionSwComponentType attributes	
shortName	Pcpt_CMscB
desc	Modeling show case for axes, curves and mapes.
properties of the ports	
properties of PPortPrototype	
shortName	P_ComAxis_Mass_Lnr_Kg_uint8
desc	Shared axis for mass
providedInterface	ComAxis_Mass_Lnr_Kg_uint8
properties of PPortPrototype	
shortName	P_PrimData_MassCorrected_Lnr_Kg_uint8
desc	Primitve data for the corrected mass in kg.
providedInterface	PrimData_MassCorrected_Lnr_Kg_uint8
properties of PPortPrototype	
shortName	P_PrimData_Time_Lnr_s_uint16
desc	Primitve data holding a time value.
providedInterface	PrimData_Time_Lnr_s_uint16
properties of PPortPrototype	
shortName	P_PrimData_ValidState_Txt_noUnit_boolean
desc	Boolean representing the data validity
providedInterface	PrimData_ValidState_Txt_noUnit_boolean
properties of RPortPrototype	
shortName	R_ModeDirection
desc	Mode to indicate a direction
requiredInterface	ModeDirection
properties of RPortPrototype	
shortName	R_PrimData_Mass_Lnr_Kg_uint8
desc	Mass in kilogram
requiredInterface	PrimData_Mass_Lnr_Kg_uint8
properties of RPortPrototype	
shortName	R_PrimData_StepsSpeed_Txt_sint8
desc	Stepwise speed indication
requiredInterface	PrimData_StepsSpeed_Txt_sint8
properties of RPortPrototype	
shortName	R_PrimData_Temperature_Lin_K_C_uint16
desc	Temperature 1 in Kelvin but displayed as degree Celsius
requiredInterface	PrimData_Temperature_Lin_K_C_uint16
properties of the components	
properties of SwComponentPrototype	
shortName	CMscB
type	CMscB
properties of SwComponentPrototype	

▽

△

shortName	CMscB_par
type	CMscB_par

Table 3.58: CompositionSwComponentType Pcpt_CMscB

3.2.2.2 ParameterSwComponentTypes

Common ParameterSwComponentType attributes	
shortName	CMscA_par
desc	Modeling show case for primitive measurement and calculation.
properties of the ports	
properties of PPortPrototype	
shortName	P_PrimCal_Mass_Lnr_Kg
desc	Primitive calibration parameter for minimum egg mass.
providedInterface	PrimCal_Mass_Lnr_Kg

Table 3.59: ParameterSwComponentType CMscA_par

Common ParameterSwComponentType attributes	
shortName	CMscD_par
desc	Modeling show case for arrays of axes and mapes.
properties of the ports	
properties of PPortPrototype	
shortName	P_Arr1dComAxis_Temp_Lin_K_uint16
desc	Array of shared axis for temperature
providedInterface	Arr1dComAxis_Temp_Lin_K_uint16
properties of PPortPrototype	
shortName	P_Arr1dMap_Time_Lnr_s_uint16
desc	Map to get time dependent on temperature and mass.
providedInterface	Arr1dMap_Time_Lnr_s_uint16

Table 3.60: ParameterSwComponentType CMscD_par

Common ParameterSwComponentType attributes	
shortName	CMscB_par
desc	Modeling show case for axes, curves and mapes.
properties of the ports	
properties of PPortPrototype	
shortName	P_ComAxis_Mass_Lnr_Kg_uint8
desc	Shared axis for mass
providedInterface	ComAxis_Mass_Lnr_Kg_uint8
properties of PPortPrototype	
shortName	P_ComAxis_Steps_Txt_sint8
desc	Shared axis for speed steps
providedInterface	ComAxis_Steps_Txt_sint8
properties of PPortPrototype	
shortName	P_ComAxis_Temp_Lin_K_uint16
desc	Shared axis for temperature
providedInterface	ComAxis_Temp_Lin_K_uint16
properties of PPortPrototype	
shortName	P_Curve_Mass_Lnr_Kg_uint8
desc	Curve to get mass according differnt speed steps.
providedInterface	Curve_Mass_Lnr_Kg_uint8
properties of PPortPrototype	
shortName	P_Map_Time_Lnr_s_uint16
desc	Map to get time dependent on temperature and mass.
providedInterface	Map_Time_Lnr_s_uint16

Table 3.61: ParameterSwComponentType CMscB_par

3.2.2.3 ApplicationSwComponentTypes

Common ApplicationSwComponentType attributes	
shortName	CMscD
desc	Modeling show case for arrays of axes and mapes.
properties of the ports	
properties of PPortPrototype	
shortName	P_Arr1dPrimData_Temperature_Lin_K_C_uint16
desc	Temperature 1 in Kelvin but displayed as degree Celsius
providedInterface	Arr1dPrimData_Temperature_Lin_K_C_uint16
properties of RPortPrototype	
shortName	R_Arr1dComAxis_Temp_Lin_K_uint16
desc	Array of shared axis for temperature
requiredInterface	Arr1dComAxis_Temp_Lin_K_uint16
properties of RPortPrototype	
shortName	R_Arr1dMap_Time_Lnr_s_uint16
desc	Map to get time dependent on temperature and mass.
requiredInterface	Arr1dMap_Time_Lnr_s_uint16
properties of RPortPrototype	
shortName	R_Arr1dPrimData_Temperature_Lin_K_C_uint16
desc	Temperature 1 in Kelvin but displayed as degree Celsius
requiredInterface	Arr1dPrimData_Temperature_Lin_K_C_uint16
properties of RPortPrototype	
shortName	R_ComAxis_Mass_Lnr_Kg_uint8
desc	Shared axis for mass
requiredInterface	ComAxis_Mass_Lnr_Kg_uint8
properties of RPortPrototype	
shortName	R_PrimData_MassCorrected_Lnr_Kg_uint8
desc	Primitve data for the corrected mass in kg.
requiredInterface	PrimData_MassCorrected_Lnr_Kg_uint8
internalBehavior	CMscD

Table 3.62: ApplicationSwComponentType CMscD

Common SwcInternalBehavior attributes	
shortName	CMscD
properties of the runnables	
properties of RunnableEntity	
shortName	CMscD_Process
symbol	CMscD_Process

Table 3.63: SwcInternalBehavior CMscD

Common ApplicationSwComponentType attributes	
shortName	CMscB
desc	Modeling show case for axes, curves and mapes.
properties of the ports	
properties of PPortPrototype	
shortName	P_PrimData_MassCorrected_Lnr_Kg_uint8
desc	Primitve data for the corrected mass in kg.
providedInterface	PrimData_MassCorrected_Lnr_Kg_uint8
properties of PPortPrototype	
shortName	P_PrimData_Time_Lnr_s_uint16
desc	Primitve data holding a time value.
providedInterface	PrimData_Time_Lnr_s_uint16
properties of PPortPrototype	
shortName	P_PrimData_ValidState_Txt_noUnit_boolean
desc	Boolean representing the data validity
providedInterface	PrimData_ValidState_Txt_noUnit_boolean
properties of RPortPrototype	
shortName	R_ComAxis_Mass_Lnr_Kg_uint8
desc	Shared axis for mass
requiredInterface	ComAxis_Mass_Lnr_Kg_uint8
properties of RPortPrototype	
shortName	R_ComAxis_Steps_Txt_sint8
desc	Shared axis for speed steps
requiredInterface	ComAxis_Steps_Txt_sint8
properties of RPortPrototype	
shortName	R_ComAxis_Temp_Lin_K_uint16
desc	Shared axis for temperature
requiredInterface	ComAxis_Temp_Lin_K_uint16
properties of RPortPrototype	
shortName	R_Curve_Mass_Lnr_Kg_uint8
desc	Curve to get mass according differnt speed steps.
requiredInterface	Curve_Mass_Lnr_Kg_uint8
properties of RPortPrototype	
shortName	R_Map_Time_Lnr_s_uint16
desc	Map to get time dependent on temperature and mass.
requiredInterface	Map_Time_Lnr_s_uint16
properties of RPortPrototype	
shortName	R_ModeDirection
desc	Mode to indicate a direction
requiredInterface	ModeDirection
properties of RPortPrototype	



△

	shortName	R_PrimData_Mass_Lnr_Kg_uint8
	desc	Mass in kilogram
	requiredInterface	PrimData_Mass_Lnr_Kg_uint8
	properties of RPortPrototype	
	shortName	R_PrimData_MassCorrected_Lnr_Kg_uint8
	desc	Primitve data for the corrected mass in kg.
	requiredInterface	PrimData_MassCorrected_Lnr_Kg_uint8
	properties of RPortPrototype	
	shortName	R_PrimData_StepsSpeed_Txt_sint8
	desc	Stepwise speed indication
	requiredInterface	PrimData_StepsSpeed_Txt_sint8
	properties of RPortPrototype	
	shortName	R_PrimData_Temperature_Lin_K_C_uint16
	desc	Temperature 1 in Kelvin but displayed as degree Celsius
	requiredInterface	PrimData_Temperature_Lin_K_C_uint16
	properties of RPortPrototype	
	shortName	R_PrimData_Time_Lnr_s_uint16
	desc	Primitve data holding a time value.
	requiredInterface	PrimData_Time_Lnr_s_uint16
	properties of RPortPrototype	
	shortName	R_PrimData_ValidState_Txt_noUnit_boolean
	desc	Boolean representing the data validity
	requiredInterface	PrimData_ValidState_Txt_noUnit_boolean
internalBehavior		CMscB

Table 3.64: ApplicationSwComponentType CMscB

Common SwcInternalBehavior attributes	
shortName	CMscB
properties of the runnables	
	properties of RunnableEntity
	shortName	CMscB_Process
	desc	cyclic process for calculation
	symbol	CMscB_Process

Table 3.65: SwcInternalBehavior CMscB

3.2.2.4 ParameterInterfaces

Common ParameterInterface attributes	
shortName	Arr1dComAxis_Temp_Lin_K_uint16
desc	Array of shared axis for temperature
properties of the parameters	
properties of ParameterDataPrototype	
shortName	Arr1dComAxis_Temp_Lin_K_uint16
desc	Array of shared axis for temperature
type	ComAxis_Temp_Lin_K_uint16_ScNoOfWheels
swImplPolicy	standard
swCalibrationAccess	readWrite
swAddrMethod	CAL

Table 3.66: ParameterInterface Arr1dComAxis_Temp_Lin_K_uint16

Common ParameterInterface attributes	
shortName	Arr1dMap_Time_Lnr_s_uint16
desc	Map to get time dependent on temperature and mass.
properties of the parameters	
properties of ParameterDataPrototype	
shortName	Arr1dMap_Time_Lnr_s_uint16
desc	Map to get time dependent on temperature and mass.
type	Map_Time_Lnr_s_uint16_ScNoOfWheels
swImplPolicy	standard
swCalibrationAccess	readWrite
swAddrMethod	CAL

Table 3.67: ParameterInterface Arr1dMap_Time_Lnr_s_uint16

Common ParameterInterface attributes	
shortName	ComAxis_Mass_Lnr_Kg_uint8
desc	Shared axis for mass
properties of the parameters	
properties of ParameterDataPrototype	
shortName	ComAxis_Mass_Lnr_Kg_uint8
desc	Shared axis for mass
type	ComAxis_Mass_Lnr_Kg_uint8
swImplPolicy	standard
swCalibrationAccess	readWrite
swAddrMethod	CAL

Table 3.68: ParameterInterface ComAxis_Mass_Lnr_Kg_uint8

Common ParameterInterface attributes	
shortName	ComAxis_Steps_Txt_sint8
desc	Shared axis for speed steps
properties of the parameters	
properties of ParameterDataPrototype	
shortName	ComAxis_Steps_Txt_sint8
desc	Shared axis for speed steps
type	ComAxis_Steps_Txt_sint8
swImplPolicy	standard
swCalibrationAccess	readWrite
swAddrMethod	CAL

Table 3.69: ParameterInterface ComAxis_Steps_Txt_sint8

Common ParameterInterface attributes	
shortName	ComAxis_Temp_Lin_K_uint16
desc	Shared axis for temperature
properties of the parameters	
properties of ParameterDataPrototype	
shortName	ComAxis_Temp_Lin_K_uint16
desc	Shared axis for temperature
type	ComAxis_Temp_Lin_K_uint16
swImplPolicy	standard
swCalibrationAccess	readWrite
swAddrMethod	CAL

Table 3.70: ParameterInterface ComAxis_Temp_Lin_K_uint16

Common ParameterInterface attributes	
shortName	Curve_Mass_Lnr_Kg_uint8
desc	Curve to get mass according differnt speed steps.
properties of the parameters	
properties of ParameterDataPrototype	
shortName	Curve_Mass_Lnr_Kg_uint8
desc	Curve to get mass according differnt speed steps.
type	Curve_Mass_Lnr_Kg_uint8
swImplPolicy	standard
swCalibrationAccess	readWrite
swAddrMethod	CAL

Table 3.71: ParameterInterface Curve_Mass_Lnr_Kg_uint8

Common ParameterInterface attributes	
shortName	Map_Time_Lnr_s_uint16
desc	Map to get time dependent on temperature and mass.
properties of the parameters	
properties of ParameterDataPrototype	
shortName	Map_Time_Lnr_s_uint16
desc	Map to get time dependent on temperature and mass.
type	Map_Time_Lnr_s_uint16
swImplPolicy	standard
swCalibrationAccess	readWrite
swAddrMethod	CAL

Table 3.72: ParameterInterface Map_Time_Lnr_s_uint16

Common ParameterInterface attributes	
shortName	PrimCal_Mass_Lnr_Kg
desc	Primitive calibration parameter for minimum egg mass.
properties of the parameters	
properties of ParameterDataPrototype	
shortName	PrimCal_Mass_Lnr_Kg
desc	Primitive calibration parameter for minimum egg mass.
type	kg_Lnr_0_0d25_0_C8_uint8
swImplPolicy	standard
swCalibrationAccess	readWrite
swAddrMethod	CAL

Table 3.73: ParameterInterface PrimCal_Mass_Lnr_Kg

3.2.2.5 ModeSwitchInterfaces

Common ModeSwitchInterface attributes	
shortName	ModeDirection
desc	Mode to indicate a direction
properties of the modeGroups	
shortName	ModeDirection
swCalibrationAccess	readOnly
type	Direction

Table 3.74: ModeSwitchInterface ModeDirection

3.2.2.6 SenderReceiverInterfaces

Common SenderReceiverInterface attributes	
shortName	Arr1dPrimData_Temperature_Lin_K_C_uint16
desc	Temperature 1 in Kelvin but displayed as degree Celsius
properties of the dataElements	
properties of VariableDataPrototype	
shortName	Arr1dPrimData_Temperature_Lin_K_C_uint16
desc	Temperature 1 in Kelvin but displayed as degree Celsius
type	K_Celsius_Lnr_0_511d9921875_0_FFFF_uint16_ScNoOf-Wheels
swImplPolicy	standard
swCalibrationAccess	readOnly
swAddrMethod	DATA

Table 3.75: SenderReceiverInterface Arr1dPrimData_Temperature_Lin_K_C_uint16

Common SenderReceiverInterface attributes	
shortName	PrimData_Mass_Lnr_Kg_uint8
desc	Mass in kilogram
properties of the dataElements	
properties of VariableDataPrototype	
shortName	PrimData_Mass_Lnr_Kg_uint8
desc	Mass in kilogram
type	kg_Lnr_0_0d25_0_C8_uint8
swImplPolicy	standard
swCalibrationAccess	readOnly
swAddrMethod	DATA

Table 3.76: SenderReceiverInterface PrimData_Mass_Lnr_Kg_uint8

Common SenderReceiverInterface attributes	
shortName	PrimData_MassCorrected_Lnr_Kg_uint8
desc	Primitve data for the corrected mass in kg.
properties of the dataElements	
properties of VariableDataPrototype	
shortName	PrimData_MassCorrected_Lnr_Kg_uint8
desc	Primitve data for the corrected mass in kg.
type	kg_Lnr_0_0d25_0_C8_uint8
swImplPolicy	standard
swCalibrationAccess	readOnly
swAddrMethod	DATA

Table 3.77: SenderReceiverInterface PrimData_MassCorrected_Lnr_Kg_uint8

Common SenderReceiverInterface attributes	
shortName	PrimData_Temperature_Lin_K_C_uint16
desc	Temperature 1 in Kelvin but displayed as degree Celsius
properties of the dataElements	
properties of VariableDataPrototype	
shortName	PrimData_Temperature_Lin_K_C_uint16
desc	Temperature 1 in Kelvin but displayed as degree Celsius
type	K_Celsius_Lnr_0_511d9921875_0_FFFF_uint16
swImplPolicy	standard
swCalibrationAccess	readOnly
swAddrMethod	DATA

Table 3.78: SenderReceiverInterface PrimData_Temperature_Lin_K_C_uint16

Common SenderReceiverInterface attributes	
shortName	PrimData_Time_Lnr_s_uint16
desc	Primitve data holding a time value.
properties of the dataElements	
properties of VariableDataPrototype	
shortName	PrimData_Time_Lnr_s_uint16
desc	Primitve data holding a time value.
type	s_Lnr_0_8191d875_0_FFFF_uint16
swImplPolicy	standard
swCalibrationAccess	readOnly
swAddrMethod	DATA

Table 3.79: SenderReceiverInterface PrimData_Time_Lnr_s_uint16

Common SenderReceiverInterface attributes	
shortName	PrimData_ValidState_Txt_noUnit_boolean
desc	Boolean representing the data validity
properties of the dataElements	
properties of VariableDataPrototype	
shortName	PrimData_ValidState_Txt_noUnit_boolean
desc	Boolean representing the data validity
type	DataValidityType
swImplPolicy	standard
swCalibrationAccess	readOnly
swAddrMethod	DATA

Table 3.80: SenderReceiverInterface PrimData_ValidState_Txt_noUnit_boolean

3.2.2.7 ApplicationDataTypes, Category BOOLEAN

Common ApplicationDataType attributes					
shortName	DataValidityType				
category	BOOLEAN				
desc	Boolean to represent the data validity				
swCalibrationAccess	notAccessible				
unit	NoUnit				
Range					
		lowerLimit	upperLimit		
	physConstrs	0	1		
Conversion					
	category	TEXTTABLE			
	direction	compuInternalToPhys			
	desc	lowerLimit	upperLimit	vt	symbol
	-	0	0	Invalid	
	-	1	1	Valid	

Table 3.81: ApplicationDataType DataValidityType

3.2.2.8 ApplicationDataTypes, Category VALUE

Common ApplicationDataType attributes				
shortName	K_Celsius_Lnr_0_511d9921875_0_FFFF_uint16			
category	VALUE			
desc	Temperature			
swCalibrationAccess	notAccessible			
unit	K			
Range				
		lowerLimit	upperLimit	
	physConstrs	0	511.9921875	
Conversion				
	category	LINEAR		
	direction	compuInternalToPhys		
	desc	lowerLimit	upperLimit	compuNumerator / compuDenominator
	-	-	-	$Phys = \frac{0 + 0.0078125 * Internal}{1}$

Table 3.82: ApplicationDataType K_Celsius_Lnr_0_511d9921875_0_FFFF_uint16

Common ApplicationDataType attributes				
shortName	kg_Lnr_0_0d25_0_C8_uint8			
category	VALUE			
desc	Mass			
swCalibrationAccess	notAccessible			
unit	kg			
Range				
		lowerLimit	upperLimit	
	physConstrs	0	0.25	
Conversion				
	category	LINEAR		
	direction	compuInternalToPhys		
	desc	lowerLimit	upperLimit	compuNumerator / compuDenominator
	-	-	-	$Phys = \frac{0 + 0.00125 * Internal}{1}$

Table 3.83: ApplicationDataType kg_Lnr_0_0d25_0_C8_uint8

Common ApplicationDataType attributes				
shortName	NoUnit_Lnr_1_4_1_4_uint8			
category	VALUE			
swCalibrationAccess	notAccessible			
unit	NoUnit			
Range				
		lowerLimit	upperLimit	
	physConstrs	1	4	
Conversion				
	category	LINEAR		
	direction	compuInternalToPhys		
	desc	lowerLimit	upperLimit	compuNumerator / compuDenominator
	-	-	-	$Phys = \frac{0 + 1 * Internal}{1}$

Table 3.84: ApplicationDataType NoUnit_Lnr_1_4_1_4_uint8

Common ApplicationDataType attributes				
shortName	NoUnit_Lnr_1_65535_1_FFFF_uint16			
category	VALUE			
swCalibrationAccess	notAccessible			
unit	NoUnit			
Range				
		lowerLimit	upperLimit	
	physConstrs	1	65535	
Conversion				
	category	LINEAR		
	direction	compuInternalToPhys		
	desc	lowerLimit	upperLimit	compuNumerator / compuDenominator
	-	-	-	$Phys = \frac{0 + 1 * Internal}{1}$

Table 3.85: ApplicationDataType NoUnit_Lnr_1_65535_1_FFFF_uint16

Common ApplicationDataType attributes				
shortName	s_Lnr_0_8191d875_0_FFFF_uint16			
category	VALUE			
desc	cooking time in seconds			
swCalibrationAccess	notAccessible			
unit	s			
Range				
		lowerLimit	upperLimit	
	physConstrs	0	8191.875	
Conversion				
	category	LINEAR		
	direction	compuInternalToPhys		
	desc	lowerLimit	upperLimit	compuNumerator / compuDenominator
	-	-	-	$Phys = \frac{0 + 0.125 * Internal}{1}$

Table 3.86: ApplicationDataType s_Lnr_0_8191d875_0_FFFF_uint16

Common ApplicationDataType attributes					
shortName	speedSteps				
category	VALUE				
desc	Possible speed steps				
swCalibrationAccess	notAccessible				
unit	NoUnit				
Range					
		lowerLimit		upperLimit	
	physConstrs	-1		2	
Conversion					
	category	TEXTTABLE			
	direction	compuInternalToPhys			
	desc	lowerLimit	upperLimit	vt	symbol
	-	-1	-1	Stop	
	-	0	0	LightSpeed	
	-	1	1	RidiculousSpeed	
	-	2	2	LudicrousSpeed	

Table 3.87: ApplicationDataType speedSteps

Common ApplicationDataType attributes					
shortName	TxWheelNames				
category	VALUE				
desc	Wheel names				
swCalibrationAccess	notAccessible				
unit	NoUnit				
Range					
		lowerLimit		upperLimit	
	physConstrs	0		3	
Conversion					
	category	TEXTTABLE			
	direction	compuInternalToPhys			
	desc	lowerLimit	upperLimit	vt	symbol
	-	0	0	FrontLeft	
	-	1	1	FrontRight	
	-	2	2	RearLeft	
	-	3	3	RearRight	

Table 3.88: ApplicationDataType TxWheelNames

3.2.2.9 ApplicationDataTypes, Category COM_AXIS

Common ApplicationDataType attributes	
shortName	ComAxis_Temp_Lin_K_uint16
category	COM_AXIS
swCalibrationAccess	notAccessible
swRecordLayout	RL20_ME_Axis
properties of the axes (swCalprmAxisSet)	
properties of SwAxisIndividual (swCalprmAxis and swCalprmAxisTypeProps)	
swAxisIndex	1
category	COM_AXIS
inputVariableType	K_Celsius_Lnr_0_511d9921875_0_FFFF_uint16
swMaxAxisPoints	6
swMinAxisPoints	6

Table 3.89: ApplicationDataType ComAxis_Temp_Lin_K_uint16

Common ApplicationDataType attributes	
shortName	ComAxis_Steps_Txt_sint8
category	COM_AXIS
swCalibrationAccess	notAccessible
swRecordLayout	RL20_ME_Axis
properties of the axes (swCalprmAxisSet)	
properties of SwAxisIndividual (swCalprmAxis and swCalprmAxisTypeProps)	
swAxisIndex	1
category	COM_AXIS
inputVariableType	speedSteps
swMaxAxisPoints	4
swMinAxisPoints	4

Table 3.90: ApplicationDataType ComAxis_Steps_Txt_sint8

Common ApplicationDataType attributes	
shortName	ComAxis_Mass_Lnr_Kg_uint8
category	COM_AXIS
swCalibrationAccess	notAccessible
swRecordLayout	RL20_ME_Axis
properties of the axes (swCalprmAxisSet)	
properties of SwAxisIndividual (swCalprmAxis and swCalprmAxisTypeProps)	
swAxisIndex	1
category	COM_AXIS
inputVariableType	kg_Lnr_0_0d25_0_C8_uint8
swMaxAxisPoints	4
swMinAxisPoints	4

Table 3.91: ApplicationDataType ComAxis_Mass_Lnr_Kg_uint8

3.2.2.10 ApplicationDataTypes, Category CURVE

Common ApplicationDataType attributes	
shortName	Curve_Mass_Lnr_Kg_uint8
category	CURVE
swCalibrationAccess	notAccessible
swRecordLayout	RL20_ME_1DimMap
valueAxisDataType	kg_Lnr_0_0d25_0_C8_uint8
properties of the axes (swCalprmAxisSet)	
properties of SwAxisGrouped (swCalprmAxis and swCalprmAxisTypeProps)	
swAxisIndex	1
category	COM_AXIS
sharedAxisType	ComAxis_Steps_Txt_sint8

Table 3.92: ApplicationDataType Curve_Mass_Lnr_Kg_uint8

3.2.2.11 ApplicationDataTypes, Category MAP

Common ApplicationDataType attributes	
shortName	Map_Time_Lnr_s_uint16
category	MAP
swCalibrationAccess	notAccessible
swRecordLayout	RL20_ME_2DimMap
valueAxisDataType	s_Lnr_0_8191d875_0_FFFF_uint16
properties of the axes (swCalprmAxisSet)	
properties of SwAxisGrouped (swCalprmAxis and swCalprmAxisTypeProps)	
swAxisIndex	1
category	COM_AXIS
sharedAxisType	ComAxis_Temp_Lin_K_uint16
properties of SwAxisGrouped (swCalprmAxis and swCalprmAxisTypeProps)	
swAxisIndex	2
category	COM_AXIS
sharedAxisType	ComAxis_Mass_Lnr_Kg_uint8

Table 3.93: ApplicationDataType Map_Time_Lnr_s_uint16

3.2.2.12 ApplicationArrayDataTypes

Common ApplicationArrayDataType attributes	
shortName	ComAxis_Temp_Lin_K_uint16_ScNoOfWheels
category	ARRAY
swCalibrationAccess	notAccessible
properties of the elements	
properties of ApplicationArrayElement	
shortName	ComAxis_Temp_Lin_K_uint16_ScNoOfWheels
category	COM_AXIS
type	ComAxis_Temp_Lin_K_uint16
arraySizeSemantics	fixedSize
maxNumberOfElements	

Table 3.94: ApplicationArrayDataType ComAxis_Temp_Lin_K_uint16_ScNoOfWheels

Common ApplicationArrayDataType attributes	
shortName	K_Celsius_Lnr_0_511d9921875_0_FFFF_uint16_ScNoOfWheels
category	ARRAY
swCalibrationAccess	notAccessible
properties of the elements	
properties of ApplicationArrayElement	
shortName	K_Celsius_Lnr_0_511d9921875_0_FFFF_uint16_ScNoOfWheels
category	VALUE
type	K_Celsius_Lnr_0_511d9921875_0_FFFF_uint16
arraySizeSemantics	fixedSize
maxNumberOfElements	

Table 3.95: ApplicationArrayDataType K_Celsius_Lnr_0_511d9921875_0_FFFF_uint16_ScNoOfWheels

Common ApplicationArrayDataType attributes	
shortName	Map_Time_Lnr_s_uint16_ScNoOfWheels
category	ARRAY
swCalibrationAccess	notAccessible
properties of the elements	
properties of ApplicationArrayElement	
shortName	Map_Time_Lnr_s_uint16_ScNoOfWheels
category	MAP
type	Map_Time_Lnr_s_uint16
arraySizeSemantics	fixedSize
maxNumberOfElements	

Table 3.96: ApplicationArrayDataType Map_Time_Lnr_s_uint16_ScNoOfWheels

3.2.2.13 ApplicationRecordDataTypes

Common ApplicationRecordDataType attributes	
shortName	CMscC_nvm_NvBlockATyp
category	STRUCTURE
swCalibrationAccess	notAccessible
properties of the elements	
properties of ApplicationRecordElement	
shortName	PrimData_StepsSpeed_Txt_sint8
category	VALUE
type	speedSteps

▽

△

Table 3.97: ApplicationRecordDataType CMscC_nvm_NvBlockATyp

3.2.2.14 ModeDeclarationGroups

Common ModeDeclarationGroup attributes	
shortName	Direction
category	EXPLICIT_ORDER
initialMode	Halt
properties of the modeDeclarations	
properties of ModeDeclaration	
shortName	Backward
desc	Backward direction
value	2
properties of ModeDeclaration	
shortName	Forward
desc	Forward direction
value	1
properties of ModeDeclaration	
shortName	Halt
desc	Standstill
value	0

Table 3.98: ModeDeclarationGroup Direction

3.2.2.15 Units

Common Unit attributes	
shortName	Celsius
desc	Degrees Celsius
displayName	℃
offsetSiToUnit	-273.15
factorSiToUnit	1.0
physicalDimension	PD_K

Table 3.99: Unit Celsius

Common Unit attributes	
shortName	K
desc	Temperature
displayName	K
offsetSiToUnit	0.0
factorSiToUnit	1.0
physicalDimension	PD_K

Table 3.100: Unit K

Common Unit attributes	
shortName	kg
desc	Mass
displayName	kg
offsetSiToUnit	0.0
factorSiToUnit	1.0
physicalDimension	PD_kg

Table 3.101: Unit kg

Common Unit attributes	
shortName	NoUnit
desc	No Unit
displayName	-
offsetSiToUnit	0.0
factorSiToUnit	1.0
physicalDimension	PD_NoUnit

Table 3.102: Unit NoUnit

Common Unit attributes	
shortName	s
desc	Time
displayName	s
offsetSiToUnit	0.0
factorSiToUnit	1.0
physicalDimension	PD_s

Table 3.103: Unit s

3.2.2.16 PhysicalDimensions

Common PhysicalDimension attributes	
shortName	PD_K
currentExp	0
lengthExp	0
luminousIntensity-Exp	0
massExp	0
molarAmountExp	0
temperatureExp	1
timeExp	0

Table 3.104: PhysicalDimension PD_K

Common PhysicalDimension attributes	
shortName	PD_kg
currentExp	0
lengthExp	0
luminousIntensity-Exp	0
massExp	1
molarAmountExp	0
temperatureExp	0
timeExp	0

Table 3.105: PhysicalDimension PD_kg

Common PhysicalDimension attributes	
shortName	PD_NoUnit
currentExp	0
lengthExp	0
luminousIntensity-Exp	0
massExp	0
molarAmountExp	0
temperatureExp	0
timeExp	0

Table 3.106: PhysicalDimension PD_NoUnit

Common PhysicalDimension attributes	
shortName	PD_s
currentExp	0
lengthExp	0
luminousIntensity-Exp	0
massExp	0
molarAmountExp	0
temperatureExp	0
timeExp	1

Table 3.107: PhysicalDimension PD_s

3.2.2.17 SwAddrMethods

Common SwAddrMethod attributes	
shortName	CAL
desc	Calibratable constants; safety level QM. Constants will be located in different memory sections depending on the alignment of the constant.
sectionType	calprm
memoryAllocation-KeywordPolicy	addrMethodShortNameAndAlignment
section-Initialization-Policy	-
option	safetyQM

Table 3.108: SwAddrMethod CAL

Common SwAddrMethod attributes	
shortName	CODE_10MS
desc	Code of ECU-functions called every 10 ms; safety level QM.
sectionType	code
memoryAllocation-KeywordPolicy	addrMethodShortName
section-Initialization-Policy	-
option	safetyQM

Table 3.109: SwAddrMethod CODE_10MS

Common SwAddrMethod attributes	
shortName	CONST_SLOW
desc	Non calibratable constants of ECU-functions called seldom; safety level QM.
sectionType	const
memoryAllocation-KeywordPolicy	addrMethodShortName
section-Initialization-Policy	-
option	safetyQM

Table 3.110: SwAddrMethod CONST_SLOW

Common SwAddrMethod attributes	
shortName	DATA
desc	Variables of ECU-functions; safety level QM. Variables will be located in different memory sections depending on the alignment of the variable.
sectionType	var
memoryAllocation-KeywordPolicy	addrMethodShortNameAndAlignment
section-Initialization-Policy	INIT
option	safetyQM

Table 3.111: SwAddrMethod DATA

Common SwAddrMethod attributes	
shortName	DATA_NVDAT
desc	Variables stored in non-volatile memory; safety level QM.
sectionType	var
memoryAllocation-KeywordPolicy	addrMethodShortName
section-Initialization-Policy	NO-INIT
option	nvData, safetyQM

Table 3.112: SwAddrMethod DATA_NVDAT

4 Structured Requirements

Structured Requirements are available at different sections in the meta model. There exists a clear definition of the elements of StructuredReq. Based on these elements a variety of possible use cases can be applied. The following show cases shall illustrate this:

- provide specification items as requirements (4.1)
- provide diagnostic requirements (4.2)
- provide decomposition of requirements (4.3)
- provide additional information for configuration (5)

Due to the fact that StructuredReq offer general usable attributes, they are filled in a context-specific manner to reflect the dedicated use case. No new and additional elements have to be introduced in the meta model because StructuredReq still provides all needed entities.

4.1 Specification items as requirements

AUTOSAR uses StructuredReq within AUTOSAR RS documents [TPS_STDT_00060]. They are automatically generated based on the specification generation process. Theses specification items use the attributes of StructuredReq including Traceable, to realize the up tracing. A dedicated example is illustrated in listing 4.1.

```
<STRUCTURED-REQ>
  <SHORT-NAME>SRS_Com_02040</SHORT-NAME>
  <LONG-NAME>
    <L-4 L="EN">AUTOSAR COM and LargeDataCOM shall be configured by using
       XML as configuration language</L-4>
  </LONG-NAME>
  <CATEGORY>REQUIREMENT_ITEM</CATEGORY>
  <DESCRIPTION>
    <P>
      <L-1 L="EN">AUTOSAR COM and LargeDataCOM shall be configured by
         using XML as configuration language as defined by ECU
         Configuration Template.</L-1>
    </P>
    <P>
      <L-1 L="EN">It is up to the <TT TYPE="Cite">CP_2d_SWS_2d_COM</TT>
         and <TT TYPE="Cite">CP_2d_SWS_2d_LargeDataCOM</TT> to define the
          configuration parameters themselves.</L-1>
    </P>
  </DESCRIPTION>
  <RATIONALE>
    <P>
      <L-1 L="EN">Having a unique configuration language within AUTOSAR.</
         L-1>
    </P>
  </RATIONALE>
```

```
    <DEPENDENCIES>
      <P>
        <L-1 L="EN">–</L-1>
      </P>
    </DEPENDENCIES>
    <USE-CASE>
      <P>
        <L-1 L="EN">Configuration of AUTOSAR COM and LargeDataCOM</L-1>
      </P>
    </USE-CASE>
    <SUPPORTING-MATERIAL>
      <P>
        <L-1 L="EN"><TT TYPE="Cite">CP_2d_TPS_2d_ECUConfiguration</TT></L-1>
      </P>
    </SUPPORTING-MATERIAL>
  </STRUCTURED-REQ>
```

Listing 4.1: Example for `StructuredReq` in RS AUTOSAR Specification

The rendering of the `StructuredReq` in the RS AUTOSAR Specification is given in figure 4.1.

[SRS_Com_02040] AUTOSAR COM and LargeDataCOM shall be configured by using XML as configuration language ⌈

Description:	AUTOSAR COM and LargeDataCOM shall be configured by using XML as configuration language as defined by ECU Configuration Template. It is up to the [3] and [4] to define the configuration parameters themselves.
Rationale:	Having a unique configuration language within AUTOSAR.
Use Case:	Configuration of AUTOSAR COM and LargeDataCOM
Dependencies:	–
Supporting Material:	[9]

⌋

Figure 4.1: Spec item represented by `StructuredReq`

Using `StructuredReq` a complete requirements description and tracing is available.

4.2 Diagnostic requirements

In this context the `StructuredReq` is embedded in `DiagnosticAccessPermission`. Mainly the description of the two conditions are shown here.

```
<DIAGNOSTIC-ACCESS-PERMISSION>
  <SHORT-NAME>exampleAccessPermission</SHORT-NAME>
  <INTRODUCTION>
    <STRUCTURED-REQ>
      <SHORT-NAME>precondition</SHORT-NAME>
      <CATEGORY>DIAG_ACCESS_PERM_PRE_COND</CATEGORY>
      <DESCRIPTION>
        <P>
```

```
          <L-1 L="EN">This is a textual description of a pre-condition</L-1
              >
          </P>
        </DESCRIPTION>
      </STRUCTURED-REQ>
      <STRUCTURED-REQ>
        <SHORT-NAME>runcondition</SHORT-NAME>
        <CATEGORY>DIAG_ACCESS_PERM_RUN_COND</CATEGORY>
        <DESCRIPTION>
          <P>
            <L-1 L="EN">This is a textual description of a run-condition</L-1
              >
          </P>
        </DESCRIPTION>
      </STRUCTURED-REQ>
    </INTRODUCTION>
    <DIAGNOSTIC-SESSION-REFS>
      <DIAGNOSTIC-SESSION-REF DEST="DIAGNOSTIC-SESSION">/AUTOSAR/UseCase_230/
          ExampleSession</DIAGNOSTIC-SESSION-REF>
    </DIAGNOSTIC-SESSION-REFS>
    <SECURITY-LEVEL-REFS>
      <SECURITY-LEVEL-REF DEST="DIAGNOSTIC-SECURITY-LEVEL">/AUTOSAR/
          UseCase_230/ExampleSecurityLevel</SECURITY-LEVEL-REF>
    </SECURITY-LEVEL-REFS>
</DIAGNOSTIC-ACCESS-PERMISSION>
```

Listing 4.2: Example for the definition of pre- and run-conditions for `DiagnosticAccessPermission`

4.3 Decomposition of requirements

The `StructuredReq` can also be used for decomposition of requirements.

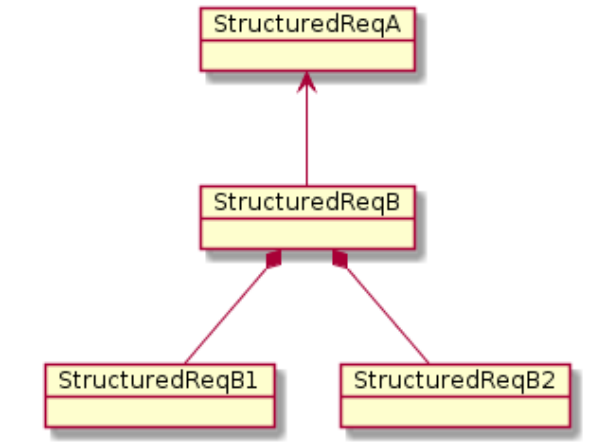


Figure 4.2: Decomposition of `StructuredReq`

In figure 4.2 the `StructuredReqB` has an upstream requirement to `StructuredReqA` realized by `Traceable` using `<TRACE-REF>`. The `StructuredReqB` itself is decomposed by `StructuredReqB1` and `StructuredReqB2`. This is realized by the

attribute dependencies of StructuredReq. It contains two <TRACE>s which in turn use <TRACE-REF> again.

```
<STRUCTURED-REQ>
  <SHORT-NAME>A</SHORT-NAME>
  <CATEGORY>REQUIREMENT_ITEM</CATEGORY>
  <DATE>2019-07-19</DATE>
  <ISSUED-BY>OEM</ISSUED-BY>
  <TYPE>valid</TYPE>
  <IMPORTANCE>High</IMPORTANCE>
</STRUCTURED-REQ>
<STRUCTURED-REQ>
  <SHORT-NAME>B</SHORT-NAME>
  <CATEGORY>REQUIREMENT_ITEM</CATEGORY>
  <TRACE-REFS>
    <TRACE-REF BASE="ArTrace" DEST="STRUCTURED-REQ">DecompositionExample/A
      </TRACE-REF>
  </TRACE-REFS>
  <DATE>2019-07-19</DATE>
  <ISSUED-BY>OEM</ISSUED-BY>
  <TYPE>valid</TYPE>
  <IMPORTANCE>High</IMPORTANCE>
  <DEPENDENCIES>
    <TRACE>
      <SHORT-NAME>DECOMPB1</SHORT-NAME>
      <LONG-NAME>
        <L-4 L="EN">Decomposition element 1</L-4>
      </LONG-NAME>
      <CATEGORY>DECOMPOSITION</CATEGORY>
      <TRACE-REFS>
        <TRACE-REF BASE="ArTrace" DEST="STRUCTURED-REQ">
          DecompositionExample/B1</TRACE-REF>
      </TRACE-REFS>
    </TRACE>
    <TRACE>
      <SHORT-NAME>DECOMPB2</SHORT-NAME>
      <LONG-NAME>
        <L-4 L="EN">Decomposition element 2</L-4>
      </LONG-NAME>
      <CATEGORY>DECOMPOSITION</CATEGORY>
      <TRACE-REFS>
        <TRACE-REF BASE="ArTrace" DEST="STRUCTURED-REQ">
          DecompositionExample/B2</TRACE-REF>
      </TRACE-REFS>
    </TRACE>
  </DEPENDENCIES>
</STRUCTURED-REQ>
<STRUCTURED-REQ>
  <SHORT-NAME>B1</SHORT-NAME>
  <CATEGORY>REQUIREMENT_ITEM</CATEGORY>
  <DATE>2019-07-19</DATE>
  <ISSUED-BY>OEM</ISSUED-BY>
  <TYPE>valid</TYPE>
  <IMPORTANCE>High</IMPORTANCE>
</STRUCTURED-REQ>
<STRUCTURED-REQ>
```

```
  <SHORT-NAME>B2</SHORT-NAME>
  <CATEGORY>REQUIREMENT_ITEM</CATEGORY>
  <DATE>2019-07-19</DATE>
  <ISSUED-BY>OEM</ISSUED-BY>
  <TYPE>valid</TYPE>
  <IMPORTANCE>High</IMPORTANCE>
</STRUCTURED-REQ>
```

Listing 4.3: Example for decomposition of `StructuredReq`

5 Integration Requirements

Creating an AUTOSAR Target Configuration (i.e. Ecu Configuration on the Classic Platform and APMC on the Adaptive Platform) is a challenging task that requires a deep understanding of the AUTOSAR standards, meticulous attention to detail, and effective use of tools and methodology. In figure 5.1 a high level workflow of the Target Configuration creation and refinement is sketched.

Although many configuration parameters can be directly derived from Upstream descriptions (*t1* in figure 5.1, like System Extract, Diagnostic Extract, etc.) there are still a significant number of configuration parameters that the integrator of the ECU/Machine has to define.

The goal of Integration Requirements is to reduce the number of undefined configuration parameters after the initial Target Configuration derivation. Once the initial Target Configuration is produced, Integration Requirements can be applied to assign values to some of the yet undefined configuration parameters (*t2* in figure 5.1).

Integration Requirements provide configuration parameter value definitions that can be categorized as either `mandatory`, `suggested`, or `information`. Mandatory values are those that the Target Configuration shall have for all occurrences of a specific configuration parameter. Suggested values serve as a starting point for the Target Configuration if the parameter is not yet defined, but it is allowed to deviate from these suggested values. Information is given as a hint to the integrator as how to set the value of a configuration parameter.

Even when the integrator works with the Target Configuration and modifies or adds further configuration elements, the Integration Requirements can be applied repeatedly to continuously enhance the Target Configuration (also *t2* in figure 5.1).

The Integration Requirements can also be used to check the Target Configuration and make sure that `mandatory` Integration Requirements values are actually configured as required by the Integration Requirements definition. This is shown in *t3* of figure 5.1, where the current Target Configuration Values are compared to the Integration Requirements. The resulting report can be used to continuously improve the Target Configuration and resolve many issues already during the integration phase *t3*.

The resulting report can also be used as a basis for discussions with customers. Handling deviations from Integration Requirements should be documented and potentially approved by customers as shown in *t4*. This is further elaborated in section 5.5.

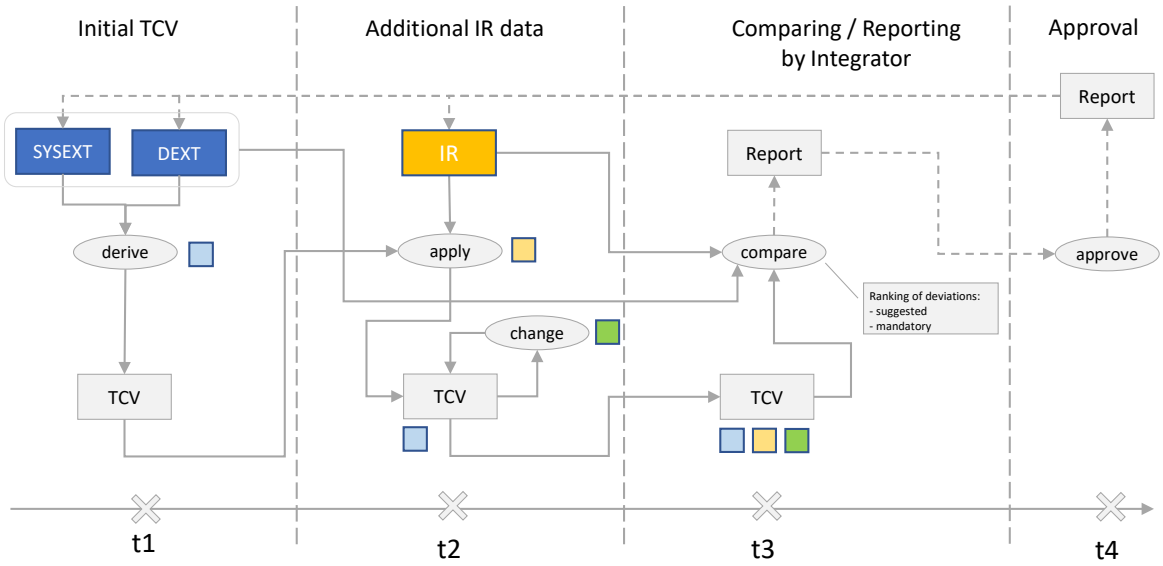


Figure 5.1: Target Configuration Workflow

The methodological framework of Integration Requirements is the meta-class `StructuredReq`, providing a dedicated set of attributes to characterize Integration Requirements values. The concrete design and parametrization of the attributes of `StructuredReq` in the context of the Integration Requirements is defined in 5.1.

Mandatory integration requirements will always create a parameter. If a parameter with the same value already exists, it's recommended to skip the creation. Suggested integration requirements will only create a parameter if it doesn't exist. This means that it's advantageous to first apply all mandatory integration requirements, and afterwards apply the suggested ones.

5.1 Defining Integration Requirements using `StructuredReq`

Using `StructuredReq` in the scope of the Integration Requirements leads to concrete defined values of some attributes.

If not otherwise stated the attributes of `StructuredReq` are used according to their general definition.

5.1.1 `StructuredReq.shortName`

The value of `StructuredReq.shortName` is used to identify the Integration Requirement and make them referable for further processing and documentation. This may be useful for tracing of change and reporting (see section 5.5).

5.1.2 StructuredReq.category

The value of StructuredReq.category shall be set to REQUIREMENT_ITEM. This identifies this StructuredReq as a dedicated requirement.

5.1.3 StructuredReq.importance

The value of StructuredReq.importance shall be one of the following strings:

- mandatory
- suggested
- information

A StructuredReq with importance equal to mandatory defines a binding value setting for that parameter. Any deviation of the value shall result in a error reporting.

A StructuredReq with importance equal to suggested defines a suggested value value setting for that parameter. Any deviation of the value may result in a warning reporting.

A StructuredReq with importance equal to information provides some documentation on how this parameter may be configured. No format checks are expected for the reporting on this Integration Requirement.

5.1.4 StructuredReq.rationale

The StructuredReq.rationale provides the possibility to describe some justification why this Integration Requirement is applied in this way.

5.1.5 StructuredReq.useCase

The StructuredReq.useCase attribute shall be set to Integration Requirement. This identifies this StructuredReq as a dedicated Integration Requirement.

5.1.6 StructuredReq.description

The StructuredReq.description is semi-structured according to the following definition. Any Integration Requirement implements the structure (or parts) as follows:

- description
  - List
    - Item with XML attribute SI = DefinitionReference

* Item with XML attribute SI = Description
* Item with XML attribute SI = ARVersion
* Item with XML attribute SI = ModuleVersion
* Item with XML attribute SI = SetValue
* Item with XML attribute SI = SupportedLowValue
* Item with XML attribute SI = SupportedHighValue
* Item with XML attribute SI = SupportedValue

5.1.6.1 DefinitionReference

Defines an Xref to the target configuration parameter definition this Integration Requirement applies to.

The reference can either point to the Standardized AUTOSAR Target Configuration Parameter Definition or to the Vendor Specific Target Configuration Parameter Definition.

If the DefinitionReference refers to a Standardized AUTOSAR Target Configuration Parameter Definition, then the corresponding Vendor Specific Target Configuration Parameter Definition needs to be derived. This approach is useful if the Integration Requirements shall be applicable to all AUTOSAR implementations, regardless which vendor has implemented the basic software and thus derived his specific Vendor Specific Target Configuration Parameter Definition. Thus Vendor Specific Target Configuration Parameter Definition references are explicitly avoided in the Integration Requirement DefinitionReference.

If the DefinitionReference refers to a Vendor Specific Target Configuration Parameter Definition, then already the specific vendor specific parameter is addressed. This Integration Requirement is specific to that vendors AUTOSAR implementation. But with this approach also custom vendor specific parameters can be the target of Integration Requirements.

```
<ITEM SI="DefinitionReference">
  <P>
    <L-1 L="EN"><XREF><REFERRABLE-REF DEST="ECUC-BOOLEAN-PARAM-DEF">/
      AUTOSAR/EcucDefs/CanIf/CanIfInitCfg/CanIfRxPduCfg/
      CanIfRxPduReadData</REFERRABLE-REF></XREF></L-1>
  </P>
</ITEM>
```

Listing 5.1: Example DefinitionReference

5.1.6.2 Description

Allows to provide a description for this Integration Requirement. This can be used to describe the motivation and background why this Integration Requirement is defined that way.

5.1.6.3 ARVersion

Defines the supported AUTOSAR versions this Integration Requirement applies to.

```
<ITEM SI="ARVersion">
  <P>
    <L-1 L="EN">4.4.0</L-1>
  </P>
  <P>
    <L-1 L="EN">R25-11</L-1>
  </P>
</ITEM>
```

Listing 5.2: Example `ARVersion`

5.1.6.4 ModuleVersion

Defines the supported BSW Module implementation versions this Integration Requirement applies to.

```
<ITEM SI="ModuleVersion">
  <P>
    <L-1 L="EN">12.1.0</L-1>
  </P>
</ITEM>
```

Listing 5.3: Example `ModuleVersion`

5.1.6.5 SetValue

Defines the value to be set by this Integration Requirement.

If no `SetValue` is defined, than this Integration Requirement defines the allowed range (`SupportedLowValue` and `SupportedHighValue`) / subset (`SupportedValue`) of possible values to be used.

The defined value shall respect the type of target configuration parameter this Integration Requirement applies to.

```
<ITEM SI="SetValue">
  <P>
    <L-1 L="EN">true</L-1>
  </P>
</ITEM>
```

Listing 5.4: Example `SetValue`

5.1.6.6 SupportedLowValue

Defines the lowest allowed value to be set by this Integration Requirement. This is applicable for Numerical Value parameters.

The defined value shall respect the type of target configuration parameter this Integration Requirement applies to.

```
<ITEM SI="SupportedLowValue">
  <P>
    <L-1 L="EN">0</L-1>
  </P>
</ITEM>
<ITEM SI="SupportedHighValue">
  <P>
    <L-1 L="EN">1048575</L-1>
  </P>
</ITEM>
```

Listing 5.5: Example `SupportedLowValue` and `SupportedHighValue`

5.1.6.7 SupportedHighValue

Defines the highest allowed value to be set by this Integration Requirement. This is applicable for Numerical Value parameters.

The defined value shall respect the type of target configuration parameter this Integration Requirement applies to.

For example see listing for `SupportedLowValue`

5.1.6.8 SupportedValue

Defines possible values to be set by this Integration Requirement. This is applicable for Enumerations and String parameters.

This is a list of allowed values the referenced parameter value is restricted to use.

If at least one SupportedValue is defined, then

- Enumeration: only one of the explicitly listed SupportedValues is allowed to be used
- String: only strings which match one of the defined SupportedValues are allowed.

If no SupportedValue is defined, then

- Enumeration: all available enumeration values of the enumeration parameter definition are supported
- String: no restriction on the string parameter value is defined.

The defined value shall respect the type of target configuration parameter this Integration Requirement applies to.

```
<ITEM SI="SupportedValue">
  <P>
    <L-1 L="EN">TRUE</L-1>
  </P>
  <P>
    <L-1 L="EN">FALSE</L-1>
  </P>
</ITEM>
```

Listing 5.6: Example SupportedValue

5.2 Applying Integration Requirements

5.2.1 Single configuration value in a single container

An example of a simple integration requirement is shown in listing 5.7.

The Integration Requirement shall be applied to the configuration parameter CanIfDevErrorDetect. This parameter is a *singular* and *not optional* configuration parameter of the CanIf module. This means, if a configuration for the CanIf module exists, then exactly one configuration value for the parameter CanIfDevErrorDetect shall exists and be set to the value FALSE.

```
<STRUCTURED-REQ>
  <SHORT-NAME>CanIfDevErrorDetect</SHORT-NAME>
  <LONG-NAME>
    <L-4 L="EN">CanIfDevErrorDetect</L-4>
  </LONG-NAME>
  <CATEGORY>REQUIREMENT_ITEM</CATEGORY>
  <DATE>2025-02-20</DATE>
  <ISSUED-BY>OEM</ISSUED-BY>
  <TYPE>valid</TYPE>
  <IMPORTANCE>mandatory</IMPORTANCE>
  <DESCRIPTION>
```

```
    <LIST>
      <ITEM SI="Description">
        <P>
          <L-1 L="EN">Switches the development error detection and
              notification on or off. * true: detection and notification is
               enabled. * false: detection and notification is disabled.</L
              -1>
        </P>
      </ITEM>
      <ITEM SI="DefinitionReference">
        <P>
          <L-1 L="EN"><XREF><REFERRABLE-REF DEST="ECUC-BOOLEAN-PARAM-DEF">
              /AUTOSAR/EcucDefs/CanIf/CanIfPublicCfg/CanIfDevErrorDetect</
              REFERRABLE-REF></XREF></L-1>
        </P>
      </ITEM>
      <ITEM SI="SetValue">
        <P>
          <L-1 L="EN">FALSE</L-1>
        </P>
      </ITEM>
      <ITEM SI="SupportedValue">
        <P>
          <L-1 L="EN">TRUE</L-1>
        </P>
        <P>
          <L-1 L="EN">FALSE</L-1>
        </P>
      </ITEM>
      <ITEM SI="ARVersion">
        <P>
          <L-1 L="EN">4.4.0</L-1>
        </P>
        <P>
          <L-1 L="EN">R25-11</L-1>
        </P>
      </ITEM>
    </LIST>
  </DESCRIPTION>
  <RATIONALE>
    <P>
      <L-1 L="EN">The feature development error detection shall be
          disabled for production code else it has to be ensured that there
           will be no side effects from this code.</L-1>
    </P>
  </RATIONALE>
  <USE-CASE>
    <P>
      <L-1 L="EN">IntegrationRequirement</L-1>
    </P>
  </USE-CASE>
</STRUCTURED-REQ>
```

Listing 5.7: Example 1 for integration requirement using `StructuredReq`

The listing 5.8 shows a simplified ecu configuration values model for a `CanIf` module.

Already a setting for the configuration parameter CanIfVersionInfoApi in the container CanIfPublicCfg is provided. However, there is no setting for the value of CanIfDevErrorDetect.

```
<ECUC-MODULE-CONFIGURATION-VALUES>
  <SHORT-NAME>CanIf</SHORT-NAME>
  <DEFINITION-REF DEST="ECUC-MODULE-DEF">/AUTOSAR/EcucDefs/CanIf</
     DEFINITION-REF>
  <CONTAINERS>
    <ECUC-CONTAINER-VALUE>
      <SHORT-NAME>CanIfPublicCfg</SHORT-NAME>
      <DEFINITION-REF DEST="ECUC-CONTAINER-DEF">/AUTOSAR/EcucDefs/CanIf/
         CanIfPublicCfg</DEFINITION-REF>
      <PARAMETER-VALUES>
        <ECUC-NUMERICAL-PARAM-VALUE>
          <DEFINITION-REF DEST="ECUC-BOOLEAN-PARAM-DEF">/AUTOSAR/EcucDefs/
             CanIf/CanIfPublicCfg/CanIfVersionInfoApi</DEFINITION-REF>
          <VALUE>TRUE</VALUE>
        </ECUC-NUMERICAL-PARAM-VALUE>
      </PARAMETER-VALUES>
    </ECUC-CONTAINER-VALUE>
  </CONTAINERS>
</ECUC-MODULE-CONFIGURATION-VALUES>
```

Listing 5.8: Example 1 for ecu configuration values

If the Integration Requirements of listing 5.7 are applied to the configuration values of listing 5.8, then the configuration values model of listing 5.9 results: the configuration values are extended with a value FALSE for CanIfDevErrorDetect.

```
<ECUC-MODULE-CONFIGURATION-VALUES>
  <SHORT-NAME>CanIf</SHORT-NAME>
  <DEFINITION-REF DEST="ECUC-MODULE-DEF">/AUTOSAR/EcucDefs/CanIf</
     DEFINITION-REF>
  <CONTAINERS>
    <ECUC-CONTAINER-VALUE>
      <SHORT-NAME>CanIfPublicCfg</SHORT-NAME>
      <DEFINITION-REF DEST="ECUC-CONTAINER-DEF">/AUTOSAR/EcucDefs/CanIf/
         CanIfPublicCfg</DEFINITION-REF>
      <PARAMETER-VALUES>
        <ECUC-NUMERICAL-PARAM-VALUE>
          <DEFINITION-REF DEST="ECUC-BOOLEAN-PARAM-DEF">/AUTOSAR/EcucDefs/
             CanIf/CanIfPublicCfg/CanIfVersionInfoApi</DEFINITION-REF>
          <VALUE>TRUE</VALUE>
        </ECUC-NUMERICAL-PARAM-VALUE>
        <!-- created through integration requirements -->
        <ECUC-NUMERICAL-PARAM-VALUE>
          <DEFINITION-REF DEST="ECUC-BOOLEAN-PARAM-DEF">/AUTOSAR/EcucDefs/
             CanIf/CanIfPublicCfg/CanIfDevErrorDetect</DEFINITION-REF>
          <VALUE>FALSE</VALUE>
        </ECUC-NUMERICAL-PARAM-VALUE>
      </PARAMETER-VALUES>
    </ECUC-CONTAINER-VALUE>
  </CONTAINERS>
```

```
</ECUC-MODULE-CONFIGURATION-VALUES>
```

Listing 5.9: Example 1 for ecu configuration values after applying integration requirements

Integration Requirements can also be applied to the adaptive platform function cluster configuration. This approach is identical to classic platform.

An example for the adaptive platform function cluster Integration Requirements is shown in listing 5.10, where the configuration parameter `PersistencyCentralStorageUri` is defined to have a `suggested` value of *"somePath"*.

```
<STRUCTURED-REQ>
  <SHORT-NAME>PersistencyCentralStorageUri</SHORT-NAME>
  <LONG-NAME>
    <L-4 L="EN">PersistencyCentralStorageUri</L-4>
  </LONG-NAME>
  <CATEGORY>REQUIREMENT_ITEM</CATEGORY>
  <DATE>2025-02-20</DATE>
  <ISSUED-BY>OEM</ISSUED-BY>
  <TYPE>valid</TYPE>
  <IMPORTANCE>suggested</IMPORTANCE>
  <DESCRIPTION>
    <LIST>
      <ITEM SI="Description">
        <P>
          <L-1 L="EN">Defines the location of the central storage of this
              Persistency instance as a URI.</L-1>
        </P>
      </ITEM>
      <ITEM SI="DefinitionReference">
        <P>
          <L-1 L="EN"><XREF><REFERRABLE-REF DEST="APMC-STRING-PARAM-DEF">/
              FunctionalClusterDefinitions/Persistency/PersistencyGeneral/
              PersistencyCentralStorageUri</REFERRABLE-REF></XREF></L-1>
        </P>
      </ITEM>
      <ITEM SI="SetValue">
        <P>
          <L-1 L="EN">somePath</L-1>
        </P>
      </ITEM>
      <ITEM SI="ARVersion">
        <P>
          <L-1 L="EN">R25-11</L-1>
        </P>
      </ITEM>
    </LIST>
  </DESCRIPTION>
  <RATIONALE>
    <P>
      <L-1 L="EN">OEM defines the central storage location explicitly for
          all projects.</L-1>
    </P>
  </RATIONALE>
  <USE-CASE>
```

```
    <P>
      <L-1 L="EN">IntegrationRequirement</L-1>
    </P>
  </USE-CASE>
</STRUCTURED-REQ>
```

Listing 5.10: Example for adaptive platform integration requirement using `Structured-Req`

5.2.2 Single configuration value in multiple containers

The example in listing 5.11 illustrates the definition of an Integration Requirements which may be applied several times: `CanIfRxPduDataLengthCheck` is a boolean configuration parameter defined as part of the configuration container `CanIfRxPduCfg`. `CanIfRxPduCfg` has an upper multiplicity greater than 1, thus many instances of `CanIfRxPduCfg` may exists in an ecu configuration values model.

```
<STRUCTURED-REQ>
  <SHORT-NAME>CanIfRxPduDataLengthCheck</SHORT-NAME>
  <LONG-NAME>
    <L-4 L="EN">CanIfRxPduDataLengthCheck</L-4>
  </LONG-NAME>
  <CATEGORY>REQUIREMENT_ITEM</CATEGORY>
  <DATE>2025-02-20</DATE>
  <ISSUED-BY>OEM</ISSUED-BY>
  <TYPE>valid</TYPE>
  <IMPORTANCE>mandatory</IMPORTANCE>
  <DESCRIPTION>
    <LIST>
      <ITEM SI="Description">
        <P>
          <L-1 L="EN">This parameter switches the message specific data
              length check. True: Data length check will be executed during
               the reception of this PDU. False: No data length check will
              be executed during the reception of this PDU.</L-1>
        </P>
      </ITEM>
      <ITEM SI="DefinitionReference">
        <P>
          <L-1 L="EN"><XREF S=""><REFERRABLE-REF DEST="ECUC-BOOLEAN-PARAM-
              DEF">/AUTOSAR/EcucDefs/CanIf/CanIfInitCfg/CanIfRxPduCfg/
              CanIfRxPduDataLengthCheck</REFERRABLE-REF></XREF></L-1>
        </P>
      </ITEM>
      <ITEM SI="SetValue">
        <P>
          <L-1 L="EN">FALSE</L-1>
        </P>
      </ITEM>
      <ITEM SI="SupportedValue">
        <P>
          <L-1 L="EN">TRUE</L-1>
        </P>
        <P>
```

```
          <L-1 L="EN">FALSE</L-1>
        </P>
      </ITEM>
      <ITEM SI="ARVersion">
        <P>
          <L-1 L="EN">R25-11</L-1>
        </P>
      </ITEM>
    </LIST>
  </DESCRIPTION>
  <RATIONALE>
    <P>
      <L-1 L="EN">OEM decision: DLC check shall always be disabled.</L-1>
    </P>
  </RATIONALE>
  <USE-CASE>
    <P>
      <L-1 L="EN">IntegrationRequirement</L-1>
    </P>
  </USE-CASE>
</STRUCTURED-REQ>
```

Listing 5.11: Example 2 for integration requirement for plural application

In the example configuration model of listing 5.12 2 instances of `CanIfRxPduCfg` are defined: *CanIfRxPduCfg17* and *CanIfRxPduCfg32*.

```
<ECUC-MODULE-CONFIGURATION-VALUES>
  <SHORT-NAME>CanIf</SHORT-NAME>
  <DEFINITION-REF DEST="ECUC-MODULE-DEF">/AUTOSAR/EcucDefs/CanIf</
     DEFINITION-REF>
  <CONTAINERS>
    <ECUC-CONTAINER-VALUE>
      <SHORT-NAME>CanIfInitCfg</SHORT-NAME>
      <DEFINITION-REF DEST="ECUC-CONTAINER-DEF">/AUTOSAR/EcucDefs/CanIf/
         CanIfInitCfg</DEFINITION-REF>
      <SUB-CONTAINERS>
        <ECUC-CONTAINER-VALUE>
          <SHORT-NAME>CanIfRxPduCfg17</SHORT-NAME>
          <DEFINITION-REF DEST="ECUC-CONTAINER-DEF">/AUTOSAR/EcucDefs/
             CanIf/CanIfInitCfg/CanIfRxPduCfg</DEFINITION-REF>
          <PARAMETER-VALUES>
            <ECUC-NUMERICAL-PARAM-VALUE>
              <DEFINITION-REF DEST="ECUC-INTEGER-PARAM-DEF">/AUTOSAR/
                 EcucDefs/CanIf/CanIfInitCfg/CanIfRxPduCfg/CanIfRxPduId</
                 DEFINITION-REF>
              <VALUE>17</VALUE>
            </ECUC-NUMERICAL-PARAM-VALUE>
          </PARAMETER-VALUES>
        </ECUC-CONTAINER-VALUE>
        <ECUC-CONTAINER-VALUE>
          <SHORT-NAME>CanIfRxPduCfg32</SHORT-NAME>
          <DEFINITION-REF DEST="ECUC-CONTAINER-DEF">/AUTOSAR/EcucDefs/
             CanIf/CanIfInitCfg/CanIfRxPduCfg</DEFINITION-REF>
          <PARAMETER-VALUES>
            <ECUC-NUMERICAL-PARAM-VALUE>
```

```
            <DEFINITION-REF DEST="ECUC-INTEGER-PARAM-DEF">/AUTOSAR/
               EcucDefs/CanIf/CanIfInitCfg/CanIfRxPduCfg/CanIfRxPduId</
               DEFINITION-REF>
            <VALUE>32</VALUE>
          </ECUC-NUMERICAL-PARAM-VALUE>
        </PARAMETER-VALUES>
      </ECUC-CONTAINER-VALUE>
    </SUB-CONTAINERS>
  </ECUC-CONTAINER-VALUE>
 </CONTAINERS>
</ECUC-MODULE-CONFIGURATION-VALUES>
```

Listing 5.12: Example 2 for ecu configuration values

If the Integration Requirements of listing 5.11 are applied to the configuration values of listing 5.12, then the configuration values model of listing 5.13 results: every configuration value container of definition `CanIfRxPduCfg` is extended with a value `FALSE` for `CanIfRxPduDataLengthCheck`.

```
<ECUC-MODULE-CONFIGURATION-VALUES>
  <SHORT-NAME>CanIf</SHORT-NAME>
  <DEFINITION-REF DEST="ECUC-MODULE-DEF">/AUTOSAR/EcucDefs/CanIf</
     DEFINITION-REF>
  <CONTAINERS>
    <ECUC-CONTAINER-VALUE>
      <SHORT-NAME>CanIfInitCfg</SHORT-NAME>
      <DEFINITION-REF DEST="ECUC-CONTAINER-DEF">/AUTOSAR/EcucDefs/CanIf/
         CanIfInitCfg</DEFINITION-REF>
      <SUB-CONTAINERS>
        <ECUC-CONTAINER-VALUE>
          <SHORT-NAME>CanIfRxPduCfg17</SHORT-NAME>
          <DEFINITION-REF DEST="ECUC-CONTAINER-DEF">/AUTOSAR/EcucDefs/
             CanIf/CanIfInitCfg/CanIfRxPduCfg</DEFINITION-REF>
          <PARAMETER-VALUES>
            <ECUC-NUMERICAL-PARAM-VALUE>
              <DEFINITION-REF DEST="ECUC-INTEGER-PARAM-DEF">/AUTOSAR/
                 EcucDefs/CanIf/CanIfInitCfg/CanIfRxPduCfg/CanIfRxPduId</
                 DEFINITION-REF>
              <VALUE>17</VALUE>
            </ECUC-NUMERICAL-PARAM-VALUE>
            <!-- created through integration requirements -->
            <ECUC-NUMERICAL-PARAM-VALUE>
              <DEFINITION-REF DEST="ECUC-BOOLEAN-PARAM-DEF">/AUTOSAR/
                 EcucDefs/CanIf/CanIfInitCfg/CanIfRxPduCfg/
                 CanIfRxPduDataLengthCheck</DEFINITION-REF>
              <VALUE>FALSE</VALUE>
            </ECUC-NUMERICAL-PARAM-VALUE>
          </PARAMETER-VALUES>
        </ECUC-CONTAINER-VALUE>
        <ECUC-CONTAINER-VALUE>
          <SHORT-NAME>CanIfRxPduCfg32</SHORT-NAME>
          <DEFINITION-REF DEST="ECUC-CONTAINER-DEF">/AUTOSAR/EcucDefs/
             CanIf/CanIfInitCfg/CanIfRxPduCfg</DEFINITION-REF>
          <PARAMETER-VALUES>
            <ECUC-NUMERICAL-PARAM-VALUE>
```

```
            <DEFINITION-REF DEST="ECUC-INTEGER-PARAM-DEF">/AUTOSAR/
                EcucDefs/CanIf/CanIfInitCfg/CanIfRxPduCfg/CanIfRxPduId</
                DEFINITION-REF>
            <VALUE>32</VALUE>
          </ECUC-NUMERICAL-PARAM-VALUE>
          <!-- created through integration requirements -->
          <ECUC-NUMERICAL-PARAM-VALUE>
            <DEFINITION-REF DEST="ECUC-BOOLEAN-PARAM-DEF">/AUTOSAR/
                EcucDefs/CanIf/CanIfInitCfg/CanIfRxPduCfg/
                CanIfRxPduDataLengthCheck</DEFINITION-REF>
            <VALUE>FALSE</VALUE>
          </ECUC-NUMERICAL-PARAM-VALUE>
        </PARAMETER-VALUES>
      </ECUC-CONTAINER-VALUE>
    </SUB-CONTAINERS>
  </ECUC-CONTAINER-VALUE>
 </CONTAINERS>
</ECUC-MODULE-CONFIGURATION-VALUES>
```

Listing 5.13: Example 2 for ecu configuration values after applying integration requirements

5.2.3 Multiple configuration values in a container

Another case is when the upper multiplicity in the parameter definition is bigger than one. In that case, the convention is that one configuration value is created. When checking, the integration requirement is applied to all existing configuration values.

The case of multiple values in one container and multiple values in several containers are not distinguished. Both cases are handled in the same way.

More sophisticated mechanisms to specify the values might be useful (e.g. creating a certain number of values following a given list or a pattern), are currently not defined by this document, but could be added in the future.

5.2.4 Tracing configuration changes

For debugging, analysis and conflict resolution, it is beneficial to store the fact that a parameter was created due to an integration requirement. This can be done by a configuration tool in a vendor specific manner.

This requires storing additional information together with the configuration values, which can increase file size and slow down tools. However, in many cases it's not that important to exactly know where a parameter came from, and it's enough to know which parameter matches to which integration requirement. To avoid storing the information, it alternatively can be deduced in the report stage. Any parameter that matches a mandatory integration requirement can be assumed to be caused by that requirement, and the same applies to any parameter matching a recommended one.

Note that AUTOSAR does not prescribe whether or not this information is stored and also does not specify a format.

If a tool vendor chooses to store this information in the model, it could be persisted in the `Annotation` element of the configuration value. As this is essentially unstructured data, a tool vendor would have to define the structure of the data. One possibility is to store the integration requirement shortName (or ShortName path) in the `label`, and store the type (recommended or mandatory) in the `annotationText`

This information can be used for conflict resolution between recommended and mandatory parameters - if there is a conflict between two parameter values, one matching (or created by) a mandatory integration requirement, and one matching (or created by) a suggested one, the recommended value can be deleted and the mandatory one kept.

5.3 Ranges in Integration Requirements

So far, this document covered cases where an integration requirement prescribes a single configuration value. There is also the use case to prescribe a range of allowed values. For example, a parameter can take the values 0-255, but in the context of a certain project only the values 10-26 make sense.

Another example is to restrict allowed literal values - the parameter definition allows FEATURE_A, FEATURE_B, FEATURE_C and FEATURE_D. If the integration requirement forbids FEATURE_A and FEATURE_D, this leaves only FEATURE_B and FEATURE_C as allowed values. It's also possible to specify a list of allowed values.

This shows that ranges can either be formulated as a range (or list) of allowed values, or as a range (or list) of forbidden values.

Please note that, due to their nature, integration requirements prescribing ranges cannot directly set a parameter to a fixed value. They can only be used to check a configuration model against the requirements.

5.4 Possible Conflicts caused by applied Integration Requirements

This section will describe the handling of possible conflicts resulting by applying Integration Requirements to parameter values. These can either result from conflicting Integration Requirements (one requires the value true, another one the value false), from a difference between an integration requirement and a value derived from the upstream mapping, or from a difference between an integration requirement and a manually configured value.

By definition, conflicts can only occur for mandatory Integration Requirements, and not for suggested ones, because violating a suggested integration requirement is not an error.

If two sources would lead to a conflicting configuration value, the recommended approach is to create both values and flag this as an error. The affected parties then have to agree on the correct value.

This document does not define a method for conflict resolution. Therefore, there are also no defined precedence rules and no suggested order of applying Integration Requirements. It is, however, assumed that the upstream mappings are processed first, since these can lead to the creation of configuration containers and values to which later integration requirements apply. It is also recommended to apply mandatory requirements before suggested ones, but this depends on the workflow and might not always be possible.

Any conflict resolution requires conflict detection. If the parameter definition has an upper multiplicity of 1, this means the resulting configuration model will violate the constraint and can easily be detected as erroneous. If the upper multiplicity is greater than 1, the conflicting values will coexist in the model and cannot be detected by checking the configuration model against the parameter definitions. However, since an integration requirement applies to all instances of a parameter (see chapter 5.2.3), and any non-range integration requirement leads to a configuration value being created, the conflict can be detected by checking the created configuration model against all Integration Requirements. It will violate both sets of Integration Requirements.

5.4.1 Merge of Integration Requirements

As mentioned, this document does not define a method for conflict resolution. It also currently does not define a merging mechanism. The assumption is that Integration Requirements from several sources are applied independently from each other in any order. Nevertheless, this chapter contains a short explanation about merging, and especially the case of ranges.

5.4.1.1 No merge of fixed values

Because Integration Requirements for fixed values always create values, this means that the Integration Requirements themselves do not have to be merged. Instead, merging is only done in the configuration model, in the same way how split configurations are merged. The resulting configuration then has to be checked against the original parameter definitions as well as against all Integration Requirements.

5.4.1.2 Merge of ranges

In the case of ranges, a similar idea applies: The resulting configuration is checked against all Integration Requirements. This will also show if there are ranges that conflict. For example, if integration_requirement_1 prescribes a range of 21-44 for param_a, integration_requirement_2 prescribes a range of 10-12 for the same

param_a, then these ranges conflict, and the resulting configuration will always violate one of the requirements.

The result is the same as merging the ranges, and then checking against the merged range. However, it would be very beneficial if a tool checks ranges for conflicts, since it is otherwise left to the integrator to find the conflict. This could lead to a lot of trial and error (set param_a to 11, get a violation of integration_requirement_1, set it to 32, get a violation of integration_requirement_2).

Finding conflicts between Integration Requirements specifying ranges are created if the sets of values that the requirements specify have no overlap at all. For a pair of Integration Requirements specifying the included range, the following cases (for an excluded range, the conditions are reversed; for more than two Integration Requirements, the rules are applied to a pair, then the pair is merged into one rule, the next rule is applied to the merged rule, etc.):

- the two ranges are identical - the sets of allowed values are the same. In case of number ranges: the lower bound of one range equals the lower bound of the other range AND the upper bound of one range equals the upper bound of the other range (trivial case)
- one range lies fully within another one - the sets of allowed values have some overlap. In case of numbers: the lower bound of one range is larger than or equal to the lower bound of the other range AND the upper bound of one range is smaller than or equal to the upper bound of the other range
- one range lies partially within another one - the sets of allowed values have some overlap. In case of numbers: the lower bound of one range is larger than or equal to the lower bound of the other range OR the upper bound of one range is smaller than or equal to the upper bound of the other range
- the ranges are disjunct - the sets of allowed values have no overlap. In case of numbers: the lower bound of one range is larger than the upper bound of the other range OR the upper bound of one range is smaller than the lower bound of the other range

In the first case, one integration requirement is superfluous. In the second and third case, the combined range consists of the set of values allowed by both requirements (in case of numbers from maximum of the lower bounds to the minimum of the upper bounds). The last case creates a conflict, since the set of values allowed by both ranges is empty. This can be detected by a tool and reported.

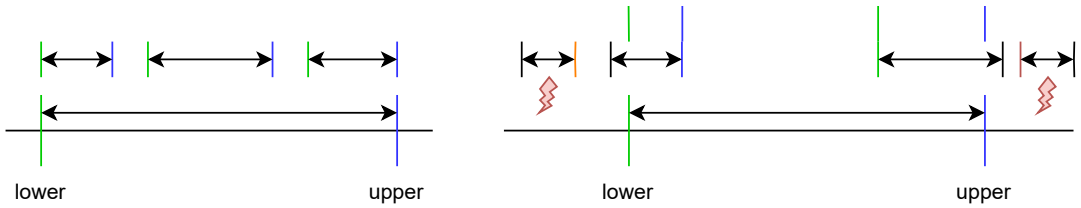


Figure 5.2: Compatible and incompatible ranges. Left side: The green bars show the case "lower bound of one range is larger than or equal to the lower bound of the other range". Blue bars show "the upper bound of one range is smaller than or equal to the upper bound of the other range". Right side: partial overlaps result in a combined range that covers both lower and upper bounds (top row, green and blue bars). Orange bar shows "the higher bound of one range is smaller than the lower bound of the other range", red bar shows "the upper bound of one range is larger than the lower bound of the other range".

5.4.1.3 Order dependency between suggested and mandatory integration requirements

As mentioned in the introduction, mandatory integration requirements will always create a parameter, while suggested integration requirements will only create a parameter if it does not exist.

If integration requirements are applied at various times in the workflow, this will lead to different results, depending on the order of application (it introduces an order dependency). This is problematic if the mandatory value and the suggested value differ. If the mandatory integration requirement is applied first, the recommended one will not create a value (since one already exists), the resulting configuration contains one value (the mandatory one). Conversely, if the recommended one is applied first, the resulting configuration will contain two values (recommended and mandatory).

If the source of configuration is stored (see 5.2.4), then this conflict can be detected and resolved immediately. Otherwise, it can be detected during reporting, when the configuration values are checked against the Integration Requirements.

5.5 Reporting

In the motivation of the Integration Requirements the reporting aspect is already mentioned. The reporting shall base on a formally standardized way to ensure consistent data comparison with the original configuration parameters derived from Upstream descriptions (*t1* in figure 5.3) and the Integration Requirements (*t2* in figure 5.3).

The integrator changes or adds further configuration elements. The change information (change and reason) will be added to affected elements, means on parameter value level. This take place in *t2* in figure 5.3.

In a subsequent step the finalized Target Configuration is compared with values derived from Upstream descriptions and the mandatory Integration Requirements values. This takes place in *t3* in figure 5.3 on parameter value level too.

All agglomerated change information available up to that point on parameter value level will be transferred to design level (System Extract, Diagnostic Extract, etc.). This step is essential to enable an automatic processing of the changes and deviations from the original data. From a technical point of view the feedback loop could also be realized by using the `StructuredReq` class to compose the report.

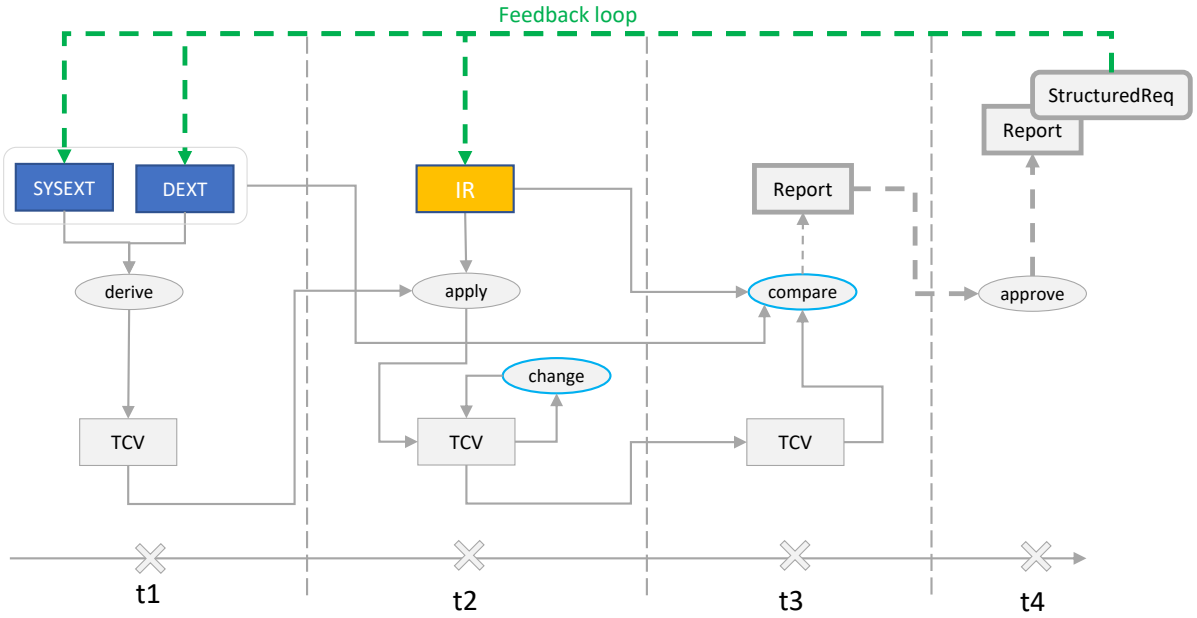


Figure 5.3: Feedback loop of change information

A Mentioned Class Tables

For the sake of completeness, this chapter contains a set of class tables representing meta-classes mentioned in the context of this document but which are not contained directly in the scope of describing specific meta-model semantics.

Class	***ARElement*** (abstract)			
Note	An element that can be defined stand-alone, i.e. without being part of another element (except for packages of course).			
Base	*ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Subclasses	*AbstractMessageToSecureChannelMapping*, AclObjectSet, AclOperation, AclPermission, AclRole, AliasNameSet, ApplicabilityInfoSet, ApplicationPartition, *AutosarDataType*, *BaseType*, BlueprintMappingSet, BswEntryRelationshipSet, BswModuleDescription, BswModuleEntry, BuildActionManifest, CalibrationParameterValueSet, ClientIdDefinitionSet, Collection, CompuMethod, ConsistencyNeedsBlueprintSet, ConstantSpecification, ConstantSpecificationMappingSet, CpSoftwareCluster, CpSoftwareClusterBinaryManifestDescriptor, CpSoftwareClusterMappingSet, CpSoftwareClusterResourcePool, CryptoEllipticCurveProps, CryptoServiceCertificate, CryptoServiceKey, CryptoServicePrimitive, CryptoServiceQueue, CryptoSignatureScheme, DataConstr, DataTransformationSet, DataTypeMappingSet, DdsCpConfig, *DiagnosticCommonElement*, DiagnosticConnection, DiagnosticContributionSet, DltArgumentPropsSet, DltContext, DltEcu, Documentation, E2EProfileCompatibilityProps, EcucDefinitionCollection, EcucDestinationUriDefSet, EcucModuleConfigurationValues, EcucModuleDef, EcucValueCollection, EthIpProps, EthTcpIpIcmpProps, EthTcpIpProps, EvaluatedVariantSet, FMFeature, FMFeatureMap, FMFeatureModel, FMFeatureSelectionSet, FirewallRule, FlatMap, GeneralPurposeConnection, HwCategory, HwElement, HwType, *IEEE1722TpConnection*, IPSecConfigProps, IPv6ExtHeaderFilterSet, *IdsCommonElement*, IdsDesign, *Implementation*, ImpositionTimeDefinitionGroup, InterpolationRoutineMappingSet, J1939ControllerApplication, KeywordSet, LifeCycleInfoSet, LifeCycleStateDefinitionGroup, LogAndTraceMessageCollectionSet, MacSecGlobalKayProps, MacSecParticipantSet, McFunction, McGroup, ModeDeclarationGroup, ModeDeclarationMappingSet, OsTaskProxy, PhysicalDimension, PhysicalDimensionMappingSet, *PortInterface*, PortInterfaceMappingSet, PortPrototypeBlueprint, PostBuildVariantCriterion, PostBuildVariantCriterionValueSet, PredefinedVariant, RapidPrototypingScenario, SdgDef, *SecureComProps*, *SecurityConnectivityAssociation*, SignalServiceTranslationPropsSet, SomeipSdClientEventGroupTimingConfig, SomeipSdClientServiceInstanceConfig, SomeipSdServerEventGroupTimingConfig, SomeipSdServerServiceInstanceConfig, SwAddrMethod, SwAxisType, SwComponentMappingConstraints, *SwComponentType*, SwRecordLayout, SwSystemconst, SwSystemconstantValueSet, SwcBswMapping, System, SystemComSpecDefinitionSet, SystemSignal, SystemSignalGroup, TDCpSoftwareClusterMappingSet, TcpOptionFilterSet, *TimingExtension*, TlsConnectionGroup, TlvDataIdDefinitionSet, TransformationPropsSet, Unit, UnitGroup, *UploadablePackageElement*, ViewMapSet			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table A.1: ARElement

Class	**ARPackage**			
Note	AUTOSAR package, allowing to create top level packages to structure the contained ARElements. ARPackages are open sets. This means that in a file based description system multiple files can be used to partially describe the contents of a package. This is an extended version of MSR's SW-SYSTEM.			
Base	*ARObject*, *AtpBlueprint*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	ARPackage.arPackage, AUTOSAR.arPackage			
Attribute	***Type***	***Mult.***	***Kind***	***Note***

▽

△

Class	**ARPackage**			
arPackage	ARPackage	*	aggr	This represents a sub package within an ARPackage, thus allowing for an unlimited package hierarchy. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=arPackage, arPackage.shortName, ar Package.variationPoint.shortLabel vh.latestBindingTime=blueprintDerivationTime xml.sequenceOffset=30
element	PackageableElement	*	aggr	Elements that are part of this package **Stereotypes:** atpVariation **Tags:** atp.Splitkey=element, element.shortName, element.variationPoint.shortLabel vh.latestBindingTime=systemDesignTime xml.sequenceOffset=20
referenceBase	ReferenceBase	*	aggr	This denotes the reference bases for the package. This is the basis for all relative references within the package. The base needs to be selected according to the base attribute within the references. **Tags:** atp.Splitkey=referenceBase, referenceBase.shortLabel xml.sequenceOffset=10

Table A.2: ARPackage

Class	**AliasNameSet**			
Note	This meta-class represents a set of AliasNames. The AliasNameSet can for example be an input to the A2L-Generator. **Tags:** atp.recommendedPackage=AliasNameSets			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
aliasName	AliasNameAssignment	*	aggr	AliasNames contained in the AliasNameSet. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=aliasName, aliasName.shortLabel, alias Name.variationPoint.shortLabel vh.latestBindingTime=preCompileTime

Table A.3: AliasNameSet

Class	**Annotation**			
Note	This is a plain annotation which does not have further formal data.			
Base	*ARObject*, *GeneralAnnotation*			
Aggregated by	*EcucAbstractReferenceValue*.annotation, *EcucParameterValue*.annotation, HwAttributeValue.annotation, *Identifiable*.annotation, PostBuildVariantCriterionValue.annotation, SwDataDefProps.annotation, Sw SystemconstValue.annotation			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
–	–	–	–	–

Table A.4: Annotation

Class	**AnyInstanceRef**			
Note	Describes a reference to any instance in an AUTOSAR model. This is the most generic form of an instance ref. Refer to the superclass notes for more details.			
Base	*ARObject*, *AtpInstanceRef*			
Aggregated by	ApmcInstanceReferenceValue.value, ApmcUpstreamDocInstanceReferenceValue.value, ApmcUri InstanceReferenceValue.value, Collection.collectedInstance, Collection.sourceInstance, Documentation Context.feature, EcucInstanceReferenceValue.value, FlatInstanceDescriptor.ecuExtractReference, Flat InstanceDescriptor.upstreamReference, RptContainer.byPassPoint, RptHook.rptArHook, SecurityEvent ReportInstanceValue.object, ViewMap.firstElementInstance, ViewMap.secondElementInstance			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
base	AtpClassifier	0..1	ref	This is the base from which navigation path begins. **Stereotypes:** atpDerived
contextElement (ordered)	AtpPrototype	*	ref	This is one step in the navigation path specified by the instance ref.
target	AtpFeature	0..1	ref	This is the target of the instance ref.

Table A.5: AnyInstanceRef

Class	**ApplicationArrayDataType**			
Note	An application data type which is an array, each element is of the same application data type. **Tags:** atp.recommendedPackage=ApplicationDataTypes			
Base	*ARElement*, *ARObject*, *ApplicationCompositeDataType*, *ApplicationDataType*, *AtpBlueprint*, *Atp Classifier*, *AtpType*, *AutosarDataType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
dynamicArray SizeProfile	String	0..1	attr	Specifies the profile which the array will follow if it is a variable size array.
element	ApplicationArray Element	0..1	aggr	This association implements the concept of an array element. That is, in some cases it is necessary to be able to identify single array elements, e.g. as input values for an interpolation routine. **Tags:** atp.Splitkey=element, element.shortName

Table A.6: ApplicationArrayDataType

Class	**ApplicationArrayElement**			
Note	Describes the properties of the elements of an application array data type.			
Base	*ARObject*, *ApplicationCompositeElementDataPrototype*, *AtpFeature*, *AtpPrototype*, *DataPrototype*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	ApplicationArrayDataType.element, *AtpClassifier*.atpFeature			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
arraySize Handling	ArraySizeHandling Enum	0..1	attr	The way how the size of the array is handled.
arraySize Semantics	ArraySizeSemantics Enum	0..1	attr	This attribute controls how the information about the array size shall be interpreted.
indexDataType	ApplicationPrimitive DataType	0..1	ref	This reference can be taken to assign a `CompuMethod` of category `TEXTTABLE` to the array. The texttable entries associate a textual value to an index number such that the element with that index number is represented by a symbolic name. **Tags:** atp.Splitkey=indexDataType

▽

△

Class	**ApplicationArrayElement**			
maxNumberOf Elements	PositiveInteger	0..1	attr	The maximum number of elements that the array can contain. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime

Table A.7: ApplicationArrayElement

Class	***ApplicationCompositeDataType*** (abstract)			
Note	Abstract base class for all application data types composed of other data types.			
Base	*ARElement*, *ARObject*, *ApplicationDataType*, *AtpBlueprint*, *AtpClassifier*, *AtpType*, *AutosarDataType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Subclasses	ApplicationArrayDataType, ApplicationRecordDataType			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table A.8: ApplicationCompositeDataType

Class	***ApplicationCompositeElementDataPrototype*** (abstract)			
Note	This class represents a data prototype which is aggregated within a composite application data type (record or array). It is introduced to provide a better distinction between target and context in instance Refs.			
Base	*ARObject*, *AtpFeature*, *AtpPrototype*, *DataPrototype*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	ApplicationArrayElement, ApplicationRecordElement			
Aggregated by	*AtpClassifier*.atpFeature			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
type	ApplicationDataType	0..1	tref	This represents the corresponding data type. **Stereotypes:** isOfType **Tags:** atp.Splitkey=type

Table A.9: ApplicationCompositeElementDataPrototype

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

Table A.10: ApplicationDataType

Class	ApplicationPrimitiveDataType			
Note	A primitive data type defines a set of allowed values. **Tags:** atp.recommendedPackage=ApplicationDataTypes			
Base	*ARElement*, *ARObject*, *ApplicationDataType*, *AtpBlueprint*, *AtpClassifier*, *AtpType*, *AutosarDataType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table A.11: ApplicationPrimitiveDataType

Class	ApplicationRecordDataType			
Note	An application data type which can be decomposed into prototypes of other application data types. **Tags:** atp.recommendedPackage=ApplicationDataTypes			
Base	*ARElement*, *ARObject*, *ApplicationCompositeDataType*, *ApplicationDataType*, *AtpBlueprint*, *Atp Classifier*, *AtpType*, *AutosarDataType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
element (ordered)	ApplicationRecord Element	*	aggr	Specifies an element of a record. The aggregation of `ApplicationRecordElement` is subject to variability with the purpose to support the conditional existence of elements inside a `ApplicationRecordData` Type. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime

Table A.12: ApplicationRecordDataType

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

Table A.13: ApplicationRecordElement

Class	ApplicationSwComponentType
Note	The `ApplicationSwComponentType` is used to represent the application software. **Tags:** atp.recommendedPackage=SwComponentTypes
Base	*ARElement*, *ARObject*, *AtomicSwComponentType*, *AtpBlueprint*, *AtpClassifier*, *AtpType*, *Collectable Element*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *SwComponentType*

▽

△

Class	**ApplicationSwComponentType**			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table A.14: ApplicationSwComponentType

Enumeration	**ArraySizeSemanticsEnum**
Note	This enumeration provides options for how the number of elements in an array object shall be interpreted.
Aggregated by	ApplicationArrayElement.arraySizeSemantics, DiagnosticDataElement.arraySizeSemantics, ImplementationDataTypeElement.arraySizeSemantics, SwTextProps.arraySizeSemantics
Literal	***Description***
fixedSize	This means that the array object will always have a fixed number of elements. **Tags:** atp.EnumerationLiteralIndex=0
variableSize	This implies that the actual number of elements in the array object might vary at run-time. The configured size represents the maximum number of elements in the array object. **Tags:** atp.EnumerationLiteralIndex=1

Table A.15: ArraySizeSemanticsEnum

Class	**AssemblySwConnector**			
Note	`AssemblySwConnector`s are exclusively used to connect `SwComponentPrototype`s in the context of a `CompositionSwComponentType`.			
Base	*ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *SwConnector*			
Aggregated by	*AtpClassifier*.atpFeature, CompositionSwComponentType.connector			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
provider	AbstractProvidedPort Prototype	0..1	iref	Instance of providing port. **Tags:** atp.Splitkey=provider, provider.contextComponent, provider.targetPPort **InstanceRef implemented by:** PPortInComposition InstanceRef
requester	AbstractRequiredPort Prototype	0..1	iref	Instance of requiring port. **Tags:** atp.Splitkey=requester, requester.context Component, requester.targetRPort **InstanceRef implemented by:** RPortInComposition InstanceRef

Table A.16: AssemblySwConnector

Class	***AtomicSwComponentType*** (abstract)
Note	An atomic software component is atomic in the sense that it cannot be further decomposed and distributed across multiple ECUs.
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *SwComponentType*
Subclasses	ApplicationSwComponentType, ComplexDeviceDriverSwComponentType, EcuAbstractionSwComponent Type, NvBlockSwComponentType, SensorActuatorSwComponentType, ServiceProxySwComponent Type, ServiceSwComponentType
Aggregated by	ARPackage.element

▽

△

Class	***AtomicSwComponentType*** (abstract)			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
internalBehavior	SwcInternalBehavior	0..1	aggr	The `SwcInternalBehavior`s owned by an `AtomicSwComponentType` can be located in a different physical file. Therefore the aggregation is <<atp Splitable>>. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=internalBehavior, internalBehavior.shortName, internalBehavior.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
symbolProps	SymbolProps	0..1	aggr	This represents the `SymbolProps` for the `AtomicSwComponentType`. **Tags:** atp.Splitkey=symbolProps, symbolProps.shortName

Table A.17: AtomicSwComponentType

Class	***AutosarDataPrototype*** (abstract)			
Note	Base class for prototypical roles of an `AutosarDataType`.			
Base	*ARObject*, *AtpFeature*, *AtpPrototype*, *DataPrototype*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	ArgumentDataPrototype, ParameterDataPrototype, VariableDataPrototype			
Aggregated by	*AtpClassifier*.atpFeature			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
type	AutosarDataType	0..1	tref	This represents the corresponding data type. **Stereotypes:** isOfType **Tags:** atp.Splitkey=type

Table A.18: AutosarDataPrototype

Class	**CompositionSwComponentType**			
Note	A `CompositionSwComponentType` aggregates `SwComponentPrototype`s (that in turn are typed by `SwComponentType`)s as well as `SwConnector`s for primarily connecting `SwComponentPrototype`s among each others and towards the surface of the `CompositionSwComponentType`. By this means, a hierarchical structures of software-components can be created. **Tags:** atp.recommendedPackage=SwComponentTypes			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *SwComponentType*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***

▽

△

Class	**CompositionSwComponentType**			
component	SwComponent Prototype	*	aggr	The instantiated components that are part of this composition. The aggregation of SwComponentPrototype is subject to variability with the purpose to support the conditional existence of a SwComponentPrototype. Please be aware: if the conditional existence of SwComponentPrototypes is resolved post-build, the deselected SwComponentPrototypes are still contained in the ECUs build but the instances are inactive in that they are not scheduled by the RTE. The use case for having 0 components owned by the CompositionSwComponentType could be to deliver an empty CompositionSwComponentType to e.g. a supplier for filling the internal structure. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=component, component.shortName, component.variationPoint.shortLabel vh.latestBindingTime=postBuild
connector	SwConnector	*	aggr	SwConnectors have the principal ability to establish a connection among PortPrototypes. They can have many roles in the context of a CompositionSwComponentType. Details are refined by subclasses. The aggregation of SwConnectors is subject to variability with the purpose to support variant data flow. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=connector, connector.shortName, connector.variationPoint.shortLabel vh.latestBindingTime=postBuild
constantValue Mapping	ConstantSpecification MappingSet	*	ref	Reference to the ConstantSpecificationMapping to be applied for initValues of PPortComSpecs and RPortComSpec. **Tags:** atp.Splitkey=constantValueMapping
dataType Mapping	DataTypeMappingSet	*	ref	Reference to the DataTypeMappingSet to be applied for the used ApplicationDataTypes in PortInterfaces. Background: when developing subsystems it may happen that ApplicationDataTypes are used on the surface of CompositionSwComponentTypes. In this case it would be reasonable to be able to also provide the intended mapping to the ImplementationDataTypes. However, this mapping shall be informal and not technically binding for the implementors mainly because the RTE generator is not concerned about the CompositionSwComponentTypes. Rationale: if the mapping of ApplicationDataTypes on the delegated and inner PortPrototype matches then the mapping to ImplementationDataTypes is not impacting compatibility. **Tags:** atp.Splitkey=dataTypeMapping

▽

△

Class	**CompositionSwComponentType**			
instantiation RTEEventProps	InstantiationRTEEvent Props	*	aggr	This allows to define instantiation specific properties for RTE Events, in particular for instance specific scheduling. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=instantiationRTEEventProps, instantiation RTEEventProps.shortLabel, instantiationRTEEvent Props.variationPoint.shortLabel vh.latestBindingTime=codeGenerationTime This Attribute is only used by the AUTOSAR Classic Platform.
physical Dimension Mapping	PhysicalDimension MappingSet	0..1	ref	This reference identifies the `PhysicalDimensionMappingSet` that is applicable in the context of the enclosing `CompositionSwComponentType`. The `PhysicalDimensionMapping`s contained in the `PhysicalDimensionMappingSet` shall be taken into account for the assessment of the compatibility of `PhysicalDimension`s in the context of creation of a `PortInterfaceMapping` in the scope of the `CompositionSwComponentType`. **Tags:** atp.Splitkey=physicalDimensionMapping

Table A.19: CompositionSwComponentType

Class	**CompuConstTextContent**			
Note	This meta-class represents the textual content of a scale.			
Base	*ARObject*, *CompuConstContent*			
Aggregated by	CompuConst.compuConstContentType			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
vt	VerbatimString	0..1	attr	This represents a textual constant in the computation method.

Table A.20: CompuConstTextContent

Class	**CompuMethod**			
Note	This meta-class represents the ability to express the relationship between a physical value and the mathematical representation. Note that this is still independent of the technical implementation in data types. It only specifies the formula how the internal value corresponds to its physical pendant. **Tags:** atp.recommendedPackage=CompuMethods			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
compuInternal ToPhys	Compu	0..1	aggr	This specifies the computation from internal values to physical values. **Tags:** atp.Splitkey=compuInternalToPhys xml.sequenceOffset=80

▽

△

Class	**CompuMethod**			
compuPhysTo Internal	Compu	0..1	aggr	This represents the computation from physical values to the internal values. **Tags:** atp.Splitkey=compuPhysToInternal xml.sequenceOffset=90
displayFormat	DisplayFormatString	0..1	attr	This property specifies, how the physical value shall be displayed e.g. in documents or measurement and calibration tools. **Tags:** xml.sequenceOffset=20
unit	Unit	0..1	ref	This is the physical unit of the Physical values for which the CompuMethod applies. **Tags:** atp.Splitkey=unit xml.sequenceOffset=30

Table A.21: CompuMethod

Class	**CompuRationalCoeffs**			
Note	This meta-class represents the ability to express a rational function by specifying the coefficients of nominator and denominator.			
Base	*ARObject*			
Aggregated by	CompuScaleRationalFormula.compuRationalCoeffs			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
compu Denominator	CompuNominator Denominator	0..1	aggr	This is the denominator of the expression. **Tags:** xml.sequenceOffset=30
compu Numerator	CompuNominator Denominator	0..1	aggr	This is the numerator of the rational expression. **Tags:** xml.sequenceOffset=20

Table A.22: CompuRationalCoeffs

Class	**CompuScale**			
Note	This meta-class represents the ability to specify one segment of a segmented computation method.			
Base	*ARObject*			
Aggregated by	CompuScales.compuScale			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
a2lDisplayText	String	0..1	attr	The value of this attribute shall be taken for generating one display text (specifically the OutVal) within the equivalent of the enclosing `CompuMethod` in A2L.
compuInverse Value	CompuConst	0..1	aggr	This is the inverse value of the constraint. This supports the case that the scale is not reversible per se. **Tags:** atp.Splitkey=compuInverseValue xml.sequenceOffset=60
compuScale Contents	CompuScaleContents	0..1	aggr	This represents the computation details of the scale. **Tags:** xml.roleElement=false xml.roleWrapperElement=false xml.sequenceOffset=70 xml.typeElement=false xml.typeWrapperElement=false

▽

△

Class	**CompuScale**			
desc	MultiLanguageOverview Paragraph	0..1	aggr	<desc> represents a general but brief description of the object in question. **Tags:** atp.Splitkey=desc xml.sequenceOffset=30
lowerLimit	Limit	0..1	attr	This specifies the lower limit of the scale. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=40
mask	PositiveUnlimitedInteger	0..1	attr	In difference to all the other computational methods every COMPU-SCALE will be applied including the bit MASK. Therefore it is allowed for this type of COMPU-METHOD, that COMPU-SCALES 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 COMPU-SCALE elements. **Tags:** xml.sequenceOffset=35
shortLabel	Identifier	0..1	attr	This element specifies a short name for the particular scale. The name can for example be used to derive a programming language identifier. **Tags:** xml.sequenceOffset=20
symbol	CIdentifier	0..1	attr	The symbol, if provided, is used by code generators to get a C identifier for the CompuScale. The name will be used as is for the code generation, therefore it needs to be unique within the generation context. **Tags:** xml.sequenceOffset=25
upperLimit	Limit	0..1	attr	This specifies the upper limit of a of the scale. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=50

Table A.23: CompuScale

Class	**DataConstr**			
Note	This meta-class represents the ability to specify constraints on data. **Tags:** atp.recommendedPackage=DataConstrs			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
dataConstrRule	DataConstrRule	*	aggr	This is one particular rule within the data constraints. **Tags:** atp.Splitkey=dataConstrRule xml.roleElement=true xml.roleWrapperElement=true xml.sequenceOffset=30 xml.typeElement=false xml.typeWrapperElement=false

Table A.24: DataConstr

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

Table A.25: DataConstrRule

Class	***DataPrototype*** (abstract)			
Note	Base class for prototypical roles of any data type.			
Base	*ARObject*, *AtpFeature*, *AtpPrototype*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	*ApplicationCompositeElementDataPrototype*, *AutosarDataPrototype*			
Aggregated by	*AtpClassifier*.atpFeature			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
swDataDef Props	SwDataDefProps	0..1	aggr	This property allows to specify data definition properties which apply on data prototype level. **Tags:** atp.Splitkey=swDataDefProps

Table A.26: DataPrototype

Class	**DataTypeMap**			
Note	This class represents the relationship between `ApplicationDataType` and its implementing `AbstractImplementationDataType`.			
Base	*ARObject*			
Aggregated by	DataTypeMappingSet.dataTypeMap			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
applicationData Type	ApplicationDataType	0..1	ref	This is the corresponding `ApplicationDataType` **Tags:** atp.Splitkey=applicationDataType
implementation DataType	AbstractImplementation DataType	0..1	ref	This is the corresponding `AbstractImplementationDataType`. **Tags:** atp.Splitkey=implementationDataType

Table A.27: DataTypeMap

Class	**DataTypeMappingSet**			
Note	This class represents a list of mappings between `ApplicationDataType`s and `ImplementationDataType`s. In addition, it can contain mappings between `ImplementationDataType`s and `ModeDeclarationGroup`s. **Tags:** atp.recommendedPackage=DataTypeMappingSets			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
dataTypeMap	DataTypeMap	*	aggr	This is one particular association between an `ApplicationDataType` and its `AbstractImplementationDataType`. **Tags:** atp.Splitkey=dataTypeMap
modeRequest TypeMap	ModeRequestTypeMap	*	aggr	This is one particular association between an `ModeDeclarationGroup` and its `AbstractImplementationDataType`. **Tags:** atp.Splitkey=modeRequestTypeMap

Table A.28: DataTypeMappingSet

Class	**DiagnosticAccessPermission**			
Note	This represents the specification of whether a given service can be accessed according to the existence of meta-classes referenced by a particular DiagnosticAccessPermission. In other words, this meta-class acts as a mapping element between several (otherwise unrelated) pieces of information that are put into context for the purpose of checking for access rights. **Tags:** atp.recommendedPackage=DiagnosticAccessPermissions			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *Identifiable*, *Multilanguage Referrable*, *PackageableElement*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
authentication Enabled	DiagnosticAuthRole Proxy	0..1	aggr	The existence of this aggregation indicates that an authentication is foreseen. The details are clarified by the aggregated class. **Tags:** atp.Splitkey=authenticationEnabled
diagnostic Session	DiagnosticSession	*	ref	This represents the associated DiagnosticSessions **Tags:** atp.Splitkey=diagnosticSession
environmental Condition	Diagnostic EnvironmentalCondition	0..1	ref	This represents the environmental conditions associated with the access permission. **Tags:** atp.Splitkey=environmentalCondition
securityLevel	DiagnosticSecurityLevel	*	ref	This represents the associated DiagnosticSecurityLevels **Tags:** atp.Splitkey=securityLevel

Table A.29: DiagnosticAccessPermission

Class	**EcuInstance**
Note	ECUInstances are used to define the ECUs used in the topology. The type of the ECU is defined by a reference to an ECU specified with the ECU resource description. **Tags:** atp.recommendedPackage=EcuInstances
Base	*ARObject*, *CollectableElement*, *FibexElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*
Aggregated by	ARPackage.element



△

Class	**EcuInstance**			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
associatedComIPduGroup	ISignalIPduGroup	*	ref	With this reference it is possible to identify which ISignalIPduGroups are applicable for which Communication Connector/ ECU. Only top level ISignalIPduGroups shall be referenced by an EcuInstance. If an ISignalIPduGroup contains other ISignalIPduGroups than these contained ISignalIPduGroups shall not be referenced by the EcuInstance. Contained ISignalIPduGroups are associated to an Ecu Instance via the top level ISignalIPduGroup. **Tags:** atp.Splitkey=associatedComIPduGroup This Attribute is only used by the AUTOSAR Classic Platform.
associatedConsumedProvidedServiceInstanceGroup	ConsumedProvidedServiceInstanceGroup	*	ref	With this reference it is possible to identify which ConsumedProvidedServiceInstanceGroups are applicable for which ECUInstance. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=associatedConsumedProvidedServiceInstanceGroup, associatedConsumedProvidedServiceInstanceGroup.consumedProvidedServiceInstanceGroup, associatedConsumedProvidedServiceInstanceGroup.variationPoint.shortLabel vh.latestBindingTime=postBuild
associatedPdurIPduGroup	PdurIPduGroup	*	ref	With this reference it is possible to identify which PduR IPdu Groups are applicable for which Communication Connector/ ECU. **Tags:** atp.Splitkey=associatedPdurIPduGroup
channelSynchronousWakeup	Boolean	0..1	attr	If this parameter is available and set to true, then all available channels will be woken up as soon as at least one channel wakeup occurs. If PNCs are configured, then all PNCs will be requested upon a channel wakeup.
clientIdRange	ClientIdRange	0..1	aggr	Restriction of the Client Identifier for this Ecu to an allowed range of numerical values. The Client Identifier of the transaction handle is generated by the client RTE for inter-Ecu Client/Server communication. **Tags:** atp.Splitkey=clientIdRange
comConfigurationGwTimeBase	TimeValue	0..1	attr	The period between successive calls to Com_MainFunctionRouteSignals of the AUTOSAR COM module in seconds. This Attribute is only used by the AUTOSAR Classic Platform.
comConfigurationRxTimeBase	TimeValue	0..1	attr	The period between successive calls to Com_MainFunctionRx of the AUTOSAR COM module in seconds. This Attribute is only used by the AUTOSAR Classic Platform.
comConfigurationTxTimeBase	TimeValue	0..1	attr	The period between successive calls to Com_MainFunctionTx of the AUTOSAR COM module in seconds. This Attribute is only used by the AUTOSAR Classic Platform.

▽

△

Class	**EcuInstance**			
comEnable MDTForCyclic Transmission	Boolean	0..1	attr	Enables for the Com module of this EcuInstance the minimum delay time monitoring for cyclic and repeated transmissions (TransmissionModeTiming has cyclic Timing assigned or eventControlledTiming with numberOf Repetitions > 0). This Attribute is only used by the AUTOSAR Classic Platform.
commController	Communication Controller	*	aggr	CommunicationControllers of the ECU. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=commController, commController.short Name, commController.variationPoint.shortLabel vh.latestBindingTime=postBuild
connector	Communication Connector	*	aggr	All channels controlled by a single controller. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=connector, connector.shortName, connector.variationPoint.shortLabel vh.latestBindingTime=postBuild
dltConfig	DltConfig	0..1	aggr	Describes the Dlt configuration on this EcuInstance. **Tags:** atp.Splitkey=dltConfig This Attribute is only used by the AUTOSAR Classic Platform.
doIpConfig	DoIpConfig	0..1	aggr	DoIp configuration on this EcuInstance. **Tags:** atp.Splitkey=doIpConfig atp.Status=draft This Attribute is only used by the AUTOSAR Classic Platform.
ecuTaskProxy	OsTaskProxy	*	ref	Reference to OsTaskProxies assigned to the Ecu Instance. **Tags:** atp.Splitkey=ecuTaskProxy This Attribute is only used by the AUTOSAR Classic Platform.
ethSwitchPort Group Derivation	Boolean	0..1	attr	Defines whether the derivation of SwitchPortGroups based on VLAN and/or CouplingPort.pncMapping shall be performed for this EcuInstance. If not defined the derivation shall not be done. This Attribute is only used by the AUTOSAR Classic Platform.
firewallRule	StateDependentFirewall	*	ref	Firewall rules defined in the context of an EcuInstance. **Tags:** atp.Splitkey=firewallRule
j1939Node	J1939Node	*	aggr	Optional collection of J1939Nodes defined on this Ecu Instance. **Tags:** atp.Splitkey=j1939Node, j1939Node.shortName This Attribute is only used by the AUTOSAR Classic Platform.
partition	EcuPartition	*	aggr	Optional definition of Partitions within an Ecu. **Tags:** atp.Splitkey=partition, partition.shortName This Attribute is only used by the AUTOSAR Classic Platform.

▽

△

Class	**EcuInstance**			
pncNmRequest	Boolean	0..1	attr	Defines if this EcuInstance shall request Nm on all its PhysicalChannels which have Nm variant set to FULL each time a PNC is requested.
pncPrepare SleepTimer	TimeValue	0..1	attr	Time in seconds the PNC state machine shall wait in PNC_PREPARE_SLEEP.
pnc Synchronous Wakeup	Boolean	0..1	attr	If this parameter is available and set to true then all available PNCs will be woken up as soon as a channel wakeup occurs. This is ensured by adding all PNCs to all channel wakeup sources during upstream mapping.
pnResetTime	TimeValue	0..1	attr	Specifies the runtime of the reset timer in seconds. This reset time is valid for the reset of PN requests in the EIRA and in the ERA.
sleepMode Supported	Boolean	0..1	attr	Specifies whether the ECU instance may be put to a "low power mode" • true: sleep mode is supported • false: sleep mode is not supported Note: This flag may only be set to "true" if the feature is supported by both hardware and basic software. This Attribute is only used by the AUTOSAR Classic Platform.
tcpIpIcmpProps	EthTcpIpIcmpProps	0..1	ref	EcuInstance specific ICMP (Internet Control Message Protocol) attributes **Tags:** atp.Splitkey=tcpIpIcmpProps This Attribute is only used by the AUTOSAR Classic Platform.
tcpIpProps	EthTcpIpProps	0..1	ref	EcuInstance specific TcpIp Stack attributes. **Tags:** atp.Splitkey=tcpIpProps This Attribute is only used by the AUTOSAR Classic Platform.
wakeUpOver BusSupported	Boolean	0..1	attr	Driver support for wakeup over Bus. This Attribute is only used by the AUTOSAR Classic Platform.

Table A.30: EcuInstance

Class	**EcucModuleConfigurationValues**			
Note	Head of the configuration of one Module. A Module can be a BSW module as well as the RTE and ECU Infrastructure. As part of the BSW module description, the EcucModuleConfigurationValues element has two different roles: The recommendedConfiguration contains parameter values recommended by the BSW module vendor. The preconfiguredConfiguration contains values for those parameters which are fixed by the implementation and cannot be changed. These two EcucModuleConfigurationValues are used when the base EcucModuleConfigurationValues (as part of the base ECU configuration) is created to fill parameters with initial values. **Tags:** atp.recommendedPackage=EcucModuleConfigurationValuess This Class is only used by the AUTOSAR Classic Platform.			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***

▽

△

Class	**EcucModuleConfigurationValues**			
container	EcucContainerValue	*	aggr	Aggregates all containers that belong to this module configuration. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=container, container.shortName, container.variationPoint.shortLabel vh.latestBindingTime=postBuild xml.sequenceOffset=10
definition	EcucModuleDef	0..1	ref	Reference to the definition of this EcucModule ConfigurationValues element. Typically, this is a vendor specific module configuration. **Tags:** atp.Splitkey=definition xml.sequenceOffset=-10
ecucDefEdition	RevisionLabelString	0..1	attr	This is the version info of the ModuleDef ECUC Parameter definition to which this values conform to / are based on. For the Definition of ModuleDef ECUC Parameters the AdminData shall be used to express the semantic changes. The compatibility rules between the definition and value revision labels is up to the module's vendor.
implementation ConfigVariant	EcucConfiguration VariantEnum	0..1	attr	Specifies the kind of deliverable this EcucModule ConfigurationValues element provides. If this element is not used in a particular role (e.g. preconfigured Configuration or recommendedConfiguration) then the value shall be one of VariantPreCompile, VariantLink Time, VariantPostBuild. **Tags:** atp.Status=obsolete
module Description	BswImplementation	0..1	ref	Referencing the BSW module description, which this EcucModuleConfigurationValues element is configuring. This is optional because the EcucModuleConfiguration Values element is also used to configure the ECU infrastructure (memory map) or Application SW-Cs. However in case the EcucModuleConfigurationValues are used to configure the module, the reference is mandatory in order to fetch module specific "common" published information. **Tags:** atp.Splitkey=moduleDescription
postBuildVariant Used	Boolean	0..1	attr	Indicates whether a module implementation has or plans to have (i.e., introduced at link or post-build time) new post-build variation points. TRUE means yes, FALSE means no. If the attribute is not defined, FALSE semantics shall be assumed.

Table A.31: EcucModuleConfigurationValues

Class	**EcucValueCollection**			
Note	This represents the anchor point of the ECU configuration description. **Tags:** atp.recommendedPackage=EcucValueCollections This Class is only used by the AUTOSAR Classic Platform.			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***

▽

△

Class	**EcucValueCollection**			
ecucValue	EcucModule ConfigurationValues	*	ref	References to the configuration of individual software modules that are present on this ECU. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=ecucValue, ecucValue.ecucModule ConfigurationValues, ecucValue.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
ecuExtract	System	0..1	ref	Represents the extract of the System Configuration that is relevant for the ECU configured with that ECU Configuration Description. **Tags:** atp.Splitkey=ecuExtract

Table A.32: EcucValueCollection

Class	**FlatInstanceDescriptor**			
Note	Represents exactly one node (e.g. a component instance or data element) of the instance tree of a software system. The purpose of this element is to map the various nested representations of this instance to a flat representation and assign a unique name (shortName) to it. Use cases: • Specify unique names of measurable data to be used by MCD tools • Specify unique names of calibration data to be used by MCD tool • Specify a unique name for an instance of a component prototype in the ECU extract of the system description Note that in addition it is possible to assign alias names via AliasNameAssignment. This Class is only used by the AUTOSAR Classic Platform.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	FlatMap.instance			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
bsw Implementation	BswImplementation	0..1	ref	Reference to BswImplementation that defines the context for the AutosarDataPrototype that is referenced by FlatInstanceDescriptor.dataPrototype. **Tags:** atp.Splitkey=bswImplementation
dataPrototype	AutosarDataPrototype	0..1	ref	Reference to a DataPrototype that is defined in the Bsw InternalBehavior in the context of a BswImplementation that is defined by the FlatInstanceDescriptor.bsw Implementation reference. **Tags:** atp.Splitkey=dataPrototype

▽

△

Class	**FlatInstanceDescriptor**			
ecuExtract Reference	AtpFeature	0..1	iref	Refers to the instance in the ECU extract. This is valid only, if the FlatMap is used in the context of an ECU extract. The reference shall be such that it uniquely defines the object instance. For example, if a data prototype is declared as a role within an SwcInternalBehavior, it is not enough to state the SwcInternalBehavior as context and the aggregated data prototype as target. In addition, the reference shall also include the complete path identifying instance of the component prototype and the Atomic SoftwareComponentType, which is refered by the particular SwcInternalBehavior. **Tags:** atp.Splitkey=ecuExtractReference, ecuExtract Reference.contextElement, ecuExtractReference.target xml.sequenceOffset=40 **InstanceRef implemented by:** AnyInstanceRef
role	Identifier	0..1	attr	The role denotes the particular role of the downstream memory location described by this FlatInstanceDescriptor. It applies to use case where one upstream object results in multiple downstream objects, e.g. ModeDeclaration GroupPrototypes which are measurable. In this case the RTE will provide locations for current mode, previous mode and next mode.
rtePluginProps	RtePluginProps	0..1	aggr	The properties of a communication graph with respect to the utilization of RTE Implementation Plug-in. **Tags:** atp.Splitkey=rtePluginProps
swDataDef Props	SwDataDefProps	0..1	aggr	The properties of this FlatInstanceDescriptor. **Tags:** atp.Splitkey=swDataDefProps
upstream Reference	AtpFeature	0..1	iref	Refers to the instance in the context of an "upstream" description, which could be: the SYSTEM_ DESCRIPTION, or SYSTEM_EXTRACT, or ECU_ SYSTEM_DESCRIPTION, or SW_CLUSTER_SYSTEM_ DESCRIPTION, or the basic software module description (in this case only the target reference of the AnyInstance Ref is needed), or (if a flat map is used in preliminary context) a description of an atomic component or composition. This reference is optional in case the flat map is used in ECU context. The reference shall be such that it uniquely defines the object instance in the given context. For example, if a data prototype is declared as a role within an SwcInternal Behavior, it is not enough to state the Swc Internal Behavior as context and the aggregated data prototype as target. In addition, the reference shall also include the complete path identifying the instance of the component prototype that contains the particular instance of Swc InternalBehavior. **Tags:** atp.Splitkey=upstreamReference, upstream Reference.contextElement, upstreamReference.target xml.sequenceOffset=20 **InstanceRef implemented by:** AnyInstanceRef

Table A.33: FlatInstanceDescriptor

Class	**FlatMap**			
Note	Contains a flat list of references to software objects. This list is used to identify instances and to resolve name conflicts. The scope is given by the RootSwCompositionPrototype for which it is used, i.e. it can be applied to a system, system extract or ECU-extract. An instance of FlatMap may also be used in a preliminary context, e.g. in the scope of a software component before integration into a system. In this case it is not referred by a RootSwComposition Prototype. **Tags:** atp.recommendedPackage=FlatMaps This Class is only used by the AUTOSAR Classic Platform.			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
instance	FlatInstanceDescriptor	*	aggr	A descriptor instance aggregated in the flat map. The variation point accounts for the fact, that the system in scope can be subject to variability, and thus the existence of some instances is variable. The aggregation has been made splitable because the content might be contributed by different stakeholders at different times in the workflow. Plus, the overall size might be so big that eventually it becomes more manageable if it is distributed over several files. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=instance, instance.shortName, instance.variationPoint.shortLabel vh.latestBindingTime=postBuild

Table A.34: FlatMap

Class	***GeneralAnnotation*** (abstract)			
Note	This class represents textual comments (called annotations) which relate to the object in which it is aggregated. These annotations are intended for use during the development process for transferring information from one step of the development process to the next one. The approach is similar to the "yellow pads" ... This abstract class can be specialized in order to add some further formal properties.			
Base	*ARObject*			
Subclasses	Annotation, ClientServerAnnotation, DelegatedPortAnnotation, IoHwAbstractionServerAnnotation, Mode PortAnnotation, NvDataPortAnnotation, ParameterPortAnnotation, *SenderReceiverAnnotation*, Trigger PortAnnotation			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
annotation Origin	String	0..1	attr	This attribute identifies the origin of the annotation. It is an arbitrary string since it can be an individual's name as well as the name of a tool or even the name of a process step. **Tags:** xml.sequenceOffset=30
annotationText	DocumentationBlock	0..1	aggr	This is the text of the annotation. **Tags:** atp.Splitkey=annotationText xml.sequenceOffset=40
label	MultilanguageLong Name	0..1	aggr	This is the headline for the annotation. **Tags:** atp.Splitkey=label xml.sequenceOffset=20

Table A.35: GeneralAnnotation

Class	***Identifiable*** (abstract)			
Note	Instances of this class can be referred to by their identifier (within the namespace borders). In addition to this, Identifiables are objects which contribute significantly to the overall structure of an AUTOSAR description. In particular, Identifiables might contain Identifiables.			
Base	*ARObject*, *MultilanguageReferrable*, *Referrable*			
Subclasses	ARPackage, *AbstractDoIpLogicAddressProps*, *AbstractEvent*, *AbstractImplementationDataTypeElement*, *AbstractSecurityEventFilter*, *AbstractSecurityIdsmInstanceFilter*, *AbstractServiceInstance*, AppOsTask ProxyToEcuTaskProxyMapping, ApplicationEndpoint, ApplicationError, ApplicationPartitionToEcuPartition Mapping, AppliedStandard, AsynchronousServerCallResultPoint, *AtpBlueprint*, *AtpClassifier*, *Atp Feature*, AutosarOperationArgumentInstance, AutosarVariableInstance, *BinaryManifestAddressable Object*, BinaryManifestItemDefinition, *BinaryManifestResource*, BinaryManifestResourceDefinition, Block State, BswInternalTriggeringPoint, BswModuleDependency, *BuildActionEntity*, BuildActionEnvironment, CanSecSecureChannel, CanTpAddress, CanTpChannel, CanTpNode, Chapter, ClientIdDefinition, Client ServerOperation, Code, *CollectableElement*, ComManagementMapping, *CommConnectorPort*, *CommunicationConnector*, *CommunicationController*, Compiler, ConsistencyNeeds, ConsumedEvent Group, *CouplingElementAbstractDetails*, CouplingPort, *CouplingPortAbstractShaper*, *CouplingPort StructuralElement*, *CpSoftwareClusterResource*, CpSoftwareClusterResourceToApplicationPartition Mapping, CpSoftwareClusterToApplicationPartitionMapping, CpSoftwareClusterToEcuInstanceMapping, CpSoftwareClusterToResourceMapping, *CryptoServiceMapping*, CyclicHandlingComDataToOsTask ProxyMapping, DataPrototypeGroup, DataPrototypeTransformationPropsIdent, DataTransformation, *Dds AbstractServiceInstanceElementCp*, DdsCpDomain, DdsCpPartition, DdsCpQosProfile, DdsCpTopic, DependencyOnArtifact, *DiagEventDebounceAlgorithm*, DiagnosticAuthTransmitCertificateEvaluation, DiagnosticConnectedIndicator, DiagnosticDataElement, DiagnosticDebounceAlgorithmProps, Diagnostic ExtendedDataRecordElement, DiagnosticFunctionInhibitSource, DiagnosticParameterElement, *DiagnosticRoutineSubfunction*, DltApplication, DltArgument, DltArgumentProps, DltLogChannel, Dlt Message, DoIpInterface, DoIpLogicAddress, DoIpRoutingActivation, ECUMapping, *EOCExecutableEntity RefAbstract*, EcuPartition, EcuPartitionToCoreMapping, EcucContainerValue, *EcucDefinitionElement*, EcucDestinationUriDef, EcucEnumerationLiteralDef, EcucQuery, EcucValidationCondition, Ethernet WakeupSleepOnDatalineConfig, EventHandler, ExclusiveArea, *ExecutableEntity*, *ExecutionTime*, FMAttributeDef, FMFeatureMapAssertion, FMFeatureMapCondition, FMFeatureMapElement, FMFeature Relation, FMFeatureRestriction, FMFeatureSelection, FlatInstanceDescriptor, FlexrayTpConnection Control, FlexrayTpNode, FlexrayTpPduPool, *FrameTriggering*, GlobalTimeGateway, *GlobalTimeMaster*, *GlobalTimeSlave*, *HeapUsage*, HwAttributeDef, HwAttributeLiteralDef, HwPin, HwPinGroup, *IEEE1722 TpAcfBus*, *IEEE1722TpAcfBusPart*, IPSecRule, IPv6ExtHeaderFilterList, ISignalToIPduMapping, ISignal Triggering, *IdentCaption*, ImpositionTime, InternalTriggeringPoint, J1939Node, J1939SharedAddress Cluster, J1939TpNode, Keyword, LifeCycleState, LinScheduleTable, LinTpNode, Linker, MacAddress VlanMembership, MacMulticastGroup, MacSecKayParticipant, McDataInstance, MemorySection, Mode Declaration, ModeDeclarationMapping, ModeSwitchPoint, ModeSwitchSenderComSpecProps, Network Endpoint, *NmCluster*, NmEcu, *NmNode*, NvBlockDescriptor, *PackageableElement*, ParameterAccess, PduActivationRoutingGroup, PduToFrameMapping, PduTriggering, PerInstanceMemory, *Physical Channel*, PortElementToCommunicationResourceMapping, PortGroup, *PortInterfaceMapping*, Queued ReceiverComSpecProps, ResourceConsumption, RootSwCompositionPrototype, RptComponent, Rpt Container, RptExecutableEntity, RptExecutableEntityEvent, RptExecutionContext, RptProfile, RptService Point, RteEventInCompositionSeparation, RteEventInCompositionToOsTaskProxyMapping, RteEventIn SystemSeparation, RteEventInSystemToOsTaskProxyMapping, RunnableEntityGroup, *SdgAttribute*, Sdg Class, SecOcJobRequirement, SecureCommunicationAuthenticationProps, SecureCommunication FreshnessProps, SecurityEventContextDataElement, SecurityEventContextProps, *ServerCallPoint*, ServerComSpecProps, *ServiceNeeds*, SignalServiceTranslationElementProps, SignalServiceTranslation EventProps, SignalServiceTranslationProps, SocketAddress, SomeipTpChannel, *StackUsage*, Static SocketConnection, StructuredReq, SwGenericAxisParamType, SwServiceArg, SwcServiceDependency, SwcToApplicationPartitionMapping, SwcToEcuMapping, SwcToImplMapping, SwitchAsynchronousTraffic ShaperGroupEntry, SwitchAtsInstanceEntry, SwitchFlowMeteringEntry, SwitchStreamFilterActionDest PortModification, SwitchStreamFilterEntry, SwitchStreamFilterRule, SwitchStreamGateEntry, Switch StreamIdentification, SystemMapping, SystemSignalGroupToCommunicationResourceMapping, System SignalToCommunicationResourceMapping, TDCpSoftwareClusterMapping, TDCpSoftwareCluster ResourceMapping, TcpOptionFilterList, *TimingClock*, TimingClockSyncAccuracy, TimingCondition, *TimingConstraint*, *TimingDescription*, TimingExtensionResource, TimingModeInstance, TlsCryptoCipher Suite, TlsCryptoCipherSuiteProps, Topic1, TpAddress, TraceableTable, TraceableText, *TracedFailure*, TransformationISignalPropsIdent, *TransformationProps*, TransformationTechnology, Trigger, Variable Access, VariationPointProxy, ViewMap, VlanConfig, WaitPoint			
Attribute	**Type**	**Mult.**	**Kind**	**Note**

▽

△

Class	***Identifiable*** (abstract)			
adminData	AdminData	0..1	aggr	This represents the administrative data for the identifiable object. **Tags:** atp.Splitkey=adminData xml.sequenceOffset=-40
annotation	Annotation	*	aggr	Possibility to provide additional notes while defining a model element (e.g. the ECU Configuration Parameter Values). These are not intended as documentation but are mere design notes. **Tags:** atp.Splitkey=annotation xml.sequenceOffset=-25
category	CategoryString	0..1	attr	The category is a keyword that specializes the semantics of the Identifiable. It affects the expected existence of attributes and the applicability of constraints. **Tags:** xml.sequenceOffset=-50
desc	MultiLanguageOverview Paragraph	0..1	aggr	This represents a general but brief (one paragraph) description what the object in question is about. It is only one paragraph! Desc is intended to be collected into overview tables. This property helps a human reader to identify the object in question. More elaborate documentation, (in particular how the object is built or used) should go to "introduction". **Tags:** atp.Splitkey=desc xml.sequenceOffset=-60
introduction	DocumentationBlock	0..1	aggr	This represents more information about how the object in question is built or is used. Therefore it is a DocumentationBlock. **Tags:** atp.Splitkey=introduction xml.sequenceOffset=-30
uuid	String	0..1	attr	The purpose of this attribute is to provide a globally unique identifier for an instance of a meta-class. The values of this attribute should be globally unique strings prefixed by the type of identifier. For example, to include a DCE UUID as defined by The Open Group, the UUID would be preceded by "DCE:". The values of this attribute may be used to support merging of different AUTOSAR models. The form of the UUID (Universally Unique Identifier) is taken from a standard defined by the Open Group (was Open Software Foundation). This standard is widely used, including by Microsoft for COM (GUIDs) and by many companies for DCE, which is based on CORBA. The method for generating these 128-bit IDs is published in the standard and the effectiveness and uniqueness of the IDs is not in practice disputed. If the id namespace is omitted, DCE is assumed. An example is "DCE:2fac1234-31f8-11b4-a222-08002b34c003". The uuid attribute has no semantic meaning for an AUTOSAR model and there is no requirement for AUTOSAR tools to manage the timestamp. **Tags:** xml.attribute=true

Table A.36: Identifiable

Class	**ImplementationDataType**			
Note	Describes a reusable data type on the implementation level. This will typically correspond to a typedef in C-code. **Tags:** atp.recommendedPackage=ImplementationDataTypes			
Base	*ARElement*, *ARObject*, *AbstractImplementationDataType*, *AtpBlueprint*, *AtpClassifier*, *AtpType*, *Autosar DataType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
dynamicArray SizeProfile	String	0..1	attr	Specifies the profile which the array will follow in case this data type is a variable size array.
isStructWith Optional Element	Boolean	0..1	attr	This attribute is only valid if the attribute category is set to STRUCTURE. If set to true, this attribute indicates that the ImplementationDataType has been created with the intention to define at least one element of the structure as optional.
subElement (ordered)	ImplementationData TypeElement	*	aggr	Specifies an element of an array, struct, or union data type. The aggregation of `ImplementationDataTypeElement` is subject to variability with the purpose to support the conditional existence of elements inside a `ImplementationDataType` representing a structure. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
symbolProps	SymbolProps	0..1	aggr	This represents the SymbolProps for the Implementation DataType. **Tags:** atp.Splitkey=symbolProps, symbolProps.short Name
typeEmitter	NameToken	0..1	attr	This attribute is used to control which part of the AUTOSAR toolchain is supposed to trigger data type definitions.

Table A.37: ImplementationDataType

Class	**InstantiationDataDefProps**			
Note	This is a general class allowing to apply additional `SwDataDefProps` to particular instantiations of a `DataPrototype`. Typically the accessibility and further information like alias names for a particular data is modeled on the level of `DataPrototype`s (especially `VariableDataPrototype`s, `ParameterDataPrototype`s). But due to the recursive structure of the meta-model concerning data types (a composite (data) type consists out of data prototypes) a part of the MCD information is described in the data type (in case of `ApplicationCompositeDataType`). This is a strong restriction in the reuse of data typed because the data type should be re-used for different `VariableDataPrototype`s and `ParameterDataPrototype`s to guarantee type compatibility on C-implementation level (e.g. data of a Port is stored in PIM or a `ParameterDataPrototype` used as ROM Block and shall be typed by the same data type as NVRAM Block). This class overcomes such a restriction if applied properly.			
Base	*ARObject*			
Aggregated by	NvBlockDescriptor.instantiationDataDefProps, ParameterSwComponentType.instantiationDataDefProps, SwcInternalBehavior.instantiationDataDefProps			
Attribute	***Type***	***Mult.***	***Kind***	***Note***

▽

△

Class	**InstantiationDataDefProps**			
parameter Instance	AutosarParameterRef	0..1	aggr	This reference identifies the particular `DataPrototype` (defined in the context of a composite `ParameterDataPrototype`) on which the swDataDef Props shall be applied. **Tags:** atp.Splitkey=parameterInstance
swDataDef Props	SwDataDefProps	0..1	aggr	These are the particular data definition properties which shall be applied **Tags:** atp.Splitkey=swDataDefProps
variableInstance	AutosarVariableRef	0..1	aggr	This reference identifies the particular `DataPrototype` (defined in the context of a composite `VariableDataPrototype`) on which the swDataDef Props shall be applied. **Tags:** atp.Splitkey=variableInstance

Table A.38: InstantiationDataDefProps

Class	***InternalBehavior*** (abstract)			
Note	Common base class (abstract) for the internal behavior of both software components and basic software modules/clusters.			
Base	*ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	BswInternalBehavior, SwcInternalBehavior			
Aggregated by	*AtpClassifier*.atpFeature			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
constant Memory	ParameterData Prototype	*	aggr	Describes a read only memory object containing characteristic value(s) implemented by this Internal Behavior. The shortName of ParameterDataPrototype has to be equal to the ''C' identifier of the described constant. The characteristic value(s) might be shared between Sw ComponentPrototypes of the same SwComponentType. The aggregation of constantMemory is subject to variability with the purpose to support variability in the software component or module implementations. Typically different algorithms in the implementation are requiring different number of memory objects. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=constantMemory, constantMemory.short Name, constantMemory.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
constantValue Mapping	ConstantSpecification MappingSet	*	ref	Reference to the ConstantSpecificationMapping to be applied for the particular InternalBehavior **Tags:** atp.Splitkey=constantValueMapping
dataType Mapping	DataTypeMappingSet	*	ref	Reference to the DataTypeMapping to be applied for the particular InternalBehavior **Tags:** atp.Splitkey=dataTypeMapping

▽

△

Class	***InternalBehavior*** (abstract)			
exclusiveArea	ExclusiveArea	*	aggr	This specifies an ExclusiveArea for this InternalBehavior. The exclusiveArea is local to the component resp. module. The aggregation of ExclusiveAreas is subject to variability. Note: the number of ExclusiveAreas might vary due to the conditional existence of RunnableEntities or BswModuleEntities. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=exclusiveArea, exclusiveArea.shortName, exclusiveArea.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
exclusiveArea NestingOrder	ExclusiveAreaNesting Order	*	aggr	This represents the set of ExclusiveAreaNestingOrder owned by the InternalBehavior. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=exclusiveAreaNestingOrder, exclusiveArea NestingOrder.shortName, exclusiveAreaNesting Order.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
staticMemory	VariableDataPrototype	*	aggr	Describes a read and writeable static memory object representing measurerment variables implemented by this software component. The term "static" is used in the meaning of "non-temporary" and does not necessarily specify a linker encapsulation. This kind of memory is only supported if supportsMultipleInstantiation is FALSE. The shortName of the VariableDataPrototype has to be equal with the ''C' identifier of the described variable. The aggregation of staticMemory is subject to variability with the purpose to support variability in the software component's implementations. Typically different algorithms in the implementation are requiring different number of memory objects. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=staticMemory, staticMemory.shortName, staticMemory.variationPoint.shortLabel vh.latestBindingTime=preCompileTime

Table A.39: InternalBehavior

Class	**Item**			
Note	This meta-class represents one particular item in a list.			
Base	*ARObject*, *DocumentViewSelectable*, *Paginateable*			
Aggregated by	List.item			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
itemContents	DocumentationBlock	0..1	aggr	this represents the actual content of the item. It is composed of a DocumentationBlock. This way it is possible to use simple paragraphs to nested lists, formula, figures or notes. **Tags:** xml.roleElement=false xml.roleWrapperElement=false xml.typeElement=false xml.typeWrapperElement=false

Table A.40: Item

Class	List			
Note	This meta-class represents the ability to express a list. The kind of list is specified in the attribute.			
Base	*ARObject*, *DocumentViewSelectable*, *Paginateable*			
Aggregated by	DocumentationBlock.list			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
item	Item	*	aggr	this represents a particular list item. Note that this is again a documentation block.Therefore lists can be arbitrarily nested. It is discouraged to have a very deep nesting. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=postBuild xml.roleElement=true xml.roleWrapperElement=false xml.sequenceOffset=20 xml.typeElement=false xml.typeWrapperElement=false
type	ListEnum	0..1	attr	The type of the list. Default is "UNNUMBER" **Tags:** xml.attribute=true

Table A.41: List

Class	McDataInstance			
Note	Describes the specific properties of one data instance in order to support measurement and/or calibration of this data instance. The most important attributes are: • Its shortName is copied from the ECU Flat map (if applicable) and will be used as identifier and for display by the MC system. • The category is copied from the corresponding data type (ApplicationDataType if defined, otherwise ImplementationDataType) as far as applicable. • The symbol is the one used in the programming language. It will be used to find out the actual memory address by the final generation tool with the help of linker generated information. It is assumed that in the M1 model this part and all the aggregated and referred elements (with the exception of the Flat Map and the references from ImplementationElementInParameterInstanceRef and McAccessDetails) are completely generated from "upstream" information. This means, that even if an element like e.g. a CompuMethod is only used via reference here, it will be copied into the M1 artifact which holds the complete McSupportData for a given Implementation.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	McDataInstance.subElement, McSupportData.mcParameterInstance, McSupportData.mcVariable Instance			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
arraySize	PositiveInteger	0..1	attr	The existence of this attribute turns the data instance into an array of data. The attribute determines the size of the array in terms of number of elements.
displayIdentifier	McdIdentifier	0..1	attr	An optional attribute to be used to set the ASAM ASAP2 DISPLAY_IDENTIFIER attribute.

△

Class	**McDataInstance**			
flatMapEntry	FlatInstanceDescriptor	0..1	ref	Reference to the corresponding entry in the ECU Flat Map. This allows to trace back to the original specification of the generated data instance. This link shall be added by the RTE generator mainly for documentation purposes. The reference is optional because • The McDataInstance may represent an array or struct in which only the subElements correspond to FlatMap entries. • The McDataInstance may represent a task local buffer for rapid prototyping access which is different from the "main instance" used for measurement access. **Tags:** atp.Splitkey=flatMapEntry This Attribute is only used by the AUTOSAR Classic Platform.
instanceIn Memory	ImplementationElement InParameterInstance Ref	0..1	aggr	Reference to the corresponding data instance in the description of calibration data structures published by the RTE generator. This is used to support emulation methods inside the ECU, it is not required for A2L generation. **Tags:** atp.Splitkey=instanceInMemory, instanceIn Memory.context, instanceInMemory.target
mcDataAccess Details	McDataAccessDetails	0..1	aggr	Refers to "upstream" information on how the RTE uses this data instance. Use Case: Rapid Prototyping **Tags:** atp.Splitkey=mcDataAccessDetails
mcData Assignment	RoleBasedMcData Assignment	*	aggr	An assignment between McDataInstances. This supports the indication of related McDataElement implementing the of "RP global buffer", "RP global measurement buffer", "RP enabler flag". **Tags:** atp.Splitkey=mcDataAssignment
resulting Properties	SwDataDefProps	0..1	aggr	These are the generated properties resulting from decisions taken by the RTE generator for the actually implemented data instance. Only those properties are relevant here, which are needed for the measurement and calibration system. **Tags:** atp.Splitkey=resultingProperties
resultingRptSw Prototyping Access	RptSwPrototyping Access	0..1	aggr	Describes the implemented accessibility of data and modes by the rapid prototyping tooling. **Tags:** atp.Splitkey=resultingRptSwPrototypingAccess atp.Status=obsolete
role	Identifier	0..1	attr	An optional attribute to be used for additional information on the role of this data instance, for example in the context of rapid prototyping.
rptImplPolicy	RptImplPolicy	0..1	aggr	Describes the implemented code preparation for rapid prototyping at data accesses for a hook based bypassing. **Tags:** atp.Splitkey=rptImplPolicy atp.Status=obsolete
subElement (ordered)	McDataInstance	*	aggr	This relation indicates, that the target element is part of a "struct" which is given by the source element. This information will be used by the final generator to set up the correct addressing scheme. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime

▽

△

Class	**McDataInstance**			
symbol	SymbolString	0..1	attr	This String is used to determine the memory address during final generation of the MC configuration data (e.g. "A2L" file) . It shall be the name of the element in the programming language such that it can be identified in linker generated information. In case the McDataInstance is part of composite data in the programming language, the symbol String may include parts denoting the element context, unless the context is given by the symbol attribute of an enclosing McDataInstance. This means in particular for the C language that the "." character shall be used as a separator between the name of a "struct" variable the name of one of its elements. The symbol can differ from the shortName in case of generated C data declarations. It is an optional attribute since it may be missing in case the instance represents an element (e.g. a single array element) which has no name in the linker map.

Table A.42: McDataInstance

Enumeration	**MemoryAllocationKeywordPolicyType**
Note	Enumeration to specify the name pattern of the Memory Allocation Keyword.
Aggregated by	SwAddrMethod.memoryAllocationKeywordPolicy
Literal	*Description*
addrMethodShort Name	The MemorySection shortNames of referring MemorySections and therefore the belonging Memory Allocation Keywords in the code are build with the shortName of the SwAddrMethod. This is the default value if the attribute does not exist. **Tags:** atp.EnumerationLiteralIndex=0
addrMethodShort NameAndAlignment	The MemorySection shortNames of referring MemorySections and therefore the belonging Memory Allocation Keywords in the code are build with the shortName of the SwAddrMethod and a variable alignment postfix. Thereby the alignment postfix needs to be consistent with the alignment attribute of the related MemorySection. **Tags:** atp.EnumerationLiteralIndex=1

Table A.43: MemoryAllocationKeywordPolicyType

Enumeration	**MemorySectionType**
Note	Enumeration to specify the essential nature of the data which can be allocated in a common memory class by the means of the AUTOSAR Memory Mapping.
Aggregated by	SwAddrMethod.sectionType
Literal	*Description*
calibrationVariables	This memory section is reserved for "virtual variables" that are computed by an MCD system during a measurement session but do not exist in the ECU memory. **Tags:** atp.EnumerationLiteralIndex=2
calprm	To be used for calibratable constants of ECU-functions. **Tags:** atp.EnumerationLiteralIndex=3
code	To be used for mapping code to application block, boot block, external flash etc. **Tags:** atp.EnumerationLiteralIndex=4

▽

△

Enumeration	**MemorySectionType**
configData	Constants with attributes that show that they reside in one segment for module configuration. **Tags:** atp.EnumerationLiteralIndex=5
const	To be used for global or static constants. **Tags:** atp.EnumerationLiteralIndex=6
excludeFromFlash	This memory section is reserved for "virtual parameters" that are taken for computing the values of so-called dependent parameter of an MCD system. Dependent Parameters that are not at the same time "virtual parameters" are allocated in the ECU memory. Virtual parameters, on the other hand, are not allocated in the ECU memory. Virtual parameters exist in the ECU Hex file for the purpose of being considered (for computing the values of dependent parameters) during an offline-calibration session. **Tags:** atp.EnumerationLiteralIndex=7
var	To be used for global or static variables. The expected initialization is specified with the attribute sectionInitializationPolicy. **Tags:** atp.EnumerationLiteralIndex=9

Table A.44: MemorySectionType

Class	**ModeDeclaration**			
Note	Declaration of one Mode. The name and semantics of a specific mode is not defined in the meta-model.			
Base	*ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, ModeDeclarationGroup.modeDeclaration			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
value	PositiveInteger	0..1	attr	The RTE shall take the value of this attribute for generating the source code representation of this Mode Declaration.

Table A.45: ModeDeclaration

Class	**ModeDeclarationGroup**			
Note	A collection of Mode Declarations. Also, the initial mode is explicitly identified. **Tags:** atp.recommendedPackage=ModeDeclarationGroups			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *UploadableDesignElement*, *Uploadable PackageElement*			
Aggregated by	ARPackage.element			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
initialMode	ModeDeclaration	0..1	ref	The initial mode of the ModeDeclarationGroup. This mode is active before any mode switches occurred. **Tags:** atp.Splitkey=initialMode
mode Declaration	ModeDeclaration	*	aggr	The ModeDeclarations collected in this ModeDeclaration Group. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=modeDeclaration, modeDeclaration.shortName, modeDeclaration.variationPoint.shortLabel vh.latestBindingTime=blueprintDerivationTime

▽

△

Class	**ModeDeclarationGroup**			
modeManager ErrorBehavior	ModeErrorBehavior	0..1	aggr	This represents the ability to define the error behavior expected by the mode manager in case of errors on the mode user side (e.g. terminated mode user). **Tags:** atp.Splitkey=modeManagerErrorBehavior This Attribute is only used by the AUTOSAR Classic Platform.
modeTransition	ModeTransition	*	aggr	This represents the avaliable ModeTransitions of the ModeDeclarationGroup **Tags:** atp.Splitkey=modeTransition, modeTransition.shortName This Attribute is only used by the AUTOSAR Classic Platform.
modeUserError Behavior	ModeErrorBehavior	0..1	aggr	This represents the definition of the error behavior expected by the mode user in case of errors on the mode manager side (e.g. terminated mode manager). **Tags:** atp.Splitkey=modeUserErrorBehavior This Attribute is only used by the AUTOSAR Classic Platform.
onTransition Value	PositiveInteger	0..1	attr	The value of this attribute shall be taken into account by the RTE generator for programmatically representing a value used for the transition between two statuses. This Attribute is only used by the AUTOSAR Classic Platform.

Table A.46: ModeDeclarationGroup

Class	**ModeDeclarationGroupPrototype**			
Note	The ModeDeclarationGroupPrototype specifies a set of Modes (ModeDeclarationGroup) which is provided or required in the given context.			
Base	*ARObject*, *AtpFeature*, *AtpPrototype*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, BswModuleDescription.providedModeGroup, BswModuleDescription.requiredModeGroup, FirewallStateSwitchInterface.firewallStateMachine, FunctionGroupSet.functionGroup, ModeSwitchInterface.modeGroup, Process.processStateMachine, StateManagementStateNotification.stateMachine			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
swCalibration Access	SwCalibrationAccessEnum	0..1	attr	This allows for specifying whether or not the enclosing ModeDeclarationGroupPrototype can be measured at run-time. This Attribute is only used by the AUTOSAR Classic Platform.
type	ModeDeclarationGroup	0..1	tref	The "collection of ModeDeclarations" (= ModeDeclarationGroup) supported by a component **Stereotypes:** isOfType **Tags:** atp.Splitkey=type

Table A.47: ModeDeclarationGroupPrototype

Class	ModeSwitchInterface			
Note	A mode switch interface declares a `ModeDeclarationGroupPrototype` to be sent and received. **Tags:** atp.recommendedPackage=PortInterfaces			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
modeGroup	ModeDeclarationGroup Prototype	0..1	aggr	The `ModeDeclarationGroupPrototype` of this mode interface. **Tags:** atp.Splitkey=modeGroup, modeGroup.shortName

Table A.48: ModeSwitchInterface

Class	NumericalValueSpecification			
Note	A numerical ValueSpecification which is intended to be assigned to a Primitive data element. Note that the numerical value is a variant, it can be computed by a formula.			
Base	*ARObject*, *ValueSpecification*			
Aggregated by	ApplicationAssocMapElementValueSpecification.key, ApplicationAssocMapElementValue Specification.value, ArrayValueSpecification.element, CalibrationParameterValue.applInitValue, CalibrationParameterValue.implInitValue, ConstantSpecification.valueSpec, CryptoService Key.developmentValue, DiagnosticEnvDataCondition.compareValue, DiagnosticEnvDataElement Condition.compareValue, DiagnosticEnvSovdDataCondition.compareValue, FieldSenderComSpec.init Value, ISignal.initValue, ISignal.timeoutSubstitutionValue, NonqueuedReceiverComSpec.initValue, NonqueuedReceiverComSpec.timeoutSubstitutionValue, NonqueuedSenderComSpec.initValue, Nv ProvideComSpec.ramBlockInitValue, NvProvideComSpec.romBlockInitValue, NvRequireComSpec.init Value, ParameterDataPrototype.initValue, ParameterProvideComSpec.initValue, ParameterRequireCom Spec.initValue, PersistencyDataRequiredComSpec.initValue, PersistencyKeyValuePair.initValue, Port DefinedArgumentValue.value, PortPrototypeBlueprintInitValue.value, RecordValueSpecification.field, *StateManagementCompareCondition*.compareValue, SwDataDefProps.invalidValue, VariableData Prototype.initValue			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
value	Numerical	0..1	attr	This is the value itself. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime

Table A.49: NumericalValueSpecification

Class	PPortPrototype			
Note	Component port providing a certain port interface.			
Base	*ARObject*, *AbstractProvidedPortPrototype*, *AtpFeature*, *AtpPrototype*, *Identifiable*, *Multilanguage Referrable*, *PortPrototype*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, *SwComponentType*.port			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
provided Interface	PortInterface	0..1	tref	The interface that this port provides. **Stereotypes:** isOfType **Tags:** atp.Splitkey=providedInterface

Table A.50: PPortPrototype

Class	ParameterAccess			
Note	The presence of a ParameterAccess implies that a RunnableEntity needs access to a ParameterDataPrototype.			
Base	*ARObject*, *AbstractAccessPoint*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, RunnableEntity.parameterAccess			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
accessed Parameter	AutosarParameterRef	0..1	aggr	Reference to the accessed calibration parameter. **Tags:** atp.Splitkey=accessedParameter
swDataDef Props	SwDataDefProps	0..1	aggr	This allows denote instance and access specific properties, mainly input values and common axis. **Tags:** atp.Splitkey=swDataDefProps

Table A.51: ParameterAccess

Class	ParameterDataPrototype			
Note	A ParameterDataPrototype represents a formalized generic piece of information that is typically immutable by the application software layer, but mutable by measurement and calibration tools. ParameterDataPrototype is used in various contexts and the specific context gives the otherwise generic ParameterDataPrototype a dedicated semantics.			
Base	*ARObject*, *AtpFeature*, *AtpPrototype*, *AutosarDataPrototype*, *DataPrototype*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, BswInternalBehavior.perInstanceParameter, *InternalBehavior*.constantMemory, NvBlockDescriptor.romBlock, ParameterInterface.parameter, SwcInternalBehavior.perInstanceParameter, SwcInternalBehavior.sharedParameter			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
initValue	ValueSpecification	0..1	aggr	Specifies initial value(s) of the ParameterDataPrototype **Tags:** atp.Splitkey=initValue

Table A.52: ParameterDataPrototype

Class	ParameterInterface			
Note	A parameter interface declares a number of parameter and characteristic values to be exchanged between parameter components and software components. **Tags:** atp.recommendedPackage=PortInterfaces			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpClassifier*, *AtpType*, *CollectableElement*, *DataInterface*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
parameter	ParameterDataPrototype	*	aggr	The ParameterDataPrototype of this ParameterInterface. **Tags:** atp.Splitkey=parameter, parameter.shortName

Table A.53: ParameterInterface

Class	ParameterProvideComSpec			
Note	"Communication" specification that applies to parameters on the provided side of a connection.			
Base	*ARObject*, *PPortComSpec*			
Aggregated by	*AbstractProvidedPortPrototype*.providedComSpec, PortPrototypeBlueprint.providedComSpec			
Attribute	**Type**	**Mult.**	**Kind**	**Note**

▽

△

Class	**ParameterProvideComSpec**			
initValue	ValueSpecification	0..1	aggr	The initial value applicable for the corresponding ParameterDataPrototype. **Tags:** atp.Splitkey=initValue
parameter	ParameterData Prototype	0..1	ref	The ParameterDataPrototype to which the Parameter ComSpec applies. **Stereotypes:** atpIdentityContributor

Table A.54: ParameterProvideComSpec

Class	**ParameterSwComponentType**			
Note	The ParameterSwComponentType defines parameters and characteristic values accessible via provided Ports. The provided values are the same for all connected SwComponentPrototypes **Tags:** atp.recommendedPackage=SwComponentTypes			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *SwComponentType*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
constant Mapping	ConstantSpecification MappingSet	*	ref	Reference to the ConstantSpecificationMapping to be applied for the particular ParameterSwComponentType **Tags:** atp.Splitkey=constantMapping
dataType Mapping	DataTypeMappingSet	*	ref	Reference to the DataTypeMapping to be applied for the particular ParameterSwComponentType **Tags:** atp.Splitkey=dataTypeMapping
instantiation DataDefProps	InstantiationDataDef Props	*	aggr	The purpose of this is that within the context of a given SwComponentType some data def properties of individual instantiations can be modified. The aggregation of InstantiationDataDefProps is subject to variability with the purpose to support the conditional existence of PortPrototypes **Stereotypes:** atpVariation **Tags:** atp.Splitkey=instantiationDataDefProps, instantiationData DefProps.variationPoint.shortLabel vh.latestBindingTime=preCompileTime

Table A.55: ParameterSwComponentType

Class	**PhysConstrs**			
Note	This meta-class represents the ability to express physical constraints. Therefore it has (in opposite to InternalConstrs) a reference to a Unit.			
Base	*ARObject*			
Aggregated by	DataConstrRule.physConstrs			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
lowerLimit	Limit	0..1	attr	This specifies the lower limit of the constraint. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=20

▽

△

Class	**PhysConstrs**			
maxDiff	Numerical	0..1	attr	Maximum difference that is permitted between two consecutive values if the constraint is applied to an axis. **Tags:** xml.sequenceOffset=60
maxGradient	Numerical	0..1	attr	This element specifies the maximum slope that may be used in curves and maps. **Tags:** xml.sequenceOffset=50
monotony	MonotonyEnum	0..1	attr	This specifies the monotony constraints on the data object. Note that this applies only to curves and maps. **Tags:** xml.sequenceOffset=70
unit	Unit	0..1	ref	This is the unit to which the physical constraints relate to. In particular, it is the physical unit of the specified limits. **Tags:** atp.Splitkey=unit xml.sequenceOffset=80
upperLimit	Limit	0..1	attr	This specifies the upper limit of the constraint. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=30

Table A.56: PhysConstrs

Class	**PhysicalDimension**			
Note	This class represents a physical dimension. If the physical dimension of two units is identical, then a conversion between them is possible. The conversion between units is related to the definition of the physical dimension. Note that the equivalence of the exponents does not per se define the convertibility. For example Energy and Torque share the same exponents (Nm). Please note further the value of an exponent does not necessarily have to be an integer number. It is also possible that the value yields a rational number, e.g. to compute the square root of a given physical quantity. In this case the exponent value would be a rational number where the numerator value is 1 and the denominator value is 2. **Tags:** atp.recommendedPackage=PhysicalDimensions			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
currentExp	Numerical	0..1	attr	This attribute represents the exponent of the physical dimension "electric current". **Tags:** xml.sequenceOffset=50
lengthExp	Numerical	0..1	attr	The exponent of the physical dimension "length". **Tags:** xml.sequenceOffset=20
luminous IntensityExp	Numerical	0..1	attr	The exponent of the physical dimension "luminous intensity". **Tags:** xml.sequenceOffset=80
massExp	Numerical	0..1	attr	The exponent of the physical dimension "mass". **Tags:** xml.sequenceOffset=30

▽

△

Class	PhysicalDimension			
molarAmount Exp	Numerical	0..1	attr	The exponent of the physical dimension "quantity of substance". **Tags:** xml.sequenceOffset=70
temperatureExp	Numerical	0..1	attr	The exponent of the physical dimension "temperature". **Tags:** xml.sequenceOffset=60
timeExp	Numerical	0..1	attr	The exponent of the physical dimension "time". **Tags:** xml.sequenceOffset=40

Table A.57: PhysicalDimension

Class	***PortInterface*** (abstract)			
Note	Abstract base class for an interface that is either provided or required by a port of a software component.			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Subclasses	ClientServerInterface, *DataInterface*, ModeSwitchInterface, TriggerInterface			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
isService	Boolean	0..1	attr	This flag is set if the `PortInterface` is to be used for communication between an • `ApplicationSwComponentType` or • `ServiceProxySwComponentType` or • `SensorActuatorSwComponentType` or • `ComplexDeviceDriverSwComponentType` • `ServiceSwComponentType` • `EcuAbstractionSwComponentType` and a `ServiceSwComponentType` (namely an AUTOSAR Service) located on the same ECU. Otherwise the flag is not set. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=blueprintDerivationTime This Attribute is only used by the AUTOSAR Classic Platform.
serviceKind	ServiceProviderEnum	0..1	attr	This attribute provides further details about the nature of the applied service. This Attribute is only used by the AUTOSAR Classic Platform.

Table A.58: PortInterface

Class	***PortPrototype*** (abstract)			
Note	Base class for the ports of an AUTOSAR software component. The aggregation of PortPrototypes is subject to variability with the purpose to support the conditional existence of ports.			
Base	*ARObject*, *AtpFeature*, *AtpPrototype*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	*AbstractProvidedPortPrototype*, *AbstractRequiredPortPrototype*			
Aggregated by	*AtpClassifier*.atpFeature, *SwComponentType*.port			
Attribute	***Type***	***Mult.***	***Kind***	***Note***

▽

△

Class	***PortPrototype*** (abstract)			
clientServer Annotation	ClientServerAnnotation	*	aggr	Annotation of this `PortPrototype` with respect to client/ server communication. **Tags:** atp.Splitkey=clientServerAnnotation
delegatedPort Annotation	DelegatedPort Annotation	0..1	aggr	Annotations on this delegated port. **Tags:** atp.Splitkey=delegatedPortAnnotation
ioHwAbstraction Server Annotation	IoHwAbstractionServer Annotation	*	aggr	Annotations on this IO Hardware Abstraction port. **Tags:** atp.Splitkey=ioHwAbstractionServerAnnotation
modePort Annotation	ModePortAnnotation	*	aggr	Annotations on this mode port. **Tags:** atp.Splitkey=modePortAnnotation
nvDataPort Annotation	NvDataPortAnnotation	*	aggr	Annotations on this non voilatile data port. **Tags:** atp.Splitkey=nvDataPortAnnotation
parameterPort Annotation	ParameterPort Annotation	*	aggr	Annotations on this parameter port. **Tags:** atp.Splitkey=parameterPortAnnotation
senderReceiver Annotation	SenderReceiver Annotation	*	aggr	Collection of annotations of this ports sender/receiver communication. **Tags:** atp.Splitkey=senderReceiverAnnotation
triggerPort Annotation	TriggerPortAnnotation	*	aggr	Annotations on this trigger port. **Tags:** atp.Splitkey=triggerPortAnnotation

Table A.59: PortPrototype

Class	**RPortPrototype**			
Note	Component port requiring a certain port interface.			
Base	*ARObject*, *AbstractRequiredPortPrototype*, *AtpFeature*, *AtpPrototype*, *Identifiable*, *Multilanguage Referrable*, *PortPrototype*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, *SwComponentType*.port			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
mayBe Unconnected	Boolean	0..1	attr	If set to true, this attribute indicates that the enclosing `RPortPrototype` may be left unconnected and that this aspect has explicitly been considered in the software-component's design. This Attribute is only used by the AUTOSAR Classic Platform.
required Interface	PortInterface	0..1	tref	The interface that this port requires. **Stereotypes:** isOfType **Tags:** atp.Splitkey=requiredInterface

Table A.60: RPortPrototype

Class	***RTEEvent*** (abstract)
Note	Abstract base class for all RTE-related events
Base	*ARObject*, *AbstractEvent*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *Multilanguage Referrable*, *Referrable*
Subclasses	AsynchronousServerCallReturnsEvent, BackgroundEvent, DataReceiveErrorEvent, DataReceivedEvent, DataSendCompletedEvent, DataWriteCompletedEvent, ExternalTriggerOccurredEvent, InitEvent, InternalTriggerOccurredEvent, ModeSwitchedAckEvent, OperationInvokedEvent, OsTaskExecutionEvent, SwcModeManagerErrorEvent, SwcModeSwitchEvent, TimingEvent, TransformerHardErrorEvent
Aggregated by	*AtpClassifier*.atpFeature, SwcInternalBehavior.event

▽

△

Class	***RTEEvent*** (abstract)			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
disabledMode	ModeDeclaration	*	iref	Reference to the Modes that disable the Event. **Tags:** atp.Splitkey=disabledMode, disabledMode.contextPort, disabledMode.contextModeDeclarationGroupPrototype, disabledMode.targetModeDeclaration **InstanceRef implemented by:** RModeInAtomicSwcInstanceRef
startOnEvent	RunnableEntity	0..1	ref	The referenced `RunnableEntity` starts when the corresponding `RTEEvent` is raised. **Tags:** atp.Splitkey=startOnEvent

Table A.61: RTEEvent

Primitive	**Ref**			
Note	This primitive denotes a name based reference. For detailed syntax see the xsd.pattern. • first slash (relative or absolute reference) [optional] • Identifier [required] • a sequence of slashes and Identifiers [optional] This primitive is used by the meta-model tools to create the references. **Tags:** xml.xsd.customType=REF xml.xsd.pattern=/?[a-zA-Z][a-zA-Z0-9_]{0,127}(/[a-zA-Z][a-zA-Z0-9_]{0,127})* xml.xsd.type=string			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
base	Identifier	0..1	attr	This attribute reflects the base to be used for this reference. **Tags:** xml.attribute=true
blueprintValue	String	0..1	attr	This represents a description that documents how the value shall be defined when deriving objects from the blueprint. **Tags:** atp.Status=valid xml.attribute=true
index	PositiveInteger	0..1	attr	This attribute supports the use case to point on specific elements in an array. This is in particular required if arrays are used to implement particular data objects. The counting of array indices starts with the value 0, i.e. the index of the first array element is 0. **Tags:** xml.attribute=true

Table A.62: Ref

Class	***Referrable*** (abstract)			
Note	Instances of this class can be referred to by their identifier (while adhering to namespace borders).			
Base	*ARObject*			
Subclasses	*AtpDefinition*, BswDistinguishedPartition, *BswModuleCallPoint*, BswModuleClientServerEntry, BswVariableAccess, CouplingPortTrafficClassAssignment, *DiagnosticEnvModeElement*, EthernetPriorityRegeneration, ExclusiveAreaNestingOrder, *HwDescriptionEntity*, *ImplementationProps*, LinSlaveConfigIdent, ModeTransition, *MultilanguageReferrable*, PncMappingIdent, *SingleLanguageReferrable*, SoConIPduIdentifier, TpConnectionIdent			
Attribute	***Type***	***Mult.***	***Kind***	***Note***

▽

△

Class	***Referrable*** (abstract)			
shortName	Identifier	1	attr	This specifies an identifying shortName for the object. It needs to be unique within its context and is intended for humans but even more for technical reference. **Stereotypes:** atpIdentityContributor **Tags:** xml.enforceMinMultiplicity=true xml.sequenceOffset=-100
shortName Fragment	ShortNameFragment	*	aggr	This specifies how the Referrable.shortName is composed of several shortNameFragments. **Tags:** atp.Splitkey=shortNameFragment xml.sequenceOffset=-90

Table A.63: Referrable

Class	**RootSwCompositionPrototype**			
Note	The RootSwCompositionPrototype represents the top-level-composition of software components within a given System. According to the use case of the System, this may for example be a more or less complete VFB description, the software of a System Extract or the software of a flat ECU Extract with only atomic SWCs. Therefore the RootSwComposition will only occasionally contain all atomic software components that are used in a complete VFB System. The OEM is primarily interested in the required functionality and the interfaces defining the integration of the Software Component into the System. The internal structure of such a component contains often substantial intellectual property of a supplier. Therefore a top-level software composition will often contain empty compositions which represent subsystems. The contained SwComponentPrototypes are fully specified by their SwComponentTypes (including Port Prototypes, PortInterfaces, VariableDataPrototypes, SwcInternalBehavior etc.), and their ports are interconnected using SwConnectorPrototypes.			
Base	*ARObject*, *AtpFeature*, *AtpPrototype*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, System.rootSoftwareComposition			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
calibration ParameterValue Set	CalibrationParameter ValueSet	*	ref	Used CalibrationParameterValueSet for instance specific initialization of calibration parameters. **Tags:** atp.Splitkey=calibrationParameterValueSet This Attribute is only used by the AUTOSAR Classic Platform.
flatMap	FlatMap	0..1	ref	The FlatMap used in the scope of this RootSw CompositionPrototype. **Tags:** atp.Splitkey=flatMap This Attribute is only used by the AUTOSAR Classic Platform.
software Composition	CompositionSw ComponentType	0..1	tref	We assume that there is exactly one top-level composition that includes all Component instances of the system. **Stereotypes:** isOfType **Tags:** atp.Splitkey=softwareComposition

Table A.64: RootSwCompositionPrototype

Class	**RunnableEntity**
Note	A `RunnableEntity` represents the smallest code-fragment that is provided by an `AtomicSwComponentType` and are executed under control of the RTE. `RunnableEntity`s are for instance set up to respond to data reception or operation invocation on a server.

▽

△

Class	**RunnableEntity**			
Base	*ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *ExecutableEntity*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, SwcInternalBehavior.runnable			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
argument (ordered)	RunnableEntity Argument	*	aggr	This represents the formal definition of an argument to a RunnableEntity.
asynchronous ServerCall ResultPoint	AsynchronousServer CallResultPoint	*	aggr	The server call result point admits a runnable to fetch the result of an asynchronous server call. The aggregation of AsynchronousServerCallResultPoint is subject to variability with the purpose to support the conditional existence of client server PortPrototypes and the variant existence of server call result points in the implementation. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=asynchronousServerCallResultPoint, asynchronousServerCallResultPoint.shortName, asynchronousServerCallResultPoint.variationPoint.shortLabel vh.latestBindingTime=preCompileTime This Attribute is only used by the AUTOSAR Classic Platform.
canBeInvoked Concurrently	Boolean	0..1	attr	If the value of this attribute is set to "true" the enclosing `RunnableEntity` can be invoked concurrently (even for one instance of the corresponding `AtomicSwComponentType`). This implies that it is the responsibility of the implementation of the `RunnableEntity` to take care of this form of concurrency.
dataRead Access	VariableAccess	*	aggr	RunnableEntity has implicit read access to dataElement of a sender-receiver PortPrototype or nv data of a nv data PortPrototype. The aggregation of dataReadAccess is subject to variability with the purpose to support the conditional existence of sender receiver ports or the variant existence of dataReadAccess in the implementation. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=dataReadAccess, dataReadAccess.shortName, dataReadAccess.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
dataReceive PointBy Argument	VariableAccess	*	aggr	RunnableEntity has explicit read access to dataElement of a sender-receiver PortPrototype or nv data of a nv data PortPrototype. The result is passed back to the application by means of an argument in the function signature. The aggregation of dataReceivePointByArgument is subject to variability with the purpose to support the conditional existence of sender receiver PortPrototype or the variant existence of data receive points in the implementation. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=dataReceivePointByArgument, dataReceivePointByArgument.shortName, dataReceivePointByArgument.variationPoint.shortLabel vh.latestBindingTime=preCompileTime

▽

△

Class	**RunnableEntity**			
dataReceive PointByValue	VariableAccess	*	aggr	RunnableEntity has explicit read access to dataElement of a sender-receiver PortPrototype or nv data of a nv data PortPrototype. The result is passed back to the application by means of the return value. The aggregation of dataReceivePointBy Value is subject to variability with the purpose to support the conditional existence of sender receiver ports or the variant existence of data receive points in the implementation. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=dataReceivePointByValue, dataReceivePoint ByValue.shortName, dataReceivePointByValue.variation Point.shortLabel vh.latestBindingTime=preCompileTime
dataSendPoint	VariableAccess	*	aggr	RunnableEntity has explicit write access to dataElement of a sender-receiver PortPrototype or nv data of a nv data PortPrototype. The aggregation of dataSendPoint is subject to variability with the purpose to support the conditional existence of sender receiver PortPrototype or the variant existence of data send points in the implementation. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=dataSendPoint, dataSendPoint.shortName, dataSendPoint.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
dataWrite Access	VariableAccess	*	aggr	RunnableEntity has implicit write access to dataElement of a sender-receiver PortPrototype or nv data of a nv data PortPrototype. The aggregation of dataWriteAccess is subject to variability with the purpose to support the conditional existence of sender receiver ports or the variant existence of dataWriteAccess in the implementation. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=dataWriteAccess, dataWriteAccess.short Name, dataWriteAccess.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
external TriggeringPoint	ExternalTriggeringPoint	*	aggr	The aggregation of ExternalTriggeringPoint is subject to variability with the purpose to support the conditional existence of trigger ports or the variant existence of external triggering points in the implementation. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=externalTriggeringPoint, externalTriggering Point.ident.shortName, externalTriggeringPoint.variation Point.shortLabel vh.latestBindingTime=preCompileTime

▽

△

Class	**RunnableEntity**			
internal TriggeringPoint	InternalTriggeringPoint	*	aggr	The aggregation of InternalTriggeringPoint is subject to variability with the purpose to support the variant existence of internal triggering points in the implementation. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=internalTriggeringPoint, internalTriggering Point.shortName, internalTriggeringPoint.variation Point.shortLabel vh.latestBindingTime=preCompileTime
modeAccess Point	ModeAccessPoint	*	aggr	The runnable has a mode access point. The aggregation of ModeAccessPoint is subject to variability with the purpose to support the conditional existence of mode ports or the variant existence of mode access points in the implementation. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=modeAccessPoint, modeAccess Point.ident.shortName, modeAccessPoint.variation Point.shortLabel vh.latestBindingTime=preCompileTime
modeSwitch Point	ModeSwitchPoint	*	aggr	The runnable has a mode switch point. The aggregation of ModeSwitchPoint is subject to variability with the purpose to support the conditional existence of mode ports or the variant existence of mode switch points in the implementation. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=modeSwitchPoint, modeSwitchPoint.short Name, modeSwitchPoint.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
parameter Access	ParameterAccess	*	aggr	The presence of a ParameterAccess implies that a RunnableEntity needs read only access to a Parameter DataPrototype which may either be local or within a Port Prototype. The aggregation of ParameterAccess is subject to variability with the purpose to support the conditional existence of parameter ports and component local parameters as well as the variant existence of Parameter Access (points) in the implementation. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=parameterAccess, parameterAccess.short Name, parameterAccess.variationPoint.shortLabel vh.latestBindingTime=preCompileTime

▽

△

Class	**RunnableEntity**			
readLocal Variable	VariableAccess	*	aggr	The presence of a readLocalVariable implies that a RunnableEntity needs read access to a VariableData Prototype in the role of implicitInterRunnableVariable or explicitInterRunnableVariable. The aggregation of readLocalVariable is subject to variability with the purpose to support the conditional existence of implicitInterRunnableVariable and explicit InterRunnableVariable or the variant existence of read LocalVariable (points) in the implementation. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=readLocalVariable, readLocalVariable.short Name, readLocalVariable.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
serverCallPoint	ServerCallPoint	*	aggr	The RunnableEntity has a ServerCallPoint. The aggregation of ServerCallPoint is subject to variability with the purpose to support the conditional existence of client server PortPrototypes or the variant existence of server call points in the implementation. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=serverCallPoint, serverCallPoint.shortName, serverCallPoint.variationPoint.shortLabel vh.latestBindingTime=preCompileTime This Attribute is only used by the AUTOSAR Classic Platform.
symbol	CIdentifier	0..1	attr	The symbol describing this `RunnableEntity`'s entry point. This is considered the API of the `RunnableEntity` and is required during the RTE contract phase.
waitPoint	WaitPoint	*	aggr	The `WaitPoint` associated with the `RunnableEntity`. **Tags:** atp.Splitkey=waitPoint, waitPoint.shortName
writtenLocal Variable	VariableAccess	*	aggr	The presence of a writtenLocalVariable implies that a RunnableEntity needs write access to a VariableData Prototype in the role of implicitInterRunnableVariable or explicitInterRunnableVariable. The aggregation of writtenLocalVariable is subject to variability with the purpose to support the conditional existence of implicitInterRunnableVariable and explicit InterRunnableVariable or the variant existence of written LocalVariable (points) in the implementation. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=writtenLocalVariable, writtenLocal Variable.shortName, writtenLocalVariable.variation Point.shortLabel vh.latestBindingTime=preCompileTime

Table A.65: RunnableEntity

Class	**SenderReceiverInterface**
Note	A sender/receiver interface declares a number of data elements to be sent and received. **Tags:** atp.recommendedPackage=PortInterfaces
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpClassifier*, *AtpType*, *CollectableElement*, *DataInterface*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable*
Aggregated by	ARPackage.element

▽

△

Class	**SenderReceiverInterface**			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
dataElement	VariableDataPrototype	*	aggr	The data elements of this `SenderReceiverInterface`. **Tags:** atp.Splitkey=dataElement, dataElement.shortName
invalidationPolicy	InvalidationPolicy	*	aggr	InvalidationPolicy for a particular dataElement **Tags:** atp.Splitkey=invalidationPolicy
metaDataItemSet	MetaDataItemSet	*	aggr	This aggregation defines fixed sets of meta-data items associated with `dataElement`s of the enclosing `SenderReceiverInterface` **Tags:** atp.Splitkey=metaDataItemSet

Table A.66: SenderReceiverInterface

Class	**StructuredReq**			
Note	This represents a structured requirement. This is intended for a case where specific requirements for features are collected. Note that this can be rendered as a labeled list.			
Base	*ARObject*, *DocumentViewSelectable*, *Identifiable*, *MultilanguageReferrable*, *Paginateable*, *Referrable*, *Traceable*			
Aggregated by	DocumentationBlock.structuredReq			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
appliesTo	StandardNameEnum	*	attr	This attribute represents the platform the requirement is assigned to. **Tags:** xml.namePlural=APPLIES-TO-DEPENDENCIES xml.sequenceOffset=25
conflicts	DocumentationBlock	0..1	aggr	This represents an informal specification of conflicts. **Tags:** atp.Splitkey=conflicts xml.sequenceOffset=40
date	DateTime	0..1	attr	This represents the date when the requirement was initiated. **Tags:** xml.sequenceOffset=5
dependencies	DocumentationBlock	0..1	aggr	This represents an informal specification of dependencies. Note that upstream tracing should be formalized in the property trace provided by the superclass Traceable. **Tags:** atp.Splitkey=dependencies xml.sequenceOffset=30
description	DocumentationBlock	0..1	aggr	This represents the general description of the requirement. **Tags:** atp.Splitkey=description xml.sequenceOffset=10
importance	String	0..1	attr	This allows to represent the importance of the requirement. **Tags:** xml.sequenceOffset=8
issuedBy	String	0..1	attr	This represents the person, organization or authority which issued the requirement. **Tags:** xml.sequenceOffset=6

▽

△

Class	**StructuredReq**			
rationale	DocumentationBlock	0..1	aggr	This represents the rationale of the requirement. **Tags:** atp.Splitkey=rationale xml.sequenceOffset=20
remark	DocumentationBlock	0..1	aggr	This represents an informal remark. Note that this is not modeled as annotation, since these remark is still essential part of the requirement. **Tags:** atp.Splitkey=remark xml.sequenceOffset=60
supporting Material	DocumentationBlock	0..1	aggr	This represents an informal specification of the supporting material. **Tags:** atp.Splitkey=supportingMaterial xml.sequenceOffset=50
testedItem	Traceable	*	ref	This association represents the ability to trace on the same specification level. This supports for example the of acceptance tests. **Tags:** atp.Splitkey=testedItem xml.sequenceOffset=70
type	String	0..1	attr	This attribute allows to denote the type of requirement to denote for example is it an "enhancement", "new feature" etc. **Tags:** xml.sequenceOffset=7
useCase	DocumentationBlock	0..1	aggr	This describes the relevant use cases. Note that formal references to use cases should be done in the trace relation. **Tags:** atp.Splitkey=useCase xml.sequenceOffset=35

Table A.67: StructuredReq

Class	**SwAddrMethod**			
Note	Used to assign a common addressing method, e.g. common memory section, to data or code objects. These objects could actually live in different modules or components. **Tags:** atp.recommendedPackage=SwAddrMethods			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
memory Allocation KeywordPolicy	MemoryAllocation KeywordPolicyType	0..1	attr	Enumeration to specify the name pattern of the Memory Allocation Keyword.
option	Identifier	*	attr	This attribute introduces the ability to specify further intended properties of the MemorySection in with the related objects shall be placed. These properties are handled as to be selected. The intended options are mentioned in the list. In the Memory Mapping configuration, this option list is used to determine an appropriate MemMapAddressing ModeSet.

▽

△

Class	**SwAddrMethod**			
section Initialization Policy	SectionInitialization PolicyType	0..1	attr	Specifies the expected initialization of the variables (inclusive those which are implementing VariableData Prototypes). Therefore this is an implementation constraint for initialization code of BSW modules (especially RTE) as well as the start-up code which initializes the memory segment to which the AutosarData Prototypes referring to the SwAddrMethod's are later on mapped. If the attribute is not defined it has the identical semantic as the attribute value "INIT"
sectionType	MemorySectionType	0..1	attr	Defines the type of memory sections which can be associated with this addressing method.

Table A.68: SwAddrMethod

Class	**SwAxisGrouped**			
Note	An SwAxisGrouped is an axis which is shared between multiple calibration parameters.			
Base	*ARObject*, *SwCalprmAxisTypeProps*			
Aggregated by	SwCalprmAxis.swCalprmAxisTypeProps			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
sharedAxisType	ApplicationPrimitive DataType	0..1	ref	This is the datatype of the calibration parameter providing the shared axis.
swAxisIndex	AxisIndexType	0..1	attr	Describes which axis of the referenced calibration parameter provides the values for the group axis. The index satisfies the following convention: • 0 = value axis. in this case, the interpolation result of the referenced parameter is used as a base point index. • The index should only be specified if the parameter under swCalprm contains more than one axis. It is standard practice for the axis index of parameters with more than one axis, to be set to 1, if data has not been assigned to swAxisIndex. **Tags:** xml.sequenceOffset=20
swCalprmRef	SwCalprmRefProxy	1	aggr	This property specifies the calibration parameter which serves as the input axis. In AUTOSAR, the type of the referenced Calibration parameter shall be compatible to the type specified by sharedAxisType. Please note that the multiplicity of this aggregation cannot be set to 0..1 based on the non-mainstream schema generation instructions defined at the aggregation. However, the multiplicity has to be factually considered 0..1 (i.e. a SwAxisGrouped that does not aggregate the role swCalprmRef is still valid according to the XML schema, depending on the use case documented in [constr_1015]). **Tags:** xml.roleElement=false xml.roleWrapperElement=false xml.sequenceOffset=30 xml.typeElement=false xml.typeWrapperElement=false

Table A.69: SwAxisGrouped

Class	**SwAxisIndividual**			
Note	This meta-class describes an axis integrated into a parameter (field etc.). The integration makes this individual to each parameter. The so-called grouped axis represents the counterpart to this. It is conceived as an independent parameter (see class SwAxisGrouped).			
Base	*ARObject*, *SwCalprmAxisTypeProps*			
Aggregated by	SwCalprmAxis.swCalprmAxisTypeProps			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
compuMethod	CompuMethod	0..1	ref	This is the compuMethod which is expected for the axis. It is used in early stages if the particular input-value is not yet available. **Tags:** xml.sequenceOffset=30
dataConstr	DataConstr	0..1	ref	Refers to constraints, e.g. for plausibility checks. **Tags:** xml.sequenceOffset=80
inputVariable Type	ApplicationPrimitive DataType	0..1	ref	This is the datatype of the input value for the axis. This allows to define e.g. a type of curve, where the input value is finalized at the access point.
swAxisGeneric	SwAxisGeneric	0..1	aggr	this specifies the properties of a generic axis if applicable. **Tags:** xml.sequenceOffset=90
swMaxAxis Points	Integer	0..1	attr	Maximum number of base points contained in the axis of a map or curve. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=60
swMinAxis Points	Integer	0..1	attr	Minimum number of base points contained in the axis of a map or curve. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=70
swVariableRef (ordered)	SwVariableRefProxy	*	aggr	Refers to input variables of the axis. It is possible to specify more than one variable. Here the following is valid: • The variable with the highest priority shall be given first. It is used in the generation of the code and is also displayed first in the application system. • All variables referenced shall be of the same physical nature. This is usually detected in that the conversion formulae affected refer back to the same SI-units. In AUTOSAR this ensured by the constraint, that the referenced input variables shall use a type compatible to "inputVariableType". • This multiple referencing allows a base point distribution for more than one input variable to be used. One example of this are the temperature curves which can depend both on the induction air temperature and the engine temperature. These variables can be displayed simultaneously by MCD systems (adjustment systems), enabling operating points to be shown in the curves. **Tags:** xml.roleElement=false xml.roleWrapperElement=true xml.sequenceOffset=20 xml.typeElement=false xml.typeWrapperElement=false



△

Class	**SwAxisIndividual**			
unit	Unit	0..1	ref	This represents the physical unit of the input value of the axis. It is provided to support the case that the particular input variable is not yet known. **Tags:** xml.sequenceOffset=40

Table A.70: SwAxisIndividual

Class	**SwBaseType**			
Note	This meta-class represents a base type used within ECU software. **Tags:** atp.recommendedPackage=BaseTypes			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *BaseType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table A.71: SwBaseType

Enumeration	**SwCalibrationAccessEnum**
Note	Determines the access rights to a data object w.r.t. measurement and calibration.
Aggregated by	ModeDeclarationGroupPrototype.swCalibrationAccess, SwCalprmAxis.swCalibrationAccess, SwDataDefProps.swCalibrationAccess
Literal	***Description***
notAccessible	The element will not be accessible via MCD tools, i.e. will not appear in the ASAP file. **Tags:** atp.EnumerationLiteralIndex=0
readOnly	The element will only appear as read-only in an ASAP file. **Tags:** atp.EnumerationLiteralIndex=1
readWrite	The element will appear in the ASAP file with both read and write access. **Tags:** atp.EnumerationLiteralIndex=2

Table A.72: SwCalibrationAccessEnum

Class	**SwCalprmAxis**			
Note	This element specifies an individual input parameter axis (abscissa).			
Base	*ARObject*			
Aggregated by	SwCalprmAxisSet.swCalprmAxis			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
category	CalprmAxisCategoryEnum	0..1	attr	This property specifies the category of a particular axis. **Tags:** xml.sequenceOffset=30
displayFormat	DisplayFormatString	0..1	attr	This property specifies how the axis values shall be displayed e.g. in documents or in measurement and calibration tools. **Tags:** xml.sequenceOffset=100

▽

△

Class	SwCalprmAxis			
swAxisIndex	AxisIndexType	0..1	attr	This attribute specifies which axis is specified by the containing SwCalprmAxis. For example in a curve this is usually "1". In a map this is "1" or "2". **Tags:** xml.sequenceOffset=20
swCalibration Access	SwCalibrationAccess Enum	0..1	attr	Describes the applicability of parameters and variables. **Tags:** xml.sequenceOffset=90
swCalprmAxis TypeProps	SwCalprmAxisType Props	0..1	aggr	specific properties depending on the type of the axis. **Tags:** xml.roleElement=false xml.roleWrapperElement=false xml.sequenceOffset=40 xml.typeElement=true xml.typeWrapperElement=false

Table A.73: SwCalprmAxis

Class	SwCalprmAxisSet			
Note	This element specifies the input parameter axes (abscissas) of parameters (and variables, if these are used adaptively).			
Base	*ARObject*			
Aggregated by	SwDataDefProps.swCalprmAxisSet			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
swCalprmAxis	SwCalprmAxis	*	aggr	One axis belonging to this SwCalprmAxisSet **Tags:** atp.Splitkey=swCalprmAxis xml.roleElement=true xml.roleWrapperElement=false xml.sequenceOffset=20 xml.typeElement=false xml.typeWrapperElement=false

Table A.74: SwCalprmAxisSet

Class	SwComponentPrototype			
Note	Role of a software component within a composition.			
Base	*ARObject*, *AtpFeature*, *AtpPrototype*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, CompositionSwComponentType.component			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
type	SwComponentType	0..1	tref	Type of the instance. **Stereotypes:** isOfType **Tags:** atp.Splitkey=type

Table A.75: SwComponentPrototype

Class	***SwComponentType*** (abstract)
Note	Base class for AUTOSAR software components.
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*

▽

△

Class	**SwComponentType** (abstract)			
Subclasses	*AtomicSwComponentType*, CompositionSwComponentType, ParameterSwComponentType			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
consistency Needs	ConsistencyNeeds	*	aggr	This represents the collection of `ConsistencyNeeds` owned by the enclosing `SwComponentType`. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=consistencyNeeds, consistencyNeeds.shortName, consistencyNeeds.variationPoint.shortLabel vh.latestBindingTime=preCompileTime This Attribute is only used by the AUTOSAR Classic Platform.
port	PortPrototype	*	aggr	The `PortPrototype`s through which this `SwComponentType` can communicate. The aggregation of `PortPrototype` is subject to variability with the purpose to support the conditional existence of `PortPrototype`s. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=port, port.shortName, port.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
portGroup	PortGroup	*	aggr	A port group being part of this component. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=portGroup, portGroup.shortName, portGroup.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
swcMapping Constraint	SwComponentMapping Constraints	*	ref	Reference to constraints that are valid for this SwComponentType. **Tags:** atp.Splitkey=swcMappingConstraint This Attribute is only used by the AUTOSAR Classic Platform.
swComponent Documentation	SwComponent Documentation	0..1	aggr	This adds a documentation to the `SwComponentType`. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=swComponentDocumentation, swComponentDocumentation.variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=-10
unitGroup	UnitGroup	*	ref	This allows for the specification of which `UnitGroup`s are relevant in the context of referencing `SwComponentType`. **Tags:** atp.Splitkey=unitGroup This Attribute is only used by the AUTOSAR Classic Platform.

Table A.76: SwComponentType

Class	«atpVariation» **SwDataDefProps**
Note	This class is a collection of properties relevant for data objects under various aspects. One could consider this class as a "pattern of inheritance by aggregation". The properties can be applied to all objects of all classes in which SwDataDefProps is aggregated. Note that not all of the attributes or associated elements are useful all of the time. Hence, the process definition (e.g. expressed with an OCL or a Document Control Instance MSR-DCI) has the task of implementing limitations. SwDataDefProps covers various aspects: • Structure of the data element for calibration use cases: is it a single value, a curve, or a map, but also the recordLayouts which specify how such elements are mapped/converted to the DataTypes in the programming language (or in AUTOSAR). This is mainly expressed by properties like swRecordLayout and swCalprmAxisSet • Implementation aspects, mainly expressed by swImplPolicy, swAddrMethod, baseType, implementationDataType and additionalNativeTypeQualifier • Access policy for the MCD system, mainly expressed by swCalibrationAccess • Semantics of the data element, mainly expressed by compuMethod and/or unit, dataConstr, invalid Value • Code generation policy provided by swRecordLayout **Tags:** vh.latestBindingTime=codeGenerationTime
Base	*ARObject*
Aggregated by	*AutosarDataType*.swDataDefProps, CompositeNetworkRepresentation.networkRepresentation, Cpp ImplementationDataTypeElement.swDataDefProps, *DataPrototype*.swDataDefProps, DataPrototype TransformationProps.networkRepresentationProps, DiagnosticDataElement.swDataDefProps, Diagnostic EnvDataElementCondition.swDataDefProps, DiagnosticExtendedDataRecordElement.swDataDefProps, DiagnosticSovdPrimitiveContentElement.swDataDefProps, DltArgumentProps.networkRep, FlatInstance Descriptor.swDataDefProps, ImplementationDataTypeElement.swDataDefProps, InstantiationDataDef Props.swDataDefProps, ISignal.networkRepresentationProps, McDataInstance.resultingProperties, ParameterAccess.swDataDefProps, PerInstanceMemory.swDataDefProps, *ReceiverComSpec*.network Representation, SecurityEventContextDataElement.networkRepresentation, *SenderComSpec*.network Representation, SomeipDataPrototypeTransformationProps.networkRepresentation, SwPointerTarget Props.swDataDefProps, SwServiceArg.swDataDefProps, SwSystemconst.swDataDefProps, System Signal.physicalProps

Attribute	**Type**	**Mult.**	**Kind**	**Note**
additionalNative TypeQualifier	NativeDeclarationString	0..1	attr	This attribute is used to declare native qualifiers of the programming language which can neither be deduced from the baseType (e.g. because the data object describes a pointer) nor from other more abstract attributes. Examples are qualifiers like "volatile", "strict" or "enum" of the C-language. All such declarations have to be put into one string. **Tags:** xml.sequenceOffset=235
annotation	Annotation	*	aggr	This aggregation allows to add annotations (yellow pads ...) related to the current data object. **Tags:** atp.Splitkey=annotation xml.roleElement=true xml.roleWrapperElement=true xml.sequenceOffset=20 xml.typeElement=false xml.typeWrapperElement=false
baseType	SwBaseType	0..1	ref	Base type associated with the containing data object. **Tags:** atp.Splitkey=baseType xml.sequenceOffset=50

▽

△

Class	«atpVariation» **SwDataDefProps**			
compuMethod	CompuMethod	0..1	ref	Computation method associated with the semantics of this data object. **Tags:** atp.Splitkey=compuMethod xml.sequenceOffset=180
dataConstr	DataConstr	0..1	ref	Data constraint for this data object. **Tags:** atp.Splitkey=dataConstr xml.sequenceOffset=190
displayFormat	DisplayFormatString	0..1	attr	This property describes how a number is to be rendered e.g. in documents or in a measurement and calibration system. **Tags:** xml.sequenceOffset=210
display Presentation	DisplayPresentation Enum	0..1	attr	This attribute controls the presentation of the related data for measurement and calibration tools.
implementation DataType	AbstractImplementation DataType	0..1	ref	This association denotes the ImplementationDataType of a data declaration via its aggregated SwDataDefProps. It is used whenever a data declaration is not directly referring to a base type. Especially • redefinition of an ImplementationDataType via a "typedef" to another ImplementationDatatype • the target type of a pointer (see SwPointerTarget Props), if it does not refer to a base type directly • the data type of an array or record element within an ImplementationDataType, if it does not refer to a base type directly • the data type of an SwServiceArg, if it does not refer to a base type directly **Tags:** atp.Splitkey=implementationDataType xml.sequenceOffset=215
invalidValue	ValueSpecification	0..1	aggr	Optional value to express invalidity of the actual data element. **Tags:** atp.Splitkey=invalidValue xml.sequenceOffset=255
stepSize	Float	0..1	attr	This attribute can be used to define a value which is added to or subtracted from the value of a DataPrototype when using up/down keys while calibrating.
swAddrMethod	SwAddrMethod	0..1	ref	Addressing method related to this data object. Via an association to the same SwAddrMethod it can be specified that several DataPrototypes shall be located in the same memory without already specifying the memory section itself. **Tags:** atp.Splitkey=swAddrMethod xml.sequenceOffset=30
swAlignment	AlignmentType	0..1	attr	The attribute describes the intended typical alignment of the DataPrototype. If the attribute is not defined the alignment is determined by the swBaseType size and the memoryAllocationKeywordPolicy of the referenced Sw AddrMethod. **Tags:** xml.sequenceOffset=33

▽

△

Class	«atpVariation» **SwDataDefProps**			
swBit Representation	SwBitRepresentation	0..1	aggr	Description of the binary representation in case of a bit variable. **Tags:** atp.Splitkey=swBitRepresentation xml.sequenceOffset=60
swCalibration Access	SwCalibrationAccess Enum	0..1	attr	Specifies the read or write access by MCD tools for this data object. **Tags:** xml.sequenceOffset=70
swCalprmAxis Set	SwCalprmAxisSet	0..1	aggr	This specifies the properties of the axes in case of a curve or map etc. This is mainly applicable to calibration parameters. **Tags:** atp.Splitkey=swCalprmAxisSet xml.sequenceOffset=90
swComparison Variable	SwVariableRefProxy	*	aggr	Variables used for comparison in an MCD process. **Tags:** atp.Splitkey=swComparisonVariable xml.sequenceOffset=170 xml.typeElement=false
swData Dependency	SwDataDependency	0..1	aggr	Describes how the value of the data object has to be calculated from the value of another data object (by the MCD system). **Tags:** atp.Splitkey=swDataDependency xml.sequenceOffset=200
swHostVariable	SwVariableRefProxy	0..1	aggr	Contains a reference to a variable which serves as a host-variable for a bit variable. Only applicable to bit objects. **Tags:** atp.Splitkey=swHostVariable xml.sequenceOffset=220 xml.typeElement=false
swImplPolicy	SwImplPolicyEnum	0..1	attr	Implementation policy for this data object. **Tags:** xml.sequenceOffset=230
swIntended Resolution	Numerical	0..1	attr	The purpose of this element is to describe the requested quantization of data objects early on in the design process. The resolution ultimately occurs via the conversion formula present (compuMethod), which specifies the transition from the physical world to the standardized world (and vice-versa) (here, "the slope per bit" is present implicitly in the conversion formula). In the case of a development phase without a fixed conversion formula, a pre-specification can occur through swIntendedResolution. The resolution is specified in the physical domain according to the property "unit". **Tags:** xml.sequenceOffset=240
swInterpolation Method	Identifier	0..1	attr	This is a keyword identifying the mathematical method to be applied for interpolation. The keyword needs to be related to the interpolation routine which needs to be invoked. **Tags:** xml.sequenceOffset=250

▽

△

Class	«atpVariation» **SwDataDefProps**			
swIsVirtual	Boolean	0..1	attr	This element distinguishes virtual objects. Virtual objects do not appear in the memory, their derivation is much more dependent on other objects and hence they shall have a swDataDependency . **Tags:** xml.sequenceOffset=260
swPointerTarget Props	SwPointerTargetProps	0..1	aggr	Specifies that the containing data object is a pointer to another data object. **Tags:** atp.Splitkey=swPointerTargetProps xml.sequenceOffset=280
swRecord Layout	SwRecordLayout	0..1	ref	Record layout for this data object. **Tags:** atp.Splitkey=swRecordLayout xml.sequenceOffset=290
swRefresh Timing	MultidimensionalTime	0..1	aggr	This element specifies the frequency in which the object involved shall be or is called or calculated. This timing can be collected from the task in which write access processes to the variable run. But this cannot be done by the MCD system. So this attribute can be used in an early phase to express the desired refresh timing and later on to specify the real refresh timing. **Tags:** atp.Splitkey=swRefreshTiming xml.sequenceOffset=300
swTextProps	SwTextProps	0..1	aggr	the specific properties if the data object is a text object. **Tags:** atp.Splitkey=swTextProps xml.sequenceOffset=120
swValueBlock Size	Numerical	0..1	attr	This represents the size of a Value Block **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=80
swValueBlock SizeMult (ordered)	Numerical	*	attr	This attribute is used to specify the dimensions of a value block (VAL_BLK) for the case that that value block has more than one dimension. The dimensions given in this attribute are ordered such that the first entry represents the first dimension, the second entry represents the second dimension, and so on. For one-dimensional value blocks the attribute swValue BlockSize shall be used and this attribute shall not exist. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
unit	Unit	0..1	ref	Physical unit associated with the semantics of this data object. This attribute applies if no compuMethod is specified. If both units (this as well as via compuMethod) are specified the units shall be compatible. **Tags:** atp.Splitkey=unit xml.sequenceOffset=350

▽

△

Class	«atpVariation» **SwDataDefProps**			
valueAxisData Type	ApplicationPrimitive DataType	0..1	ref	The referenced ApplicationPrimitiveDataType represents the primitive data type of the value axis within a compound primitive (e.g. curve, map). It supersedes CompuMethod, Unit, and BaseType. **Tags:** atp.Splitkey=valueAxisDataType xml.sequenceOffset=355

Table A.77: SwDataDefProps

Enumeration	**SwImplPolicyEnum**
Note	Specifies the implementation strategy with respect to consistency mechanisms of variables.
Aggregated by	BswInternalTriggeringPoint.swImplPolicy, InternalTriggeringPoint.swImplPolicy, SwDataDefProps.swImplPolicy, Trigger.swImplPolicy
Literal	*Description*
const	forced implementation such that the running software within the ECU shall not modify it. For example implemented with the "const" modifier in C. This can be applied for parameters (not for those in NVRAM) as well as argument data prototypes. **Tags:** atp.EnumerationLiteralIndex=0
fixed	This data element is fixed. In particular this indicates, that it might also be implemented e.g. as in place data, (#DEFINE). **Tags:** atp.EnumerationLiteralIndex=1
measurementPoint	The data element is created for measurement purposes only. The data element is never read directly within the ECU software. In contrast to a "standard" data element in an unconnected provide port is, this unconnection is guaranteed for measurementPoint data elements. **Tags:** atp.EnumerationLiteralIndex=2
queued	The content of the data element is queued and the data element has 'event' semantics, i.e. data elements are stored in a queue and all data elements are processed in 'first in first out' order. The queuing is intended to be implemented by RTE Generator. This value is not applicable for parameters. **Tags:** atp.EnumerationLiteralIndex=3
standard	This is applicable for all kinds of data elements. For variable data prototypes the 'last is best' semantics applies. For parameter there is no specific implementation directive. **Tags:** atp.EnumerationLiteralIndex=4

Table A.78: SwImplPolicyEnum

Class	**SwcImplementation**			
Note	This meta-class represents a specialization of the general Implementation meta-class with respect to the usage in application software. **Tags:** atp.recommendedPackage=SwcImplementations This Class is only used by the AUTOSAR Classic Platform.			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *Implementation*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
behavior	SwcInternalBehavior	0..1	ref	The internal behavior implemented by this Implementation. **Tags:** atp.Splitkey=behavior

▽

△

Class	**SwcImplementation**			
perInstance MemorySize	PerInstanceMemory Size	*	aggr	Allows a definition of the size of the per-instance memory for this implementation. The aggregation of PerInstance MemorySize is subject to variability with the purpose to support variability in the software components implementations. Typically different algorithms in the implementation are requiring different number of memory objects, in this case PerInstanceMemory. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=perInstanceMemorySize, perInstance MemorySize.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
required RTEVendor	String	0..1	attr	Identify a specific RTE vendor. This information is potentially important at the time of integrating (in particular: linking) the application code with the RTE. The semantics is that (if the association exists) the corresponding code has been created to fit to the vendor-mode RTE provided by this specific vendor. Attempting to integrate the code with another RTE generated in vendor mode is in general not possible.

Table A.79: SwcImplementation

Class	**SwcInternalBehavior**			
Note	The `SwcInternalBehavior` of an `AtomicSwComponentType` describes the relevant aspects of the software-component with respect to the RTE, i.e. the `RunnableEntity`s and the `RTEEvent`s they respond to.			
Base	*ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *InternalBehavior*, *Multilanguage Referrable*, *Referrable*			
Aggregated by	*AtomicSwComponentType*.internalBehavior, *AtpClassifier*.atpFeature			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
arTypedPer Instance Memory	VariableDataPrototype	*	aggr	Defines an AUTOSAR typed memory-block that needs to be available for each instance of the SW-component. This is typically only useful if `supportsMultipleInstantiation` is set to "true" or if the component defines NVRAM access via permanent blocks. The aggregation of `arTypedPerInstanceMemory` is subject to variability with the purpose to support variability in the software component's implementations. Typically different algorithms in the implementation are requiring different number of memory objects. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=arTypedPerInstanceMemory, arTypedPer InstanceMemory.shortName, arTypedPerInstance Memory.variationPoint.shortLabel vh.latestBindingTime=preCompileTime

▽

△

Class	SwcInternalBehavior			
event	RTEEvent	*	aggr	This is a `RTEEvent` specified for the particular `SwcInternalBehavior`. The aggregation of `RTEEvent` is subject to variability with the purpose to support the conditional existence of `RTEEvent`s. Note: the number of `RTEEvent`s might vary due to the conditional existence of `PortPrototype`s using `DataReceivedEvent`s or due to different scheduling needs of algorithms. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=event, event.shortName, event.variation Point.shortLabel vh.latestBindingTime=preCompileTime
exclusiveArea Policy	SwcExclusiveArea Policy	*	aggr	Options how to generate the ExclusiveArea related APIs. When no SwcExclusiveAreaPolicy is specified for an ExclusiveArea the default values apply. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=exclusiveAreaPolicy, exclusiveArea Policy.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
explicitInter Runnable Variable	VariableDataPrototype	*	aggr	Implement state message semantics for establishing communication among runnables of the same component. The aggregation of explicitInterRunnable Variable is subject to variability with the purpose to support variability in the software components implementations. Typically different algorithms in the implementation are requiring different number of memory objects. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=explicitInterRunnableVariable, explicitInter RunnableVariable.shortName, explicitInterRunnable Variable.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
implicitInter Runnable Variable	VariableDataPrototype	*	aggr	Implement state message semantics for establishing communication among runnables of the same component. The aggregation of implicitInterRunnable Variable is subject to variability with the purpose to support variability in the software components implementations. Typically different algorithms in the implementation are requiring different number of memory objects. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=implicitInterRunnableVariable, implicitInter RunnableVariable.shortName, implicitInterRunnable Variable.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
includedData TypeSet	IncludedDataTypeSet	*	aggr	The `includedDataTypeSet` is used by a software component for its implementation. **Tags:** atp.Splitkey=includedDataTypeSet
includedMode Declaration GroupSet	IncludedMode DeclarationGroupSet	*	aggr	This aggregation represents the included Mode DeclarationGroups **Tags:** atp.Splitkey=includedModeDeclarationGroupSet

▽

△

Class	**SwcInternalBehavior**			
instantiation DataDefProps	InstantiationDataDefProps	*	aggr	The purpose of this is that within the context of a given SwComponentType some data def properties of individual instantiations can be modified. The aggregation of InstantiationDataDefProps is subject to variability with the purpose to support the conditional existence of Port Prototypes and component local memories like "per InstanceParameter" or "arTypedPerInstanceMemory". **Stereotypes:** atpVariation **Tags:** atp.Splitkey=instantiationDataDefProps, instantiationDataDefProps.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
perInstanceMemory	PerInstanceMemory	*	aggr	Defines a per-instance memory object needed by this software component. The aggregation of PerInstanceMemory is subject to variability with the purpose to support variability in the software components implementations. Typically different algorithms in the implementation are requiring different number of memory objects. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=perInstanceMemory, perInstanceMemory.shortName, perInstanceMemory.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
perInstanceParameter	ParameterDataPrototype	*	aggr	Defines parameter(s) or characteristic value(s) that needs to be available for each instance of the software-component. This is typically only useful if supportsMultipleInstantiation is set to "true". The aggregation of perInstanceParameter is subject to variability with the purpose to support variability in the software components implementations. Typically different algorithms in the implementation are requiring different number of memory objects. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=perInstanceParameter, perInstanceParameter.shortName, perInstanceParameter.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
portAPIOption	PortAPIOption	*	aggr	Options for generating the signature of port-related calls from a runnable to the RTE and vice versa. The aggregation of PortPrototypes is subject to variability with the purpose to support the conditional existence of ports. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=portAPIOption, portAPIOption.port, portAPIOption.variationPoint.shortLabel vh.latestBindingTime=preCompileTime

▽

△

Class	SwcInternalBehavior			
runnable	RunnableEntity	*	aggr	This is a `RunnableEntity` specified for the particular `SwcInternalBehavior`. The aggregation of `RunnableEntity` is subject to variability with the purpose to support the conditional existence of `RunnableEntity`s. Note: the number of `RunnableEntity`s might vary due to the conditional existence of `PortPrototype`s using `DataReceivedEvent`s or due to different scheduling needs of algorithms. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=runnable, runnable.shortName, runnable.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
service Dependency	SwcService Dependency	*	aggr	Defines the requirements on AUTOSAR Services for a particular item. The aggregation of SwcServiceDependency is subject to variability with the purpose to support the conditional existence of ports as well as the conditional existence of ServiceNeeds. The SwcServiceDependency owned by an SwcInternal Behavior can be located in a different physical file in order to support that SwcServiceDependency might be provided in later development steps or even by different expert domain (e.g OBD expert for Obd related Service Needs) tools. Therefore the aggregation is <<atp Splitable>>. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=serviceDependency, service Dependency.shortName, serviceDependency.variation Point.shortLabel vh.latestBindingTime=preCompileTime
shared Parameter	ParameterData Prototype	*	aggr	Defines parameter(s) or characteristic value(s) shared between SwComponentPrototypes of the same Sw ComponentType The aggregation of sharedParameter is subject to variability with the purpose to support variability in the software components implementations. Typically different algorithms in the implementation are requiring different number of memory objects. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=sharedParameter, sharedParameter.short Name, sharedParameter.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
supports Multiple Instantiation	Boolean	0..1	attr	Indicate whether the corresponding software-component can be multiply instantiated on one ECU. In this case the attribute will result in an appropriate component API on programming language level (with or without instance handle).
variationPoint Proxy	VariationPointProxy	*	aggr	Proxy of a variation points in the C/C++ implementation. **Tags:** atp.Splitkey=variationPointProxy, variationPoint Proxy.shortName

Table A.80: SwcInternalBehavior

Class	**System**			
Note	The top level element of the System Description. The System description defines five major elements: Topology, Software, Communication, Mapping and Mapping Constraints. The System element directly aggregates the elements describing the Software, Mapping and Mapping Constraints; it contains a reference to an ASAM FIBEX description specifying Communication and Topology. **Tags:** atp.recommendedPackage=Systems			
Base	*ARElement*, *ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *UploadableDesignElement*, *UploadablePackageElement*			
Aggregated by	ARPackage.element, *AtpClassifier*.atpFeature			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
clientId DefinitionSet	ClientIdDefinitionSet	*	ref	Set of Client Identifiers that are used for inter-ECU client-server communication in the System. **Tags:** atp.Splitkey=clientIdDefinitionSet This Attribute is only used by the AUTOSAR Classic Platform.
containerIPdu HeaderByte Order	ByteOrderEnum	0..1	attr	Defines the byteOrder of the header in ContainerIPdus. This Attribute is only used by the AUTOSAR Classic Platform.
ecuExtract Version	RevisionLabelString	0..1	attr	Version number of the Ecu Extract. This Attribute is only used by the AUTOSAR Classic Platform.
fibexElement	FibexElement	*	ref	Reference to ASAM FIBEX elements specifying Communication and Topology. All Fibex Elements used within a System Description shall be referenced from the System Element. atpVariation: In order to describe a product-line, all Fibex Elements can be optional. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=fibexElement, fibexElement.fibexElement, fibexElement.variationPoint.shortLabel vh.latestBindingTime=postBuild
interpolation Routine MappingSet	InterpolationRoutine MappingSet	*	ref	This reference identifies the InterpolationRoutineMapping Sets that are relevant in the context of the enclosing System. **Tags:** atp.Splitkey=interpolationRoutineMappingSet This Attribute is only used by the AUTOSAR Classic Platform.
j1939Shared AddressCluster	J1939SharedAddress Cluster	*	aggr	Collection of J1939Clusters that share a common address space for the routing of messages. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=j1939SharedAddressCluster, j1939Shared AddressCluster.shortName, j1939SharedAddress Cluster.variationPoint.shortLabel vh.latestBindingTime=postBuild This Attribute is only used by the AUTOSAR Classic Platform.



△

Class	System			
mapping	SystemMapping	*	aggr	Aggregation of all mapping aspects (mapping of SW components to ECUs, mapping of data elements to signals, and mapping constraints). In order to support OEM / Tier 1 interaction and shared development for one common System this aggregation is atpSplitable and atpVariation. The content of System Mapping can be provided by several parties using different names for the SystemMapping. This element is not required when the System description is used for a network-only use-case. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=mapping, mapping.shortName, mapping.variationPoint.shortLabel vh.latestBindingTime=postBuild
pncVector Length	PositiveInteger	0..1	attr	Length of the partial networking request release information vector (in bytes).
pncVectorOffset	PositiveInteger	0..1	attr	Absolute offset (with respect to the NM-PDU) of the partial networking request release information vector that is defined in bytes as an index starting with 0.
rootSoftware Composition	RootSwComposition Prototype	0..1	aggr	Aggregation of the root software composition, containing all software components in the System in a hierarchical structure. This element is not required when the System description is used for a network-only use-case. atpVariation: The RootSwCompositionPrototype can vary. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=rootSoftwareComposition, rootSoftware Composition.shortName, rootSoftware Composition.variationPoint.shortLabel vh.latestBindingTime=systemDesignTime This Attribute is only used by the AUTOSAR Classic Platform.
swCluster	CpSoftwareCluster	*	ref	CP Software Clusters of this System **Stereotypes:** atpVariation **Tags:** atp.Splitkey=swCluster, swCluster.cpSoftwareCluster, sw Cluster.variationPoint.shortLabel vh.latestBindingTime=systemDesignTime This Attribute is only used by the AUTOSAR Classic Platform.
systemCom SpecDefinition	SystemComSpec DefinitionSet	*	ref	Reference to the set of ComSpec definitions that are used for inter-ECU communication in the System. **Tags:** atp.Splitkey=systemComSpecDefinition
system Documentation	Chapter	*	aggr	Possibility to provide additional documentation while defining the System. The System documentation can be composed of several chapters. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=systemDocumentation, system Documentation.shortName, system Documentation.variationPoint.shortLabel vh.latestBindingTime=systemDesignTime xml.sequenceOffset=-10 This Attribute is only used by the AUTOSAR Classic Platform.

▽

△

Class	**System**			
systemVersion	RevisionLabelString	0..1	attr	Version number of the System Description.

Table A.81: System

Class	**TimingEvent**			
Note	This event is used to start `RunnableEntity`s that shall be executed periodically.			
Base	*ARObject*, *AbstractEvent*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *RTEEvent*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, SwcInternalBehavior.event			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
offset	TimeValue	0..1	attr	The value makes an assumption about the time offset of the first activation of the `RunnableEntity` triggered by the mapped `TimingEvent` relative to the periodic activation of the time base of this `TimingEvent`. Unit: second.
period	TimeValue	0..1	attr	Period of timing event in seconds. The value of this attribute shall be greater than zero.

Table A.82: TimingEvent

Class	***Traceable*** (abstract)			
Note	This meta class represents the ability to be subject to tracing within an AUTOSAR model. Note that it is expected that its subclasses inherit either from MultilanguageReferrable or from Identifiable. Nevertheless it also inherits from MultilanguageReferrable in order to provide a common reference target for all Traceables.			
Base	*ARObject*, *MultilanguageReferrable*, *Referrable*			
Subclasses	StructuredReq, *TimingConstraint*, TraceableTable, TraceableText			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
trace	Traceable	*	ref	This association represents the ability to trace to upstream requirements / constraints. This supports for example the bottom up tracing ProjectObjectives <- MainRequirements <- Features <- RequirementSpecs <- BSW/AI **Tags:** atp.Splitkey=trace xml.sequenceOffset=20

Table A.83: Traceable

Class	**Unit**
Note	This is a physical measurement unit. All units that might be defined should stem from SI units. In order to convert one unit into another factor and offset are defined. For the calculation from SI-unit to the defined unit the factor (factorSiToUnit) and the offset (offsetSiTo Unit) are applied as follows: x [{unit}] := y * {siUnit} * factorSiToUnit [{unit}/{siUnit}] + offsetSiToUnit [{unit}] For the calculation from a unit to SI-unit the reciprocal of the factor (factorSiToUnit) and the negation of the offset (offsetSiToUnit) are applied. y [{siUnit}] := (x*{unit} - offsetSiToUnit [{unit}]) / factorSiToUnit [{unit}/{siUnit}] **Tags:** atp.recommendedPackage=Units

▽

△

Class	Unit			
Base	ARElement, ARObject, CollectableElement, Identifiable, MultilanguageReferrable, Packageable Element, Referrable			
Aggregated by	ARPackage.element			
Attribute	Type	Mult.	Kind	Note
displayName	SingleLanguageUnit Names	0..1	aggr	This specifies how the unit shall be displayed in documents or in user interfaces of tools.The displayName corresponds to the Unit.Display in an ASAM MCD-2MC file. **Tags:** atp.Splitkey=displayName xml.sequenceOffset=20
factorSiToUnit	Float	0..1	attr	This is the factor for the conversion from SI Units to units. The inverse is used for conversion from units to SI Units. **Tags:** xml.sequenceOffset=30
offsetSiToUnit	Float	0..1	attr	This is the offset for the conversion from and to siUnits. **Tags:** xml.sequenceOffset=40
physical Dimension	PhysicalDimension	0..1	ref	This association represents the physical dimension to which the unit belongs to. Note that only values with units of the same physical dimensions might be converted. **Tags:** atp.Splitkey=physicalDimension xml.sequenceOffset=50

Table A.84: Unit

Class	UnitGroup			
Note	This meta-class represents the ability to specify a logical grouping of units.The category denotes the unit system that the referenced units are associated to. In this way, e.g. country-specific unit systems (CATEGORY="COUNTRY") can be defined as well as specific unit systems for certain application domains. In the same way a group of equivalent units, can be defined which are used in different countries, by setting CATEGORY="EQUIV_UNITS". KmPerHour and MilesPerHour could such be combined to one group named "vehicle_speed". The unit MeterPerSec would not belong to this group because it is normally not used for vehicle speed. But all of the mentioned units could be combined to one group named "speed". Note that the UnitGroup does not ensure the physical compliance of the units. This is maintained by the physical dimension. **Tags:** atp.recommendedPackage=UnitGroups			
Base	ARElement, ARObject, CollectableElement, Identifiable, MultilanguageReferrable, Packageable Element, Referrable			
Aggregated by	ARPackage.element			
Attribute	Type	Mult.	Kind	Note
unit	Unit	*	ref	This represents one particular unit in the UnitGroup. **Tags:** atp.Splitkey=unit xml.sequenceOffset=20

Table A.85: UnitGroup

Class	**ValueSpecification** (abstract)			
Note	Base class for expressions leading to a value which can be used to initialize a data object.			
Base	*ARObject*			
Subclasses	*AbstractRuleBasedValueSpecification*, ApplicationValueSpecification, *CompositeValueSpecification*, ConstantReference, NotAvailableValueSpecification, NumericalValueSpecification, ReferenceValue Specification, TextValueSpecification			
Aggregated by	ApplicationAssocMapElementValueSpecification.key, ApplicationAssocMapElementValue Specification.value, ArrayValueSpecification.element, CalibrationParameterValue.applInitValue, CalibrationParameterValue.implInitValue, ConstantSpecification.valueSpec, CryptoService Key.developmentValue, DiagnosticEnvDataCondition.compareValue, DiagnosticEnvDataElement Condition.compareValue, DiagnosticEnvSovdDataCondition.compareValue, FieldSenderComSpec.init Value, ISignal.initValue, ISignal.timeoutSubstitutionValue, NonqueuedReceiverComSpec.initValue, NonqueuedReceiverComSpec.timeoutSubstitutionValue, NonqueuedSenderComSpec.initValue, Nv ProvideComSpec.ramBlockInitValue, NvProvideComSpec.romBlockInitValue, NvRequireComSpec.init Value, ParameterDataPrototype.initValue, ParameterProvideComSpec.initValue, ParameterRequireCom Spec.initValue, PersistencyDataRequiredComSpec.initValue, PersistencyKeyValuePair.initValue, Port DefinedArgumentValue.value, PortPrototypeBlueprintInitValue.value, RecordValueSpecification.field, *StateManagementCompareCondition*.compareValue, SwDataDefProps.invalidValue, VariableData Prototype.initValue			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
shortLabel	Identifier	0..1	attr	This can be used to identify particular value specifications for human readers, for example elements of a record type.

Table A.86: ValueSpecification

Class	**VariableDataPrototype**			
Note	A `VariableDataPrototype` represents a formalized generic piece of information that is typically mutable by the application software layer. `VariableDataPrototype` is used in various contexts and the specific context gives the otherwise generic `VariableDataPrototype` a dedicated semantics.			
Base	*ARObject*, *AtpFeature*, *AtpPrototype*, *AutosarDataPrototype*, *DataPrototype*, *Identifiable*, *Multilanguage Referrable*, *Referrable*			
Aggregated by	ApplicationInterface.indication, *AtpClassifier*.atpFeature, BswInternalBehavior.arTypedPerInstance Memory, BswModuleDescription.providedData, BswModuleDescription.requiredData, BulkNvData Descriptor.bulkNvBlock, DiagnosticSovdAccessArgument.contentObject, *InternalBehavior*.staticMemory, NvBlockDescriptor.ramBlock, NvDataInterface.nvData, SenderReceiverInterface.dataElement, Service Interface.event, SwcInternalBehavior.arTypedPerInstanceMemory, SwcInternalBehavior.explicitInter RunnableVariable, SwcInternalBehavior.implicitInterRunnableVariable			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
initValue	ValueSpecification	0..1	aggr	Specifies initial value(s) of the VariableDataPrototype **Tags:** atp.Splitkey=initValue

Table A.87: VariableDataPrototype

Class	**Xref**			
Note	This represents a cross-reference within documentation.			
Base	*ARObject*			
Aggregated by	*MixedContentForOverviewParagraph*.xref, *MixedContentForParagraph*.xref, *MixedContentFor Verbatim*.xref			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
label1	SingleLanguageLong Name	0..1	aggr	This allows to specify a replacement text which shall be rendered if showContent is selected.
referrable	Referrable	0..1	ref	This establishes the reference in Autosar style
resolutionPolicy	ResolutionPolicyEnum	0..1	attr	Indicates if the content of the xref element follow a dedicated resolution policy. **Tags:** xml.attribute=true

▽

△

Class	Xref			
showContent	ShowContentEnum	0..1	attr	Indicates if the content of the xref element shall be rendered. **Tags:** xml.attribute=true
showResource AliasName	ShowResourceAlias NameEnum	0..1	attr	This indicates if the alias names of the referenced objects shall be rendered. This means this is some kind of backward searching: look whether there is an alias for the referenced object, if yes, print it. If there is more than one AliasNameSet, Xref might render all of those. If no alias is found and showResourceShortName is set to NoShowShortName, then the shortName of the reference target shall be displayed. By this showResourceAlias Name is similar to showResourceShortName but shows the aliasName instead of the shortName. **Tags:** xml.attribute=true
showResource Category	ShowResource CategoryEnum	0..1	attr	Indicates if the category of the referenced resource shall be rendered. **Tags:** xml.attribute=true
showResource LongName	ShowResourceLong NameEnum	0..1	attr	Indicates if the longName of the referenced resource shall be rendered. **Tags:** xml.attribute=true
showResource Number	ShowResourceNumber Enum	0..1	attr	Indicates if the Number of the referenced resource shall be shown. **Tags:** xml.attribute=true
showResource Page	ShowResourcePage Enum	0..1	attr	Indicates if the page number of the referenced resource shall be shown. **Tags:** xml.attribute=true
showResource ShortName	ShowResourceShort NameEnum	0..1	attr	Indicates if the shortJName of the referenced resource shall be shown. **Tags:** xml.attribute=true
showResource Type	ShowResourceType Enum	0..1	attr	Indicates if the type of the referenced Resource shall be shown. **Tags:** xml.attribute=true
showSee	ShowSeeEnum	0..1	attr	Indicates if the word "see " shall be shown before the reference. **Tags:** xml.attribute=true

Table A.88: Xref