Document Title	Specification of RTE
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	084
Document Classification	Standard

Document Version	3.5.0
Document Status	Final
Part of Release	4.1
Revision	3

Document Change History

Date	Version	Changed by	Change Description
31.03.2014	3.5.0	AUTOSAR Release Management	• Various fixes and clarifications
29.10.2013	3.4.0	AUTOSAR Release Management	• Various fixes and clarifications
05.03.2013	3.3.0	AUTOSAR Administration	• Adapted to new version of meta model • Bypass support added • Support for parameter serialization of client-server communication added • Support for inter-partition communication of BSW modules added • General consolidation and bug fixes

26.10.2011	3.2.0	AUTOSAR Administration	• Adapted to new version of meta model • Support for mixed compu methods with categories `SCALE_LINEAR_AND_TEXTTABLE` and `SCALE_RATIONAL_AND_TEXTTABLE` added • Support for compatibility of partial record types added • Consolidation of signal invalidation, data conversion, and out-of-range handling • General consolidation and bug fixes
29.10.2010	3.1.0	AUTOSAR Administration	• Adapted to new version of meta model • Backward compatibility to implicit communication behavior of AUTOSAR 2.1/3.0/3.1 added • Support of inter-runnable variables extended to composite data types • Clarification which API calls shall be implemented as macro accesses to the component data structure in compatibility mode (see [SWS_Rte_01156]) • General consolidation and bug fixes
18.12.2009	3.0.0	AUTOSAR Administration	• Adapted to new version of meta model • RTE and Basic Software Scheduler merged • Support of multi core architectures added • Re-scaling at ports added • API enhancements added

04.02.2009	2.1.0	AUTOSAR Administration	• updated VFB-Tracing: changes [SWS_Rte_01327],[SWS_Rte_01328] • unconnected R-Ports are supported: changed [SWS_Rte_01329], [SWS_Rte_03019]; added [SWS_Rte_01330], [SWS_Rte_01331], [SWS_Rte_01333], [SWS_Rte_01334], rte_sws_1336, rte_sws_1337, [SWS_Rte_01346], [SWS_Rte_02621], [SWS_Rte_02638], [SWS_Rte_02639], [SWS_Rte_02640], [SWS_Rte_03785], [SWS_Rte_05099], rte_sws_5100, [SWS_Rte_05101], [SWS_Rte_05102] • incompatible function declarations: changed [SWS_Rte_01018], [SWS_Rte_01019], [SWS_Rte_01020]; added [SWS_Rte_05107], [SWS_Rte_05108], [SWS_Rte_05109]; removed rte_sws_6030. • Insufficient RTE server mapping requirement: changed [SWS_Rte_02204].
15.02.2008	2.0.1	AUTOSAR Administration	Layout adaptations
20.12.2007	2.0.0	AUTOSAR Administration	• Adapted to new version of meta model • "RTE ECU Configuration" added • Calibration and measurement revised • Document meta information extended • Small layout adaptations made

31.01.2007	1.1.1	AUTOSAR Administration	• "Advice for users" revised • "Revision Information" added
01.12.2006	1.1.0	AUTOSAR Administration	Updated for AUTOSAR Release 2.1. • Adapted to new version of meta model • New feature 'debouncing of runnable activation' • New feature 'runnable activation offset' • 'Measurement and Calibration' added • Semantics of implicit communication enhanced • Legal disclaimer revised
18.07.2006	1.0.1	AUTOSAR Administration	Second release. Additional features integrated, adapted to updated version of meta-model.
05.05.2006	1.0.0	AUTOSAR Administration	Initial release

Advice to users of AUTOSAR Specification Documents

AUTOSAR specifications may contain exemplary items (exemplary reference models, "use cases", and/or references to exemplary technical solutions, devices, processes or software).

Any such exemplary items are contained in the specifications for illustration purposes only, and they themselves are not part of the AUTOSAR Standard. Neither their presence in such specifications, nor any later documentation of AUTOSAR conformance of products actually implementing such exemplary items, imply that intellectual property rights covering such exemplary items are licensed under the same rules as applicable to the AUTOSAR Standard.

Table of Contents

References

[1] Virtual Functional Bus
AUTOSAR_EXP_VFB

[2] Software Component Template
AUTOSAR_TPS_SoftwareComponentTemplate

[3] Specification of Communication
AUTOSAR_SWS_COM

[4] Specification of Operating System
AUTOSAR_SWS_OS

[5] Specification of ECU Configuration
AUTOSAR_TPS_ECUConfiguration

[6] Methodology
AUTOSAR_TR_Methodology

[7] Specification of ECU State Manager
AUTOSAR_SWS_ECUStateManager

[8] System Template
AUTOSAR_TPS_SystemTemplate

[9] Basic Software Module Description Template
AUTOSAR_TPS_BSWModuleDescriptionTemplate

[10] Generic Structure Template
AUTOSAR_TPS_GenericStructureTemplate

[11] Glossary
AUTOSAR_TR_Glossary

[12] General Requirements on Basic Software Modules
AUTOSAR_SRS_BSWGeneral

[13] Requirements on Runtime Environment
AUTOSAR_SRS_RTE

[14] Specification of Timing Extensions
AUTOSAR_TPS_TimingExtensions

[15] Layered Software Architecture
AUTOSAR_EXP_LayeredSoftwareArchitecture

[16] Specification of ECU Resource Template
AUTOSAR_TPS_ECUResourceTemplate

[17] Specification of I/O Hardware Abstraction
AUTOSAR_SWS_IOHardwareAbstraction

[18] Requirements on Operating System
AUTOSAR_SRS_OS

[19] Requirements on Communication
AUTOSAR_SRS_COM

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

[21] Specification of NVRAM Manager
AUTOSAR_SWS_NVRAMManager

[22] Guide to Multi-Core Systems
AUTOSAR_EXP_MultiCoreGuide

[23] Specification of Development Error Tracer
AUTOSAR_SWS_DevelopmentErrorTracer

[24] Gemeinsames Subset der MISRA C Guidelines
HIS_SubSet_MISRA_C_1.0.3.pdf

[25] Specification of Memory Mapping
AUTOSAR_SWS_MemoryMapping

[26] Specification of Debugging in AUTOSAR
AUTOSAR_SWS_Debugging

[27] General Specification of Basic Software Modules
AUTOSAR_SWS_BSWGeneral

[28] Specification of Compiler Abstraction
AUTOSAR_SWS_CompilerAbstraction

[29] Specification of Standard Types
AUTOSAR_SWS_StandardTypes

[30] Specification of Bit Handling Routines
AUTOSAR_SWS_BFXLibrary

[31] Specification of Diagnostic Log and Trace
AUTOSAR_SWS_DiagnosticLogAndTrace

[32] Collection of constraints on AUTOSAR M1 models
AUTOSAR_TR_AutosarModelConstraints

Note on XML examples

This specification includes examples in XML based on the AUTOSAR metamodel available at the time of writing. These examples are included as illustrations of configurations and their expected outcome but should not be considered part of the specification.

1 Introduction

This document contains the software specification of the AUTOSAR Run-Time Environment (*RTE*) and the *Basic Software Scheduler*. Basically, the RTE together with the OS, AUTOSAR COM and other Basic Software Modules is the implementation of the Virtual Functional Bus concepts (*VFB*, [1]). The RTE implements the AUTOSAR Virtual Functional Bus interfaces and thereby realizes the communication between AUTOSAR software-components.

This document describes how these concepts are realized within the RTE. Furthermore, the Application Programming Interface (*API*) of the RTE and the interaction of the RTE with other basic software modules is specified.

The *Basic Software Scheduler* offers concepts and services to integrate Basic Software Modules Hence, the *Basic Software Scheduler*

- embed *Basic Software Module* implementations into the AUTOSAR OS context
- trigger main processing functions of the *Basic Software Module*s
- apply data consistency mechanisms for the *Basic Software Module*s
- to communicate modes between *Basic Software Module*s

1.1 Scope

This document is intended to be the main reference for developers of an RTE generator tool or of a concrete RTE implementation respectively. The document is also the reference for developers of AUTOSAR software-components and basic software modules that interact with the RTE, since it specifies the application programming interface of the RTE and therefore the mechanisms for accessing the RTE functionality. Furthermore, this specification should be read by the AUTOSAR working groups that are closely related to the RTE (see Section 1.2 below), since it describes the interfaces of the RTE to these modules as well as the behavior / functionality the RTE expects from them.

This document is structured as follows. After this general introduction, Chapter 2 gives a more detailed introduction of the concepts of the RTE. Chapter 3 describes how an RTE is generated in the context of the overall AUTOSAR methodology. Chapter 4 is the central part of this document. It specifies the RTE functionality in detail. The RTE API is described in Chapter 5.

The appendix of this document consists of five parts: Appendix A lists the restrictions to the AUTOSAR metamodel that this version of the RTE specification relies on. Appendix B explicitly lists all external requirements, i.e. all requirements that are not about the RTE itself but specify the assumptions on the environment and the input of an RTE generator. In Appendix C some HIS MISRA rules are listed that are likely to be violated by RTE code, and the rationale why these violations may occur.

Note that Chapters 1 and 2, as well as Appendix C do not contain any requirements and are thus intended for information only.

Chapters 4 and 5 are probably of most interest for developers of an RTE Generator. Chapters 2, 3, 5 are important for developers of AUTOSAR software-components and basic software modules. The most important chapters for related AUTOSAR work packages would be Chapters 4, 5, as well as Appendix B.

The specifications in this document do not define details of the implementation of a concrete RTE or RTE generator respectively. Furthermore, aspects of the ECU- and system-generation process (like e.g. the mapping of SW-Cs to ECUs, or schedulability analysis) are also not in the scope of this specification. Nevertheless, it is specified what input the RTE generator expects from these configuration phases.

1.2 Dependency to other AUTOSAR specifications

The main documents that served as input for the specification of the RTE are the specification of the Virtual Functional Bus [1] and the specification of the Software Component Template [2]. Also of primary importance are the specifications of those Basic Software modules that closely interact with the RTE (or vice versa). These are especially the communication module [3] and the operating system [4]. The main input of an RTE generator is described (among others) in the ECU Configuration Description. Therefore, the corresponding specification [5] is also important for the RTE specification. Furthermore, as the process of RTE generation is an important part of the overall AUTOSAR Methodology, the corresponding document [6] is also considered.

The following list shows the specifications that are closely interdependent to the specification of the RTE:

- Specification of the Virtual Functional Bus [1]
- Specification of the Software Component Template [2]
- Specification of AUTOSAR COM [3]
- Specification of AUTOSAR OS [4]
- Specification of ECU State Manager and Communication Manager [7]
- Specification of ECU Configuration [5]
- Specification of System Description / Generation [8]

- AUTOSAR Methodology [6]
- Specification of BSW Module Description Template [9]
- AUTOSAR Generic Structure Template [10]

1.3 Acronyms and Abbreviations

All abbreviations used throughout this document – except the ones listed here – can be found in the official AUTOSAR glossary [11].

1.4 Technical Terms

All technical terms used throughout this document – except the ones listed here – can be found in the official AUTOSAR glossary [11] or the Software Component Template Specification [2].

Term	Description
mode switch port	The port for receiving (or sending) a mode switch notification. For this purpose, a *mode switch port* is typed by a `ModeSwitchInterface`.
mode user	An *AUTOSAR SW-C* or *AUTOSAR Basic Software Module* that depends on modes is called a mode user. The dependency can occur through a `SwcModeSwitchEvent`/`BswModeSwitchEvent`, a `ModeAccessPoint` for a provided/required `mode switch port`, or a `accessedModeGroup` for a `providedModeGroup`/`requiredModeGroup` `ModeDeclarationGroupPrototype`. See also section 4.4.1.
mode manager	Entering and leaving modes is initiated by a *mode manager*. A *mode manager* is either a software component that provides a p-port typed by a `ModeSwitchInterface` or a BSW module which defines in its `BswModuleDescription` a *ModeDeclarationGroupPrototype* in the role *providedModeGroup*. See also section 4.4.2.
application mode manager	An *application mode manager* is an *AUTOSAR software-component* that provides the service of switching modes. The modes of an `application mode manager` do not have to be standardized.

mode switch notification	The communication of a mode switch from the mode manager to the mode user using either the ModeSwitchInterface or *providedModeGroup* and *requiredModeGroup ModeDeclarationGroupPrototype*s is called *mode switch notification*.
mode machine instance	The instances of mode machines or *ModeDeclarationGroup*s are defined by the *ModeDeclarationGroupPrototype*s of the mode managers. Since a mode switch is not executed instantaneously, The RTE or *Basic Software Scheduler* has to maintain it's own states. For each mode manager's ModeDeclarationGroupPrototype, RTE or *Basic Software Scheduler* has one state machine. This state machine is called *mode machine instance*. For all mode users of the same mode manager's *ModeDeclarationGroupPrototype*, RTE and *Basic Software Scheduler* uses the same *mode machine instance*. See also section 4.4.2.
common mode machine instance	A 'common mode machine instance' is a special 'mode machine instance' shared by BSW Modules and SW-Cs: The RTE Generator creates only one mode machine instance if a *ModeDeclarationGroupPrototype* instantiated in a port of a software-component is synchronized (*synchronizedModeGroup* of a *SwcBswMapping*) with a *providedModeGroup ModeDeclarationGroupPrototype* of a Basic Software Module instance. The related mode machine instance is called common mode machine instance.
ModeDisablingDependency	A *RTEEvent* (respectively a *BswEvent*) that starts a *Runnable Entity* (respectively a *Basic Software Schedulable Entity*) can contain a *disabledMode* (respectively *disabledInMode*) association which *references* a *ModeDeclaration*. This association is called *ModeDisablingDependency* in this document.
mode disabling dependent ExecutableEntity	A mode disabling dependent *Runnable Entity* or a *Basic Software Schedulable Entity* is triggered by an RTEEvent respectively a BswEvent with a ModeDisablingDependency. RTE and *Basic Software Scheduler* prevent the start of those *Runnable Entity* or *Basic Software Schedulable Entity* by the RTEEvent / BswEvent, when the corresponding mode disabling is active. See also section 4.4.1.

mode disabling	When a 'mode disabling' is active, RTE and *Basic Software Scheduler* disables the start of mode disabling dependent ExecutableEntitys. The 'mode disabling' is active during the mode that is referenced in the mode disabling dependency and during the transitions that enter and leave this mode. See also section 4.4.1.
OnEntry ExecutableEntity	A *Runnable Entity* or a *Basic Software Schedulable Entity* that is triggered by a SwcModeSwitchEvent respectively a BswModeSwitchEvent with *ModeActivationKind* 'entry' is triggered on entering the mode. It is called *OnEntry ExecutableEntity*. See also section 4.4.1.
OnExit ExecutableEntity	A *Runnable Entity* or a *Basic Software Schedulable Entity* that is triggered by a SwcModeSwitchEvent respectively a BswModeSwitchEvent with *ModeActivationKind* 'exit' is triggered on exiting the mode. It is called *OnExit ExecutableEntity*. See also section 4.4.1.
OnTransition ExecutableEntity	A *Runnable Entity* or a *Basic Software Schedulable Entity* that is triggered by a SwcModeSwitchEvent respectively a BswModeSwitchEvent with *ModeActivationKind* 'transition' is triggered on a transition between the two specified modes. It is called *OnTransition ExecutableEntity*. See also section 4.4.1.
mode switch acknowledge ExecutableEntity	A *Runnable Entity* or a *Basic Software Schedulable Entity* that is triggered by a ModeSwitchedAckEvent respectively a BswModeSwitchedAckEventconnected to the mode manager's ModeDeclarationGroupPrototype. It is called *mode switch acknowledge ExecutableEntity*. See also section 4.4.1.
server runnable	A server that is triggered by an OperationInvokedEvent. It has a mixed behavior between a runnable and a function call. In certain situations, RTE can implement the client server communication as a simple function call.
server ExecutableEntity	A server that is triggered either by an OperationInvokedEvent or by an BswOperationInvokedEvent. In certain situations, RTE can implement the client server communication as a simple function call.

runnable activation	The activation of a runnable is linked to the RTEEvent that leads to the execution of the runnable. It is defined as the incident that is referred to by the RTEEvent. E.g., for a timing event, the corresponding runnable is activated, when the timer expires, and for a data received event, the runnable is activated when the data is received by the RTE.
Basic Software Schedulable Entity activation	The activation of a *Basic Software Schedulable Entity* is defined as the activation of the task that contains the *Basic Software Schedulable Entity* and eventually includes setting a flag that tells the glue code in the task which *Basic Software Schedulable Entity* is to be executed.
runnable start	A runnable is started by the calling the C-function that implements the runnable from within a started task.
Basic Software Schedulable Entity start	A *Basic Software Schedulable Entity* is started by the calling the C-function that implements the *Basic Software Schedulable Entity* from within a started task.
Trigger Emitter	A *Trigger Emitter* has the ability to release triggers which in turn are activating triggered ExecutableEntitys. *Trigger Emitter* are described by the meta model with provide trigger ports, Trigger in role releasedTrigger, InternalTriggeringPoints and BswInternalTriggeringPoints.
Trigger Source	A *Trigger Source* administrate the particular *Trigger* and informs the RTE or *Basic Software Scheduler* if the *Trigger* is raised. A *Trigger Source* has dedicated provide trigger port(s) or / and *releasedTrigger Trigger*(s) to communicate to the Trigger Sink(s).
Trigger Sink	A *Trigger Sink* relies on the activation of *Runnable Entities* or *Basic Software Schedulable Entities* if a particular *Trigger* is raised. A *Trigger Sink* has a dedicated require trigger port(s) or / and *requiredTrigger Trigger*(s) to communicate to the Trigger Source(s).
trigger port	A PortPrototype which is typed by an TriggerInterface

triggered ExecutableEntity	A *Runnable Entity* or a *Basic Software Schedulable Entity* that is triggered at least by one `ExternalTriggerOccurredEvent` / `BswExternalTriggerOccurredEvent` or `InternalTriggerOccurredEvent` / `BswInternalTriggerOccurredEvent`. In particular cases, the Trigger Event Communication or the *Inter Runnable Triggering* is implemented by RTE or *Basic Software Scheduler* as a direct function call of the *triggered ExecutableEntity* by the triggering *ExecutableEntity*.
triggered runnable	A *Runnable Entity* that is triggered at least by one `ExternalTriggerOccurredEvent` or `InternalTriggerOccurredEvent`. In particular cases, the Trigger Event Communication or the *Inter Runnable Triggering* is implemented by RTE as a direct function call of the *triggered runnable* by the triggering runnable.
triggered Basic Software Schedulable Entity	A *Basic Software Schedulable Entity* that is triggered at least by one `BswExternalTriggerOccurredEvent` or `BswInternalTriggerOccurredEvent`. In particular cases, the *Trigger Event Communication* or the *Inter Basic Software Schedulable Entity Triggering* is implemented by *Basic Software Scheduler* as a direct function call of the *triggered ExecutableEntity* by the triggering *ExecutableEntity*.
execution-instance	An execution-instance of a `ExecutableEntity` is one instance or call context of an `ExecutableEntity` with respect to concurrent execution, see section 4.2.3.
inter-ECU communication	The communication between ECUs, typically using COM is called `inter-ECU` communication in this document.
inter-partition communication	The communication within one ECU but between different partitions, represented by different OS applications, is called `inter-partition` communication in this document. It typically involves the use of OS mechanisms like IOC or trusted function calls. The partitions can be located on different cores or use different memory sections of the ECU.
intra-partition communication	The communication within one partition of one ECU is called `intra-partition` communication. In this case, RTE can make use of internal buffers and queues for communication.

intra-ECU communication	The communication within one ECU is called intra-ECU communication in this document. It is a super set of inter-partition communication and intra-partition communication.
SystemDesignTime Variability	Variability defined with an VariationPoint or *AttributeValueVariationPoint* with latest bindingTime SystemDesignTime.
CodeGenerationTime Variability	Variability defined with an VariationPoint or *AttributeValueVariationPoint* with latest bindingTime CodeGenerationTime.
PreCompileTime Variability	Variability defined with an VariationPoint or *AttributeValueVariationPoint* with latest bindingTime PreCompileTime.
LinkTime Variability	Variability defined with an VariationPoint or *AttributeValueVariationPoint* with latest bindingTime LinkTime.
PreBuild Variability	Variability defined with an VariationPoint or *AttributeValueVariationPoint* with latest bindingTime SystemDesignTime, CodeGenerationTime, PreCompileTime or LinkTime.
PostBuild Variability	Variability defined with an VariationPoint having an postBuildVariantCriterion
Preemption Area	A preemption area defines a set of tasks which are scheduled cooperatively. Therefore tasks of one preemption area are preempting each other only at dedicated schedule points. A schedule point is not allowed to occur during the execution of a RunnableEntity.
Copy Semantic	Copy semantic means, that the accessing entities are able to read or write the "copied" data from their execution context in a non concurrent and non preempting manner. If all accessing entities are in the same Preemption Area this might not require a real physical data copy.
Primitive Data Type	Primitive data types are the types implemented by a boolean, integer (up to 32 bits), floating point, or opaque type (up to 32 bits).
NvBlockSwComponent	NvBlockSwComponent is a SwComponentPrototype typed an NvBlockSwComponentType.
'C' typed PerInstanceMemory	'C' typed PerInstanceMemory is defined with the class PerInstanceMemory. The type of the memory is defined with a 'C' typedef in the attribute typeDefinition.

AutosarDataPrototype implementation	Definitions and declarations for non automatic[1] memory objects which are allocated by the RTE and implementing AutosarDataPrototypes or their belonging status handling.
Implicit Read Access	VariableAccess aggregated in the role dataReadAccess to a VariableDataPrototype
Implicit Write Access	VariableAccess aggregated in the role dataWriteAccess to a VariableDataPrototype
Incoherent Implicit Data Access	An Implicit Read Access or an Implicit Write Access which does not belong to a Coherency Group. Therefore it is NOT referenced by any RteVariableReadAccessRef or RteVariableWriteAccessRef belonging to a RteImplicitCommunication container which RteCoherentAccess parameter is set to true.
Incoherent Implicit Read Access	An Implicit Read Access which does not belong to a Coherency Group. Therefore it is NOT referenced by any RteVariableReadAccessRef belonging to a RteImplicitCommunication container which RteCoherentAccess parameter is set to true.
Incoherent Implicit Write Access	An Implicit Write Access which does not belong to a Coherency Group. Therefore it is NOT referenced by any RteVariableWriteAccessRef belonging to a RteImplicitCommunication container which RteCoherentAccess parameter is set to true.
Coherency Group	A set of Implicit Read Accesses and Implicit Write Accesses for which the RTE cares for data coherency. Please note that in the context of this specification the definition of coherency includes that • read data values of different VariableDataPrototypes have to be from the same age, except the values are changed by Implicit Write Accesses belonging to the Coherency Group • written data values of different VariableDataPrototypes are communicated to readers NOT belonging to the Coherency Group after the last Implicit Write Access belonging to the Coherency Group.

[1]declaration with no static or external specifier defines an automatic variable

Coherent Implicit Data Access	An Implicit Read Access or an Implicit Write Access which belongs to Coherency Group. Therefore it is referenced by a RteVariableReadAccessRef or RteVariableWriteAccessRef belonging to a RteImplicitCommunication container which RteCoherentAccess parameter is set to true.
Coherent Implicit Read Access	An Implicit Read Access which belongs to Coherency Group. Therefore it is referenced by a RteVariableReadAccessRef belonging to a RteImplicitCommunication container which RteCoherentAccess parameter is set to true.
Coherent Implicit Write Access	An Implicit Write Access which belongs to Coherency Group. Therefore it is referenced by a RteVariableReadAccessRef or RteVariableWriteAccessRef belonging to a RteImplicitCommunication container which RteCoherentAccess parameter is set to true.
event semantic	When events are distributed the whole history of received events is of interest, hence they must be queued on receiver side. Therefore the software implementation policy, stated in the swImplPolicy attribute of the SwDataDefProps, will have the value 'queued'(corresponding to event distribution with a queue).
data semantic	When data is distributed, the last received value is of interest (last-is-best semantics). Therefore the software implementation policy, stated in the swImplPolicy attribute of the SwDataDefProps, shouldn't be 'queued'.
client	A client is defined as one ClientServerOperation in one RPortPrototype of one Software Component instance. For the definition of the client neither the number of ServerCallPoints nor RunnableEntity accesses to the ServerCallPoint are relevant. A Software Component instance can appear as several clients to the same server if it defines ServerCallPoints for several PortPrototypes of the same PortInterface's ClientServerOperation.
server	A server is defined as one RunnableEntity which is the target of an OperationInvokedEvent. Call serialization is on activation of RunnableEntity.

1.5 Document Conventions

Requirements in the SRS are referenced using **[SRS_Rte_<n>]** where <**n**> is the requirement id. For example, [SRS_Rte_00098].

Requirements in the SWS are marked with **[SWS_Rte_nnnnn]** ⌈ as the first text in a paragraph. The scope of the requirement is marked with the half brackets. ⌋

Constraints on the input of the RTE are marked with **[constr_<nnnn>]**.

Technical terms are typeset in monospace font, e.g. `Warp Core`.

AUTOSAR Meta Class Names and Attributes are typeset in monospace font, e.g. `ApplicationSwComponentType`. As a general rule, plural forms of AUTOSAR Meta Class Names and Attributes are created by adding "s" to the singular form, e.g. `PortPrototype`s. By this means the document resembles terminology used in the AUTOSAR XML Schema.

AUTOSAR ECU Configuration Parameters are typeset in monospace font, e.g. `RteCodeVendorId`. As a general rule, plural forms of ECU Configuration Parameters are created by adding "s" to the singular form, e.g. `RteEventToTaskMapping`s. By this means the document resembles terminology used in the ARXML file of AUTOSAR ECU Configuration Parameter Definition.

API function calls are also marked with monospace font, like `Rte_EjectWarpCore`.

1.6 Requirements Tracing

The following table references the requirements specified in [12] as well as [13] and links to the fulfillment of these. Please note that if column “Satisfied by” is empty for a specific requirement this means that this requirement is not fulfilled by this document.

Requirement	Description	Satisfied by
[SRS_BSW_00004]	All Basic SW Modules shall perform a pre-processor check of the versions of all imported include files	[SWS_Rte_07692]
[SRS_BSW_00007]	All Basic SW Modules written in C language shall conform to the MISRA C 2004 Standard.	[SWS_Rte_01168] [SWS_Rte_03715] [SWS_Rte_06804] [SWS_Rte_06805] [SWS_Rte_06806] [SWS_Rte_06807] [SWS_Rte_06808] [SWS_Rte_06809] [SWS_Rte_06810] [SWS_Rte_07086] [SWS_Rte_07300]
[SRS_BSW_00101]	The Basic Software Module shall be able to initialize variables and hardware in a separate initialization function	[SWS_Rte_07270] [SWS_Rte_07271] [SWS_Rte_07273]

Requirement	Description	Satisfied by
[SRS_BSW_00161]	The AUTOSAR Basic Software shall provide a microcontroller abstraction layer which provides a standardized interface to higher software layers	[SWS_Rte_02734]
[SRS_BSW_00300]	All AUTOSAR Basic Software Modules shall be identified by an unambiguous name	[SWS_Rte_01003] [SWS_Rte_01157] [SWS_Rte_01158] [SWS_Rte_01161] [SWS_Rte_01169] [SWS_Rte_01171] [SWS_Rte_07122] [SWS_Rte_07139] [SWS_Rte_07284] [SWS_Rte_07288] [SWS_Rte_07295] [SWS_Rte_07504] [SWS_Rte_07922]
[SRS_BSW_00305]	Data types naming convention	[SWS_Rte_01055] [SWS_Rte_01150] [SWS_Rte_02301] [SWS_Rte_03714] [SWS_Rte_03731] [SWS_Rte_03733]
[SRS_BSW_00307]	Global variables naming convention	[SWS_Rte_01171] [SWS_Rte_03712] [SWS_Rte_07284]
[SRS_BSW_00308]	AUTOSAR Basic Software Modules shall not define global data in their header files, but in the C file	[SWS_Rte_03786] [SWS_Rte_07121] [SWS_Rte_07502] [SWS_Rte_07921]
[SRS_BSW_00310]	API naming convention	[SWS_Rte_01071] [SWS_Rte_01072] [SWS_Rte_01083] [SWS_Rte_01091] [SWS_Rte_01092] [SWS_Rte_01102] [SWS_Rte_01111] [SWS_Rte_01118] [SWS_Rte_01120] [SWS_Rte_01123] [SWS_Rte_01206] [SWS_Rte_01252] [SWS_Rte_02569] [SWS_Rte_02631] [SWS_Rte_02725] [SWS_Rte_03550] [SWS_Rte_03553] [SWS_Rte_03560] [SWS_Rte_03565] [SWS_Rte_03741] [SWS_Rte_03744] [SWS_Rte_03800] [SWS_Rte_03928] [SWS_Rte_03929] [SWS_Rte_05509] [SWS_Rte_07367] [SWS_Rte_07390] [SWS_Rte_07394] [SWS_Rte_07556]
[SRS_BSW_00312]	Shared code shall be reentrant	[SWS_Rte_01012]
[SRS_BSW_00327]	Error values naming convention	[SWS_Rte_01058] [SWS_Rte_01060] [SWS_Rte_01061] [SWS_Rte_01064] [SWS_Rte_01065] [SWS_Rte_01317] [SWS_Rte_02571] [SWS_Rte_02594] [SWS_Rte_02702] [SWS_Rte_02739] [SWS_Rte_02747] [SWS_Rte_02757] [SWS_Rte_07054] [SWS_Rte_07289] [SWS_Rte_07290] [SWS_Rte_07384] [SWS_Rte_07562] [SWS_Rte_07563] [SWS_Rte_07655] [SWS_Rte_08065] [SWS_Rte_08725] [SWS_Rte_08726]
[SRS_BSW_00330]	It shall be allowed to use macros instead of functions where source code is used and runtime is critical	[SWS_Rte_01274]

Requirement	Description	Satisfied by
[SRS_BSW_00336]	Basic SW module shall be able to shutdown	[SWS_Rte_07274] [SWS_Rte_07275] [SWS_Rte_07277]
[SRS_BSW_00337]	Classification of development errors	[SWS_Rte_06630] [SWS_Rte_06631] [SWS_Rte_06632] [SWS_Rte_06633] [SWS_Rte_06634] [SWS_Rte_06635] [SWS_Rte_06637] [SWS_Rte_07675] [SWS_Rte_07676] [SWS_Rte_07682] [SWS_Rte_07683] [SWS_Rte_07684] [SWS_Rte_07685]
[SRS_BSW_00342]	It shall be possible to create an AUTOSAR ECU out of modules provided as source code and modules provided as object code, even mixed	[SWS_Rte_07511]
[SRS_BSW_00345]	BSW Modules shall support pre-compile configuration	[SWS_Rte_05103]
[SRS_BSW_00346]	All AUTOSAR Basic Software Modules shall provide at least a basic set of module files	[SWS_Rte_06638]
[SRS_BSW_00347]	A Naming seperation of different instances of BSW drivers shall be in place	[SWS_Rte_06532] [SWS_Rte_06535] [SWS_Rte_06536] [SWS_Rte_07093] [SWS_Rte_07250] [SWS_Rte_07253] [SWS_Rte_07255] [SWS_Rte_07260] [SWS_Rte_07263] [SWS_Rte_07266] [SWS_Rte_07282] [SWS_Rte_07295] [SWS_Rte_07504] [SWS_Rte_07528] [SWS_Rte_07694] [SWS_Rte_08765]
[SRS_BSW_00353]	All integer type definitions of target and compiler specific scope shall be placed and organized in a single type header	[SWS_Rte_01163] [SWS_Rte_01164] [SWS_Rte_07104] [SWS_Rte_07641]
[SRS_BSW_00397]	The configuration parameters in pre-compile time are fixed before compilation starts	[SWS_Rte_05103]
[SRS_BSW_00399]	Parameter-sets shall be located in a separate segment and shall be loaded after the code	[SWS_Rte_05104]
[SRS_BSW_00400]	Parameter shall be selected from multiple sets of parameters after code has been loaded and started	[SWS_Rte_05104]
[SRS_BSW_00405]	BSW Modules shall support multiple configuration sets	[SWS_Rte_06544] [SWS_Rte_06545]
[SRS_BSW_00407]	Each BSW module shall provide a function to read out the version information of a dedicated module implementation	[SWS_Rte_07278] [SWS_Rte_07279] [SWS_Rte_07280] [SWS_Rte_07281]
[SRS_BSW_00415]	Interfaces which are provided exclusively for one module shall be separated into a dedicated header file	[SWS_Rte_07295] [SWS_Rte_07500] [SWS_Rte_07501] [SWS_Rte_07503] [SWS_Rte_07504] [SWS_Rte_07505] [SWS_Rte_07506] [SWS_Rte_07510]

Requirement	Description	Satisfied by
[SRS_BSW_00447]	Standardizing Include file structure of BSW Modules Implementing Autosar Service	[SWS_Rte_07120]
[SRS_Rte_00003]	Tracing of sender-receiver communication	[SWS_Rte_01238] [SWS_Rte_01240] [SWS_Rte_01241] [SWS_Rte_01242] [SWS_Rte_01357] [SWS_Rte_03814] [SWS_Rte_07639]
[SRS_Rte_00004]	Tracing of client-server communication	[SWS_Rte_01238] [SWS_Rte_01240] [SWS_Rte_01241] [SWS_Rte_01242] [SWS_Rte_01357] [SWS_Rte_03814] [SWS_Rte_07639]
[SRS_Rte_00005]	The RTE generator shall support "trace" builds	[SWS_Rte_01320] [SWS_Rte_01322] [SWS_Rte_01323] [SWS_Rte_01327] [SWS_Rte_01328] [SWS_Rte_03607] [SWS_Rte_05091] [SWS_Rte_05092] [SWS_Rte_05093] [SWS_Rte_05106] [SWS_Rte_06031] [SWS_Rte_08000]
[SRS_Rte_00008]	VFB tracing configuration	[SWS_Rte_01236] [SWS_Rte_01320] [SWS_Rte_01321] [SWS_Rte_01322] [SWS_Rte_01323] [SWS_Rte_01324] [SWS_Rte_01325] [SWS_Rte_03607] [SWS_Rte_05091] [SWS_Rte_05092] [SWS_Rte_05093] [SWS_Rte_08000]
[SRS_Rte_00011]	Support for multiple Application Software Component instances.	[SWS_Rte_01012] [SWS_Rte_01013] [SWS_Rte_01016] [SWS_Rte_01126] [SWS_Rte_01148] [SWS_Rte_01349] [SWS_Rte_02001] [SWS_Rte_02002] [SWS_Rte_02008] [SWS_Rte_02009] [SWS_Rte_02015] [SWS_Rte_03015] [SWS_Rte_03711] [SWS_Rte_03716] [SWS_Rte_03717] [SWS_Rte_03718] [SWS_Rte_03719] [SWS_Rte_03720] [SWS_Rte_03721] [SWS_Rte_03722] [SWS_Rte_03793] [SWS_Rte_03806] [SWS_Rte_06031] [SWS_Rte_07132] [SWS_Rte_07194] [SWS_Rte_07225] [SWS_Rte_07837] [SWS_Rte_07838] [SWS_Rte_07839]
[SRS_Rte_00012]	Multiple instantiated AUTOSAR software components delivered as binary code shall share code.	[SWS_Rte_01007] [SWS_Rte_02015] [SWS_Rte_03015]
[SRS_Rte_00013]	Per-instance memory	[SWS_Rte_02301] [SWS_Rte_02302] [SWS_Rte_02303] [SWS_Rte_02304] [SWS_Rte_02305] [SWS_Rte_03782] [SWS_Rte_05062] [SWS_Rte_07045] [SWS_Rte_07133] [SWS_Rte_07134] [SWS_Rte_07135] [SWS_Rte_07161] [SWS_Rte_07182] [SWS_Rte_07183] [SWS_Rte_07184] [SWS_Rte_08303] [SWS_Rte_08304]
[SRS_Rte_00017]	Rejection of inconsistent component implementations	[SWS_Rte_01004] [SWS_Rte_01276] [SWS_Rte_02751] [SWS_Rte_07123] [SWS_Rte_07510]

Requirement	Description	Satisfied by
[SRS_Rte_00018]	Rejection of invalid configurations	[SWS_Rte_01287] [SWS_Rte_01313] [SWS_Rte_01358] [SWS_Rte_01373] [SWS_Rte_02009] [SWS_Rte_02051] [SWS_Rte_02204] [SWS_Rte_02254] [SWS_Rte_02500] [SWS_Rte_02526] [SWS_Rte_02529] [SWS_Rte_02579] [SWS_Rte_02662] [SWS_Rte_02663] [SWS_Rte_02664] [SWS_Rte_02670] [SWS_Rte_02706] [SWS_Rte_02723] [SWS_Rte_02724] [SWS_Rte_02730] [SWS_Rte_02733] [SWS_Rte_02738] [SWS_Rte_02750] [SWS_Rte_03010] [SWS_Rte_03014] [SWS_Rte_03018] [SWS_Rte_03019] [SWS_Rte_03526] [SWS_Rte_03527] [SWS_Rte_03594] [SWS_Rte_03605] [SWS_Rte_03755] [SWS_Rte_03764] [SWS_Rte_03813] [SWS_Rte_03817] [SWS_Rte_03820] [SWS_Rte_03823] [SWS_Rte_03826] [SWS_Rte_03831] [SWS_Rte_03851] [SWS_Rte_03862] [SWS_Rte_03950] [SWS_Rte_03951] [SWS_Rte_03970] [SWS_Rte_05111] [SWS_Rte_05149] [SWS_Rte_06502] [SWS_Rte_06503] [SWS_Rte_06504] [SWS_Rte_06505] [SWS_Rte_06508] [SWS_Rte_06509] [SWS_Rte_06511] [SWS_Rte_06547] [SWS_Rte_06548] [SWS_Rte_06610] [SWS_Rte_06613] [SWS_Rte_06719] [SWS_Rte_06724] [SWS_Rte_06732] [SWS_Rte_06768] [SWS_Rte_06769] [SWS_Rte_06770] [SWS_Rte_06801] [SWS_Rte_06802] [SWS_Rte_06803] [SWS_Rte_07005] [SWS_Rte_07006] [SWS_Rte_07007] [SWS_Rte_07026] [SWS_Rte_07028] [SWS_Rte_07039] [SWS_Rte_07044] [SWS_Rte_07057] [SWS_Rte_07075] [SWS_Rte_07101] [SWS_Rte_07135] [SWS_Rte_07157] [SWS_Rte_07170] [SWS_Rte_07175] [SWS_Rte_07181] [SWS_Rte_07190] [SWS_Rte_07191] [SWS_Rte_07192] [SWS_Rte_07343] [SWS_Rte_07347] [SWS_Rte_07353] [SWS_Rte_07356] [SWS_Rte_07357] [SWS_Rte_07402] [SWS_Rte_07403] [SWS_Rte_07516] [SWS_Rte_07524] [SWS_Rte_07545] [SWS_Rte_07548] [SWS_Rte_07549] [SWS_Rte_07564] [SWS_Rte_07588] [SWS_Rte_07610] [SWS_Rte_07621] [SWS_Rte_07638] [SWS_Rte_07640] [SWS_Rte_07642] [SWS_Rte_07654] [SWS_Rte_07662] [SWS_Rte_07667] [SWS_Rte_07670] [SWS_Rte_07681]

Requirement	Description	Satisfied by
		[SWS_Rte_07686] [SWS_Rte_07803] [SWS_Rte_07808] [SWS_Rte_07809] [SWS_Rte_07810] [SWS_Rte_07811] [SWS_Rte_07812] [SWS_Rte_07842] [SWS_Rte_07845] [SWS_Rte_07927] [SWS_Rte_08072] [SWS_Rte_08076] [SWS_Rte_08311] [SWS_Rte_08700] [SWS_Rte_08701] [SWS_Rte_08702] [SWS_Rte_08767] [SWS_Rte_08768] [SWS_Rte_08800]
[SRS_Rte_00019]	RTE is the communication infrastructure	[SWS_Rte_01231] [SWS_Rte_01264] [SWS_Rte_02527] [SWS_Rte_02528] [SWS_Rte_02610] [SWS_Rte_02611] [SWS_Rte_02612] [SWS_Rte_03000] [SWS_Rte_03001] [SWS_Rte_03002] [SWS_Rte_03004] [SWS_Rte_03005] [SWS_Rte_03007] [SWS_Rte_03008] [SWS_Rte_03760] [SWS_Rte_03761] [SWS_Rte_03762] [SWS_Rte_03769] [SWS_Rte_03775] [SWS_Rte_03776] [SWS_Rte_03795] [SWS_Rte_03796] [SWS_Rte_04515] [SWS_Rte_04516] [SWS_Rte_04520] [SWS_Rte_04522] [SWS_Rte_04526] [SWS_Rte_04527] [SWS_Rte_05063] [SWS_Rte_05065] [SWS_Rte_05084] [SWS_Rte_05085] [SWS_Rte_05500] [SWS_Rte_06000] [SWS_Rte_06011] [SWS_Rte_06023] [SWS_Rte_06024] [SWS_Rte_07662] [SWS_Rte_08001] [SWS_Rte_08002]
[SRS_Rte_00020]	Access to OS	[SWS_Rte_02250]
[SRS_Rte_00021]	Per-ECU RTE customization	[SWS_Rte_01316] [SWS_Rte_05000]
[SRS_Rte_00022]	Interaction with call-backs	[SWS_Rte_01165]
[SRS_Rte_00023]	RTE Overheads	[SWS_Rte_05053]
[SRS_Rte_00024]	Source-code AUTOSAR software components	[SWS_Rte_01000] [SWS_Rte_01195] [SWS_Rte_01315] [SWS_Rte_07120]
[SRS_Rte_00025]	Static communication	[SWS_Rte_06026]
[SRS_Rte_00027]	VFB to RTE mapping shall be semantic preserving	[SWS_Rte_01274] [SWS_Rte_02200] [SWS_Rte_02201] [SWS_Rte_02649] [SWS_Rte_02651] [SWS_Rte_02653] [SWS_Rte_02654] [SWS_Rte_02657] [SWS_Rte_07346] [SWS_Rte_08700] [SWS_Rte_08701] [SWS_Rte_08702] [SWS_Rte_08703] [SWS_Rte_08704] [SWS_Rte_08705] [SWS_Rte_08706] [SWS_Rte_08707] [SWS_Rte_08708] [SWS_Rte_08709] [SWS_Rte_08710] [SWS_Rte_08730]

Requirement	Description	Satisfied by
[SRS_Rte_00028]	"1:n" Sender-receiver communication	[SWS_Rte_01071] [SWS_Rte_01072] [SWS_Rte_01082] [SWS_Rte_01091] [SWS_Rte_01092] [SWS_Rte_01135] [SWS_Rte_02631] [SWS_Rte_02633] [SWS_Rte_02635] [SWS_Rte_04526] [SWS_Rte_06023] [SWS_Rte_06024] [SWS_Rte_07394] [SWS_Rte_07824] [SWS_Rte_07825] [SWS_Rte_07826] [SWS_Rte_07827] [SWS_Rte_08413] [SWS_Rte_08414] [SWS_Rte_08415]
[SRS_Rte_00029]	"n:1" Client-server communication	[SWS_Rte_01102] [SWS_Rte_01109] [SWS_Rte_01133] [SWS_Rte_01166] [SWS_Rte_01359] [SWS_Rte_02579] [SWS_Rte_03763] [SWS_Rte_03767] [SWS_Rte_03768] [SWS_Rte_03769] [SWS_Rte_03770] [SWS_Rte_04517] [SWS_Rte_04519] [SWS_Rte_05111] [SWS_Rte_05193] [SWS_Rte_06019] [SWS_Rte_07023] [SWS_Rte_07024] [SWS_Rte_07025] [SWS_Rte_07026] [SWS_Rte_07027] [SWS_Rte_07845] [SWS_Rte_08310]
[SRS_Rte_00031]	Multiple Runnable Entities	[SWS_Rte_01016] [SWS_Rte_01126] [SWS_Rte_01130] [SWS_Rte_01132] [SWS_Rte_02202] [SWS_Rte_06713]
[SRS_Rte_00032]	Data consistency mechanisms	[SWS_Rte_01122] [SWS_Rte_02740] [SWS_Rte_02741] [SWS_Rte_02743] [SWS_Rte_02744] [SWS_Rte_02745] [SWS_Rte_02746] [SWS_Rte_03500] [SWS_Rte_03503] [SWS_Rte_03504] [SWS_Rte_03514] [SWS_Rte_03516] [SWS_Rte_03517] [SWS_Rte_03519] [SWS_Rte_03595] [SWS_Rte_03739] [SWS_Rte_03740] [SWS_Rte_03812] [SWS_Rte_05164] [SWS_Rte_07005] [SWS_Rte_08318] [SWS_Rte_08319] [SWS_Rte_08320] [SWS_Rte_08321] [SWS_Rte_08322]
[SRS_Rte_00033]	Serialized execution of Server Runnable Entities	[SWS_Rte_02527] [SWS_Rte_02528] [SWS_Rte_02529] [SWS_Rte_02530] [SWS_Rte_04515] [SWS_Rte_04518] [SWS_Rte_04522] [SWS_Rte_07008] [SWS_Rte_08001] [SWS_Rte_08002]
[SRS_Rte_00036]	Assignment to OS Applications	[SWS_Rte_07347]
[SRS_Rte_00045]	Standardized VFB tracing interface	[SWS_Rte_01238] [SWS_Rte_01239] [SWS_Rte_01240] [SWS_Rte_01241] [SWS_Rte_01242] [SWS_Rte_01243] [SWS_Rte_01244] [SWS_Rte_01245] [SWS_Rte_01246] [SWS_Rte_01247] [SWS_Rte_01248] [SWS_Rte_01249] [SWS_Rte_01250] [SWS_Rte_01251] [SWS_Rte_01319] [SWS_Rte_01321] [SWS_Rte_01326] [SWS_Rte_03814] [SWS_Rte_06032] [SWS_Rte_07639]

Requirement	Description	Satisfied by
[SRS_Rte_00046]	Support for "Executable Entity runs inside" Exclusive Areas	[SWS_Rte_01120] [SWS_Rte_01122] [SWS_Rte_01123] [SWS_Rte_02740] [SWS_Rte_02741] [SWS_Rte_02743] [SWS_Rte_02744] [SWS_Rte_02745] [SWS_Rte_02746] [SWS_Rte_03500] [SWS_Rte_03515] [SWS_Rte_07250] [SWS_Rte_07251] [SWS_Rte_07252] [SWS_Rte_07253] [SWS_Rte_07254] [SWS_Rte_07522] [SWS_Rte_07523] [SWS_Rte_07524] [SWS_Rte_07578] [SWS_Rte_07579] [SWS_Rte_08318] [SWS_Rte_08319] [SWS_Rte_08320] [SWS_Rte_08321] [SWS_Rte_08322]
[SRS_Rte_00048]	RTE Generator input	[SWS_Rte_08769] [SWS_Rte_08770] [SWS_Rte_08771] [SWS_Rte_08772] [SWS_Rte_08773] [SWS_Rte_08774] [SWS_Rte_08775] [SWS_Rte_08776]
[SRS_Rte_00049]	Construction of task bodies	[SWS_Rte_02204] [SWS_Rte_02251] [SWS_Rte_02254] [SWS_Rte_07516]

Requirement	Description	Satisfied by
[SRS_Rte_00051]	RTE API mapping	[SWS_Rte_01053] [SWS_Rte_01055] [SWS_Rte_01119] [SWS_Rte_01123] [SWS_Rte_01132] [SWS_Rte_01146] [SWS_Rte_01148] [SWS_Rte_01153] [SWS_Rte_01156] [SWS_Rte_01159] [SWS_Rte_01197] [SWS_Rte_01266] [SWS_Rte_01268] [SWS_Rte_01269] [SWS_Rte_01274] [SWS_Rte_01276] [SWS_Rte_01280] [SWS_Rte_01281] [SWS_Rte_01282] [SWS_Rte_01283] [SWS_Rte_01284] [SWS_Rte_01285] [SWS_Rte_01286] [SWS_Rte_01287] [SWS_Rte_01288] [SWS_Rte_01289] [SWS_Rte_01290] [SWS_Rte_01293] [SWS_Rte_01294] [SWS_Rte_01296] [SWS_Rte_01297] [SWS_Rte_01298] [SWS_Rte_01299] [SWS_Rte_01300] [SWS_Rte_01301] [SWS_Rte_01302] [SWS_Rte_01303] [SWS_Rte_01304] [SWS_Rte_01305] [SWS_Rte_01306] [SWS_Rte_01307] [SWS_Rte_01308] [SWS_Rte_01309] [SWS_Rte_01310] [SWS_Rte_01312] [SWS_Rte_01313] [SWS_Rte_01342] [SWS_Rte_01343] [SWS_Rte_01349] [SWS_Rte_01354] [SWS_Rte_01355] [SWS_Rte_01363] [SWS_Rte_01364] [SWS_Rte_01365] [SWS_Rte_01366] [SWS_Rte_02301] [SWS_Rte_02302] [SWS_Rte_02588] [SWS_Rte_02589] [SWS_Rte_02607] [SWS_Rte_02608] [SWS_Rte_02613] [SWS_Rte_02614] [SWS_Rte_02615] [SWS_Rte_02616] [SWS_Rte_02617] [SWS_Rte_02618] [SWS_Rte_02619] [SWS_Rte_02620] [SWS_Rte_02621] [SWS_Rte_02623] [SWS_Rte_02632] [SWS_Rte_02666] [SWS_Rte_02676] [SWS_Rte_02677] [SWS_Rte_02678] [SWS_Rte_02679] [SWS_Rte_02730] [SWS_Rte_03014] [SWS_Rte_03562] [SWS_Rte_03567] [SWS_Rte_03602] [SWS_Rte_03603] [SWS_Rte_03605] [SWS_Rte_03706] [SWS_Rte_03707] [SWS_Rte_03716] [SWS_Rte_03717] [SWS_Rte_03718] [SWS_Rte_03719] [SWS_Rte_03720] [SWS_Rte_03721] [SWS_Rte_03723] [SWS_Rte_03725] [SWS_Rte_03726] [SWS_Rte_03730] [SWS_Rte_03731] [SWS_Rte_03733] [SWS_Rte_03734] [SWS_Rte_03739] [SWS_Rte_03740] [SWS_Rte_03746] [SWS_Rte_03752] [SWS_Rte_03791] [SWS_Rte_03799] [SWS_Rte_03801]

Requirement	Description	Satisfied by
		[SWS_Rte_03812] [SWS_Rte_03835] [SWS_Rte_03837] [SWS_Rte_03927] [SWS_Rte_03930] [SWS_Rte_03949] [SWS_Rte_03952] [SWS_Rte_05510] [SWS_Rte_05511] [SWS_Rte_06639] [SWS_Rte_06713] [SWS_Rte_07136] [SWS_Rte_07137] [SWS_Rte_07138] [SWS_Rte_07170] [SWS_Rte_07225] [SWS_Rte_07226] [SWS_Rte_07227] [SWS_Rte_07228] [SWS_Rte_07291] [SWS_Rte_07395] [SWS_Rte_07396] [SWS_Rte_07677] [SWS_Rte_07837] [SWS_Rte_07838] [SWS_Rte_07839] [SWS_Rte_07850] [SWS_Rte_07851] [SWS_Rte_08073] [SWS_Rte_08309] [SWS_Rte_08312] [SWS_Rte_08777] [SWS_Rte_08778] [SWS_Rte_08779] [SWS_Rte_08780] [SWS_Rte_08781] [SWS_Rte_08782] [SWS_Rte_08783] [SWS_Rte_08784] [SWS_Rte_08785] [SWS_Rte_08786]
[SRS_Rte_00052]	Initialization and finalization of components	[SWS_Rte_02503] [SWS_Rte_02562] [SWS_Rte_02564] [SWS_Rte_02707] [SWS_Rte_03852] [SWS_Rte_07046]
[SRS_Rte_00055]	RTE use of global namespace	[SWS_Rte_01171] [SWS_Rte_06706] [SWS_Rte_06707] [SWS_Rte_06708] [SWS_Rte_07036] [SWS_Rte_07037] [SWS_Rte_07104] [SWS_Rte_07109] [SWS_Rte_07110] [SWS_Rte_07111] [SWS_Rte_07114] [SWS_Rte_07115] [SWS_Rte_07116] [SWS_Rte_07117] [SWS_Rte_07118] [SWS_Rte_07119] [SWS_Rte_07144] [SWS_Rte_07145] [SWS_Rte_07146] [SWS_Rte_07148] [SWS_Rte_07149] [SWS_Rte_07162] [SWS_Rte_07163] [SWS_Rte_07166] [SWS_Rte_07284]
[SRS_Rte_00059]	RTE API shall pass "in" primitive data types by value	[SWS_Rte_01017] [SWS_Rte_01020] [SWS_Rte_06805] [SWS_Rte_06807] [SWS_Rte_07069] [SWS_Rte_07070] [SWS_Rte_07071] [SWS_Rte_07072] [SWS_Rte_07073] [SWS_Rte_07074] [SWS_Rte_07076] [SWS_Rte_07077] [SWS_Rte_07078] [SWS_Rte_07079] [SWS_Rte_07080] [SWS_Rte_07081] [SWS_Rte_07083] [SWS_Rte_07084] [SWS_Rte_07661] [SWS_Rte_08300]
[SRS_Rte_00060]	RTE API shall pass "in" composite data types by reference.	[SWS_Rte_01018] [SWS_Rte_05107] [SWS_Rte_05108] [SWS_Rte_06804] [SWS_Rte_06807] [SWS_Rte_07082] [SWS_Rte_07084] [SWS_Rte_07086]

Requirement	Description	Satisfied by
[SRS_Rte_00061]	"in/out" and "out" parameters	[SWS_Rte_01017] [SWS_Rte_01018] [SWS_Rte_01019] [SWS_Rte_01020] [SWS_Rte_05107] [SWS_Rte_05108] [SWS_Rte_05109] [SWS_Rte_06806] [SWS_Rte_07082] [SWS_Rte_07083] [SWS_Rte_07084] [SWS_Rte_07661]
[SRS_Rte_00062]	Local access to basic software components	[SWS_Rte_02051]
[SRS_Rte_00065]	Deterministic generation	[SWS_Rte_02514] [SWS_Rte_05150]
[SRS_Rte_00068]	Signal initial values	[SWS_Rte_02517] [SWS_Rte_03852] [SWS_Rte_05078] [SWS_Rte_07046] [SWS_Rte_07642] [SWS_Rte_07668] [SWS_Rte_08311]
[SRS_Rte_00069]	Communication timeouts	[SWS_Rte_01064] [SWS_Rte_01095] [SWS_Rte_01107] [SWS_Rte_01114] [SWS_Rte_03754] [SWS_Rte_03758] [SWS_Rte_03759] [SWS_Rte_03763] [SWS_Rte_03767] [SWS_Rte_03768] [SWS_Rte_03770] [SWS_Rte_03771] [SWS_Rte_03772] [SWS_Rte_03773] [SWS_Rte_06002] [SWS_Rte_06013] [SWS_Rte_07056] [SWS_Rte_07059] [SWS_Rte_07060] [SWS_Rte_08310]
[SRS_Rte_00070]	Invocation order of Runnable Entities	[SWS_Rte_02207]
[SRS_Rte_00072]	Activation of Runnable Entities	[SWS_Rte_01131] [SWS_Rte_01133] [SWS_Rte_01135] [SWS_Rte_01137] [SWS_Rte_01166] [SWS_Rte_01292] [SWS_Rte_01359] [SWS_Rte_02203] [SWS_Rte_02512] [SWS_Rte_02697] [SWS_Rte_02758] [SWS_Rte_03520] [SWS_Rte_03523] [SWS_Rte_03524] [SWS_Rte_03526] [SWS_Rte_03527] [SWS_Rte_03530] [SWS_Rte_03531] [SWS_Rte_03532] [SWS_Rte_05193] [SWS_Rte_06748] [SWS_Rte_06759] [SWS_Rte_06760] [SWS_Rte_06771] [SWS_Rte_07023] [SWS_Rte_07024] [SWS_Rte_07025] [SWS_Rte_07026] [SWS_Rte_07027] [SWS_Rte_07061] [SWS_Rte_07177] [SWS_Rte_07178] [SWS_Rte_07207] [SWS_Rte_07208] [SWS_Rte_07379] [SWS_Rte_07403] [SWS_Rte_07515] [SWS_Rte_07575]
[SRS_Rte_00073]	Atomic transport of Data Elements	[SWS_Rte_04527]
[SRS_Rte_00075]	API for accessing per-instance memory	[SWS_Rte_01118] [SWS_Rte_01119]

Requirement	Description	Satisfied by
[SRS_Rte_00077]	Instantiation of per-instance memory	[SWS_Rte_02303] [SWS_Rte_02304] [SWS_Rte_02305] [SWS_Rte_03782] [SWS_Rte_05062] [SWS_Rte_07045] [SWS_Rte_07133] [SWS_Rte_07161] [SWS_Rte_07182] [SWS_Rte_07183] [SWS_Rte_07184] [SWS_Rte_08303] [SWS_Rte_08304]
[SRS_Rte_00078]	Support for Data Element Invalidation	[SWS_Rte_01206] [SWS_Rte_01231] [SWS_Rte_01282] [SWS_Rte_02589] [SWS_Rte_02590] [SWS_Rte_02594] [SWS_Rte_02599] [SWS_Rte_02600] [SWS_Rte_02603] [SWS_Rte_02607] [SWS_Rte_02609] [SWS_Rte_02626] [SWS_Rte_02629] [SWS_Rte_02666] [SWS_Rte_02702] [SWS_Rte_03778] [SWS_Rte_03800] [SWS_Rte_03801] [SWS_Rte_03802] [SWS_Rte_05024] [SWS_Rte_05025] [SWS_Rte_05026] [SWS_Rte_05030] [SWS_Rte_05032] [SWS_Rte_05048] [SWS_Rte_05049] [SWS_Rte_05063] [SWS_Rte_05064] [SWS_Rte_06727] [SWS_Rte_07031] [SWS_Rte_07032] [SWS_Rte_08004] [SWS_Rte_08005] [SWS_Rte_08007] [SWS_Rte_08008] [SWS_Rte_08009] [SWS_Rte_08046] [SWS_Rte_08047] [SWS_Rte_08048] [SWS_Rte_08049] [SWS_Rte_08050] [SWS_Rte_08405] [SWS_Rte_08406] [SWS_Rte_08407] [SWS_Rte_08501]
[SRS_Rte_00079]	Single asynchronous client-server interaction	[SWS_Rte_01105] [SWS_Rte_01109] [SWS_Rte_01133] [SWS_Rte_01166] [SWS_Rte_01359] [SWS_Rte_02658] [SWS_Rte_03765] [SWS_Rte_03766] [SWS_Rte_03771] [SWS_Rte_03772] [SWS_Rte_05193] [SWS_Rte_07023] [SWS_Rte_07024] [SWS_Rte_07025] [SWS_Rte_07026] [SWS_Rte_07027] [SWS_Rte_08800]
[SRS_Rte_00080]	Multiple requests of servers	[SWS_Rte_03769] [SWS_Rte_04516] [SWS_Rte_04520]
[SRS_Rte_00082]	Standardized communication protocol	[SWS_Rte_02579] [SWS_Rte_02649] [SWS_Rte_02651] [SWS_Rte_02653] [SWS_Rte_02654] [SWS_Rte_02655] [SWS_Rte_02656] [SWS_Rte_02657] [SWS_Rte_05111] [SWS_Rte_07346] [SWS_Rte_08700] [SWS_Rte_08701] [SWS_Rte_08702] [SWS_Rte_08703] [SWS_Rte_08704] [SWS_Rte_08705] [SWS_Rte_08706] [SWS_Rte_08707] [SWS_Rte_08708] [SWS_Rte_08709] [SWS_Rte_08710] [SWS_Rte_08711] [SWS_Rte_08712] [SWS_Rte_08713] [SWS_Rte_08730]

Requirement	Description	Satisfied by
[SRS_Rte_00083]	Optimization for source-code components	[SWS_Rte_01152] [SWS_Rte_01274]
[SRS_Rte_00084]	Support infrastructural errors	[SWS_Rte_01318] [SWS_Rte_02593]
[SRS_Rte_00087]	Software Module Header File generation	[SWS_Rte_01000] [SWS_Rte_01004] [SWS_Rte_01006] [SWS_Rte_01132] [SWS_Rte_01274] [SWS_Rte_03786] [SWS_Rte_05078] [SWS_Rte_06703] [SWS_Rte_06704] [SWS_Rte_06705] [SWS_Rte_06713] [SWS_Rte_07127] [SWS_Rte_07131] [SWS_Rte_07924]
[SRS_Rte_00089]	Independent access to interface elements	[SWS_Rte_06008]
[SRS_Rte_00091]	Inter-ECU Marshalling	[SWS_Rte_02557] [SWS_Rte_04504] [SWS_Rte_04505] [SWS_Rte_04506] [SWS_Rte_04507] [SWS_Rte_04508] [SWS_Rte_04527] [SWS_Rte_05081] [SWS_Rte_05173] [SWS_Rte_08700] [SWS_Rte_08701] [SWS_Rte_08702] [SWS_Rte_08703] [SWS_Rte_08704] [SWS_Rte_08705] [SWS_Rte_08706] [SWS_Rte_08707] [SWS_Rte_08708] [SWS_Rte_08709] [SWS_Rte_08710] [SWS_Rte_08711] [SWS_Rte_08712] [SWS_Rte_08713] [SWS_Rte_08725] [SWS_Rte_08726] [SWS_Rte_08727] [SWS_Rte_08728] [SWS_Rte_08729] [SWS_Rte_08730] [SWS_Rte_08731] [SWS_Rte_08761] [SWS_Rte_08762]
[SRS_Rte_00092]	Implementation of VFB model "waitpoints"	[SWS_Rte_01358] [SWS_Rte_02740] [SWS_Rte_02741] [SWS_Rte_02743] [SWS_Rte_02744] [SWS_Rte_02745] [SWS_Rte_02746] [SWS_Rte_03010] [SWS_Rte_03018] [SWS_Rte_07402] [SWS_Rte_07846] [SWS_Rte_07847] [SWS_Rte_08318] [SWS_Rte_08319] [SWS_Rte_08320] [SWS_Rte_08321] [SWS_Rte_08322]

Requirement	Description	Satisfied by
[SRS_Rte_00094]	Communication and Resource Errors	[SWS_Rte_01034] [SWS_Rte_01084] [SWS_Rte_01086] [SWS_Rte_01093] [SWS_Rte_01094] [SWS_Rte_01095] [SWS_Rte_01103] [SWS_Rte_01104] [SWS_Rte_01105] [SWS_Rte_01106] [SWS_Rte_01107] [SWS_Rte_01112] [SWS_Rte_01113] [SWS_Rte_01114] [SWS_Rte_01207] [SWS_Rte_01259] [SWS_Rte_01260] [SWS_Rte_01261] [SWS_Rte_01262] [SWS_Rte_01318] [SWS_Rte_01330] [SWS_Rte_01331] [SWS_Rte_01333] [SWS_Rte_01334] [SWS_Rte_01339] [SWS_Rte_01344] [SWS_Rte_02524] [SWS_Rte_02525] [SWS_Rte_02571] [SWS_Rte_02572] [SWS_Rte_02578] [SWS_Rte_02598] [SWS_Rte_02602] [SWS_Rte_02674] [SWS_Rte_02721] [SWS_Rte_02727] [SWS_Rte_02728] [SWS_Rte_02729] [SWS_Rte_03606] [SWS_Rte_03774] [SWS_Rte_03785] [SWS_Rte_03853] [SWS_Rte_07258] [SWS_Rte_07374] [SWS_Rte_07375] [SWS_Rte_07376] [SWS_Rte_07392] [SWS_Rte_07393] [SWS_Rte_07636] [SWS_Rte_07637] [SWS_Rte_07650] [SWS_Rte_07651] [SWS_Rte_07652] [SWS_Rte_07659] [SWS_Rte_07660] [SWS_Rte_07673] [SWS_Rte_07820] [SWS_Rte_07821] [SWS_Rte_07822] [SWS_Rte_07823] [SWS_Rte_07848] [SWS_Rte_07849] [SWS_Rte_08301] [SWS_Rte_08302] [SWS_Rte_08727] [SWS_Rte_08728] [SWS_Rte_08729]
[SRS_Rte_00098]	Explicit Sending	[SWS_Rte_01071] [SWS_Rte_06011] [SWS_Rte_06016]
[SRS_Rte_00099]	Decoupling of interrupts	[SWS_Rte_03530] [SWS_Rte_03531] [SWS_Rte_03532] [SWS_Rte_03594] [SWS_Rte_03600]
[SRS_Rte_00100]	Compiler independent API	[SWS_Rte_01314]
[SRS_Rte_00107]	Support for INFORMATION_TYPE attribute	[SWS_Rte_01135] [SWS_Rte_01137] [SWS_Rte_01331] [SWS_Rte_02516] [SWS_Rte_02518] [SWS_Rte_02520] [SWS_Rte_02521] [SWS_Rte_02522] [SWS_Rte_02523] [SWS_Rte_02524] [SWS_Rte_02525] [SWS_Rte_02571] [SWS_Rte_02572] [SWS_Rte_02718] [SWS_Rte_02719] [SWS_Rte_02720] [SWS_Rte_02721] [SWS_Rte_02758] [SWS_Rte_04500] [SWS_Rte_06010] [SWS_Rte_06771]

Requirement	Description	Satisfied by
[SRS_Rte_00108]	Support for INIT_VALUE attribute	[SWS_Rte_01268] [SWS_Rte_02517] [SWS_Rte_04501] [SWS_Rte_04502] [SWS_Rte_05078] [SWS_Rte_06009] [SWS_Rte_07642] [SWS_Rte_07668] [SWS_Rte_07680] [SWS_Rte_07681] [SWS_Rte_08311]
[SRS_Rte_00109]	Support for RECEIVE_MODE attribute	[SWS_Rte_02519] [SWS_Rte_03018] [SWS_Rte_06002] [SWS_Rte_06012]
[SRS_Rte_00110]	Support for BUFFERING attribute	[SWS_Rte_01331] [SWS_Rte_02515] [SWS_Rte_02522] [SWS_Rte_02523] [SWS_Rte_02524] [SWS_Rte_02525] [SWS_Rte_02526] [SWS_Rte_02527] [SWS_Rte_02529] [SWS_Rte_02530] [SWS_Rte_02571] [SWS_Rte_02572] [SWS_Rte_02719] [SWS_Rte_02720] [SWS_Rte_02721] [SWS_Rte_02723] [SWS_Rte_07008]
[SRS_Rte_00111]	Support for CLIENT_MODE attribute	[SWS_Rte_01293] [SWS_Rte_01294] [SWS_Rte_06639]
[SRS_Rte_00115]	API for data consistency mechanism	[SWS_Rte_01120] [SWS_Rte_01122] [SWS_Rte_01307] [SWS_Rte_01308]
[SRS_Rte_00116]	RTE Initialization and finalization	[SWS_Rte_02535] [SWS_Rte_02536] [SWS_Rte_02538] [SWS_Rte_02544] [SWS_Rte_02569] [SWS_Rte_02570] [SWS_Rte_02584] [SWS_Rte_02585] [SWS_Rte_03852] [SWS_Rte_06766] [SWS_Rte_06767] [SWS_Rte_07046] [SWS_Rte_07270] [SWS_Rte_07586]
[SRS_Rte_00121]	Support for FILTER attribute	[SWS_Rte_05500] [SWS_Rte_05501] [SWS_Rte_05503] [SWS_Rte_08077] [SWS_Rte_08078] [SWS_Rte_08079]

Requirement	Description	Satisfied by
[SRS_Rte_00122]	Support for Transmission Acknowledgement	[SWS_Rte_01080] [SWS_Rte_01083] [SWS_Rte_01084] [SWS_Rte_01086] [SWS_Rte_01137] [SWS_Rte_01283] [SWS_Rte_01284] [SWS_Rte_01285] [SWS_Rte_01286] [SWS_Rte_01287] [SWS_Rte_01344] [SWS_Rte_02612] [SWS_Rte_02676] [SWS_Rte_02677] [SWS_Rte_02678] [SWS_Rte_02725] [SWS_Rte_02727] [SWS_Rte_02729] [SWS_Rte_02758] [SWS_Rte_03002] [SWS_Rte_03005] [SWS_Rte_03604] [SWS_Rte_03754] [SWS_Rte_03756] [SWS_Rte_03757] [SWS_Rte_03758] [SWS_Rte_03774] [SWS_Rte_03775] [SWS_Rte_03776] [SWS_Rte_05065] [SWS_Rte_05084] [SWS_Rte_05085] [SWS_Rte_05504] [SWS_Rte_06771] [SWS_Rte_07055] [SWS_Rte_07286] [SWS_Rte_07367] [SWS_Rte_07374] [SWS_Rte_07375] [SWS_Rte_07376] [SWS_Rte_07379] [SWS_Rte_07557] [SWS_Rte_07558] [SWS_Rte_07560] [SWS_Rte_07561] [SWS_Rte_07634] [SWS_Rte_07635] [SWS_Rte_07636] [SWS_Rte_07637] [SWS_Rte_07646] [SWS_Rte_07647] [SWS_Rte_07648] [SWS_Rte_07650] [SWS_Rte_07651] [SWS_Rte_07652] [SWS_Rte_07659] [SWS_Rte_07660] [SWS_Rte_07846] [SWS_Rte_07847] [SWS_Rte_07848] [SWS_Rte_07849] [SWS_Rte_07850] [SWS_Rte_07851] [SWS_Rte_07927] [SWS_Rte_08017] [SWS_Rte_08018] [SWS_Rte_08020] [SWS_Rte_08021] [SWS_Rte_08022] [SWS_Rte_08023] [SWS_Rte_08043] [SWS_Rte_08044] [SWS_Rte_08045] [SWS_Rte_08074] [SWS_Rte_08075] [SWS_Rte_08076]
[SRS_Rte_00123]	The RTE shall forward application level errors from server to client	[SWS_Rte_01103] [SWS_Rte_02576] [SWS_Rte_02577] [SWS_Rte_02578] [SWS_Rte_02593] [SWS_Rte_07925] [SWS_Rte_07926] [SWS_Rte_08705] [SWS_Rte_08706] [SWS_Rte_08709] [SWS_Rte_08710]
[SRS_Rte_00124]	API for application level errors during Client Server communication	[SWS_Rte_01103] [SWS_Rte_01130] [SWS_Rte_02573] [SWS_Rte_02575]

Requirement	Description	Satisfied by
[SRS_Rte_00126]	C language support	[SWS_Rte_01005] [SWS_Rte_01162] [SWS_Rte_01167] [SWS_Rte_01169] [SWS_Rte_03709] [SWS_Rte_03710] [SWS_Rte_03724] [SWS_Rte_07124] [SWS_Rte_07125] [SWS_Rte_07126] [SWS_Rte_07297] [SWS_Rte_07298] [SWS_Rte_07299] [SWS_Rte_07507] [SWS_Rte_07508] [SWS_Rte_07509] [SWS_Rte_07678] [SWS_Rte_07923]
[SRS_Rte_00128]	Implicit Reception	[SWS_Rte_01268] [SWS_Rte_03598] [SWS_Rte_03599] [SWS_Rte_03741] [SWS_Rte_03954] [SWS_Rte_03955] [SWS_Rte_03956] [SWS_Rte_06000] [SWS_Rte_06001] [SWS_Rte_06004] [SWS_Rte_06011] [SWS_Rte_07007] [SWS_Rte_07020] [SWS_Rte_07062] [SWS_Rte_07063] [SWS_Rte_07064] [SWS_Rte_07652] [SWS_Rte_08408]
[SRS_Rte_00129]	Implicit Sending	[SWS_Rte_03570] [SWS_Rte_03571] [SWS_Rte_03572] [SWS_Rte_03573] [SWS_Rte_03574] [SWS_Rte_03598] [SWS_Rte_03744] [SWS_Rte_03746] [SWS_Rte_03953] [SWS_Rte_03954] [SWS_Rte_03955] [SWS_Rte_03957] [SWS_Rte_05509] [SWS_Rte_06011] [SWS_Rte_07007] [SWS_Rte_07021] [SWS_Rte_07041] [SWS_Rte_07062] [SWS_Rte_07065] [SWS_Rte_07066] [SWS_Rte_07067] [SWS_Rte_07068] [SWS_Rte_07367] [SWS_Rte_07374] [SWS_Rte_07375] [SWS_Rte_07376] [SWS_Rte_07646] [SWS_Rte_07647] [SWS_Rte_07648] [SWS_Rte_07650] [SWS_Rte_07651] [SWS_Rte_07660] [SWS_Rte_08408]
[SRS_Rte_00131]	"n:1" Sender-receiver communication	[SWS_Rte_01071] [SWS_Rte_01072] [SWS_Rte_01091] [SWS_Rte_01092] [SWS_Rte_01135] [SWS_Rte_02631] [SWS_Rte_02633] [SWS_Rte_02635] [SWS_Rte_02670] [SWS_Rte_02724] [SWS_Rte_03760] [SWS_Rte_03761] [SWS_Rte_03762] [SWS_Rte_07394] [SWS_Rte_07824] [SWS_Rte_07825] [SWS_Rte_07826] [SWS_Rte_07827]
[SRS_Rte_00133]	Concurrent invocation of Runnable Entities	[SWS_Rte_02697] [SWS_Rte_03523] [SWS_Rte_07007]
[SRS_Rte_00134]	Runnable Entity categories supported by the RTE	[SWS_Rte_03574] [SWS_Rte_03954] [SWS_Rte_06003] [SWS_Rte_06007] [SWS_Rte_07062]
[SRS_Rte_00137]	API for mismatched ports	[SWS_Rte_01368] [SWS_Rte_01369] [SWS_Rte_01370]

Requirement	Description	Satisfied by
[SRS_Rte_00138]	C++ language support	[SWS_Rte_01005] [SWS_Rte_01011] [SWS_Rte_03709] [SWS_Rte_03710] [SWS_Rte_07124] [SWS_Rte_07125] [SWS_Rte_07126] [SWS_Rte_07297] [SWS_Rte_07298] [SWS_Rte_07299] [SWS_Rte_07507] [SWS_Rte_07508] [SWS_Rte_07509]
[SRS_Rte_00139]	Support for unconnected ports	[SWS_Rte_01329] [SWS_Rte_01330] [SWS_Rte_01331] [SWS_Rte_01332] [SWS_Rte_01333] [SWS_Rte_01334] [SWS_Rte_01344] [SWS_Rte_01346] [SWS_Rte_01347] [SWS_Rte_01375] [SWS_Rte_02638] [SWS_Rte_02639] [SWS_Rte_02640] [SWS_Rte_02641] [SWS_Rte_02642] [SWS_Rte_02749] [SWS_Rte_02750] [SWS_Rte_03019] [SWS_Rte_03783] [SWS_Rte_03784] [SWS_Rte_03785] [SWS_Rte_03978] [SWS_Rte_03980] [SWS_Rte_05099] [SWS_Rte_05101] [SWS_Rte_05102] [SWS_Rte_05170] [SWS_Rte_06030] [SWS_Rte_07378] [SWS_Rte_07655] [SWS_Rte_07659] [SWS_Rte_07660] [SWS_Rte_07663] [SWS_Rte_07667] [SWS_Rte_07668] [SWS_Rte_07669] [SWS_Rte_07847]
[SRS_Rte_00140]	Binary-code AUTOSAR software components	[SWS_Rte_01000] [SWS_Rte_01195] [SWS_Rte_01315] [SWS_Rte_07120]
[SRS_Rte_00141]	Explicit Reception	[SWS_Rte_01072] [SWS_Rte_01091] [SWS_Rte_01092] [SWS_Rte_06011] [SWS_Rte_07394] [SWS_Rte_07673]
[SRS_Rte_00142]	Support for InterRunnableVariables	[SWS_Rte_01303] [SWS_Rte_01304] [SWS_Rte_01305] [SWS_Rte_01306] [SWS_Rte_01350] [SWS_Rte_01351] [SWS_Rte_02636] [SWS_Rte_03516] [SWS_Rte_03517] [SWS_Rte_03519] [SWS_Rte_03550] [SWS_Rte_03553] [SWS_Rte_03560] [SWS_Rte_03562] [SWS_Rte_03565] [SWS_Rte_03567] [SWS_Rte_03580] [SWS_Rte_03582] [SWS_Rte_03583] [SWS_Rte_03584] [SWS_Rte_03589] [SWS_Rte_07007] [SWS_Rte_07022] [SWS_Rte_07187]

Requirement	Description	Satisfied by
[SRS_Rte_00143]	Mode Switches	[SWS_Rte_02500] [SWS_Rte_02503] [SWS_Rte_02504] [SWS_Rte_02512] [SWS_Rte_02544] [SWS_Rte_02546] [SWS_Rte_02562] [SWS_Rte_02563] [SWS_Rte_02564] [SWS_Rte_02587] [SWS_Rte_02630] [SWS_Rte_02631] [SWS_Rte_02634] [SWS_Rte_02661] [SWS_Rte_02662] [SWS_Rte_02663] [SWS_Rte_02664] [SWS_Rte_02665] [SWS_Rte_02667] [SWS_Rte_02668] [SWS_Rte_02669] [SWS_Rte_02675] [SWS_Rte_02679] [SWS_Rte_02706] [SWS_Rte_02707] [SWS_Rte_02708] [SWS_Rte_02730] [SWS_Rte_06766] [SWS_Rte_06767] [SWS_Rte_06768] [SWS_Rte_06769] [SWS_Rte_06770] [SWS_Rte_06772] [SWS_Rte_06773] [SWS_Rte_06774] [SWS_Rte_06775] [SWS_Rte_06776] [SWS_Rte_06777] [SWS_Rte_06778] [SWS_Rte_06779] [SWS_Rte_06780] [SWS_Rte_06785] [SWS_Rte_06786] [SWS_Rte_06787] [SWS_Rte_06788] [SWS_Rte_06789] [SWS_Rte_06790] [SWS_Rte_06791] [SWS_Rte_06792] [SWS_Rte_06793] [SWS_Rte_06794] [SWS_Rte_06795] [SWS_Rte_06796] [SWS_Rte_06797] [SWS_Rte_07056] [SWS_Rte_07057] [SWS_Rte_07058] [SWS_Rte_07059] [SWS_Rte_07060] [SWS_Rte_07150] [SWS_Rte_07151] [SWS_Rte_07152] [SWS_Rte_07153] [SWS_Rte_07154] [SWS_Rte_07155] [SWS_Rte_07157] [SWS_Rte_07173] [SWS_Rte_07259] [SWS_Rte_07533] [SWS_Rte_07535] [SWS_Rte_07559] [SWS_Rte_07564]

Requirement	Description	Satisfied by
[SRS_Rte_00144]	RTE shall support the notification of mode switches via AUTOSAR interfaces	[SWS_Rte_02508] [SWS_Rte_02544] [SWS_Rte_02546] [SWS_Rte_02549] [SWS_Rte_02566] [SWS_Rte_02567] [SWS_Rte_02568] [SWS_Rte_02624] [SWS_Rte_02627] [SWS_Rte_02628] [SWS_Rte_02659] [SWS_Rte_02660] [SWS_Rte_02732] [SWS_Rte_02738] [SWS_Rte_03858] [SWS_Rte_03859] [SWS_Rte_06742] [SWS_Rte_06743] [SWS_Rte_06744] [SWS_Rte_06745] [SWS_Rte_06746] [SWS_Rte_06747] [SWS_Rte_06766] [SWS_Rte_06767] [SWS_Rte_06772] [SWS_Rte_06773] [SWS_Rte_06774] [SWS_Rte_06775] [SWS_Rte_06776] [SWS_Rte_06777] [SWS_Rte_06778] [SWS_Rte_06779] [SWS_Rte_06780] [SWS_Rte_06781] [SWS_Rte_06782] [SWS_Rte_06783] [SWS_Rte_06784] [SWS_Rte_06785] [SWS_Rte_06786] [SWS_Rte_06787] [SWS_Rte_06788] [SWS_Rte_06789] [SWS_Rte_06790] [SWS_Rte_06791] [SWS_Rte_06792] [SWS_Rte_06793] [SWS_Rte_06794] [SWS_Rte_06795] [SWS_Rte_06796] [SWS_Rte_06797] [SWS_Rte_07155] [SWS_Rte_07262] [SWS_Rte_07540] [SWS_Rte_07640] [SWS_Rte_07666] [SWS_Rte_08500] [SWS_Rte_08504] [SWS_Rte_08505] [SWS_Rte_08506] [SWS_Rte_08509] [SWS_Rte_08510]
[SRS_Rte_00145]	Compatibility mode	[SWS_Rte_01151] [SWS_Rte_01216] [SWS_Rte_01234] [SWS_Rte_01257] [SWS_Rte_01277] [SWS_Rte_01279] [SWS_Rte_01326] [SWS_Rte_03794]
[SRS_Rte_00146]	Vendor mode	[SWS_Rte_01234]
[SRS_Rte_00147]	Support for communication infrastructure time-out notification	[SWS_Rte_02589] [SWS_Rte_02590] [SWS_Rte_02599] [SWS_Rte_02600] [SWS_Rte_02604] [SWS_Rte_02607] [SWS_Rte_02609] [SWS_Rte_02610] [SWS_Rte_02611] [SWS_Rte_02629] [SWS_Rte_02666] [SWS_Rte_02703] [SWS_Rte_02710] [SWS_Rte_03759] [SWS_Rte_05021] [SWS_Rte_05022] [SWS_Rte_08004] [SWS_Rte_08061] [SWS_Rte_08062] [SWS_Rte_08501]

Requirement	Description	Satisfied by
[SRS_Rte_00148]	Support "Specification of Memory Mapping"	[SWS_Rte_03788] [SWS_Rte_05088] [SWS_Rte_05089] [SWS_Rte_06741] [SWS_Rte_07047] [SWS_Rte_07048] [SWS_Rte_07049] [SWS_Rte_07050] [SWS_Rte_07051] [SWS_Rte_07052] [SWS_Rte_07053] [SWS_Rte_07194] [SWS_Rte_07195] [SWS_Rte_07589] [SWS_Rte_07590] [SWS_Rte_07591] [SWS_Rte_07592] [SWS_Rte_07593] [SWS_Rte_07594] [SWS_Rte_07595] [SWS_Rte_07596] [SWS_Rte_07830] [SWS_Rte_07831] [SWS_Rte_07832]
[SRS_Rte_00149]	Support "Specification of Compiler Abstraction"	[SWS_Rte_01164] [SWS_Rte_03787] [SWS_Rte_07194] [SWS_Rte_07195] [SWS_Rte_07593] [SWS_Rte_07594] [SWS_Rte_07595] [SWS_Rte_07596] [SWS_Rte_07641]
[SRS_Rte_00150]	Support "Specification of Platform Types"	[SWS_Rte_01164] [SWS_Rte_07641]
[SRS_Rte_00152]	Support for port-defined argument values	[SWS_Rte_01166] [SWS_Rte_01360]
[SRS_Rte_00153]	Support for Measurement	[SWS_Rte_03900] [SWS_Rte_03901] [SWS_Rte_03902] [SWS_Rte_03903] [SWS_Rte_03904] [SWS_Rte_03950] [SWS_Rte_03951] [SWS_Rte_03972] [SWS_Rte_03973] [SWS_Rte_03974] [SWS_Rte_03975] [SWS_Rte_03976] [SWS_Rte_03977] [SWS_Rte_03978] [SWS_Rte_03979] [SWS_Rte_03980] [SWS_Rte_03981] [SWS_Rte_03982] [SWS_Rte_05087] [SWS_Rte_05101] [SWS_Rte_05102] [SWS_Rte_05120] [SWS_Rte_05121] [SWS_Rte_05122] [SWS_Rte_05123] [SWS_Rte_05124] [SWS_Rte_05125] [SWS_Rte_05136] [SWS_Rte_05168] [SWS_Rte_05169] [SWS_Rte_05170] [SWS_Rte_05172] [SWS_Rte_05174] [SWS_Rte_05175] [SWS_Rte_05176] [SWS_Rte_06700] [SWS_Rte_06701] [SWS_Rte_06702] [SWS_Rte_06726] [SWS_Rte_07160] [SWS_Rte_07174] [SWS_Rte_07197] [SWS_Rte_07198] [SWS_Rte_07344] [SWS_Rte_07349]

Requirement	Description	Satisfied by
[SRS_Rte_00154]	Support for Calibration	[SWS_Rte_03835] [SWS_Rte_03905] [SWS_Rte_03906] [SWS_Rte_03907] [SWS_Rte_03908] [SWS_Rte_03909] [SWS_Rte_03910] [SWS_Rte_03911] [SWS_Rte_03912] [SWS_Rte_03913] [SWS_Rte_03914] [SWS_Rte_03915] [SWS_Rte_03916] [SWS_Rte_03922] [SWS_Rte_03932] [SWS_Rte_03933] [SWS_Rte_03934] [SWS_Rte_03935] [SWS_Rte_03936] [SWS_Rte_03942] [SWS_Rte_03943] [SWS_Rte_03947] [SWS_Rte_03948] [SWS_Rte_03949] [SWS_Rte_03958] [SWS_Rte_03959] [SWS_Rte_03960] [SWS_Rte_03961] [SWS_Rte_03962] [SWS_Rte_03963] [SWS_Rte_03964] [SWS_Rte_03965] [SWS_Rte_03968] [SWS_Rte_03970] [SWS_Rte_03971] [SWS_Rte_05112] [SWS_Rte_05145] [SWS_Rte_05194] [SWS_Rte_07029] [SWS_Rte_07030] [SWS_Rte_07033] [SWS_Rte_07034] [SWS_Rte_07035] [SWS_Rte_07096] [SWS_Rte_07185] [SWS_Rte_07186] [SWS_Rte_07693]
[SRS_Rte_00155]	API to access calibration parameters	[SWS_Rte_01252] [SWS_Rte_01300] [SWS_Rte_03835] [SWS_Rte_03927] [SWS_Rte_03928] [SWS_Rte_03929] [SWS_Rte_03930] [SWS_Rte_03949] [SWS_Rte_03952] [SWS_Rte_07093] [SWS_Rte_07094] [SWS_Rte_07095]
[SRS_Rte_00156]	Support for different calibration data emulation methods	[SWS_Rte_03905] [SWS_Rte_03906] [SWS_Rte_03908] [SWS_Rte_03909] [SWS_Rte_03910] [SWS_Rte_03911] [SWS_Rte_03913] [SWS_Rte_03914] [SWS_Rte_03915] [SWS_Rte_03916] [SWS_Rte_03922] [SWS_Rte_03932] [SWS_Rte_03933] [SWS_Rte_03934] [SWS_Rte_03935] [SWS_Rte_03936] [SWS_Rte_03942] [SWS_Rte_03943] [SWS_Rte_03947] [SWS_Rte_03948] [SWS_Rte_03960] [SWS_Rte_03961] [SWS_Rte_03962] [SWS_Rte_03963] [SWS_Rte_03964] [SWS_Rte_03965] [SWS_Rte_03968] [SWS_Rte_03970] [SWS_Rte_03971] [SWS_Rte_05145]
[SRS_Rte_00157]	Support for calibration parameters in NVRAM	[SWS_Rte_03936]
[SRS_Rte_00158]	Support separation of calibration parameters	[SWS_Rte_03907] [SWS_Rte_03908] [SWS_Rte_03911] [SWS_Rte_03912] [SWS_Rte_03959] [SWS_Rte_05145] [SWS_Rte_05194] [SWS_Rte_07096]
[SRS_Rte_00159]	Sharing of calibration parameters	[SWS_Rte_02749] [SWS_Rte_02750] [SWS_Rte_03958] [SWS_Rte_05112] [SWS_Rte_07186]

Requirement	Description	Satisfied by
[SRS_Rte_00160]	Debounced start of Runnable Entities	[SWS_Rte_02697]
[SRS_Rte_00161]	Activation offset of Runnable Entities	[SWS_Rte_07000]
[SRS_Rte_00162]	"1:n" External Trigger communication	[SWS_Rte_07200] [SWS_Rte_07201] [SWS_Rte_07207] [SWS_Rte_07212] [SWS_Rte_07213] [SWS_Rte_07214] [SWS_Rte_07215] [SWS_Rte_07216] [SWS_Rte_07218] [SWS_Rte_07229] [SWS_Rte_07543]
[SRS_Rte_00163]	Support for InterRunnableTriggering	[SWS_Rte_07203] [SWS_Rte_07204] [SWS_Rte_07208] [SWS_Rte_07220] [SWS_Rte_07221] [SWS_Rte_07223] [SWS_Rte_07224] [SWS_Rte_07226] [SWS_Rte_07227] [SWS_Rte_07228] [SWS_Rte_07229] [SWS_Rte_07555]
[SRS_Rte_00164]	Ensure a unique naming of generated types visible in the global namespace.	[SWS_Rte_06706] [SWS_Rte_06707] [SWS_Rte_06708] [SWS_Rte_07110] [SWS_Rte_07111] [SWS_Rte_07114] [SWS_Rte_07115] [SWS_Rte_07116] [SWS_Rte_07117] [SWS_Rte_07118] [SWS_Rte_07119] [SWS_Rte_07144] [SWS_Rte_07145] [SWS_Rte_07146]
[SRS_Rte_00165]	Suppress identical "C" type re-definitions.	[SWS_Rte_07105] [SWS_Rte_07107] [SWS_Rte_07112] [SWS_Rte_07113] [SWS_Rte_07134] [SWS_Rte_07143] [SWS_Rte_07167] [SWS_Rte_07169]
[SRS_Rte_00166]	Use the AUTOSAR Standard Types in the global namespace if the AUTOSAR data type is mapped to an AUTOSAR Standard Type.	[SWS_Rte_07036] [SWS_Rte_07037] [SWS_Rte_07104] [SWS_Rte_07109] [SWS_Rte_07148] [SWS_Rte_07149] [SWS_Rte_07162] [SWS_Rte_07163] [SWS_Rte_07166]
[SRS_Rte_00167]	Encapsulate a Software Component local name space.	[SWS_Rte_01004] [SWS_Rte_01276] [SWS_Rte_02575] [SWS_Rte_03809] [SWS_Rte_03810] [SWS_Rte_03854] [SWS_Rte_05051] [SWS_Rte_05052] [SWS_Rte_06513] [SWS_Rte_06515] [SWS_Rte_06518] [SWS_Rte_06519] [SWS_Rte_06520] [SWS_Rte_06530] [SWS_Rte_06541] [SWS_Rte_06542] [SWS_Rte_06551] [SWS_Rte_06552] [SWS_Rte_06716] [SWS_Rte_06717] [SWS_Rte_06718] [SWS_Rte_07122] [SWS_Rte_07123] [SWS_Rte_07132] [SWS_Rte_07140] [SWS_Rte_07410] [SWS_Rte_07411] [SWS_Rte_07412] [SWS_Rte_08401] [SWS_Rte_08402] [SWS_Rte_08416]
[SRS_Rte_00168]	Typing of RTE API.	[SWS_Rte_07104]

Requirement	Description	Satisfied by
[SRS_Rte_00169]	Map code and memory allocated by the RTE to memory sections.	[SWS_Rte_05088] [SWS_Rte_05089] [SWS_Rte_06741] [SWS_Rte_07047] [SWS_Rte_07048] [SWS_Rte_07049] [SWS_Rte_07050] [SWS_Rte_07051] [SWS_Rte_07052] [SWS_Rte_07053] [SWS_Rte_07589] [SWS_Rte_07590] [SWS_Rte_07591] [SWS_Rte_07592]
[SRS_Rte_00170]	Provide used memory sections description	[SWS_Rte_05086] [SWS_Rte_05089] [SWS_Rte_06725]
[SRS_Rte_00171]	Support for fixed and constant data	[SWS_Rte_03930]
[SRS_Rte_00176]	Sharing of NVRAM data	[SWS_Rte_07301]
[SRS_Rte_00177]	Support of NvBlockComponentType	[SWS_Rte_07303] [SWS_Rte_07312] [SWS_Rte_07317] [SWS_Rte_07343] [SWS_Rte_07353] [SWS_Rte_07355] [SWS_Rte_07398] [SWS_Rte_07399] [SWS_Rte_07632] [SWS_Rte_07633] [SWS_Rte_08063] [SWS_Rte_08064]
[SRS_Rte_00178]	Data consistency of NvBlockComponentType	[SWS_Rte_07310] [SWS_Rte_07311] [SWS_Rte_07315] [SWS_Rte_07316] [SWS_Rte_07319] [SWS_Rte_07350] [SWS_Rte_07601] [SWS_Rte_07602] [SWS_Rte_07613] [SWS_Rte_07614]
[SRS_Rte_00179]	Support of Update Flag for Data Reception	[SWS_Rte_07385] [SWS_Rte_07386] [SWS_Rte_07387] [SWS_Rte_07390] [SWS_Rte_07391] [SWS_Rte_07392] [SWS_Rte_07393] [SWS_Rte_07654] [SWS_Rte_07689]
[SRS_Rte_00180]	DataSemantics range check during runtime	[SWS_Rte_01371] [SWS_Rte_01372] [SWS_Rte_01374] [SWS_Rte_03839] [SWS_Rte_03840] [SWS_Rte_03841] [SWS_Rte_03842] [SWS_Rte_03843] [SWS_Rte_03845] [SWS_Rte_03846] [SWS_Rte_03847] [SWS_Rte_03848] [SWS_Rte_03849] [SWS_Rte_03861] [SWS_Rte_07038] [SWS_Rte_08016] [SWS_Rte_08024] [SWS_Rte_08025] [SWS_Rte_08026] [SWS_Rte_08027] [SWS_Rte_08028] [SWS_Rte_08029] [SWS_Rte_08030] [SWS_Rte_08031] [SWS_Rte_08032] [SWS_Rte_08033] [SWS_Rte_08034] [SWS_Rte_08035] [SWS_Rte_08036] [SWS_Rte_08037] [SWS_Rte_08038] [SWS_Rte_08039] [SWS_Rte_08040] [SWS_Rte_08041] [SWS_Rte_08042] [SWS_Rte_08065]
[SRS_Rte_00181]	Conversion between internal and network data types	[SWS_Rte_03827] [SWS_Rte_03828] [SWS_Rte_06737] [SWS_Rte_06738] [SWS_Rte_07828] [SWS_Rte_07829] [SWS_Rte_07844]

Requirement	Description	Satisfied by
[SRS_Rte_00182]	Self Scaling Signals at Port Interfaces	[SWS_Rte_01374] [SWS_Rte_03815] [SWS_Rte_03816] [SWS_Rte_03817] [SWS_Rte_03818] [SWS_Rte_03819] [SWS_Rte_03820] [SWS_Rte_03821] [SWS_Rte_03822] [SWS_Rte_03823] [SWS_Rte_03829] [SWS_Rte_03830] [SWS_Rte_03831] [SWS_Rte_03832] [SWS_Rte_03833] [SWS_Rte_03855] [SWS_Rte_03856] [SWS_Rte_03857] [SWS_Rte_03860] [SWS_Rte_07038] [SWS_Rte_07091] [SWS_Rte_07092] [SWS_Rte_07099] [SWS_Rte_07925] [SWS_Rte_07926] [SWS_Rte_07928] [SWS_Rte_08801]
[SRS_Rte_00183]	RTE Read API returning the dataElement value	[SWS_Rte_07394] [SWS_Rte_07395] [SWS_Rte_07396]
[SRS_Rte_00184]	RTE Status "Never Received"	[SWS_Rte_07381] [SWS_Rte_07382] [SWS_Rte_07383] [SWS_Rte_07384] [SWS_Rte_07643] [SWS_Rte_07644] [SWS_Rte_07645] [SWS_Rte_08005] [SWS_Rte_08008] [SWS_Rte_08009] [SWS_Rte_08046] [SWS_Rte_08047] [SWS_Rte_08048]
[SRS_Rte_00185]	RTE API with Rte_IFeedback	[SWS_Rte_02589] [SWS_Rte_02590] [SWS_Rte_02608] [SWS_Rte_02666] [SWS_Rte_03836] [SWS_Rte_07367] [SWS_Rte_07374] [SWS_Rte_07375] [SWS_Rte_07376] [SWS_Rte_07378] [SWS_Rte_07379] [SWS_Rte_07646] [SWS_Rte_07647] [SWS_Rte_07648] [SWS_Rte_07650] [SWS_Rte_07651] [SWS_Rte_07652] [SWS_Rte_07660]

Requirement	Description	Satisfied by
[SRS_Rte_00189]	A2L Generation Support	[SWS_Rte_05087] [SWS_Rte_05118] [SWS_Rte_05119] [SWS_Rte_05120] [SWS_Rte_05121] [SWS_Rte_05122] [SWS_Rte_05123] [SWS_Rte_05124] [SWS_Rte_05125] [SWS_Rte_05126] [SWS_Rte_05127] [SWS_Rte_05128] [SWS_Rte_05129] [SWS_Rte_05130] [SWS_Rte_05131] [SWS_Rte_05132] [SWS_Rte_05133] [SWS_Rte_05135] [SWS_Rte_05136] [SWS_Rte_05137] [SWS_Rte_05138] [SWS_Rte_05139] [SWS_Rte_05140] [SWS_Rte_05141] [SWS_Rte_05142] [SWS_Rte_05143] [SWS_Rte_05144] [SWS_Rte_05152] [SWS_Rte_05153] [SWS_Rte_05154] [SWS_Rte_05155] [SWS_Rte_05156] [SWS_Rte_05157] [SWS_Rte_05158] [SWS_Rte_05159] [SWS_Rte_05160] [SWS_Rte_05161] [SWS_Rte_05162] [SWS_Rte_06702] [SWS_Rte_06726] [SWS_Rte_07097] [SWS_Rte_08313] [SWS_Rte_08314] [SWS_Rte_08315] [SWS_Rte_08316] [SWS_Rte_08317]
[SRS_Rte_00190]	Support for variable-length Data Types	[SWS_Rte_07813] [SWS_Rte_07814]
[SRS_Rte_00191]	Support for Variant Handling	[SWS_Rte_05168] [SWS_Rte_05169] [SWS_Rte_05174] [SWS_Rte_05175] [SWS_Rte_05176] [SWS_Rte_06500] [SWS_Rte_06501] [SWS_Rte_06507] [SWS_Rte_06509] [SWS_Rte_06510] [SWS_Rte_06512] [SWS_Rte_06543] [SWS_Rte_06546] [SWS_Rte_06547] [SWS_Rte_06549] [SWS_Rte_06550] [SWS_Rte_06553] [SWS_Rte_06611] [SWS_Rte_06612] [SWS_Rte_08066] [SWS_Rte_08067] [SWS_Rte_08068] [SWS_Rte_08069] [SWS_Rte_08070]
[SRS_Rte_00192]	Support multiple trace clients	[SWS_Rte_05086] [SWS_Rte_05091] [SWS_Rte_05092] [SWS_Rte_05093] [SWS_Rte_05106] [SWS_Rte_06725]
[SRS_Rte_00193]	Support for Runnable Entity execution chaining	[SWS_Rte_07800] [SWS_Rte_07802]
[SRS_Rte_00195]	No activation of Runnable Entities in terminated or restarting partitions	[SWS_Rte_07604] [SWS_Rte_07606]
[SRS_Rte_00196]	Inter-partition communication consistency	[SWS_Rte_02761] [SWS_Rte_05147] [SWS_Rte_07610]
[SRS_Rte_00200]	Support of unconnected R-Ports	[SWS_Rte_01330] [SWS_Rte_01331] [SWS_Rte_01333] [SWS_Rte_01334] [SWS_Rte_03785] [SWS_Rte_07655] [SWS_Rte_07663]

Requirement	Description	Satisfied by
[SRS_Rte_00201]	Contract Phase with Variant Handling support	[SWS_Rte_05104] [SWS_Rte_06500] [SWS_Rte_06502] [SWS_Rte_06505] [SWS_Rte_06514] [SWS_Rte_06515] [SWS_Rte_06516] [SWS_Rte_06518] [SWS_Rte_06519] [SWS_Rte_06520] [SWS_Rte_06521] [SWS_Rte_06522] [SWS_Rte_06523] [SWS_Rte_06524] [SWS_Rte_06525] [SWS_Rte_06526] [SWS_Rte_06527] [SWS_Rte_06528] [SWS_Rte_06529] [SWS_Rte_06530] [SWS_Rte_06531] [SWS_Rte_06539] [SWS_Rte_06540] [SWS_Rte_06541] [SWS_Rte_06542] [SWS_Rte_06543] [SWS_Rte_06546] [SWS_Rte_06551] [SWS_Rte_06552] [SWS_Rte_06620] [SWS_Rte_06638]
[SRS_Rte_00202]	Support for array size variants	[SWS_Rte_06500] [SWS_Rte_06505] [SWS_Rte_06543] [SWS_Rte_06546]
[SRS_Rte_00203]	API to read system constant	[SWS_Rte_03854] [SWS_Rte_06513] [SWS_Rte_06514] [SWS_Rte_06517]
[SRS_Rte_00204]	Support the selection / de-selection of SWC prototypes	[SWS_Rte_05104] [SWS_Rte_06544] [SWS_Rte_06545] [SWS_Rte_06601]
[SRS_Rte_00206]	Support the selection of a signal provider	[SWS_Rte_05104] [SWS_Rte_06544] [SWS_Rte_06545] [SWS_Rte_06601] [SWS_Rte_06602] [SWS_Rte_06603] [SWS_Rte_06604] [SWS_Rte_06605] [SWS_Rte_06606]
[SRS_Rte_00207]	Support N to M communication patterns while unresolved variations are affecting these communications.	[SWS_Rte_05104] [SWS_Rte_06544] [SWS_Rte_06545] [SWS_Rte_06601] [SWS_Rte_06602] [SWS_Rte_06603] [SWS_Rte_06604] [SWS_Rte_06605] [SWS_Rte_06606]
[SRS_Rte_00210]	Support for inter OS application communication	[SWS_Rte_02728] [SWS_Rte_02732] [SWS_Rte_02752] [SWS_Rte_02753] [SWS_Rte_02754] [SWS_Rte_02755] [SWS_Rte_02756] [SWS_Rte_03853] [SWS_Rte_07606] [SWS_Rte_08400] [SWS_Rte_08504] [SWS_Rte_08506]
[SRS_Rte_00211]	Cyclic time based scheduling of BSW Schedulable Entities	[SWS_Rte_02697] [SWS_Rte_07282] [SWS_Rte_07283] [SWS_Rte_07514] [SWS_Rte_07574] [SWS_Rte_07584]
[SRS_Rte_00212]	Activation Offset of BSW Schedulable Entities	[SWS_Rte_07520]

Requirement	Description	Satisfied by
[SRS_Rte_00213]	Mode Switches for BSW Modules	[SWS_Rte_02500] [SWS_Rte_02562] [SWS_Rte_02563] [SWS_Rte_02564] [SWS_Rte_02587] [SWS_Rte_02630] [SWS_Rte_02661] [SWS_Rte_02662] [SWS_Rte_02663] [SWS_Rte_02664] [SWS_Rte_02665] [SWS_Rte_02667] [SWS_Rte_02668] [SWS_Rte_02669] [SWS_Rte_02707] [SWS_Rte_02708] [SWS_Rte_07055] [SWS_Rte_07150] [SWS_Rte_07151] [SWS_Rte_07152] [SWS_Rte_07153] [SWS_Rte_07154] [SWS_Rte_07157] [SWS_Rte_07173] [SWS_Rte_07258] [SWS_Rte_07259] [SWS_Rte_07260] [SWS_Rte_07282] [SWS_Rte_07283] [SWS_Rte_07286] [SWS_Rte_07292] [SWS_Rte_07293] [SWS_Rte_07294] [SWS_Rte_07514] [SWS_Rte_07530] [SWS_Rte_07531] [SWS_Rte_07532] [SWS_Rte_07534] [SWS_Rte_07535] [SWS_Rte_07538] [SWS_Rte_07539] [SWS_Rte_07540] [SWS_Rte_07541] [SWS_Rte_07556] [SWS_Rte_07557] [SWS_Rte_07558] [SWS_Rte_07559] [SWS_Rte_07560] [SWS_Rte_07561] [SWS_Rte_07564] [SWS_Rte_07694] [SWS_Rte_08600] [SWS_Rte_08601]
[SRS_Rte_00214]	Common Mode handling for Basic SW and Application SW	[SWS_Rte_02697] [SWS_Rte_07258] [SWS_Rte_07259] [SWS_Rte_07286] [SWS_Rte_07535] [SWS_Rte_07564] [SWS_Rte_07582] [SWS_Rte_07583]
[SRS_Rte_00215]	API for Mode switch notification to the SchM	[SWS_Rte_07255] [SWS_Rte_07256] [SWS_Rte_07261] [SWS_Rte_08507]
[SRS_Rte_00216]	Triggering of BSW Schedulable Entities by occurrence of External Trigger	[SWS_Rte_07213] [SWS_Rte_07214] [SWS_Rte_07216] [SWS_Rte_07218] [SWS_Rte_07282] [SWS_Rte_07283] [SWS_Rte_07514] [SWS_Rte_07542] [SWS_Rte_07544] [SWS_Rte_07545] [SWS_Rte_07546] [SWS_Rte_07548] [SWS_Rte_07549]
[SRS_Rte_00217]	Synchronized activation of Runnable Entities and BSW Schedulable Entities	[SWS_Rte_02697] [SWS_Rte_07218] [SWS_Rte_07549]
[SRS_Rte_00218]	API for Triggering BSW modules by Triggered Events	[SWS_Rte_07263] [SWS_Rte_07264] [SWS_Rte_07266] [SWS_Rte_07267]
[SRS_Rte_00219]	Support for interlaced execution sequences of Runnable Entities and BSW Schedulable Entities	[SWS_Rte_02697] [SWS_Rte_07517] [SWS_Rte_07518]
[SRS_Rte_00220]	ECU life cycle dependent scheduling	[SWS_Rte_02538] [SWS_Rte_07580]
[SRS_Rte_00221]	Support for "BSW integration" builds	[SWS_Rte_07569] [SWS_Rte_07585]

Requirement	Description	Satisfied by
[SRS_Rte_00222]	Support shared exclusive areas in BSW Service Modules and the corresponding Service Component	[SWS_Rte_07250] [SWS_Rte_07251] [SWS_Rte_07252] [SWS_Rte_07253] [SWS_Rte_07254] [SWS_Rte_07522] [SWS_Rte_07523] [SWS_Rte_07524] [SWS_Rte_07578] [SWS_Rte_07579]
[SRS_Rte_00223]	Callout for partition termination notification	[SWS_Rte_07330] [SWS_Rte_07331] [SWS_Rte_07334] [SWS_Rte_07335] [SWS_Rte_07617] [SWS_Rte_07619] [SWS_Rte_07620] [SWS_Rte_07622]
[SRS_Rte_00224]	Callout for partition restart request	[SWS_Rte_07188] [SWS_Rte_07336] [SWS_Rte_07338] [SWS_Rte_07339] [SWS_Rte_07340] [SWS_Rte_07341] [SWS_Rte_07342] [SWS_Rte_07643] [SWS_Rte_07644] [SWS_Rte_07645]
[SRS_Rte_00228]	Fan-out NvBlock callback function	[SWS_Rte_07623] [SWS_Rte_07624] [SWS_Rte_07625] [SWS_Rte_07626] [SWS_Rte_07627] [SWS_Rte_07628] [SWS_Rte_07629] [SWS_Rte_07630] [SWS_Rte_07631] [SWS_Rte_07671] [SWS_Rte_07672]
[SRS_Rte_00229]	Support for Variant Handling of BSW Modules	[SWS_Rte_05104] [SWS_Rte_06500] [SWS_Rte_06503] [SWS_Rte_06504] [SWS_Rte_06507] [SWS_Rte_06508] [SWS_Rte_06532] [SWS_Rte_06533] [SWS_Rte_06534] [SWS_Rte_06535] [SWS_Rte_06536] [SWS_Rte_06537] [SWS_Rte_06543] [SWS_Rte_06544] [SWS_Rte_06545] [SWS_Rte_06546] [SWS_Rte_06548]
[SRS_Rte_00230]	Triggering of BSW Schedulable Entities by occurrence of Internal Trigger	[SWS_Rte_07229] [SWS_Rte_07551] [SWS_Rte_07552] [SWS_Rte_07553] [SWS_Rte_07554]
[SRS_Rte_00231]	Support native interface between Rte and Com for Strings and uint8 arrays	[SWS_Rte_07408] [SWS_Rte_07817]
[SRS_Rte_00232]	Synchronization of runnable entities	[SWS_Rte_07804] [SWS_Rte_07805] [SWS_Rte_07806] [SWS_Rte_07807]
[SRS_Rte_00233]	Generation of the Basic Software Module Description	[SWS_Rte_05086] [SWS_Rte_05165] [SWS_Rte_05166] [SWS_Rte_05167] [SWS_Rte_05177] [SWS_Rte_05179] [SWS_Rte_05180] [SWS_Rte_05181] [SWS_Rte_05182] [SWS_Rte_05183] [SWS_Rte_05184] [SWS_Rte_05185] [SWS_Rte_05186] [SWS_Rte_05187] [SWS_Rte_05188] [SWS_Rte_05189] [SWS_Rte_05190] [SWS_Rte_05191] [SWS_Rte_05192] [SWS_Rte_06725] [SWS_Rte_07085] [SWS_Rte_08305] [SWS_Rte_08404]
[SRS_Rte_00234]	Support for Record Type sub-setting	[SWS_Rte_07091] [SWS_Rte_07092] [SWS_Rte_07099]

Requirement	Description	Satisfied by
[SRS_Rte_00235]	Support queued triggers	[SWS_Rte_06720] [SWS_Rte_06721] [SWS_Rte_06722] [SWS_Rte_06723] [SWS_Rte_07087] [SWS_Rte_07088] [SWS_Rte_07089] [SWS_Rte_07090]
[SRS_Rte_00236]	Support for ModeInterfaceMapping	[SWS_Rte_08511] [SWS_Rte_08512] [SWS_Rte_08513] [SWS_Rte_08514]
[SRS_Rte_00237]	Time recurrent activation of Runnable Entities	[SWS_Rte_06728] [SWS_Rte_06729] [SWS_Rte_06730]
[SRS_Rte_00238]	Allow enabling of RTE-Feature to get the activating Event of Executable Entity	[SWS_Rte_01126] [SWS_Rte_07194] [SWS_Rte_07195] [SWS_Rte_07282] [SWS_Rte_08051] [SWS_Rte_08052] [SWS_Rte_08053] [SWS_Rte_08054] [SWS_Rte_08055] [SWS_Rte_08056] [SWS_Rte_08057] [SWS_Rte_08058] [SWS_Rte_08059] [SWS_Rte_08060] [SWS_Rte_08071]
[SRS_Rte_00239]	Support rule-based initialization of composite DataPrototypes and compound primitive DataPrototypes	[SWS_Rte_06733] [SWS_Rte_06734] [SWS_Rte_06735] [SWS_Rte_06736] [SWS_Rte_06764] [SWS_Rte_06765]
[SRS_Rte_00240]	Support of init runnables for initialization purposes	[SWS_Rte_06748] [SWS_Rte_06749] [SWS_Rte_06750] [SWS_Rte_06751] [SWS_Rte_06752] [SWS_Rte_06753] [SWS_Rte_06754] [SWS_Rte_06755] [SWS_Rte_06756] [SWS_Rte_06757] [SWS_Rte_06758] [SWS_Rte_06759] [SWS_Rte_06760] [SWS_Rte_06761] [SWS_Rte_06762] [SWS_Rte_06767] [SWS_Rte_06768] [SWS_Rte_06769] [SWS_Rte_06770]
[SRS_Rte_00241]	Support for Local or Remote Handling of BSW Service Calls on Partitioned Systems	[SWS_Rte_08765]
[SRS_Rte_00243]	Support for inter-partition communication of BSW modules	[SWS_Rte_08733] [SWS_Rte_08734] [SWS_Rte_08735] [SWS_Rte_08736] [SWS_Rte_08737] [SWS_Rte_08738] [SWS_Rte_08739] [SWS_Rte_08743] [SWS_Rte_08744] [SWS_Rte_08747] [SWS_Rte_08748] [SWS_Rte_08751] [SWS_Rte_08752] [SWS_Rte_08753] [SWS_Rte_08754] [SWS_Rte_08755] [SWS_Rte_08756] [SWS_Rte_08763] [SWS_Rte_08764] [SWS_Rte_08765] [SWS_Rte_08766]
[SRS_Rte_00244]	Support for bypass	[SWS_Rte_07833] [SWS_Rte_07834] [SWS_Rte_07835] [SWS_Rte_07836] [SWS_Rte_07837] [SWS_Rte_07838] [SWS_Rte_07839] [SWS_Rte_07840] [SWS_Rte_07841]

2 RTE Overview

2.1 The RTE in the Context of AUTOSAR

The Run-Time Environment (RTE) is at the heart of the AUTOSAR ECU architecture. The RTE is the realization (for a particular ECU) of the interfaces of the AUTOSAR Virtual Function Bus (VFB). The RTE provides the infrastructure services that enable communication to occur between AUTOSAR software-components as well as acting as the means by which AUTOSAR software-components access basic software modules including the OS and communication service.

The RTE encompasses both the variable elements of the system infrastructure that arise from the different mappings of components to ECUs as well as standardized RTE services.

In principle the RTE can be logically divided into two sub-parts realizing:

- the communication between software components
- the scheduling of the software components

To fully describe the concept of the RTE, the Basic Software Scheduler has to be considered as well. The Basic Software Scheduler schedules the schedulable entities of the basic software modules. In some documents the schedulable entities are also called main processing functions.

Due to the situation that the same OS Task might be used for the scheduling of software components and basic software modules the scheduling part of the RTE is strongly linked with the Basic Software Scheduler and can not be clearly separated.

The RTE and the Basic Software Scheduler is generated[1] for each ECU to ensure that the RTE and Basic Software Scheduler is optimal for the ECU [SRS_Rte_00023].

2.2 AUTOSAR Concepts

This section introduces some important AUTOSAR concepts and how they are implemented within the context of the RTE.

2.2.1 AUTOSAR Software-components

In AUTOSAR, “application” software is conceptually located above the AUTOSAR RTE and consists of “AUTOSAR application software-components” that are ECU and loca-

[1] An implementation is free to *configure* rather than *generate* the RTE and Basic Software Scheduler. The remainder of this specification refers to generation for reasons of simplicity only and these references should not be interpreted as ruling out either a wholly configured, or partially generated and partially configured, RTE and Basic Software Scheduler implementation.

tion independent and “AUTOSAR sensor-actuator components” that are dependent on ECU hardware and thus not readily relocatable for reasons of performance/efficiency. This means that, subject to constraints imposed by the system designer, an AUTOSAR software-component can be deployed to any available ECU during system configuration. The RTE is then responsible for ensuring that components can communicate and that the system continues to function as expected wherever the components are deployed. Considering sensor/actuator software components, they may only directly address the local ECU abstraction. Therefore, access to remote ECU abstraction shall be done through an intermediate sensor/actuator software component which broadcasts the information on the remote ECU. Hence, moving the sensor/actuator software components on different ECUs, may then imply to also move connected devices (sensor/actuator) to the same ECU (provided that efficient access is needed).

An AUTOSAR software-component is defined by a *type* definition that defines the component's interfaces. A component type is instantiated when the component is deployed to an ECU. A component type can be instantiated more than once on the same ECU in which case the component type is said to be “multiple instantiated”. The RTE supports per-instance memory sections that enable each component instance to have private states.

The RTE supports both AUTOSAR software-components where the source is available (“source-code software-components”) [SRS_Rte_00024] and AUTOSAR software-components where only the object code (“object-code software components”) is available [SRS_Rte_00140].

Details of AUTOSAR software-components in relation to the RTE are presented in Section 4.1.3.

2.2.2 Basic Software Modules

As well as “AUTOSAR software-components” an AUTOSAR ECU includes basic software modules. Basic software modules can access the ECU abstraction layer as well as other basic software modules directly and are thus neither ECU nor location independent [2].

An “AUTOSAR software-component” *cannot* directly access basic software modules – all communication is via AUTOSAR interfaces and therefore under the control of the RTE. The requirement to not have direct access applies to all *Basic Software Modules* including the operating system [SRS_Rte_00020] and the communication service.

[2] The functionality provided by a basic software module cannot be relocated in another ECU. However, the source of some basic software modules can be reused on other ECUs.

2.2.3 Communication

The communication interface of an AUTOSAR software-component consists of several ports (which are characterized by port-interfaces). An AUTOSAR software-component can communicate through its interfaces with other AUTOSAR software-components (whether that component is located on the same ECU or on a different ECU) or with basic software modules that have ports and runnables (i.e ServiceSwComponents, EcuAbstractionSwComponents and ComplexDeviceDriverSwComponents) and are located on the same ECU. This communication can *only* occur via the component's ports. A port can be categorized by either a sender-receiver or client-server port-interface. A sender-receiver interface provides a message passing facility whereas a client-server interface provides function invocation.

2.2.3.1 Communication Paradigms

The RTE provides different paradigms for the communication between software-component instances: sender-receiver (signal passing), client-server (function invocation), mode switch, and NvBlockSwComponentType interaction.

Each communication paradigm can be applied to intra-partition software-component distribution (which includes both intra-task and inter-task distribution, within the same Partition), inter-Partition software-component distribution, and inter-ECU software-component distribution. Intra-task communication occurs between runnable entities that are mapped to the same OS task whereas inter-task communication occurs between runnable entities mapped to different tasks of the same Partition and can therefore involve a context switch. Inter-Partition communication occurs between runnable entities in components mapped to different partitions of the same ECU and therefore involve a context switch and crossing a protection boundary (memory protection, timing protection, isolation on a core). Inter-ECU communication occurs between runnable entities in components that have been mapped to different ECUs and so is inherently concurrent and involves potentially unreliable communication.

Details of the communication paradigms that are supported by the RTE are contained in Section 4.3.

2.2.3.2 Communication Modes

The RTE supports two modes for sender-receiver communication:

- **Explicit** — A component uses explicit RTE API calls to send and receive data elements [SRS_Rte_00098].
- **Implicit** — The RTE automatically reads a specified set of data elements before a runnable is invoked and automatically writes (a different) set of data elements after the runnable entity has terminated [SRS_Rte_00128] [SRS_Rte_00129].

The term “implicit” is used here since the runnable does not actively initiate the reception or transmission of data.

Implicit and explicit communication is considered in greater detail in Section 4.3.1.5.

2.2.3.3 Static Communication

[SWS_Rte_06026] ⌈ The RTE shall support static communication only. ⌋*(SRS_Rte_00025)*

Static communication includes only those communication connections where the source(s) and destination(s) of all communication is known at the point the RTE is generated. [SRS_Rte_00025]. This includes also connections which are subject to variability because the variant handling concept of AUTOSAR does only support the selection of connectors from a superset of possible connectors to define a particular variant.
Dynamic reconfiguration of communication is not supported due to the run-time and code overhead which would therefore limit the range of devices for which the RTE is suitable.

2.2.3.4 Multiplicity

As well as point to point communication (i.e. “1:1”) the RTE supports communication connections with multiple providers or requires:

- When using sender-receiver communication, the RTE supports both “1:n” (single sender with multiple receivers) [SRS_Rte_00028] and “n:1” (multiple senders and a single receiver) [SRS_Rte_00131] communication with the restriction that multiple senders are not allowed for `mode switch notifications`, see meta-model restrictions [SWS_Rte_02670].

  The execution of the multiple senders or receivers is not coordinated by the RTE. This means that the actions of different software-components are independent – the RTE does not ensure that different senders transmit data simultaneously and does not ensure that all receivers read data or receive events simultaneously.

- When using client-server communication, the RTE supports “n:1” (multiple clients and a single server) [SRS_Rte_00029] communication. The RTE does *not* support “1:n” (single client with multiple servers) client-server communication.

Irrespective of whether “1:1”, “n:1” or “1:n” communication is used, the RTE is responsible for implementing the communication connections and therefore the AUTOSAR software-component is unaware of the configuration. This permits an AUTOSAR software-component to be redeployed in a different configuration without modification.

2.2.4 Concurrency

AUTOSAR software-components have no direct access to the OS and hence there are no "tasks" in an AUTOSAR application. Instead, concurrent activity within AUTOSAR is based around `RunnableEntity`s within components that are invoked by the RTE.

The AUTOSAR VFB specification [1] defines a runnable entity as a "sequence of instructions that can be started by the Run-Time Environment". A component provides one[3] or more runnable entities [SRS_Rte_00031] and each runnable entity has exactly one entry point. An entry point defines the *symbol* within the software-component's code that provides the implementation of a runnable entity.

The RTE is responsible for invoking runnable entities – AUTOSAR software-components are not able to (dynamically) create private threads of control. Hence, all activity within an AUTOSAR application is initiated by the triggering of runnable entities by the RTE as a result of `RTEEvent`s.

An `RTEEvent` encompasses all possible situations that can trigger execution of a runnable entity by the RTE. The different classes of `RTEEvent` are defined in Section 5.7.5.

The RTE supports runnable entities in any component that has an AUTOSAR interface - this includes AUTOSAR software-components and basic software modules.[4]

Runnable entities are divided into multiple categories with each category supporting different facilities. The categories supported by the RTE are described in Section 4.2.2.3.

2.3 The RTE Generator

The RTE generator is one of a set of tools[5] that create the realization of the AUTOSAR virtual function bus for an ECU based on information in the *ECU Configuration Description*. The RTE Generator is responsible for creating the AUTOSAR software-component API functions that link AUTOSAR software-components to the OS and manage communication between AUTOSAR software-components and between AUTOSAR software-components and basic software modules.

Additionally the RTE Generator creates both the *Basic Software Scheduler* and the *Basic Software Scheduler* API functions for each particular instance of a *Basic Software Module*.

The RTE generation process for SWCs has two main phases:

[3]The VFB specification does not permit zero runnable entities.

[4]The OS and COM are basic software modules but present a *standardized interface* to the RTE and have no AUTOSAR interface. The OS and COM therefore do not have runnable entities.

[5]The RTE generator works in conjunction with other tools, for example, the OS and COM generators, to fully realize the AUTOSAR VFB.

- **RTE Contract phase** – a limited set of information about a component, principally the AUTOSAR interface definitions, is used to create an application header file for a component type. The application header file defines the “contract” between component and RTE.
- **RTE Generation phase** - all relevant information about components, their deployment to ECUs and communication connections is used to generate the RTE and optionally the Ioc configuration [4]. One RTE is generated for each ECU in the system.

The two-phase development model ensures that the RTE generated application header files are available for use for source-code AUTOSAR software-components as well as object-code AUTOSAR software-components with both types of component having access to all definitions created as part of the RTE generation process.

The RTE generation process, and the necessary inputs in each phase, are considered in more detail in chapter 3.

2.4 Design Decisions

This section details decisions that affect both the general direction that has been taken as well as the actual content of this document.

1. The role of this document is to specify RTE behavior, not RTE implementation. Implementation details should not be considered to be part of the RTE software specification unless they are explicitly marked as RTE requirements.
2. An AUTOSAR system consists of multiple ECUs each of which contains an RTE that may have been generated by different RTE generators. Consequently, the specification of how RTEs from multiple vendors interoperate is considered to be within the scope of this document.
3. The RTE does not have sufficient information to be able to derive a mapping from runnable entity to OS task. The decision was therefore taken to require that the mapping be specified as part of the RTE input.
4. Support for C++ is provided by making the C RTE API available for C++ components rather than specifying a completely separate object-oriented API. This decision was taken for two reasons; firstly the same interface for the C and C++ simplifies the learning curve and secondly a single interface greatly simplifies both the specification and any subsequent implementations.
5. There is no support within the specification for Java.
6. The AUTOSAR meta-model is a highly expressive language for defining systems however for reasons of practicality certain restrictions and constraints have been placed on the use of the meta-model. The restrictions are described in Appendix A.

3 RTE Generation Process

This chapter describes the methodology of the RTE and Basic Software Scheduler generation. For a detailed description of the overall AUTOSAR methodology refer to methodology document [6].

[SWS_Rte_02514] ⌈ The RTE generator shall produce the same RTE API, RTE code, SchM API and SchM code when the input information is the same. ⌋*(SRS_Rte_00065)*

The RTE Generator gets involved in the AUTOSAR Methodology several times in different roles. Technically the RTE Generator can be implemented as one tool which is invoked with options to switch between the different roles. Or the RTE Generator could be a set of separate tools. In the following section the individual applications of the RTE Generator are described based on the roles that are take, not necessarily the actual tools.

The RTE Generator is used in different roles for the following phases:

- RTE Contract Phase
- Basic Software Scheduler Contract Phase
- PreBuild Data Set Contract Phase
- Basic Software Scheduler Generation Phase
- RTE Generation Phase
- PreBuild Data Set Generation Phase
- PostBuild Data Set Generation Phase

RTE Generator for Software-Components

In Figure 3.1 the overall AUTOSAR Methodology wrt. Application SW-Components and the RTE Generator.

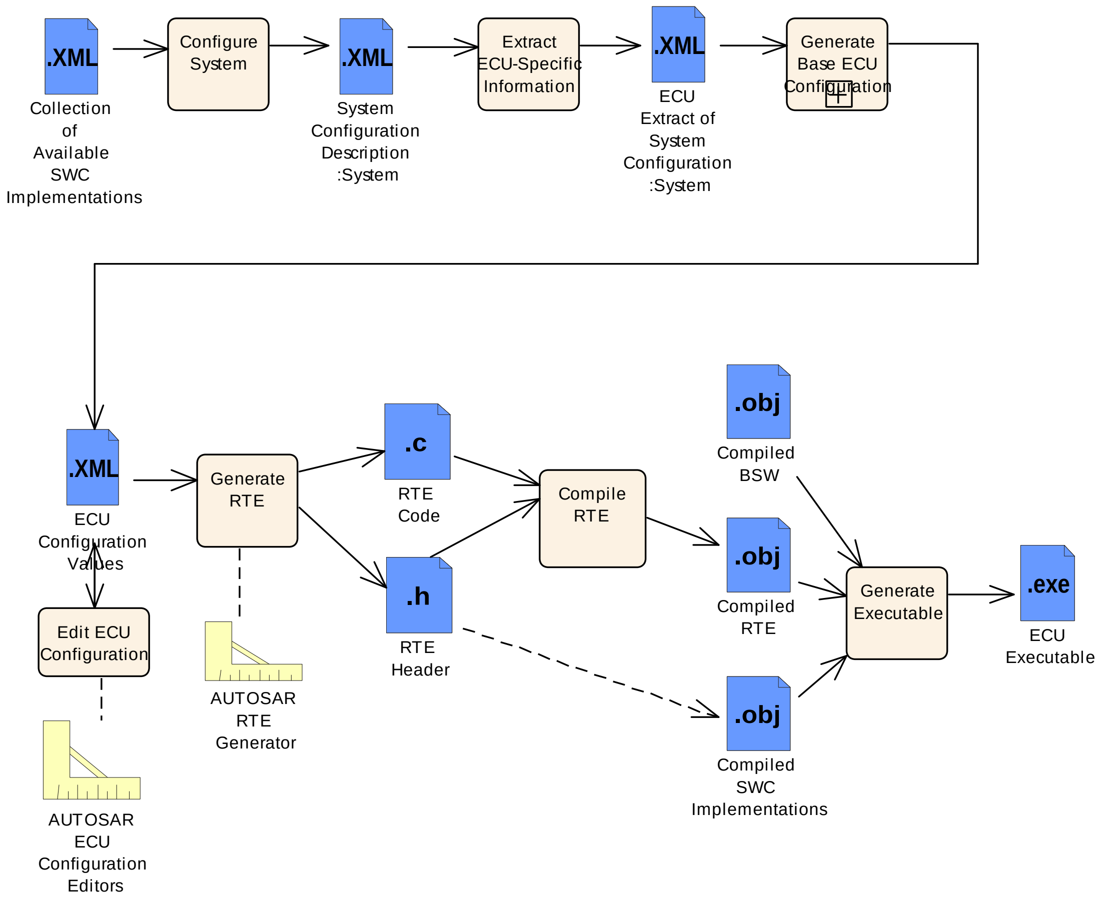


Figure 3.1: System Build Methodology

The whole vehicle functionality is described with means of `CompositionSwComponents`, `SwComponentPrototypes` and `AtomicSwComponents` [2]. In the `CompositionSwComponent` descriptions the connections between the software-components' ports are also defined. Such a collection of software-components connected to each other, without the mapping on actual ECUs, is called the VFB view.

During the 'Configure System' step the needed software-components, the available ECUs and the System Constraints are resolved into a System Configuration Description. Now the `SwComponentPrototypes` and thus the associated `AtomicSwComponents` are mapped on the available ECUs.

Since in the VFB view the communication relationships between the `AtomicSwComponents` have been described and the mapping of each `SwComponentPrototypes` and `AtomicSwComponents` to a specific ECU has been fixed, the communication matrix can be generated. In the `SwComponentType Description` (using the format of the AUTOSAR Software Component Template [2]) the data that is exchanged through ports is defined in an abstract way. Now the 'System Configuration Generator' needs to define system signals (including the actual signal length and the frames in which they will be transmitted) to be able to transmit the application data over the network. COM signals that correspond to the system signals will be later used by the 'RTE Generator' to actually transmit the application data.

In the next step the 'System Configuration Description' is split into descriptions for each individual ECU. During the generation of the Ecu Extract also the hierarchical structure of the CompositionSwComponents of the VFB view is flattened and the SwComponentPrototypes of the ECU Extract represent actual instances. The Ecu Extract only contains information necessary to configure one ECU individually and it is fed into the ECU Configuration for each ECU.

[SWS_Rte_05000] ⌈ The RTE is configured and generated for each ECU instance individually. ⌋*(SRS_Rte_00021)*

The 'ECU Configuration Editors' (see also Section 3.3) are working iteratively on the 'ECU Configuration Values' until all configuration issues are resolved. There will be the need for several configuration editors, each specialized on a specific part of ECU Configuration. So one editor might be configuring the COM stack (not the communication matrix but the interaction of the individual modules) while another editor is used to configure the RTE.

Since the configuration of a specific Basic-SW module is not entirely independent from other modules there is the need to apply the editors several times to the 'ECU Configuration Values' to ensure all configuration parameters are consistent.

Only when the configuration issues are resolved the 'RTE Generator' will be used to generate the actual RTE code (see also Section 3.4.2) which will then be compiled and linked together with the other Basic-SW modules and the software-components code.

The 'RTE Generator' needs to cope with many sources of information since the necessary information for the RTE Generator is based on the 'ECU Configuration Values' which might be distributed over several files and itself references to multiple other AUTOSAR descriptions.

[SWS_Rte_08769] ⌈ RTE Generator shall support for reading single files and of sets of files that are stored in a file system. The tool shall provide a mechanism to select a specific file and sets of files in the file system. ⌋*(SRS_Rte_00048)*

An AUTOSAR XML description can be shipped in several files. Some files could contain data types others could contain interfaces, etc.

[SWS_Rte_08770] ⌈ An RTE Generator tools SHALL support the merging of AUTOSAR models that have been split up and stored in multiple partial models while reading an set of files. Thereby the to be supported minimum granularity of an AUTOSAR model is defined by ≪atpSplitable≫. The Merging of a model also includes the resolution of references. The RTE Generator SHALL be able to read the submodels in any order. There is no preference. ⌋*(SRS_Rte_00048)*

[SWS_Rte_08771] ⌈ RTE Generator SHALL support the interpretation and creation of AUTOSAR XML descriptions. These descriptions SHALL be 'well-formed' and 'valid' as defined by the XML recommendation, W3C XML 1.1 Specification, whether used with or without the document's corresponding AUTOSAR XML schema(s). In other words: Even if the tool does not use standard XML mechanisms for validating the XML descriptions it SHALL ensure that the XML descriptions can be successfully validated against the AUTOSAR XML schema. ⌋*(SRS_Rte_00048)*

[SWS_Rte_08772] ⌈ If an RTE Generator wants to validate an AUTOSAR XML description against an AUTOSAR schema, it SHALL provide the necessary schema files in its own resources.

An RTE Generator shall use the SYSTEM-Identifier in the xsi:schemaLocation to identify an appropriate schema file. ⌋*(SRS_Rte_00048)*

[SWS_Rte_08773] ⌈ RTE Generator shall provide a serialization for XML. ⌋*(SRS_Rte_00048)*

[SWS_Rte_08774] ⌈ RTE Generator shall not change model content passed to the Generator ⌋*(SRS_Rte_00048)*

[SWS_Rte_08775] ⌈ An RTE Generator MAY support the AUTOSAR extension mechanism `SDG`s if applicable.

If the RTE Generator does not need the additional information for its intended purpose it SHALL ignore the irrelevant extensions `SDG`s. ⌋*(SRS_Rte_00048)*

[SWS_Rte_08776] ⌈ An RTE Generator may use well structured error messages. ⌋*(SRS_Rte_00048)*

The following list is a collection of proposed information items in particular applicable to log files used for exchanging information about errors.

- **ErrorCode** – A symbolic name for the message text
- **StandardErrorCode** – The reference to the AUTOSAR error code
- **ConstraintCode** – Reference to the semantic constraint mentioned in the AUTOSAR template specification.
- **Signature** – Signature of the message for duplicate checks
- **Timestamp** – A time stamp for the message
- **ShortName** – A unique identification which allows to refer to particular error messages
  This can also be used to establish references between error messages, e.g. for screening and also to trace back to root cause
- **Desc** – The human readable message text
- **Component** – Such information item may help the user to locate the problem in the model
- **BaseUrl** – An url for a base directory which can be used as basis for file references in a log file. This is typically the root direactory of a project structure.
- **ColumnNumber** – The column of the error position
- **LineNumber** – The line number of the error position

- **LongName** – The title of the error message
- **ObjectCategory** – The category of for example the involved ApplicationPrimitve-DataType (e.g.VALUE)
- **PrimaryErrorReference** – Reference to the root cause if applicable
- **ScopeEntryReference** – Reference to a scoping message if applicable
- **Object** – The shortName based reference to the AUTOSAR element which caused the error
- **ToolName** – The name of the tool which reported the error
- **ToolVersion** – The version of the tools which reported the error
- **IncidentUrl** – The Url which refers to the artifact in which the error occurs
- **Value** – The actual found value which caused the problem

This is just a rough sketch of the main steps necessary to build an ECU with AUTOSAR and how the RTE is involved in this methodology. For a more detailed description of the AUTOSAR Methodology please refer to the methodology document [6]. In the next sections the steps with RTE interaction are explained in more detail.

RTE Generator for Basic Software Scheduler

In Figure 3.2 the overall AUTOSAR Methodology wrt. Basis Software Scheduler and the RTE Generator interaction.

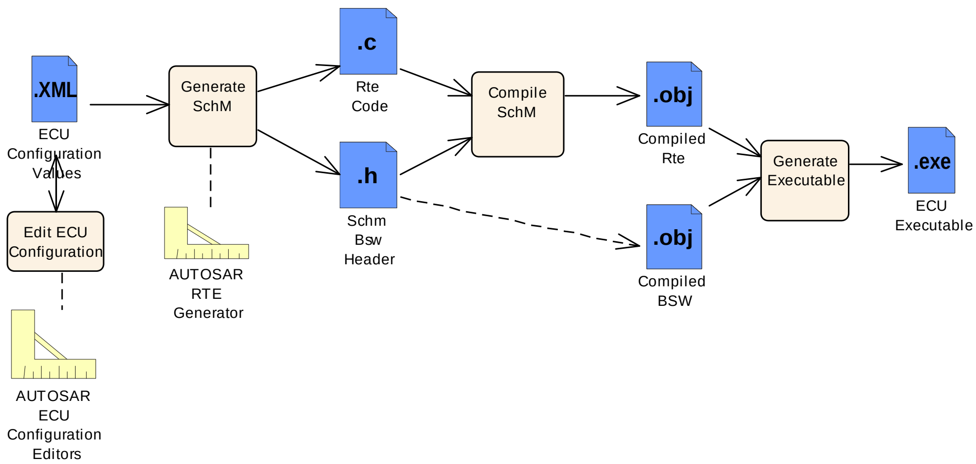


Figure 3.2: Basic Software Scheduler Methodology

The ECU Configuration phase is the start of the Basic Software Scheduler configuration where all the requirements of the different Basic Software Modules are collected. The Input information is provided in the Basic Software Module Descriptions [9] of the individual Basic Software Modules.

The Basic Software Scheduler configuration is then generated into the Basic Software Scheduler code which is compiled and built into the Ecu executable.

3.1 Contract Phase

3.1.1 RTE Contract Phase

To be able to support the AUTOSAR software-component development with RTE-specific APIs the 'Component API' (application header file) is generated from the 'software-component Internal Behavior Description' (see Figure 3.1) by the RTE Generator in the so called 'RTE Contract Phase' (see Figure 3.3).

In the software-component Interface description – which is using the AUTOSAR Software Component Template – at least the AUTOSAR Interfaces of the particular software-component have to be described. This means the software-component Types with Ports and their Interfaces. In the software-component Internal Behavior description additionally the Runnable Entities and the RTE Events are defined. From this information the RTE Generator can generate specific APIs to access the Ports and send and receive data.

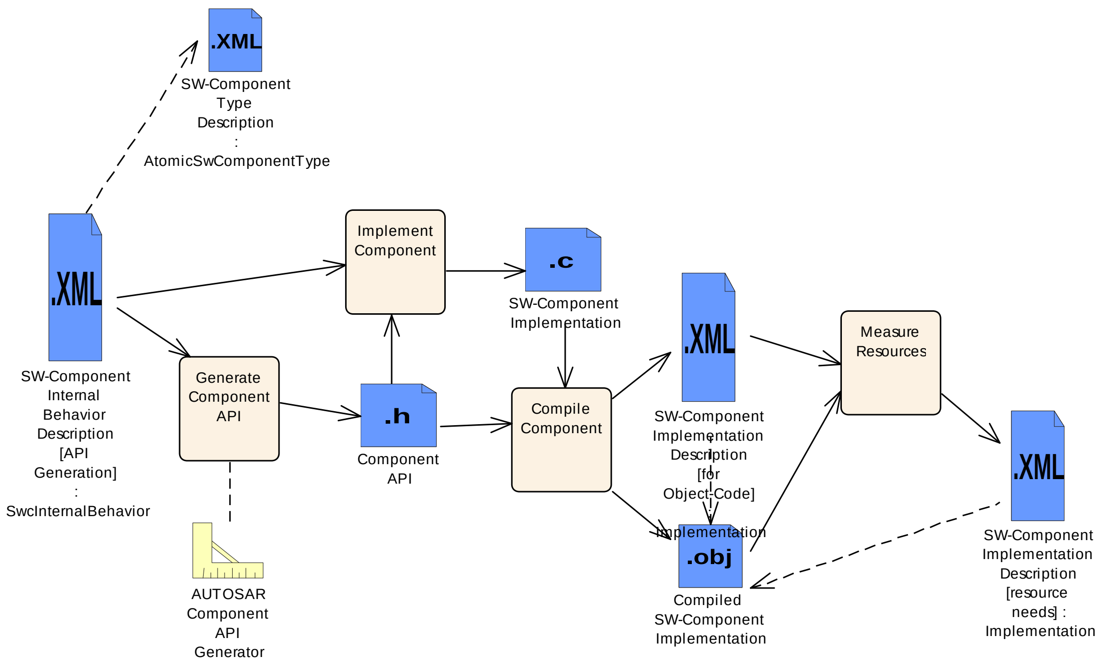


Figure 3.3: RTE Contract Phase

With the generated 'Component API' (application header file) the Software Component developer can provide the Software Component's source code without being concerned as to whether the communication will later be local or using some network(s).

It has to be considered that the AUTOSAR software-component development process is iterative and that the AUTOSAR software-component description might be changed during the development of the AUTOSAR software-component. This requires the application header file to be regenerated to reflect the changes done in the software-component description.

When the software-component has been compiled successfully the 'Component Implementation Description Generation' tool will analyze the resulting object files and enhance the software-component description with the information from the specific implementation. This includes information about the actual memory needs for ROM as well as for RAM and goes into the 'Component Implementation Description' section of the AUTOSAR Software Component Template.

Please note that in case of implemented `PreCompileTime Variability` additionally the *PreBuild Data Set Contract Phase* is required 3.2 to be able to compile the software component.

So when a software-component is delivered it will consist of the following parts:

- SW-Component Type Description
- SW-Component Internal Behavior Description
- The actual SW-Component implementation and/or compiled SW-Component
- SW-Component Implementation Description

The above listed information will be needed to provide enough information for the System Generation steps when the whole system is assembled.

3.1.2 Basic Software Scheduler Contract Phase

To be able to support the *Basic Software Module* development with *Basic Software Scheduler* specific APIs the *Module Interlink Header* (6.3.2) and *Module Interlink Types Header* (6.3.1) containing the definitions and declaration for the *Basic Software Scheduler* API related to the single *Basic Software Module* instance is generated by the RTE Generator in the so called '*Basic Software Scheduler Contract Phase*'.

The required input is

- *Basic Software Module Description* and
- *Basic Software Module Internal Behavior* and
- *Basic Software Module Implementation*

Please note that in case of implemented `PreCompileTime Variability` additionally the *PreBuild Data Set Contract Phase* is required 3.2 to be able to compile the *Basic Software Module*.

3.2 PreBuild Data Set Contract Phase

In the *RTE PreBuild Data Set Contract Phase* are the *Condition Value Macros* (see 5.3.8.2.2) generated which are required to resolve the implemented `PreBuild Variability` of a particular software component or *Basic Software Module*.

The particular values are defined via PredefinedVariants. These Predefined-Variant elements containing definition of SwSystemconstValues for SwSystem-consts which shall be applied when resolving the variability during ECU Configuration.

The output of this phase is the *RTE Configuration Header File* 5.3.8. This file is required to compile a particular variant of a software component using PreCompile-Time Variability. The *Condition Value Macros* are used for the implementation of PreCompileTime Variability with preprocessor statements and therefore are needed to run the C preprocessor resolving the implemented variability.

3.3 Edit ECU Configuration of the RTE

During the configuration of an ECU the RTE also needs to be configured. This is divided into several steps which have to be performed iteratively: The configuration of the RTE and the configuration of other modules.

So first the 'RTE Configuration Editor' needs to collect all the information needed to establish an operational RTE. This gathering includes information on the software-component instances and their communication relationships, the Runnable Entities and the involved RTE-Events and so on. The main source for all this information is the 'ECU Configuration Values', which might provide references to further descriptions like the software-component description or the System Configuration description.

An additional input source is the Specification of Timing Extensions [14]. This template can be used to specify the execution order of runnable entities (see section 'Execution order constraint'). An 'RTE Configuration Editor' can use the information to create and check the configuration of the Rte Event to Os task mapping (see section 7.6.1).

The usage of 'ECU Configuration Editors' covering different parts of the 'ECU Configuration Values' will – if there are no cyclic dependencies which do not converge – converge to a stable configuration and then the ECU Configuration process is finished. A detailed description of the ECU Configuration can be found in [5]. The next phase is the generation of the actual RTE code.

3.4 Generation Phase

After the ECU has been entirely configured the generation of the actual RTE inclusive the *Basic Software Scheduler* part can be performed. Since all the relationships to and from the other Basic-SW modules have been already resolved during the ECU Configuration phase, the generation can be performed in parallel for all modules (see Figure 3.4).

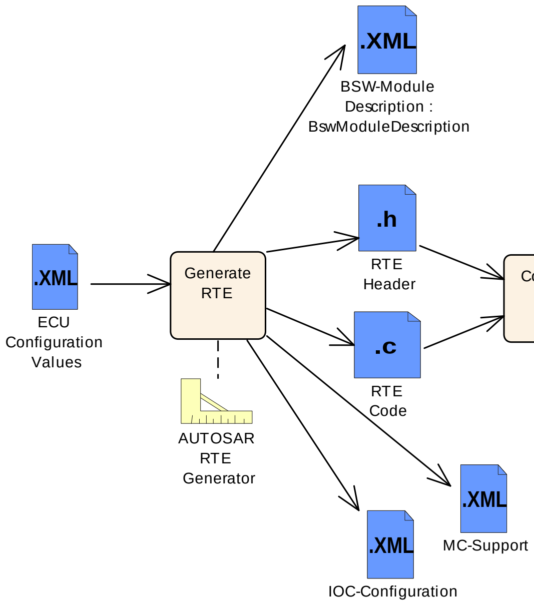


Figure 3.4: RTE Generation Phase

The *Basic Software Scheduler* is a part of the *Rte* and therefore not explicitly shown in figure 3.4.

3.4.1 Basic Software Scheduler Generation Phase

Depending on the complexity of the ECU and the cooperation model of the different software vendors it might be required to integrate the *Basic Software* stand alone without software components.

Therefore the RTE Generator has to support the generation of the *Basic Software Scheduler* without software component related RTE fragments. The *Basic Software Scheduler Generation Phase* is only applicable for software builds which are not containing any kind of software components.

[SWS_Rte_07569] ⌈ In the *Basic Software Scheduler Generation Phase* the RTE Generator shall generate the *Basic Software Scheduler* without the RTE functionality. ⌋*(SRS_Rte_00221)*

In this case the RTE Generator generates the API for *Basic Software Modules* and the *Basic Software Scheduling* code only. When the input contains software component related information this information raises an error.

For instance:

- *Application Header Files* are not generated for the software components contained in the ECU extract.
- Mapped RTEEvents are not permitted and the runnable calls are not generated into the OS task bodies. Nevertheless all OS task bodies related to the *Basic Software Scheduler* configuration are generated.
- Mode machine instances mapped to the RTE are not supported.

[SWS_Rte_07585] ⌈ In the *Basic Software Scheduler Generation Phase* the RTE Generator shall reject input configuration containing software component related information. ⌋*(SRS_Rte_00221)*

The RTE Generator in the *Basic Software Scheduler Generation Phase* is also responsible to generate additional artifacts which contribute to the further build, deployment and calibration of the ECU's software.

[SWS_Rte_06725] ⌈ The RTE Generator in *Basic Software Scheduler Generation Phase* shall provide its *Basic Software Module Description* in order to capture the generated RTE's / Basic Software Scheduler attributes. ⌋*(SRS_Rte_00170, SRS_Rte_00192, SRS_Rte_00233)*

Details about the Basic Software Module Description generation can can be found in section 3.4.3.

[SWS_Rte_06726] ⌈ The RTE Generator in *Basic Software Scheduler Generation Phase* shall provide an *MC-Support* (Measurement and Calibration) description as part of the *Basic Software Module Description*. ⌋*(SRS_Rte_00153, SRS_Rte_00189)*

Details about the *MC-Support* can be found in section 4.2.8.4.

For software builds which are containing software components the *RTE Generation Phase* 3.4.2 is applicable where the *Basic Software Scheduler* part of the RTE is generated as well.

3.4.2 RTE Generation Phase

The actual AUTOSAR software-components and Basic-SW modules code will be linked together with the RTE and *Basic Software Scheduler* code to build the entire ECU software.

Please note that in case of implemented PreCompileTime Variability additionally the *PreBuild Data Set Generation Phase* is required (see section 3.5) to be able to compile the ECU software. Further on in case of implemented PostBuild Variability *PostBuild Data Set Generation Phase* is required (see section 3.6) to be able to link the full ECU software.

The RTE Generator in the *Generation Phase* is also responsible to generate additional artifacts which contribute to the further build, deployment and calibration of the ECU's software.

[SWS_Rte_05086] ⌈ The RTE Generator in Generation Phase shall provide its *Basic Software Module Description* in order to capture the generated RTE's attributes. ⌋*(SRS_Rte_00170, SRS_Rte_00192, SRS_Rte_00233)*

Details about the Basic Software Module Description generation can can be found in section 3.4.3.

[SWS_Rte_05087] ⌈ The RTE Generator in Generation Phase shall provide an *MC-Support* (Measurement and Calibration) description as part of the *Basic Software Module Description*. ⌋*(SRS_Rte_00153, SRS_Rte_00189)*

Details about the *MC-Support* can be found in section 4.2.8.4.

[SWS_Rte_05147] ⌈ The RTE Generator in Generation Phase shall provide the configuration for the Ioc module [4] if the Ioc module is used. ⌋*(SRS_Rte_00196)*

The RTE generates the IOC configurations and uses an implementation specific deterministic generation scheme. This generation scheme can be used by implementations to reuse these IOC configurations (e.g. if the configuration switch `strictConfigurationCheck` is used).

[SWS_Rte_08400] ⌈ The RTE Generator in Generation Phase shall generate internal ImplementationDataTypes types used for IOC configuration. ⌋*(SRS_Rte_00210)*

The corresponding C data types will be generated into the *Rte_Type.h*. This *Rte_Type.h* header file will be used by the IOC to get the types for the IOC API.

Changing the RTE generator will require a new IOC configuration generation.

Details about the Ioc module can be found in section 4.3.4.1.

[SWS_Rte_08305] ⌈ The RTE Generator in Generation Phase shall ignore XML-Content categorized as ICS. ⌋*(SRS_Rte_00233)*

3.4.3 Basic Software Module Description generation

The Basic Software Module Description [9] generated by the RTE Generator in generation phase describes features of the actual RTE code. The following requirements specify which elements of the Basic Software Module Description are mandatory to be generated by the RTE Generator.

3.4.3.1 Bsw Module Description

[SWS_Rte_05165] ⌈ The RTE Generator in Generation Phase shall provide the BswModuleDescription element of the Basic Software Module Description for the generated RTE. ⌋*(SRS_Rte_00233)*

[SWS_Rte_08404] ⌈ The RTE BswModuleDescription shall be provided in ARPackage AUTOSAR_Rte according to AUTOSAR Generic Structure Template [10] (chapter "Identifying M1 elements in packages"). ⌋*(SRS_Rte_00233)*

[SWS_Rte_05177] ⌈ The RTE Generator in Generation Phase shall provide the BswModuleEntry and a reference to it from the BswModuleDescription in the role providedEntry for each *Standardized Interface* provided by the RTE (see Layered Software Architecture [15] page *tz76a* and page *94ju5*). The provided *Standardized Interface*s are the Rte Lifecycle API (section 5.8) and the SchM Lifecycle API (section 6.7). ⌋*(SRS_Rte_00233)*

[SWS_Rte_05179] ⌈ The RTE Generator in Generation Phase shall provide the BswModuleDependency in the BswModuleDescription with the role bswModuleDependency for each callback API provided by the RTE and called by the respective Basic Software Module. The reference from the BswModuleDependency to the BswModuleEntry shall be in the role expectedCallback. The calling Basic Software Module is specified in the attribute targetModuleId of the BswModuleDependency. ⌋*(SRS_Rte_00233)*

For all the APIs the RTE code is invoking in other Basic Software Modules the dependencies are described via requirement [SWS_Rte_05180].

[SWS_Rte_05180] ⌈ The RTE Generator in Generation Phase shall provide the BswModuleDependency in the BswModuleDescription with the role bswModuleDependency for each API called by the RTE in another Basic Software Module. The reference from the BswModuleDependency to the BswModuleEntry shall be in the role requiredEntry. The called Basic Software Module is specified in the attribute targetModuleId of the BswModuleDependency. ⌋*(SRS_Rte_00233)*

[SWS_Rte_07085] ⌈ If the Basic Software Module Description for the generated RTE depends from elements in Basic Software Module Descriptions of other Basic Software Modules the RTE Generator shall use the full qualified path name to this elements according the rules in "Identifying M1 elements in packages" of the document AUTOSAR Generic Structure Template [10]. ⌋*(SRS_Rte_00233)*

For instance the description of the the hook function

```
void Rte_Dlt_Task_Activate(TaskType task)
```

for the Dlt needs the ImplementationDataType "TaskType" from the OS in order to describe the data type of the SwServiceArg "task" in the description of the related BswModuleEntry.

In this case the full qualified path name to the ImplementationDataType "TaskType" shall be

```
1 AUTOSAR_OS/ImplementationDataTypes/TaskType
```

The full example about the description is given below:

```
<AR-PACKAGE>
  <SHORT-NAME>AUTOSAR_RTE</SHORT-NAME>
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>BswModuleEntrys</SHORT-NAME>
      <ELEMENTS>
        <BSW-MODULE-ENTRY>
          <SHORT-NAME>Rte_Dlt_Task_Activate</SHORT-NAME>
          <ARGUMENTS>
            <SW-SERVICE-ARG>
              <SHORT-NAME>task</SHORT-NAME>
              <CATEGORY>TYPE_REFERENCE</CATEGORY>
              <SW-DATA-DEF-PROPS>
                <SW-DATA-DEF-PROPS-VARIANTS>
                  <SW-DATA-DEF-PROPS-CONDITIONAL>
                    <IMPLEMENTATION-DATA-TYPE-REF DEST="IMPLEMENTATION-
                       DATA-TYPE">AUTOSAR_OS/ImplementationDataTypes/
                       TaskType</IMPLEMENTATION-DATA-TYPE-REF>
                  </SW-DATA-DEF-PROPS-CONDITIONAL>
                </SW-DATA-DEF-PROPS-VARIANTS>
              </SW-DATA-DEF-PROPS>
            </SW-SERVICE-ARG>
          </ARGUMENTS>
        </BSW-MODULE-ENTRY>
      </ELEMENTS>
    </AR-PACKAGE>
```

3.4.3.2 Bsw Internal Behavior

[SWS_Rte_05166] ⌈ The RTE Generator in Generation Phase shall provide the `BswInternalBehavior` element in the `BswModuleDescription` of the Basic Software Module Description for the generated RTE. ⌋ *(SRS_Rte_00233)*

[SWS_Rte_05181] ⌈ The RTE Generator in Generation Phase shall provide the `BswCalledEntity` element in the `BswInternalBehavior` for each C-function implementing the lifecycle APIs (section 5.8) and the SchM Lifecycle API (section 6.7). The `BswCalledEntity` shall have a reference to the respective `BswModuleEntry` ([SWS_Rte_05177]) in the role `implementedEntry`. ⌋ *(SRS_Rte_00233)*

[SWS_Rte_05182] ⌈ The RTE Generator in Generation Phase shall provide the `VariableDataPrototype` element in the `BswInternalBehavior` in the role `staticMemory` for each variable memory object the RTE allocates. ⌋ *(SRS_Rte_00233)*

[SWS_Rte_05183] ⌈ The RTE Generator in Generation Phase shall provide the `ParameterDataPrototype` element in the `BswInternalBehavior` in the role `constantMemory` for each constant memory object the RTE allocates. ⌋ *(SRS_Rte_00233)*

3.4.3.3 Bsw Implementation

[SWS_Rte_05167] ⌈ The RTE Generator in Generation Phase shall provide the BswImplementation element and a reference to the BswInternalBehavior of the Basic Software Module Description in the role behavior. ⌋*(SRS_Rte_00233)*

[SWS_Rte_05187] ⌈ The RTE Generator in Generation Phase shall provide the programmingLanguage element in the BswImplementation element according to the actual RTE implementation. ⌋*(SRS_Rte_00233)*

[SWS_Rte_05186] ⌈ The RTE Generator in Generation Phase shall provide the swVersion element in the BswImplementation element according to the input information from the RTE Ecu configuration ([SWS_Rte_05184], [SWS_Rte_05185]). ⌋*(SRS_Rte_00233)*

[SWS_Rte_05190] ⌈ The RTE Generator in Generation Phase shall provide the arReleaseVersion element in the BswImplementation element according to AUTOSAR release version the RTE Generator is based on. ⌋*(SRS_Rte_00233)*

[SWS_Rte_05188] ⌈ The RTE Generator in Generation Phase shall provide the usedCodeGenerator element in the BswImplementation element according to the actual RTE implementation. ⌋*(SRS_Rte_00233)*

[SWS_Rte_05189] ⌈ The RTE Generator in Generation Phase shall provide the vendorId element in the BswImplementation element according to the input information from the RTE Ecu configuration (RteCodeVendorId). ⌋*(SRS_Rte_00233)*

The RteCodeVendorId specifies the vendor id of the actual user of the RTE Generator, not the id of the RTE Vendor itself.

[SWS_Rte_05191] ⌈ If the generated RTE code is hardware specific (due to vendor specific optimizations of the RTE Generator) then the reference to the applicable HwElements from the ECU Resource Description [16] shall be provided in the BswImplementation element with the role hwElement. ⌋*(SRS_Rte_00233)*

[SWS_Rte_05192] ⌈ The RTE Generator in Generation Phase shall provide the DependencyOnArtifact element in the BswImplementation with the role generatedArtifact for all c- and header-files which are required to compile the Rte code. This does not include other Basic Software modules or Application Software. ⌋*(SRS_Rte_00233)*

Note: The use case is the support of the build-environment (automatic or manual).

Attributes shall be used in this context as follow:

- category shall be used as defined in Generic Structure Template [10] (e.g. SWSRC, SWOBJ, SWHDR)
- domain is optional and can be chosen freely
- revisionLabel shall contain the revision label out of RTE Configuration

- shortLabel is the name of artifact

Details on the description of DependencyOnArtifact can be found in the Generic Structure Template [10].

Additional elements of the *Basic Software Module Description* which shall be exported are specified in later requirements e.g. in section 4.2.8.4 and sectionsection 5.1.2.4.

3.5 PreBuild Data Set Generation Phase

During the *PreBuild Data Set Generation Phase* are the *Condition Value Macros* (see 5.3.8.2.2) generated which are required to resolve the implemented PreBuild Variability of the software components, generated RTE and *Basic Software Scheduler*.

The particular values are defined via the EcucVariationResolver configuration selecting PredefinedVariants. These PredefinedVariant elements containing definition of SwSystemconstValues for SwSystemconsts which shall be applied when resolving the variability during ECU Configuration.

The values of the *Condition Value Macros* are the results of evaluated ConditionByFormulas of the related VariationPoints. These ConditionByFormulas referencing SwSystemconsts in the formula expressions. It is supported that the assigned SwSystemconstValue might contain again a formula expressions referencing SwSystemconsts. Therefore the input might be a tree of formula expressions and SwSystemconstValues but the leaf SwSystemconstValues are required to be values which are not dependent from other SwSystemconsts to ensure that the evaluation of the tree results in a unique number.

[SWS_Rte_06610] ⌈ The RTE generator shall validate the resolved pre-build variants and check the integrity with regards to the meta model. Any meta model violation shall result in the rejection of the input configuration. ⌋*(SRS_Rte_00018)*

The output of this phase is the *RTE Configuration Header File* 5.3.8.This file is required to compile a particular variant of ECU software including software component code and RTE code using PreCompileTime Variability. The *Condition Value Macros* are used for the implementation of PreCompileTime Variability with preprocessor statements and therefore are needed to run the C preprocessor resolving the implemented variability.

3.6 PostBuild Data Set Generation Phase

In the *PostBuild Data Set Generation Phase* the PredefinedVariant values are generated which are required to resolve the implemented PostBuild Variability of the software components and generated RTE.

The output of this phase are the *RTE Post Build Variant Sets* 5.3.10. This file is required to link the ECU software and to select a particular PostBuild variant in the generated RTE code during start up when the *Basic Software Scheduler* is initialized.

[SWS_Rte_06611] ⌈ If the DET is enabled then the RTE shall generate validation code which at runtime (i.e. during initialization) validates the resolved post-build variants and check the integrity with regards to the active variants. If a violation is detected the RTE shall report a development error to the DET. To execute this validation RTE initialization will get a pointer to the `RtePostBuildVariantConfiguration` instance to allow it to validate the selected variant. ⌋*(SRS_Rte_00191)*

[SWS_Rte_06612] ⌈ The RTE generator shall create an RTE Post Build Data Set configuration (i.e. Rte_PBCfg.c) representing the collection of `PredefinedVariant` definitions (typically for each subsystem and/or system configuration) providing and defining the post build variants of the RTE. ⌋*(SRS_Rte_00191)*

Note that the Rte_PBCfg.h is generated during the Rte Generation phase. An Rte_PBCfg.c may also have to be generated at that time to reserve memory (with default values).

Additional details about these configuration files are described in section 5.3.10.

An RTE variant can consist of a collection of `PredefinedVariant`s. Each `PredefinedVariant` contains a collection of `PostBuildVariantCriterionValue`s which assigns a value to a specific `PostBuildVariantCriterion` which in turn is used to resolve the variability at runtime by evaluating a `PostBuildVariantCondition`. Different `PredefinedVariant`s could assign different values to the same `PostBuildVariantCriterion` and as such create conflicts for a specific `PostBuildVariantCriterionValueSet`. It is allowed to have different assignments if these assignment assign the same value.

[SWS_Rte_06613] ⌈ The RTE Generator shall reject configurations where different `PredefinedVariant`s assign different values to the same `PostBuildVariantCriterion` for the same `RtePostBuildVariantConfiguration`. ⌋*(SRS_Rte_00018)*

3.7 RTE Configuration interaction with other BSW Modules

The generated RTE interacts heavily with other AUTOSAR Basic Software Modules like Com and Os. The configuration values for the different BSW Modules are stored in individual structures of ECU Configuration it is however essential that the common used values are synchronized between the different BSW Module's configurations. AUTOSAR does not provide a standardized way how the individual configurations can be synchronized, it is assumed that during the generation of the BSW Modules the input information provided to the individual BSW Module is in sync with the input information provided to other (dependent) BSW Modules.

The AUTOSAR BSW Module code-generation methodology is heavily relying on the logical distinction between Configuration editors and configuration generators. These tools do not necessarily have to be implemented as two separate tools, it just shall be possible to distinguish the different roles the tools take during a certain step in the methodology.

For the RTE it is assumed that tool support for the resolution of interactions between the Rte and other BSW Modules is needed to allow an efficient configuration of the Rte. It is however not specified how and in which tools this support shall be implemented.

The RTE Generator in Generation Phase needs information about other BSW Module's configurations based on the configuration input of the Rte itself (there are references in the configuration of the Rte which point to configuration values of other BSW Modules). If during RTE Generation Phase the provided input information is inconsistent wrt. the Rte input the Rte Generator will have to consider the input as invalid configuration.

[SWS_Rte_05149] ⌈ The RTE Generator in Generation Phase shall consider errors in the Rte configuration input information as invalid configuration. ⌋*(SRS_Rte_00018)*

Due to implementation freedom of the RTE Generator it is possible to correct / update provided input configurations of other BSW Modules based on the RTE configuration requirements. But to allow a stable build process it is also possible to disallow such an update behavior.

[SWS_Rte_05150] ⌈ If the external configuration switch `strictConfigurationCheck` is set to *true* the Rte Generator shall not create or modify any configuration input. ⌋*(SRS_Rte_00065)*

If the external configuration switch `strictConfigurationCheck` (see [SWS_Rte_05148]) is set to *false* the Rte Generator may update the input configuration information of the Rte and other BSW Modules.

Example: If the Rte configuration is referencing an `OsTask` which is not configured in the provided Os configuration, the RTE Generator would behave like:

- In case [SWS_Rte_05150] applies: Only show an error message.
- Otherwise: Possible behavior: Show a warning message and modify the Os configuration to contain the `OsTask` which is referred to by the Rte configuration (Of course the Os configuration of this new `OsTask` needs to be refined afterwards).

4 RTE Functional Specification

4.1 Architectural concepts

4.1.1 Scope

In this section the concept of an AUTOSAR software-component and its usage within the RTE is introduced.

The AUTOSAR Software Component Template [2] defines the kinds of software-components within the AUTOSAR context. These are shown in Figure 4.1. The abstract SwComponentType can not be instantiated, so there can only be either a CompositionSwComponentType, a ParameterSwComponentType, or a specialized class ApplicationSwComponentType, ServiceProxySwComponentType, SensorActuatorSwComponentType, NvBlockSwComponentType, ServiceSwComponentType, ComplexDeviceDriverSwComponentType, or EcuAbstractionSwComponentType of the abstract class AtomicSwComponentType.

In the following document the term AtomicSwComponentType is used as collective term for all the mentioned non-abstract derived meta-classes.

The SwComponentType is defining the type of an AUTOSAR software-component which is independent of any usage and can be potentially re-used several times in different scenarios. In a composition the types are occurring in specific roles which are called SwComponentPrototypes. The prototype is the utilization of a type within a certain scenario. In AUTOSAR any SwComponentType can be used as a type for a prototype.

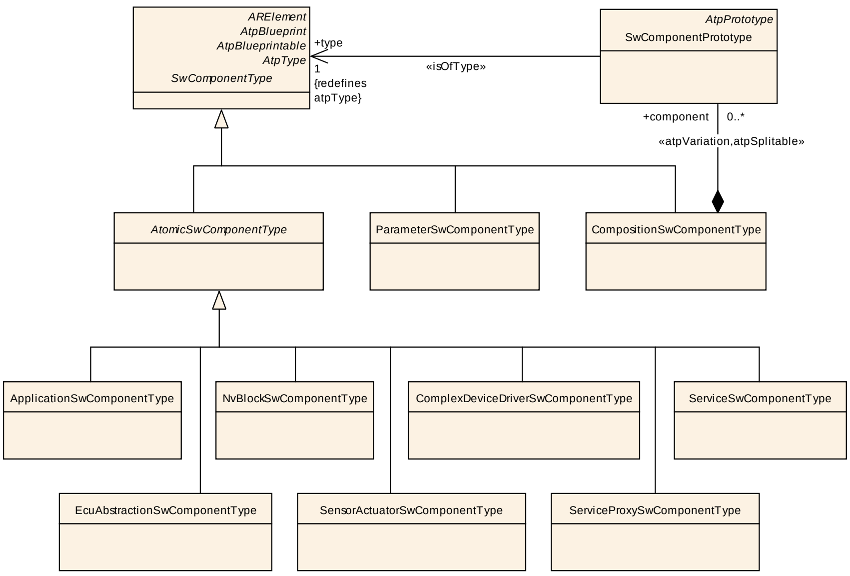


Figure 4.1: AUTOSAR software-component classification

The AUTOSAR software-components shown in Figure 4.1 are located above and below the RTE in the architectural Figure 4.2.

Below the RTE there are also software entities that have an AUTOSAR Interface. These are the AUTOSAR services, the ECU Abstraction and the Complex Device Drivers. For these software not only the AUTOSAR Interface will be described but also information about their internal structure will be available in the Basic Software Module Description.

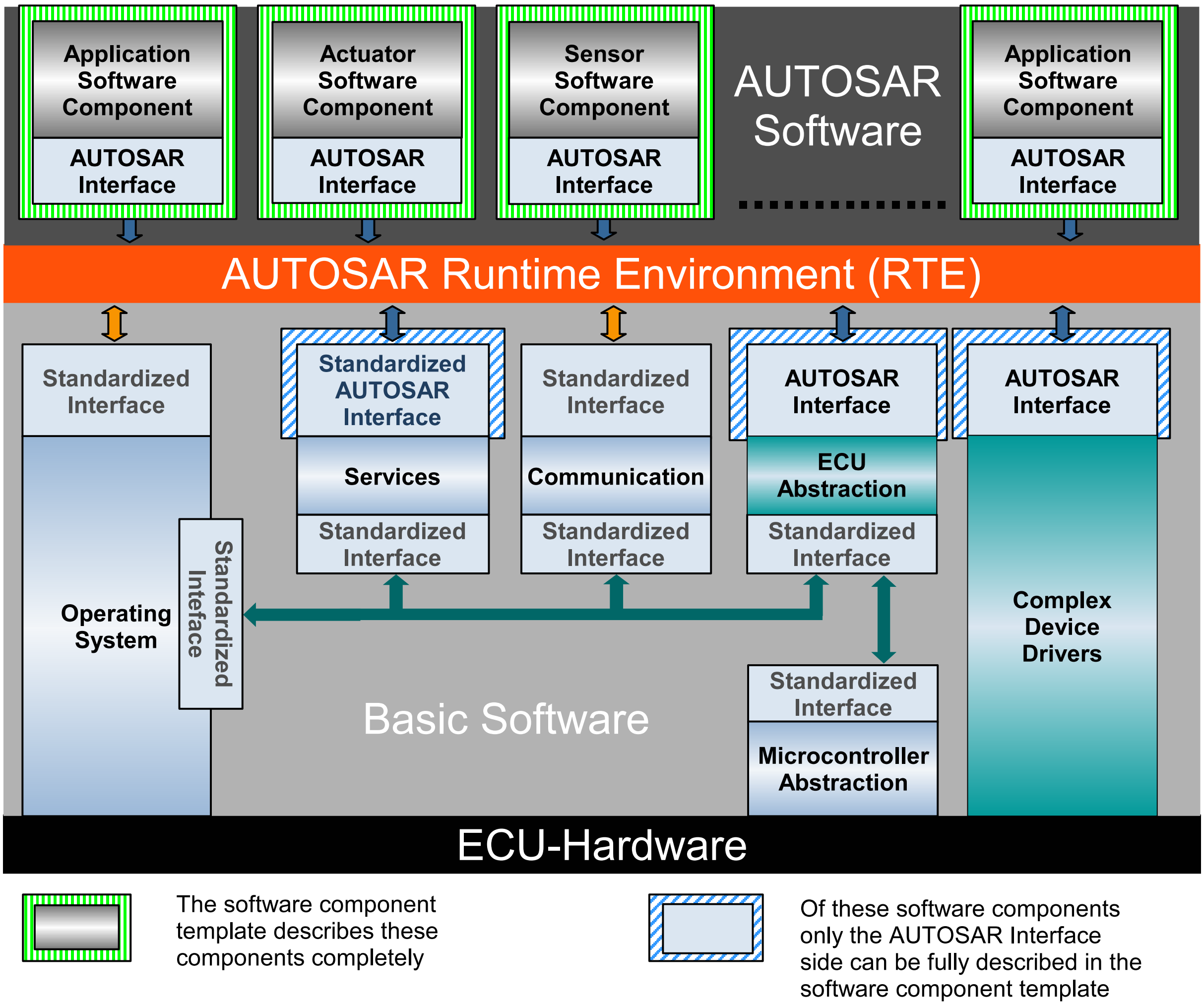


Figure 4.2: AUTOSAR ECU architecture diagram

In the next sections the different AUTOSAR software-components kinds will be described in detail with respect to their influence on the RTE.

4.1.2 RTE and Data Types

The AUTOSAR Meta Model defines `ApplicationDataType`s and `ImplementationDataType`s. A `AutosarDataPrototype` can be typed by an `ApplicationDataType` or an `ImplementationDataType`. But the RTE Generator only implements `ImplementationDataType`s as C data types and uses these C data types to type the RTE API which is related to `DataPrototype`s. Therefore it is required in the input configuration that every `ApplicationDataType` used for the typing of a

DataPrototype which is relevant for RTE generation is mapped to an ImplementationDataType with a DataTypeMap. Which DataTypeMap is applicable for an particular software component respectively Basic Software Module is defined by the DataTypeMappingSets referenced by the InternalBehavior.

[SWS_Rte_07028] ⌈ The RTE Generator shall reject input configurations containing a AutosarDataPrototype which influences the generated RTE and which is typed by an ApplicationDataType not mapped to an ImplementationDataType. ⌋*(SRS_Rte_00018)*

Nevertheless a subset of the attributes given by the ApplicationDataTypes are relevant for the RTE generator for instance

- to create the McSupportData (see section 4.2.8.4) information
- to calculate the conversion formula in case of *Data Conversion* (see section 4.3.5 and 4.3.5.3)
- to calculate numerical representation of values required for the RTE code but defined in the physical representation (e.g. initialValues and invalidValues).

[SWS_Rte_01374] ⌈ When a value is required for the RTE code and is provided as an ApplicationValueSpecification, if there is an applicable ConstantSpecificationMapping then the RTE Generator shall use the ValueSpecification referenced by its implConstant as the definitive numerical representation of the value regardless of any compuMethod. ⌋*(SRS_Rte_00180, SRS_Rte_00182)*

[SWS_Rte_07038] ⌈ When a value is required for the RTE code and is provided as an ApplicationValueSpecification, if there is no applicable ConstantSpecificationMapping then the RTE Generator shall calculate the numerical representation according to the conversion defined by an compuMethod. This shall be supported for categorys VALUE, VAL_BLK, STRUCTURE, ARRAY, and BOOLEAN. If there is no CompuMethod provided the conversion is treated like an CompuMethod of category IDENTICAL. ⌋*(SRS_Rte_00180, SRS_Rte_00182)*

In [SWS_Rte_01374] and [SWS_Rte_07038], an "applicable ConstantSpecificationMapping" is one that is aggregated by the relevant SwComponentType and which references the ApplicationValueSpecification in its applConstant.

4.1.3 RTE and AUTOSAR Software-Components

The description of an AUTOSAR software-component is divided into the sections

- hierarchical structure
- ports and interfaces
- internal behavior

- implementation

which will be addressed separately in the following sections.

[SWS_Rte_07196] ⌈ The RTE Generator shall respect the precedence of data properties defined via SwDataDefProps as defined in the *Software Component Template* [2]. ⌋

Requirement [SWS_Rte_07196] means that:

1. SwDataDefProps defined on ApplicationDataType which may be overwritten by
2. SwDataDefProps defined on ImplementationDataType which may be overwritten by
3. SwDataDefProps defined on AutosarDataPrototype which may be overwritten by
4. SwDataDefProps defined on InstantiationDataDefProps which may be overwritten by
5. SwDataDefProps defined on AccessPoint respectively Argument which may be overwritten by
6. SwDataDefProps defined on FlatInstanceDescriptor which may be overwritten by
7. SwDataDefProps defined on McDataInstance

The SwDataDefProps defined on McDataInstance are not relevant for the RTE generation but rather the documentation of the generated RTE.

Especially the attributes swAddrMethod, swCalibrationAccess, swImplPolicy and dataConstr do have an impact on the generated RTE. In the following document only the attribute names are mentioned with the semantic that this refers to the most significant one.

4.1.3.1 Hierarchical Structure of Software-Components

In AUTOSAR the structure of an E/E-system is described using the AUTOSAR Software Component Template and especially the mechanism of compositions. Such a Top Level Composition assembles subsystems and connects their ports.

Of course such a composition utilizes a lot of hierarchical levels where compositions instantiate other composition types and so on. But at some low hierarchical level each composition only consists of AtomicSwComponentType instances. And those instances of AtomicSwComponentTypes are what the RTE is going to be working with.

4.1.3.2 Ports, Interfaces and Connections

Each AUTOSAR software-component (SwComponentType) can have ports (PortPrototype). An AUTOSAR software-component has provide ports (PPortPrototype) and/or has require ports (RPortPrototype) to communicate with other AUTOSAR software-components. The requiredInterface or providedInterface (PortInterface) determines if the port is a sender/receiver or a client/server port. The attribute isService is used with AUTOSAR Services (see section 4.1.5).

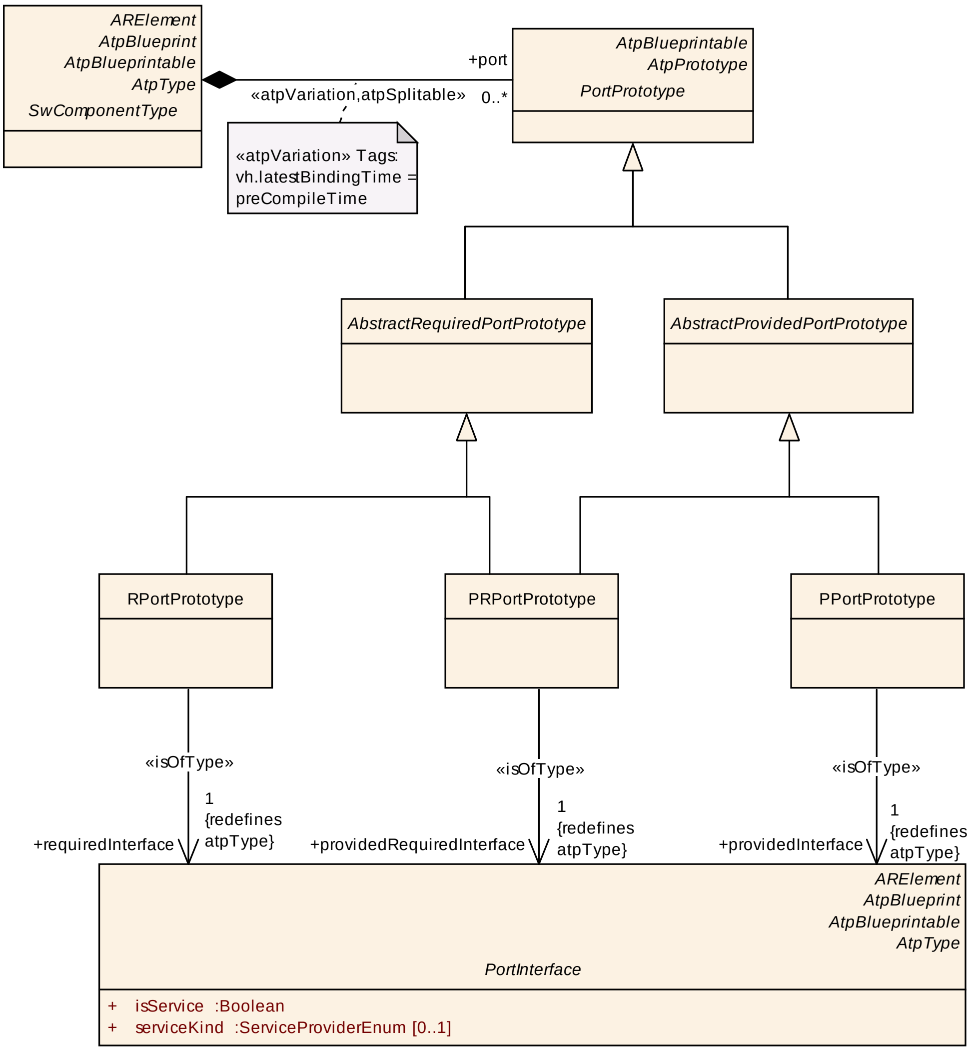


Figure 4.3: Software-Components and Ports

When compositions are built of instances the ports can be connected either within the composition or made accessible to the outside of the composition. For the connections inside a composition the AssemblySwConnector is used, while the DelegationSwConnector is used to connect ports from the inside of a composition to the outside. Ports not connected will be handled according to the requirement [SRS_Rte_00139].

The next step is to map the SW-C instances on ECUs and to establish the communication relationships. From this step the actual communication is derived, so it is now fixed if a connection between two instance's ports is going to be over a communication bus or locally within one ECU.

[SWS_Rte_02200] ⌈ The RTE shall implement the communication paths specified by the ECU Configuration description. ⌋*(SRS_Rte_00027)*

[SWS_Rte_02201] ⌈ The RTE shall implement the semantic of the communication attributes given by the AUTOSAR software-component description. The semantic of the given communication mechanism shall not change regardless of whether the communication partner is located on the same partition, on another partition of the same ECU or on a remote ECU, or whether the communication is done by the RTE itself or by the RTE calling COM or IOC. ⌋*(SRS_Rte_00027)*

E.g., according to [SWS_Rte_02200] and [SWS_Rte_02201] the RTE is not permitted to change the semantic of an asynchronous client to synchronous because both client and server are mapped to the very same ECU.

4.1.3.3 Internal Behavior

Only for `AtomicSwComponentType`s the internal structure is exposed in the `SwcInternalBehavior` description. Here the definition of the `RunnableEntity`s and used `RTEEvent`s is done (see Figure 4.4).

The AUTOSAR MetaModel enforces that there is at most one `SwcInternalBehavior` per `AtomicSwComponentType`

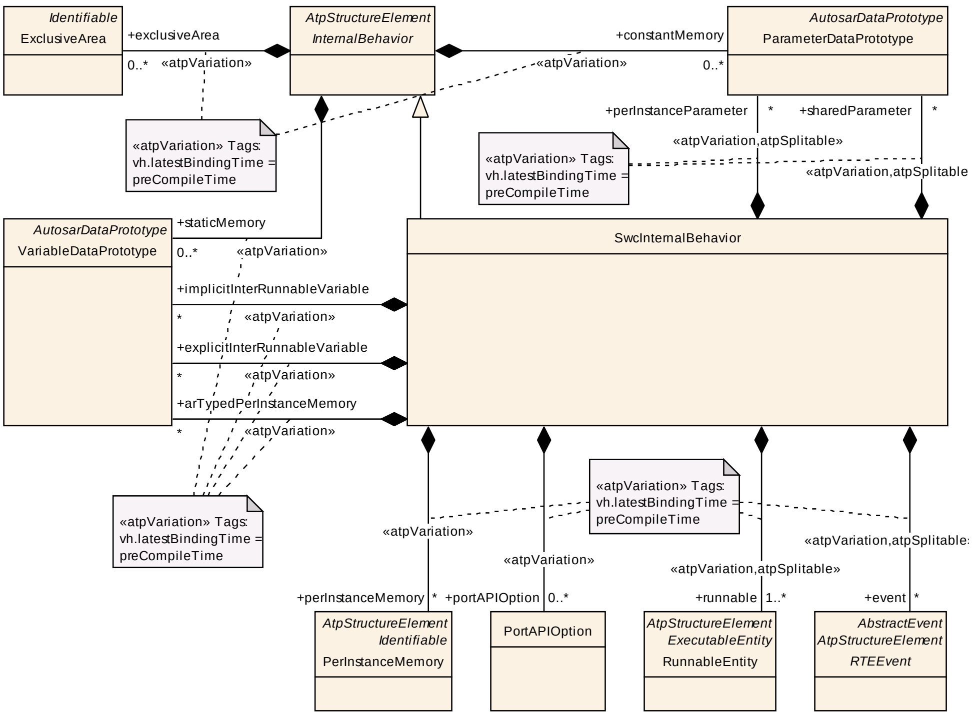


Figure 4.4: Software-component internal behavior

RunnableEntitys (also abbreviated simply as Runnable) are the smallest code fragments that are provided by AUTOSAR software-components and those basic software modules that implement *AUTOSAR Interface*s. They are represented by the meta-class RunnableEntity, see Figure 4.5.

In general, software components are composed of multiple RunnableEntitys in order to accomplish servers, receivers, feedback, etc.

[SWS_Rte_02202] ⌈The RTE shall support multiple RunnableEntitys in AUTOSAR software-components. ⌋*(SRS_Rte_00031)*

RunnableEntitys are executed in the context of an OS task, their execution is triggered by RTEEvents. Section 4.2.2.3 gives a more detailed description of the concept of RunnableEntitys, Section 4.2.2.6 discusses the problem of mapping RunnableEntitys to OS tasks. RTEEvents and the activation of RunnableEntitys by RTEEvents is treated in Section 4.2.2.4.

[SWS_Rte_02203] ⌈The RTE shall trigger the execution of RunnableEntitys in accordance with the connected RTEEvent. ⌋*(SRS_Rte_00072)*

[SWS_Rte_02204] ⌈The RTE Generator shall reject configurations where an RTEEvent instance which can start a RunnableEntity is not mapped to an OS task. The only exceptions are RunnableEntitys that are invoked by a direct function call. ⌋*(SRS_Rte_00049, SRS_Rte_00018)*

[SWS_Rte_07347] ⌈The RTE Generator shall reject configurations where RunnableEntitys of a SW-C are mapped to tasks of different partitions. ⌋*(SRS_Rte_00036, SRS_Rte_00018)*

[SWS_Rte_02207] ⌈The RTE shall respect the configured execution order of RunnableEntitys within one OS task. ⌋*(SRS_Rte_00070)*

[SWS_Rte_08768] ⌈The RTE generator shall reject configuration where the scope of a VariableAccess is violated by the system and/or ECU configuration. ⌋*(SRS_Rte_00018)*

[constr_9081] Mapping to partition vs the value of VariableAccess.scope ⌈For every connection between SwComponentPrototypes mapped to different partitions the value of VariableAccess.scope shall not be set to VariableAccessScopeEnum.communicationIntraPartition. ⌋

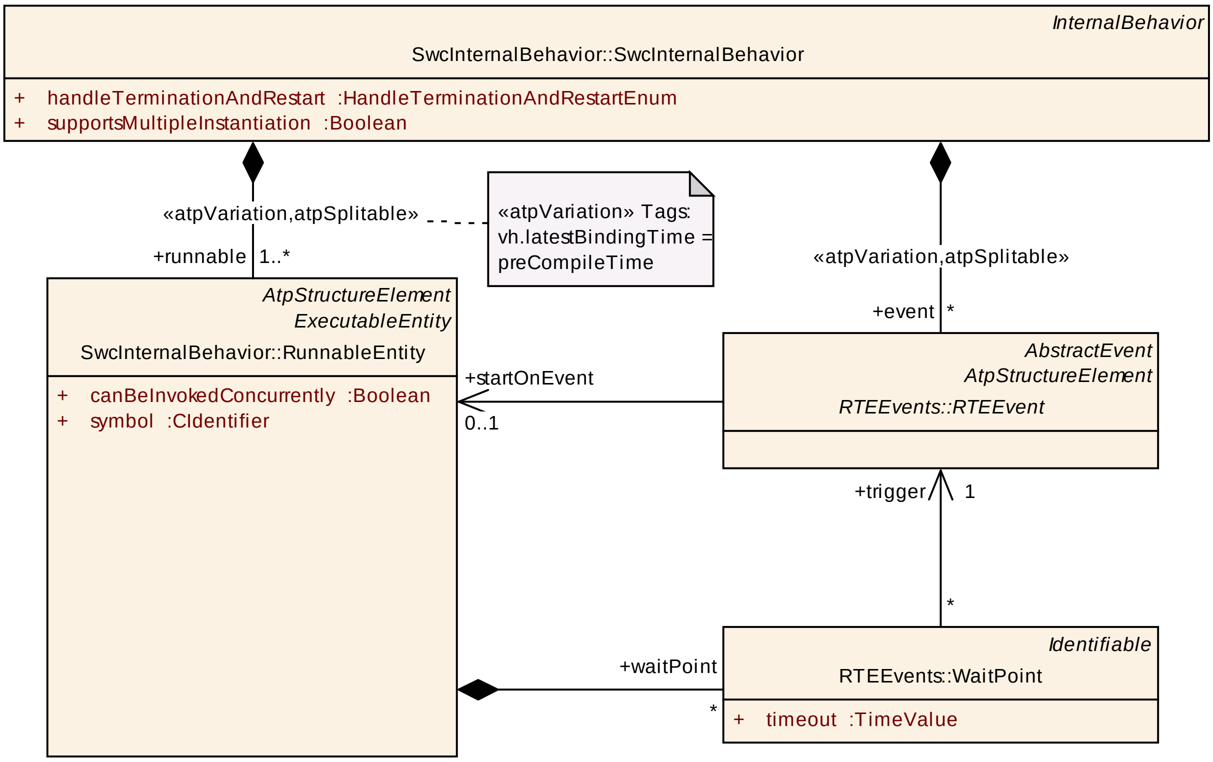


Figure 4.5: Software-component runnable entity, wait points and RTE Events

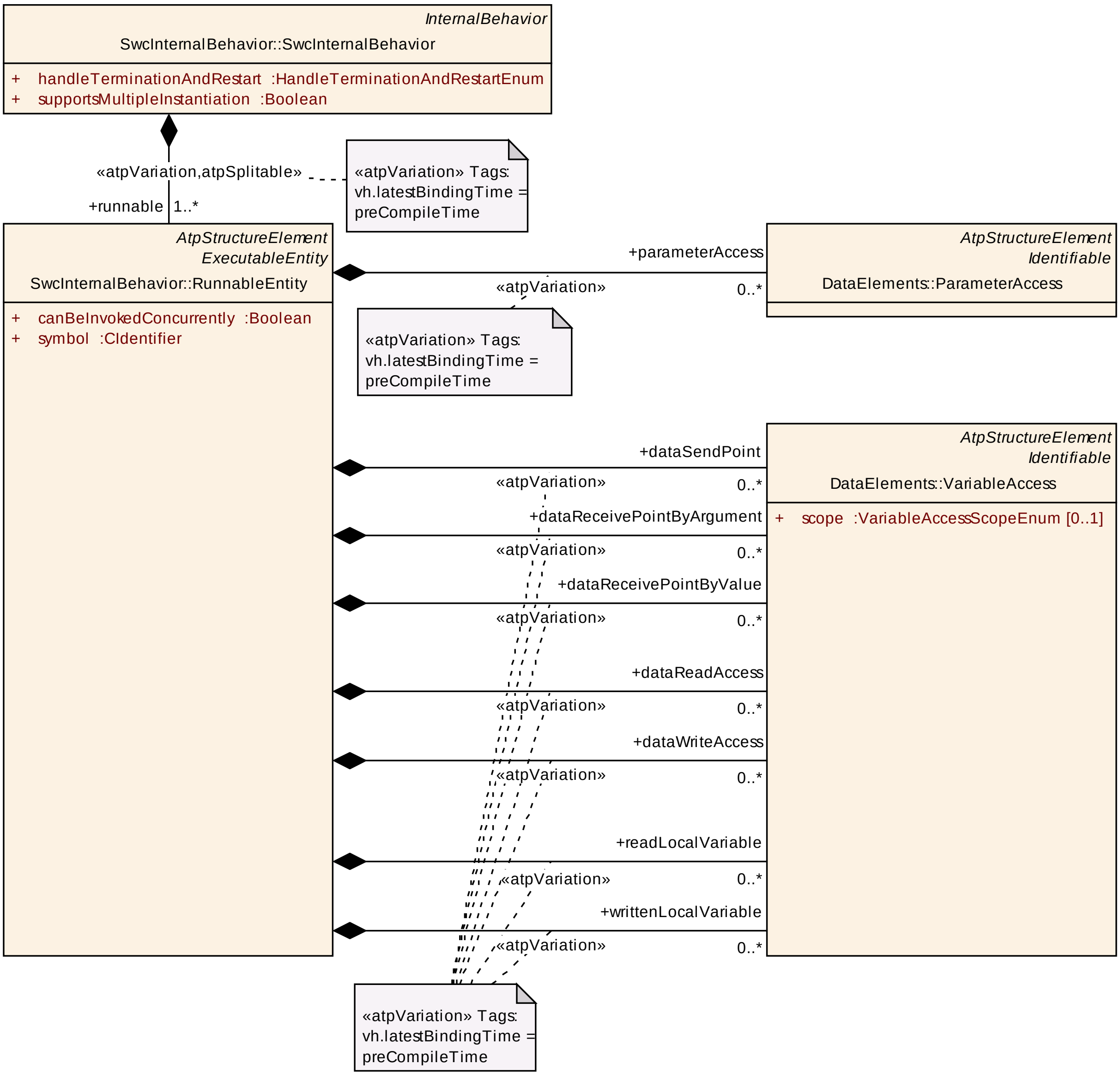


Figure 4.6: Software-component runnable entity and data accesses

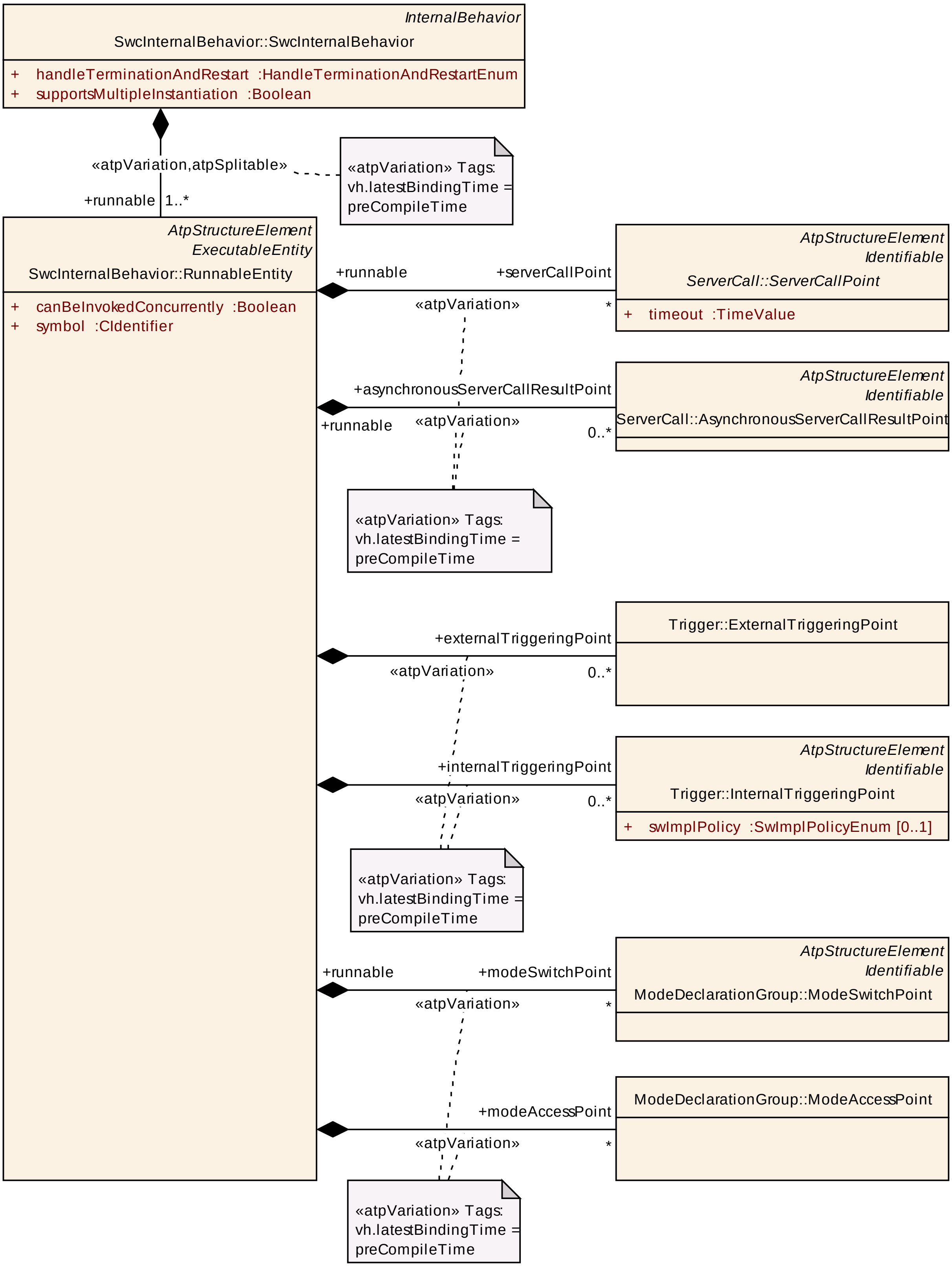


Figure 4.7: Software-component runnable entity and server invocation, trigger, and mode switches

With the information from SwcInternalBehavior a part of the setup of the AUTOSAR software-component within the RTE and the OS can already be configured. Furthermore, the information (description) of the structure (ports, interfaces) and the internal behavior of an AUTOSAR software component are sufficient for the *RTE Contract Phase*.

However, some detailed information is still missing and this is part of the Implementation description.

4.1.3.4 Implementation

In the Implementation description an actual implementation of an AUTOSAR software-component is described including the memory consumption (see Figure 4.8).

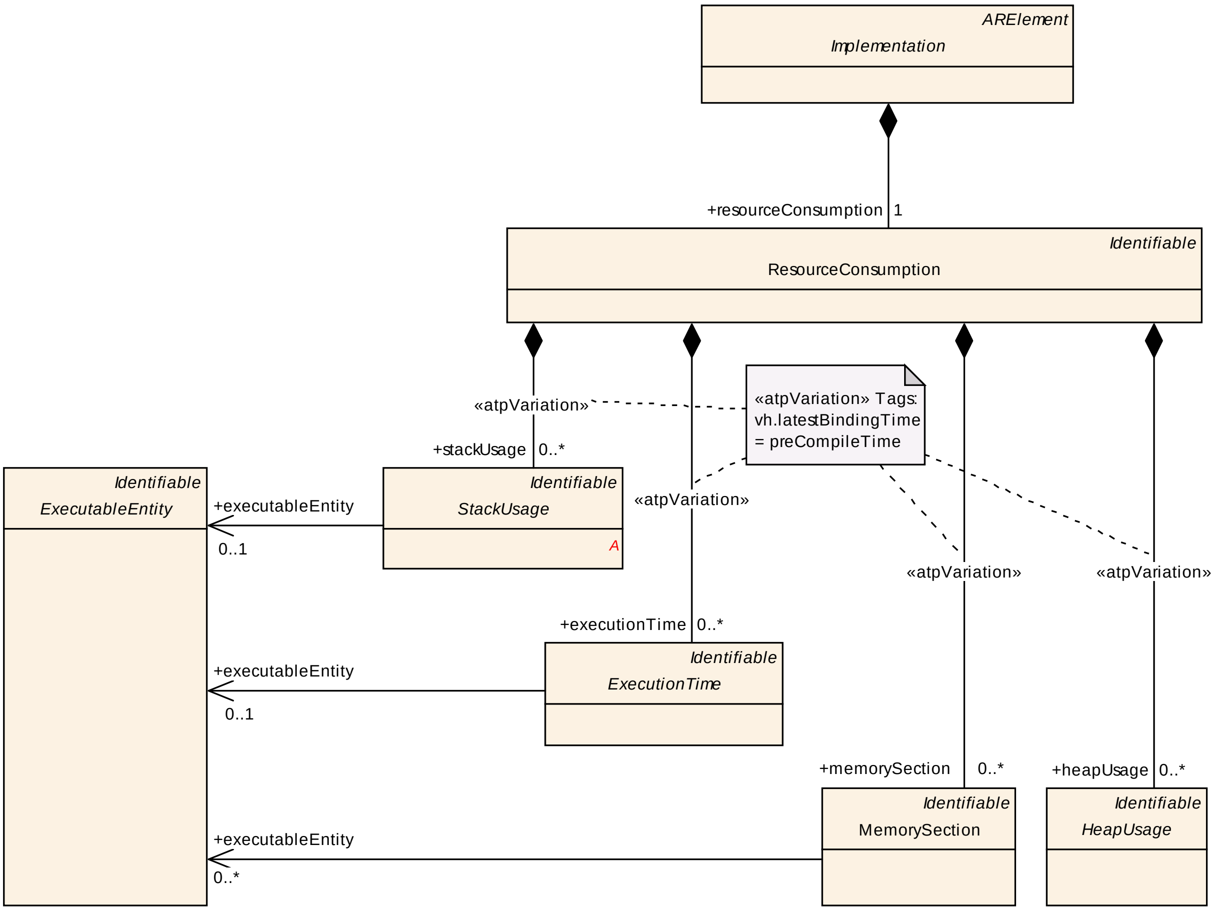


Figure 4.8: Software-component resource consumption

Note that the information from the Implementation part are only required for the *RTE Generation Phase*, if at all.

4.1.4 Instantiation

4.1.4.1 Scope and background

Generally spoken, the term *instantiation* refers to the process of deriving specific instances from a model or template. But, this process can be accomplished on different levels of abstraction. Therefore, the instance of the one level can be the model for the next.

With respect to AUTOSAR four modeling levels are distinguished. They are referred to as the levels $M3$ to $M0$.

The level $M3$ describes the concepts used to derive an AUTOSAR meta model of level $M2$. This meta model at level $M2$ defines a language in order to be able to describe specific attributes of a model at level $M1$, e.g., to be able to describe an specific type of an AUTOSAR software component. E.g., one part of the AUTOSAR meta model is called *Software Component Template* or *SW-C-T* for short and specified in [2]. It is discussed more detailed in section 4.1.3.

At level $M1$ engineers will use the defined language in order to design components or interfaces or compositions, say to describe an specific *type* of a *LightManager*. Hereby, e.g., the descriptions of the (atomic) software components will also contain an internal behavior as well as an implementation part as mentioned in section 4.1.3.

Those descriptions are input for the RTE Generator in the so-called 'Contract Phase' (see section 3.1.1). Out of this information specific APIs (in a programming language) to access ports and interfaces will be generated.

Software components generally consist of a set of Runnable Entities. They can now specifically be described in a programming language which can be refered to as "implementation". As one can see in section 4.1.3 this "implementation" then corresponds exactly to one implementation description as well as to one internal behavior description.

$M0$ refers to a specific running instance on a specific car.

Objects derived from those specified component types can only be executed in a specific run time environment (on a specific target). The objects embody the real and running implementation and shall therefore be referred to as software component instances (on modeling level $M0$). E.g., there could be two component instances derived from the same component type *LightManager* on a specific *light controller* ECU each responsible for different lights. Making instances means that it should be possible to distinguish them even though the objects are descended from the same model.

With respect to this more narrative description the *RTE* as the *run time environment* shall enable the process of instantiation. Thereby the term *instantiation* throughout the document shall refer to the process of deriving and providing explicit particular descriptions of all occuring instances of all types. Therefore, this section will address the problems which can arise out of the instantiation process and will specify the needs for AUTOSAR components and the AUTOSAR RTE respectively.

4.1.4.2 Concepts of instantiation

Regardless of the fact that the (aforementioned) instantiation of AUTOSAR software components can be generally achieved on a per-system basis, the RTE Generator restricts its view to a per-ECU customization (see [SWS_Rte_05000]).

Generally, there are two different kinds of instantiations possible:

- single instantiation – which refers to the case where only *one* object or AUTOSAR software component instance will be derived out of the AUTOSAR software component description
- multiple instantiation – which refers to the case where *multiple* objects or AUTOSAR software component instances will be derived out of the AUTOSAR software component description

[SWS_Rte_02001] ⌈ The RTE Generator shall be able to instantiate one or more AUTOSAR software component instances out of a single AUTOSAR software component description. ⌋*(SRS_Rte_00011)*

[SWS_Rte_02008] ⌈ The RTE Generator shall evaluate the attribute *supportsMultipleInstantiation* of the *SwcInternalBehavior* of an AUTOSAR software component description. ⌋*(SRS_Rte_00011)*

[SWS_Rte_02009] ⌈ The RTE Generator shall reject configurations where multiple instantiation is required, but the value of the attribute *supportsMultipleInstantiation* of the *SwcInternalBehavior* of an AUTOSAR software component description is set to *FALSE*. ⌋*(SRS_Rte_00011, SRS_Rte_00018)*

4.1.4.3 Single instantiation

Single instantiation refers to the easiest case of instantiation.

To be instantiated merely means that the code and the corresponding data of a particular `RunnableEntity` are embedded in a runtime context. In general, this is achieved by the context of an OS task (see example 4.1).

Example 4.1

Runnable entity `R1` called out of a task context:

```
TASK(Task1){
    ...
    R1();
    ...
}
```

Since the single instance of the software component is unambigous per se no additional concepts have to be added.

4.1.4.4 Multiple instantiation

[SWS_Rte_02002] ⌈ Multiple objects instantiated from a single AUTOSAR software component (type) shall be identifiable without ambiguity. ⌋*(SRS_Rte_00011)*

There are two *principle* ways to achieve this goal –

- by code duplication (of runnable entities)
- by code sharing (of reentrant runnable entities)

For now it was decided to solely concentrate on code sharing and not to support code duplication.

[SWS_Rte_03015] ⌈ The RTE only supports multiple objects instantiated from a single AUTOSAR software component by code sharing, the RTE doesn't support code duplication. ⌋*(SRS_Rte_00011, SRS_Rte_00012)*

Multiple instances can share the same code, if the code is reentrant. For a multi core controller, the possibility to share code between the cores depends on the hardware.

Example 4.2 is similar to the example 4.1, but for a software-component that support multiple instantiations, and where two instances have their `R1` `RunnableEntity` mapped to the same task.

Example 4.2

Runnable entity `R1` called for two instances out of the same task context:

```
TASK(Task1){
    ...
    R1(instance1);
    R1(instance2);
    ...
}
```

The same code for `R1` is shared by the different instances.

4.1.4.4.1 Reentrant code

In general, side effects can appear if the same code entity is invoked by different threads of execution running, namely tasks. This holds particularly true, if the invoked code entity inherits a state or memory by the means of static variables which are visible to all instances. That would mean that all instances are coupled by those static variables.

Thus, they affect each other. This would lead to data consistency problems on one hand. On the other – and that is even more important – it would introduce a new communication mechanism to AUTOSAR and this is forbidden. AUTOSAR software components can only communicate via ports.

To be complete, it shall be noted that a calling code entity also inherits the reentrancy problems of its callee. This holds especially true in case of recursive calls.

4.1.4.4.2 Unambiguous object identification

[SWS_Rte_02015] ⌈ The instantiated AUTOSAR software component objects shall be unambiguously identifiable by an *instance handle*, if multiple instantiation by sharing code is required. ⌋*(SRS_Rte_00011, SRS_Rte_00012)*

4.1.4.4.3 Multiple instantiation and Per-instance memory

An AUTOSAR SW-C can define internal memory only accessible by a SW-C instance itself. This concept is called PerInstanceMemory. The memory can only be accessed by the runnable entities of this particular instance. That means in turn, other instances don't have the possibility to access this memory.

PerInstanceMemory API principles are explained in Section 5.2.5.

The API for PerInstanceMemory is specified in Section 5.6.15.

4.1.5 RTE and AUTOSAR Services

According to the AUTOSAR glossary [11] "an AUTOSAR service is a logical entity of the Basic Software offering general functionality to be used by various AUTOSAR software components. The functionality is accessed via standardized AUTOSAR interfaces".

Therefore, AUTOSAR services provide standardized AUTOSAR Interfaces: ports typed by standardized *PortInterfaces*.

When connecting AUTOSAR service ports to ports of AUTOSAR software components the RTE maps standard RTE API calls to the symbols defined in the RTE input (i.e. XML) for the AUTOSAR service runnables of the BSW. The key technique to distinguish ECU dependent identifiers for the AUTOSAR services is called "port-defined argument values", which is described in Section 4.3.2.4. Currently "port-defined argument values" are only supported for client-server communication. It is not possible to use a pre-defined symbol for sending or receiving data.

The RTE does not pass an instance handle to the *C*-based API of AUTOSAR services since the latter are single-instantiatable (see [SWS_Rte_03806]).

As displayed on figure 4.2, there can be direct interactions between the RTE and some Basic Software Modules. This is the case of the Operating System, the AUTOSAR Communication, and the NVRAM Manager.

4.1.6 RTE and ECU Abstraction

The *ECU Abstraction* provides an interface to physical values for AUTOSAR software components. It abstracts the physical origin of signals (their pathes to the ECU hardware ports) and normalizes the signals with respect to their physical appearance (like specific values of current or voltage).

See the AUTOSAR ECU architecture in figure 4.2. From an architectural point of view the ECU Abstraction is part of the *Basic Software* layer and offers AUTOSAR interfaces to AUTOSAR software components.

Seen from the perspective of an RTE, regular AUTOSAR ports are connected. Without any restrictions all communication paradigms specified by the AUTOSAR Virtual Functional Bus (VFB) shall be applicable to the ports, interfaces and connections – sender-receiver just as well as client-server mechanisms.

However, ports of the ECU Abstraction shall always only be connected to ports of specific AUTOSAR software components: sensor or actuator software components. In this sense they are tightly coupled to a particular ECU Abstraction.

Furthermore, it must not be possible (by an RTE) to connect AUTOSAR ports of the ECU Abstraction to AUTOSAR ports of any AUTOSAR component located on a remote ECU (see [SWS_Rte_02051].

This means, e.g., that sensor-related signals coming from the ECU Abstraction are always received by an AUTOSAR sensor component located on the same ECU. The AUTOSAR sensor component will then process the received signal and deploy it to other AUTOSAR components regardless of whether they are located on the same or any remote ECU. This applies to actuator-related signals accordingly, however, the opposite way around.

[SWS_Rte_02050] ⌈ The RTE Generator shall generate a communication path between connected ports of AUTOSAR sensor or actuator software components and the ECU Abstraction in the exact same manner like for connected ports of AUTOSAR software components. ⌋

[SWS_Rte_02051] ⌈ The RTE Generator shall reject configurations which require a communication path from a AUTOSAR software component to an ECU Abstraction located on a remote ECU. ⌋*(SRS_Rte_00062, SRS_Rte_00018)*

Further information about the ECU Abstraction can be found in the corresponding specification document [17].

4.1.7 RTE and Complex Device Driver

A Complex Device Driver has an AUTOSAR Interface, therefore the RTE can deal with the communication on the Complex Device Drivers ports. The Complex Device Driver is allowed to have code entities that are not under control of the RTE but yet still may use the RTE API (e.g. ISR2, BSW main processing functions).

4.1.8 Basic Software Scheduler and Basic Software Modules

4.1.8.1 Description of a Basic Software Module

The description of a Basic Software Module is divided into the sections

- interfaces
- internal behavior
- implementation

For further details see document [9].

4.1.8.2 Basic Software Interfaces

The interface of a *Basic Software Module* is described with *Basic Software Module Entries* (*BswModuleEntry*). For the functionality of the *Basic Software Scheduler* only *BswModuleEntry*s from *BswCallType SCHEDULED* are relevant. Nevertheless for optimization purpose the analysis of the full call tree might be required which requires the consideration of all *BswModuleEntry*'s

4.1.8.3 Basic Software Internal Behavior

The *Basic Software Internal Behavior* specifies the behavior of a BSW module or a BSW cluster w.r.t. the code entities visible by the BSW Scheduler. For the *Basic Software Scheduler* mainly *Basic Software Schedulable Entities* (*BswSchedulableEntity*'s) are relevant. These are *Basic Software Module Entities*, which are designed for control by the *Basic Software Scheduler*. *Basic Software Schedulable Entities* are implementing main processing functions. Furthermore all *Basic Software Schedulable Entities* are allowed to use exclusive areas and for call tree analysis all *Basic Software Module Entities* are relevant.

[SWS_Rte_07514] ⌈ The *Basic Software Scheduler* shall support multiple *Basic Software Module Entities* in *AUTOSAR Basic Software Modules*. ⌋*(SRS_Rte_00211, SRS_Rte_00213, SRS_Rte_00216)*

[SWS_Rte_07515] ⌈ The *Basic Software Scheduler* shall trigger the execution of *Schedulable Entity*'s in accordance with the connected *BswEvent*. ⌋*(SRS_Rte_00072)*

[SWS_Rte_07516] ⌈ The RTE Generator shall reject configurations where an *BswEvent* which can start a *Schedulable Entity* is not mapped to an OS task. The exceptions are *BswEvent* that are implemented by a direct function call. ⌋*(SRS_Rte_00049, SRS_Rte_00018)*

[SWS_Rte_07517] ⌈ The RTE Generator shall respect the configured execution order of *Schedulable Entities* within one OS task. ⌋*(SRS_Rte_00219)*

[SWS_Rte_07518] ⌈ The RTE shall support the execution sequences of *Runnable Entities* and *Schedulable Entities* within the same OS task in an arbitrarily configurable order. ⌋*(SRS_Rte_00219)*

4.1.8.4 Basic Software Implementation

The implementation defines further details of the implantation of the *Basic Software Module*. The *vendorApiInfix* attribute is of particular interest, because it defines the name space extension for multiple instances of the same basic software module. Further on the category of the `codeDescriptor` specifies if the *Basic Software Module* is delivered as source code or as object.

4.1.8.5 Multiple Instances of Basic Software Modules

In difference to the multiple instantiation concept of software components, where the same component code is used for all component instances, basic software modules are multiple instantiated by creation of own code per instance in a different name space. The attribute *vendorApiInfix* allows to define name expansions required for global symbols.

4.1.8.6 AUTOSAR Services / ECU Abstraction / Complex Device Drivers

AUTOSAR Services, *ECU Abstraction* and *Complex Device Drivers* are hybrid of AUTOSAR software-component and *Basic Software Module*. These kinds of modules might use *AUTOSAR Interfaces* to communicate via RTE as well as C-API to directly access other *Basic Software Modules*. Caused by the structure of the *AUTOSAR Meta Model* some entities of the 'C' implementation have to be described twice; on the one hand by the means of the *Software Component Template* [2] and on the other hand by the means of the *Basic Software Module Description Template* [9]. Further on the dualism of port based communication between software component and non-port based communication between *Basic Software Modules* requires in some cases the coordination and synchronization between both principles. The information about elements belonging together is provided by the so called *SwcBswMapping*.

4.1.8.6.1 RunnableEntity / BswModuleEntity mapping

A *Runnable Entity* which is mapped to a *Basic Software Module Entity* has to be treated as one common entity. This means it describes an entity which can use the features of a *Runnable Entity* and a *Basic Software Module Entity* as well. For instance it supports to use the port based API as well as *Basic Software Scheduler* API in one C function.

4.1.8.6.2 Synchronized ModeDeclarationGroupPrototype

Two synchronized *ModeDeclarationGroupPrototype* are resulting in the implementation of one common mode machine instance. Consequently the call of the belonging Rte_Switch API and the SchM_Switch API are having the same effect. For optimization purpose the Rte_Switch API might just refer to the SchM_Switch API.

4.1.8.6.3 Synchronized Trigger

Two synchronized *Trigger* are behaving like one common *Trigger*. Consequently the call of the belonging Rte_Trigger API and the SchM_Trigger API are having the same effect. For optimization purpose the Rte_Trigger API might just refer to the SchM_Trigger API.

4.2 RTE and Basic Software Scheduler Implementation Aspects

4.2.1 Scope

This section describes some specific implementation aspects of an AUTOSAR RTE and the Basic Software Scheduler. It will mainly address

- the mapping of logical concepts (e.g., Runnable Entities, BSW Schedulable Entities) to technical architectures (namely, the AUTOSAR OS)
- the decoupling of pending interrupts (in the Basic Software) and the notification of AUTOSAR software components
- data consistency problems to be solved by the RTE

Therefore this section will also refer to aspects of the interaction of the AUTOSAR RTE and Basic Software Scheduler and the two modules of the AUTOSAR Basic Software with standardized interfaces (see Figure 4.9):

- the module *AUTOSAR Operating System* [18, 4]
- the module *AUTOSAR COM* [19, 3]

Having a standardized interface means *first* that the modules do not provide or request services for/of the *AUTOSAR software components* located above the RTE. They do not have ports and therefore cannot be connected to the aforementioned AUTOSAR software components. AUTOSAR OS as well as AUTOSAR COM are simply invisible for them.

Secondly AUTOSAR OS and AUTOSAR COM are used by the RTE in order to achieve the functionality requested by the AUTOSAR software components. The AUTOSAR COM module is used by the RTE to route a signal over ECU boundaries, but this mechanism is hidden to the sending as well as to the receiving AUTOSAR software

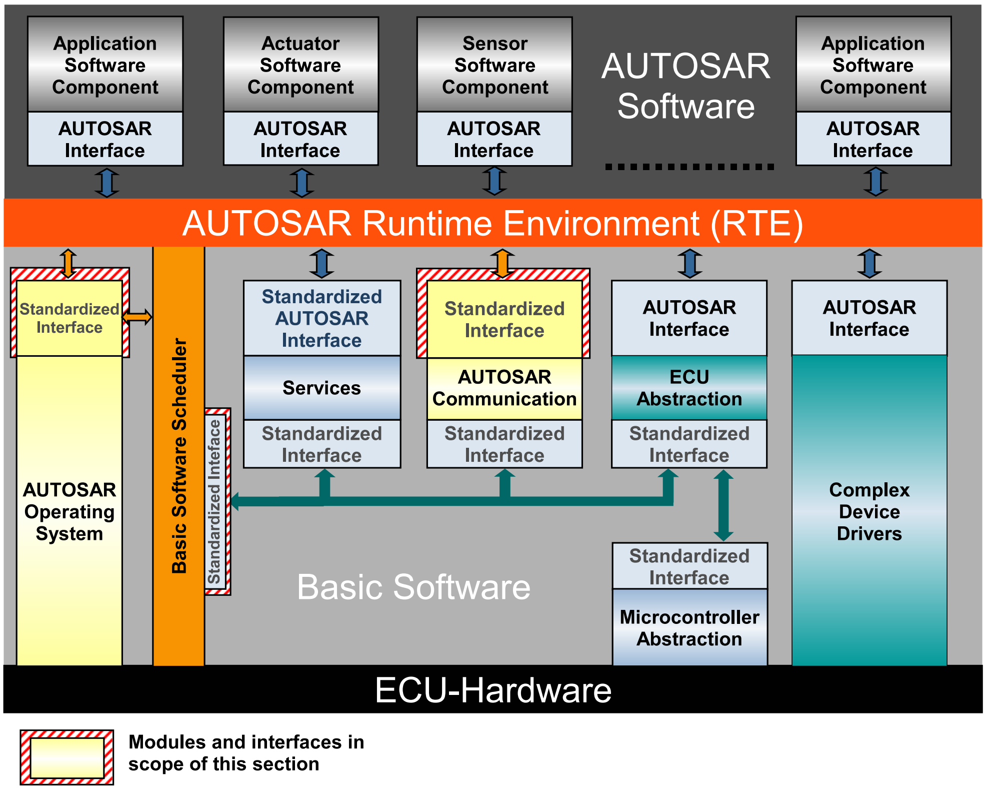


Figure 4.9: Scope of the section on Basic Software modules

component. The AUTOSAR OS module is used for two main purposes. First, OS is used by the RTE to route a signal over core and partition boundaries. Secondly, the AUTOSAR OS module is used by the RTE in order to properly schedule the single Runnables in the sense that the RTE Generator generates Task-bodies which contain then the calls to appropriate Runnables.

In this sense the RTE shall also *use* the available means to convert interrupts to notifications in a task context or to guarantee data consistency.

With respect to this view, the RTE is *thirdly* **not** a generic abstraction layer for AUTOSAR OS and AUTOSAR COM. It is generated for a specific ECU and offers the same *interface* to the AUTOSAR Software Components as the VFB. It implements the functionality of the VFB using modules of the Basic Software, including a specific implementation of AUTOSAR OS and AUTOSAR COM.

The *Basic Software Scheduler* offers services to integrate *Basic Software Modules* for all modules of all layers. Hence, the *Basic Software Scheduler* provides the following functions:

- embed *Basic Software Modules* implementations into the *AUTOSAR OS* context
- trigger `BswSchedulableEntity`s of the *Basic Software Modules*
- apply data consistency mechanisms for the *Basic Software Modules*

The integrator's task is to apply given means (of the AUTOSAR OS) in order to assemble BSW modules in a well-defined and efficient manner in a project specific context.

This also means that the BSW Scheduler only uses the AUTOSAR OS. It is not in the least a competing entity for the AUTOSAR OS scheduler.

[SWS_Rte_02250] ⌈ The RTE shall only use the AUTOSAR OS and AUTOSAR COM in order to provide the RTE functionality to the AUTOSAR components. ⌋*(SRS_Rte_00020)*

[SWS_Rte_07519] ⌈ The *Basic Software Scheduler* shall only use the *AUTOSAR OS* in order to provide the *Basic Software Scheduler* functionality to the *Basic Software Modules*. ⌋

[SWS_Rte_02251] ⌈ The RTE Generator shall construct task bodies for those tasks which contain `RunnableEntity`s and *Basic Software Schedulable Entities*. ⌋*(SRS_Rte_00049)*

The information for the construction of task bodies has to be given by the ECU Configuration description. The mapping of *Runnable Entities* to tasks is given as an input by the ECU Configuration description. The RTE Generator does not decide on the mapping of `RunnableEntity`s to tasks.

[SWS_Rte_02254] ⌈ The RTE Generator shall reject configurations where input information is missing regarding the mapping of *Runnable Entities* and *Basic Software Schedulable Entities* to OS tasks or the construction of tasks bodies. ⌋*(SRS_Rte_00049, SRS_Rte_00018)*

4.2.2 OS

This section describes the interaction between the RTE + Basic Software Scheduler and the AUTOSAR OS. The interaction is realized via the standardized interface of the OS - the AUTOSAR OS API. See Figure 4.9.

The OS is statically configured by the ECU Configuration. The RTE generator however may be allowed to create tasks and other OS objects, which are necessary for the runtime environment (see [SWS_Rte_05150]). The mapping of `RunnableEntity`s and *BSW Schedulable Entities* to OS tasks is not the job of the RTE generator. This mapping has to be done in a configuration step before, in the RTE-Configuration phase. The RTE generator is responsible for the generation of OS task bodies, which contain the calls for the `RunnableEntity`s and *BSW Schedulable Entities*. The `RunnableEntity`s and *BSW Schedulable Entities* themselves are OS independent and are not allowed to use OS service calls. The RTE and *Basic Software Scheduler* have to encapsulate such calls via the standardized RTE API respectively *Basic Software Scheduler* API.

4.2.2.1 OS Objects

Tasks

- The RTE generator has to create the task bodies, which contain the calls of the RunnableEntitys and BswSchedulableEntitys. Note that the term *task body* is used here to describe a piece of code, while the term *task* describes a configuration object of the OS.
- The RTE and *Basic Software Scheduler* controls the task activation/resumption either directly by calling OS services like SetEvent() or ActivateTask() or indirectly by initializing OS alarms or starting Schedule-Tables for time-based activation of RunnableEntitys. If the task terminates, the generated taskbody also contains the calls of TerminateTask() or ChainTask().
- The RTE generator does **not** create tasks. The mapping of RunnableEntitys and BswSchedulableEntitys to tasks is the input to the RTE generator and is therefore part of the RTE Configuration.
- The RTE configurator has to allocate the necessary tasks in the OS configuration.

OS applications

- AUTOSAR OS has in R4.0 a new feature called Inter-OS-Application Communication (IOC). IOC is generated by the OS based on the configuration partially generated by the RTE. The appropriate objects (OS-Applications) are generated by the OS, and are used by RTE to for task/runnable mapping.

Events

- The RTE and *Basic Software Scheduler* may use OS Events for the implementation of the abstract RTEEvents and BswEvents.
- The RTE and *Basic Software Scheduler* therefore may call the OS service functions SetEvent(), WaitEvent(), GetEvent() and ClearEvent().
- The used OS Events are part of the input information of the RTE generator.
- The RTE configurator has to allocate the necessary events in the OS configuration.

Resources

- The RTE and *Basic Software Scheduler* may use OS Resources (standard or internal) e.g. to implement data consistency mechanisms.
- The RTE and *Basic Software Scheduler* may call the OS services GetResource() and ReleaseResource().
- The used Resources are part of the input information of the RTE generator.
- The RTE configurator has to allocate the necessary resources (all types of resources) in the OS configuration.

Interrupt Processing

- An alternative mechanism to get consistent data access is disabling/enabling of interrupts. The AUTOSAR OS provides different service functions to handle interrupt enabling/disabling. The RTE may use these functions and must **not** use compiler/processor dependent functions for the same purpose.

Alarms

- The RTE may use Alarms for timeout monitoring of asynchronous client/server calls. The RTE is responsible for Timeout handling.
- The RTE and *Basic Software Scheduler* may setup cyclic alarms for periodic triggering of `RunnableEntity`s and `BswSchedulableEntity`s (`RunnableEntity` activation via `RTEEvent TimingEvent` respectively `BswSchedulableEntity` activation via `BswEvent BswTimingEvent`)
- The RTE and *Basic Software Scheduler* therefore may call the OS service functions `GetAlarmBase()`, `GetAlarm()`, `SetRelAlarm()`, `SetAbsAlarm()` and `CancelAlarm()`.
- The used Alarms are part of the input information of the RTE generator.
- The RTE configurator has to allocate the necessary alarms in the OS configuration.

Schedule Tables

- The RTE and *Basic Software Scheduler* may setup schedule tables for cyclic task activation (e.g. `RunnableEntity` activation via `RTEEvent TimingEvent`)
- The used schedule tables are part of the input information of the RTE generator.
- The RTE configurator has to allocate the necessary schedule tables in the OS configuration.

Common OS features

Depending on the global scheduling strategy of the OS, the RTE can make decisions about the necessary data consistency mechanisms. E.g. in an ECU, where all tasks are non-preemptive - and as the result also the global scheduling strategy of the complete ECU is non-preemptive - the RTE may optimize the generated code regarding the mechanisms for data consistency.

Hook functions

The AUTOSAR OS Specification defines hook functions as follows:

A Hook function is implemented by the user and invoked by the operating system in the case of certain incidents. In order to react to these on system or application level, there are two kinds of hook functions.

- **application-specific:** Hook functions within the scope of an individual OS Application.

- **system-specific:** Hook functions within the scope of the complete ECU (in general provided by the integrator).

If no memory protection is used (scalability classes SCC1 and SCC2) only the system-specific hook functions are available.

In the SRS the requirements to implement the system-specific hook functions were rejected [RTE00001], [RTE00101], [RTE00102] and [RTE00105], as well as the application-specific hook functions [RTE00198]. The reason for the rejection is the system (ECU) global scope of those functions. The RTE is not the only user of those functions. Other BSW modules might have requirements to use hook functions as well. This is the reason why the RTE is not able to generate these functions without the necessary information of the BSW configuration.

It is intended that the implementation of the hook functions is done by the system integrator and NOT by the RTE generator.

4.2.2.2 Basic Software Schedulable Entities

`BswSchedulableEntity`s are *Basic Software Module Entities*, which are designed for control by the BSW Scheduler. `BswSchedulableEntity`s are implementing main processing functions. The configuration of the *Basic Software Scheduler* allows mapping of `BswSchedulableEntity`s to both types; basic tasks and extended tasks.

`BswSchedulableEntity`s not mapped to a `RunnableEntity` are not allowed to enter a wait state. Therefore such `BswSchedulableEntity`s are comparable to `RunnableEntity`s of category 1. `BswSchedulableEntity`s mapped to a `RunnableEntity` can enter wait states by usage of the RTE API and such `BswSchedulableEntity`s have to be treated according the classification of the mapped `RunnableEntity`. The mapping of `BswSchedulableEntity`s to a `RunnableEntity`s is typically used for *AUTOSAR Services*, *ECU Abstraction* and *Complex Device Drivers*. See sections 4.1.8.6.

4.2.2.3 Runnable Entities

The following section describes the `RunnableEntity`s, their categories and their task-mapping aspects. The prototypes of the functions implementing `RunnableEntity`s are described in section 5.7

Runnable entities are the schedulable parts of SW-Cs. With the exception of reentrant server runnables that are invoked via direct function calls, they have to be mapped to tasks. The mapping must be described in the ECU Configuration Description. This configuration - or just the RTE relevant parts of it - is the input of the RTE generator.

All `RunnableEntity`s are activated by the RTE as a result of an `RTEEvent`. Possible activation events are described in the meta-model by using `RTEEvent`s (see section 4.2.2.4). If no RTEEvent is specified in the role startOnEvent for the RunnableEntity, the RunnableEntity is never activated by the RTE.

The categories of RunnableEntitys are described in [2].

RunnableEntitys and BswSchedulableEntitys are generalized by ExecutableEntitys.

4.2.2.4 RTE Events

The meta model describes the following RTE events:

Abbreviation	Name
T	TimingEvent
BG	BackgroundEvent
DR	DataReceivedEvent (S/R Communication only)
DRE	DataReceiveErrorEvent (S/R Communication only)
DSC	DataSendCompletedEvent (explicit S/R Communication only)
DWC	DataWriteCompletedEvent (implicit S/R Communication only)
OI	OperationInvokedEvent (C/S Communication only)
ASCR	AsynchronousServerCallReturnsEvent (C/S communication only)
MS	SwcModeSwitchEvent
MSA	ModeSwitchedAckEvent
MME	SwcModeManagerErrorEvent
ETO	ExternalTriggerOccurredEvent
ITO	InternalTriggerOccurredEvent
I	InitEvent

According to the meta model each kind of RTEEvent can either

ACT activate a RunnableEntity, or

WUP wakeup a RunnableEntity at its WaitPoints

The meta model makes no restrictions which kind of RTEEvents are referred by WaitPoints. As a consequence RTE API functions would be necessary to set up the WaitPoints for each kind of RTEEvent.

Nevertheless in some cases it seems to make no sense to implement all possible combinations of the general meta model. E.g. setting up a WaitPoint, which should be resolved by a cyclic TimingEvent . Therefore the RTE SWS defines some restrictions, which are also described in section A.

The meta model also allows, that the same RunnableEntity can be triggered by several RTEEvents. For the current approach of the RTE and restrictions see section 4.2.6.

	T	BG	DR	DRE	DSC	DWC	OI	ASCR
ACT	x	x	x	x	x	x	x	x
WUP			x		x			x
	MS	**MSA**	**MME**	**ETO**	**ITO**	**I**		
ACT	x	x	x	x	x	x		
WUP		x						

The table shows, that *activation of* *RunnableEntity* is possible for each kind of RTE-Event. For RunnableEntity activation, no explicit RTE API in the to be activated RunnableEntity is necessary. The RTE itself is responsible for the activation of the RunnableEntity depending on the configuration in the SW-C Description.

If the RunnableEntity contains a WaitPoint, it can be resolved by the assigned RTEEvent(s). Entering the WaitPoint requires an explicit call of a RTE API function. The RTE (together with the OS) has to implement the WaitPoint inside this RTE API.

The following list shows which RTE API function has to be called to set up Wait-Points.

- DataReceivedEvent: Rte_Receive()
- DataSendCompletedEvent: Rte_Feedback()
- ModeSwitchedAckEvent: Rte_SwitchAck()
- AsynchronousServerCallReturnsEvent: Rte_Result()

[SWS_Rte_01292] ⌈ When a DataReceivedEvent references a RunnableEntity and a required VariableDataPrototype and no WaitPoint references the DataReceivedEvent, the RunnableEntity shall be activated when the data is received. [SWS_Rte_01135]. ⌋*(SRS_Rte_00072)*

Requirement [SWS_Rte_01292] merely affects when the runnable is activated – an API call should still be created, according to requirement [SWS_Rte_01288], [SWS_Rte_01289], and [SWS_Rte_07395] as appropriate, to actually read the data.

4.2.2.5 BswEvents

The meta model describes the following BswEvents.

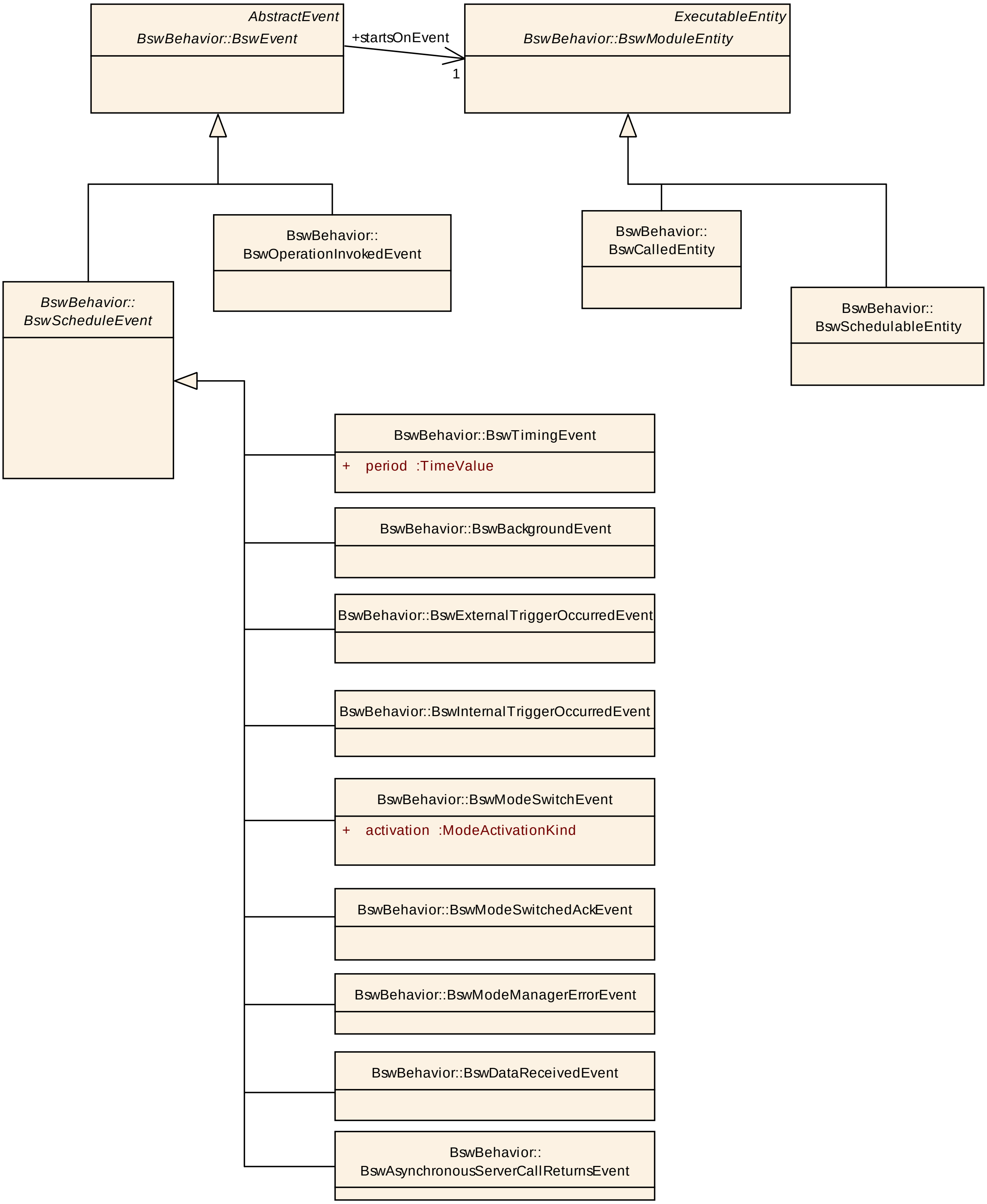


Figure 4.10: Different kinds of `BswEvents`

Similar to RTEEvents the *activation of Basic Software Schedulable Entities* is possible for each kind of BswEvent. For of BswSchedulableEntitys activation, no explicit *Basic Software Scheduler* API in the to be activated BswSchedulableEntity is necessary. The *Basic Software Scheduler* itself is responsible for the activation of the BswSchedulableEntity depending on the configuration in the *Basic Software Module Description*. In difference to RTEEvents, none of the BswEvents support WaitPoints. For more details see document [9].

4.2.2.6 Mapping of Runnable Entities and Basic Software Schedulable Entities to tasks (informative)

One of the main requirements of the RTE generator is "Construction of task bodies" [SRS_Rte_00049]. The necessary input information e.g. the mapping of RunnableEntitys and BswSchedulableEntity to tasks must be provided by the ECU configuration description.

The ECU configuration description (or an extract of it) is the input for the RTE Generator (see Figure 3.4). It is also the purpose of this document to define the necessary input information. Therefore the following scenarios may help to derive requirements for the ECU Configuration Template as well as for the RTE-generator itself.
Note: The scenarios do not cover all possible combinations.

The RTE-Configurator uses parts of the ECU Configuration of other BSW Modules, e.g. the mapping of RunnableEntitys to OsTasks. In this configuration process the RTE-Configurator expects OS objects (e.g. Tasks, Events, Alarms...) which are used in the generated RTE and *Basic Software Scheduler*.

Some figures for better understanding use the following conventions:

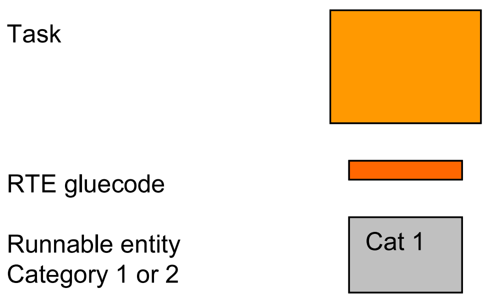


Figure 4.11: Element description

Note: The following examples are only showing RunnableEntitys. But taking the categorization of BswSchedulableEntitys defined in section 4.2.2.2 into account, the scenarios are applicable for BswSchedulableEntitys as well.

4.2.2.6.1 Scenario for mapping of RunnableEntitys to tasks

The different properties of RunnableEntitys with respect to data access and termination have to be taken into account when discussing possible scenarios of mapping RunnableEntitys to tasks.

- RunnableEntitys using VariableAccesses in the dataReadAccess or dataWriteAccess roles (implicit read and send) have to terminate.
- RunnableEntitys of category 1 can be mapped either to basic or extended tasks. (see next subsection).
- RunnableEntitys using at least one WaitPoint are of category 2.
- RunnableEntitys of category 2 that contain WaitPoints will be typically mapped to extended tasks.
- RunnableEntitys that contain a SynchronousServerCallPoint generally have to be mapped to extended tasks.
- RunnableEntitys that contain a SynchronousServerCallPoint can be mapped to basic tasks if no timeout monitoring is required and the server runnable is on the same partition.
- RunnableEntitys that contain a SynchronousServerCallPoint can be mapped to basic tasks if the server runnable is invoked directly and is itself of category 1.

Note that the runnable to task mapping scenarios supported by a particular RTE implementation might be restricted.

4.2.2.6.1.1 Scenario 1

Runnable entity category 1A: "runnable1"

- Ports: only S/R with VariableAccesses in the dataReadAccess or dataWriteAccess role
- RTEEvents: TimingEvent
- no sequence of RunnableEntitys specified
- no VariableAccess in the dataSendPoint role
- no WaitPoint

Possible mappings of "runnable1" to tasks:

Basic Task
If only one of those kinds of RunnableEntitys is mapped to a task (task contains only one RunnableEntity), or if multiple RunnableEntitys with the same activation period are mapped to the same task, a basic task can be used. In this case, the

execution order of the RunnableEntitys within the task is necessary. In case the RunnableEntitys have different activation periods, the RTE has to provide the glue-code to guarantee the correct call cycle of each RunnableEntity.

The ECU Configuration-Template has to provide the sequence of RunnableEntitys mapped to the same task, see RtePositionInTask.

Figure 4.12 shows the possible mappings of RunnableEntitys into a basic task. If and only if a sequence order is specified, more than one RunnableEntity can be mapped into a basic task.

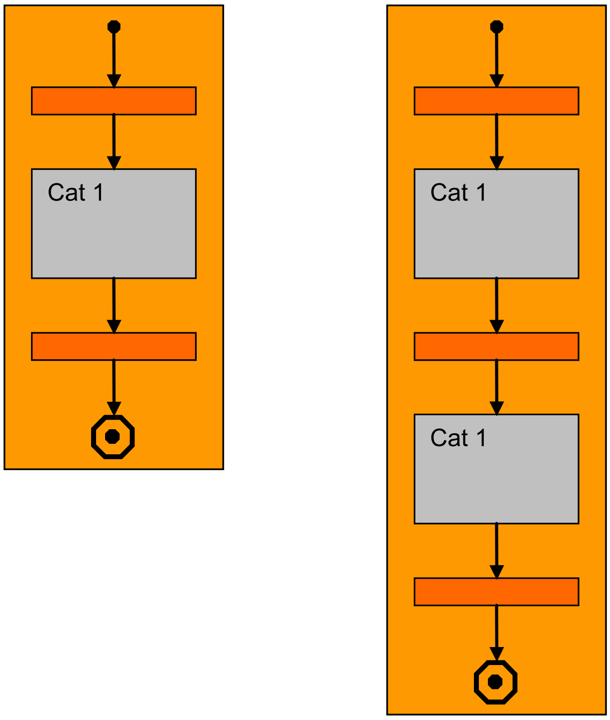


Figure 4.12: Mapping of Category 1 RunnableEntitys to Basic Tasks

Extended Task

If more than one RunnableEntity is mapped to the same task and the special condition (same activation period) does not fit, an extended task is used.

If an extended task is used, the entry points to the different RunnableEntitys might be distinguished by evaluation of different OS events. In the scenario above, the different activation periods may be provided by different OS alarms. The corresponding OS events have to be handled inside the task body. Therefore the RTE-generator needs for each task the number of assigned OS Events and their names.

The ECU Configuration has to provide the OS events assigned to the RTEEvents triggering the RunnableEntitys that are mapped to an extended task, see RteUsedOsEventRef.

Figure 4.13 shows the possible mapping of the multiple RunnableEntitys of category 1 into an Extended Task. Note: The Task does not terminate.

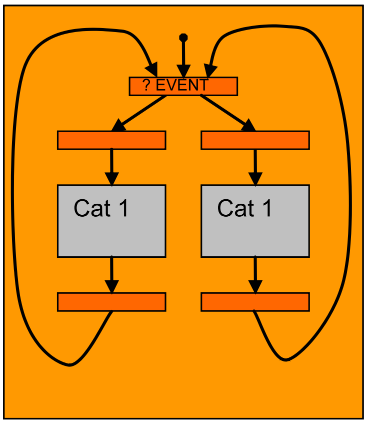


Figure 4.13: Mapping of Category 1 RunnableEntitys to Extended Tasks

For both, basic tasks and extended tasks, the ECU Configuration must provide the name of the task.

The ECU Configuration has to provide the name of the task, see OsTask.

The ECU Configuration has to provide the task type (BASIC or EXTENDED), which can be determined from the presence or absence of OS Events associated with that task, see OsTask.

4.2.2.6.1.2 Scenario 2

Runnable entity category 1B: "runnable2"

- Ports: S/R with VariableAccesses in the dataSendPoint role.
- RTEEvents: TimingEvent
- no WaitPoint

Possible mappings of "runnable2" to tasks:

The following figure shows the different mappings:

- One category 1B runnable
- More than one category 1B runnable mapped to the same basic task with a specified sequence order
- More than one category 1B runnable mapped into an extended task

The gluecode to realize the VariableAccessin the dataReadAccess and dataWriteAccess roles respectively before entering the runnable and after exiting is not necessary.

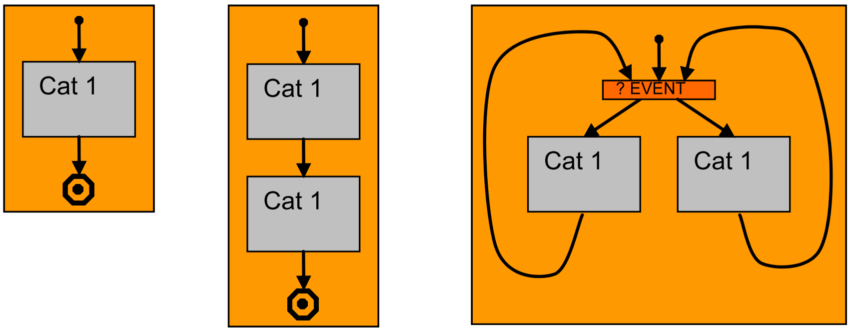


Figure 4.14: Mapping of Category 1 RunnableEntitys using no VariableAccesses in the dataReadAccess or dataWriteAccess role

4.2.2.6.1.3 Scenario 3

Runnable entity category 1A: "runnable3"

- Ports: S/R with VariableAccesses in the dataReadAccess or dataWriteAccess role
- RTEEvents: Runnable is activated by a DataReceivedEvent
- no VariableAccess in the dataSendPoint role
- no WaitPoint

There is no difference between Scenario 1 and 3. Only the RTEEvent that activates the RunnableEntity is different.

4.2.2.6.1.4 Scenario 4

Runnable entity category 2: "runnable4"

- Ports: S/R with VariableAccesses in the dataReceivePointByValue or dataReceivePointByArgument role and WaitPoint (blocking read)
- RTEEvents: WaitPoint referencing a DataReceivedEvent

Runnable is activated by an arbitrary RTEEvent (e.g. by a TimingEvent). When the RunnableEntity has entered the WaitPoint and the DataReceivedEvent occurs, the RunnableEntity resumes execution.

The runnable has to be mapped to an extended task. Normally each category 2 runnable has to be mapped to its own task. Nevertheless it is not forbidden to map multiple category 2 RunnableEntitys to the same task, though this might be restricted by an RTE generator. Mapping multiple category 2 RunnableEntitys to the same task can lead to big delay times if e.g. a WaitPoint is resolved by the incoming RTEEvent, but the task is still waiting at a different WaitPoint.

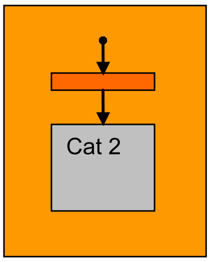


Figure 4.15: Mapping of Category 2 RunnableEntitys to Extended Tasks

4.2.2.6.1.5 Scenario 5

There are two RunnableEntitys implementing a client (category 2) and a server for synchronous C/S communication and the timeout attribute of the ServerCallPoint is 0.

On a single core, there are two ways to invoke a server synchronously:

- Simple function call for intra-partition C/S communication if the canBeInvokedConcurrently attribute of the server runnable is set and if the server runnable is of category 1. In that case the server runnable is executed in the same task context (same stack) as the client runnable that has invoked the server. The client runnable can be mapped to a basic task.
- The server runnable is mapped to its own task. If the canBeInvokedConcurrently attribute is not set, the server runnable must be mapped to a task.

  If the implementation of the synchronous server invocation does not use OS events, the client runnable can be mapped to a basic task and the task of the server runnable must have higher priority than the task of the client runnable. Furthermore, the task to which the client runnable is mapped must be preemptable. This has to be checked by the RTE generator. Activation of the server runnable can be done by ActivateTask() for a basic task or by SetEvent() for an extended task. In both cases, the task to be activated must have higher priority than the task of the client runnable to enforce a task switch (necessary, because the server invocation is synchronous).

4.2.2.6.1.6 Scenario 6

There are two RunnableEntitys implementing a client (category 2) and a server for synchronous C/S communication and the timeout attribute of the ServerCallPoint is greater than 0.

There are again two ways to invoke a server synchronously:

- Simple function call for intra-partition C/S communication if the canBeInvokedConcurrently attribute of the server runnable is set and the server is of category 1. In that case the server runnable is executed in the same task context (same stack) as the client runnable that has invoked the server and no timeout monitoring is performed (see [SWS_Rte_03768]). In this case the client runnable can be mapped to a basic task.

- The server runnable is mapped to its own task. If the `canBeInvokedConcurrently` attribute is not set, the server runnable must be mapped to a task.

  If the implementation of the timeout monitoring uses OS events, the task of the server runnable must have lower priority than the task of the client runnable and the client runnable must be mapped to an extended task. Furthermore, both tasks must be preemptable[1]. This has to be checked by the RTE generator. The notification that a timeout occurred is then notified to the client runnable by using an OS Event. In order for the client runnable to immediately react to the timeout, a task switch to the client task must be possible when the timeout occurs.

4.2.2.6.1.7 Scenario 7

Runnable entity category 2: "runnable7"

- Ports: only C/S with `AsynchronousServerCallPoint` and `WaitPoint`
- `RTEEvent`s: `AsynchronousServerCallReturnsEvent` (C/S communication only)

The mapping scenario for "runnable7", the client runnable that collects the result of the asynchronous server invocation, is similar to Scenario 4.

[1]Strictly speaking, this restriction is not necessary for the task to which the client runnable is mapped. If OS events are used to implement the timeout monitoring and the notification that the server is finished, the RTE API implementation generally uses the OS service `WaitEvent`, which is a point of rescheduling.

4.2.2.7 Monitoring of runnable execution time

This section describes how the monitoring of RunnableEntity execution time can be done.

The RTE doesn't directly support monitoring of RunnableEntitys execution time but the AUTOSAR OS support for monitoring of OsTasks execution time can be used for this purpose.

If execution time monitoring of a RunnableEntity is required a possible solution is to map the RunnableEntity alone to an OsTask and to configure the OS to monitor the execution time of the OsTask.

This solution can lead to dispatch to individual OsTasks RunnableEntitys that should be initially mapped to the same OsTask because of for example:

- requirements on execution order of the RunnableEntitys and/or
- requirements on evaluation order of the RTEEvents that activate the RunnableEntitys and
- constraints to have no preemption between the RunnableEntitys

In order to keep the control on the execution order of the RunnableEntitys, the evaluation order of the RTEEvents and the non-preemption between the RunnableEntitys when then RunnableEntitys are individually mapped to several OsTasks for the purpose of monitoring, a possible solution is to replace the calls to the C-functions of the RunnableEntitys by activations of the OsTasks to which the monitored RunnableEntitys are mapped.

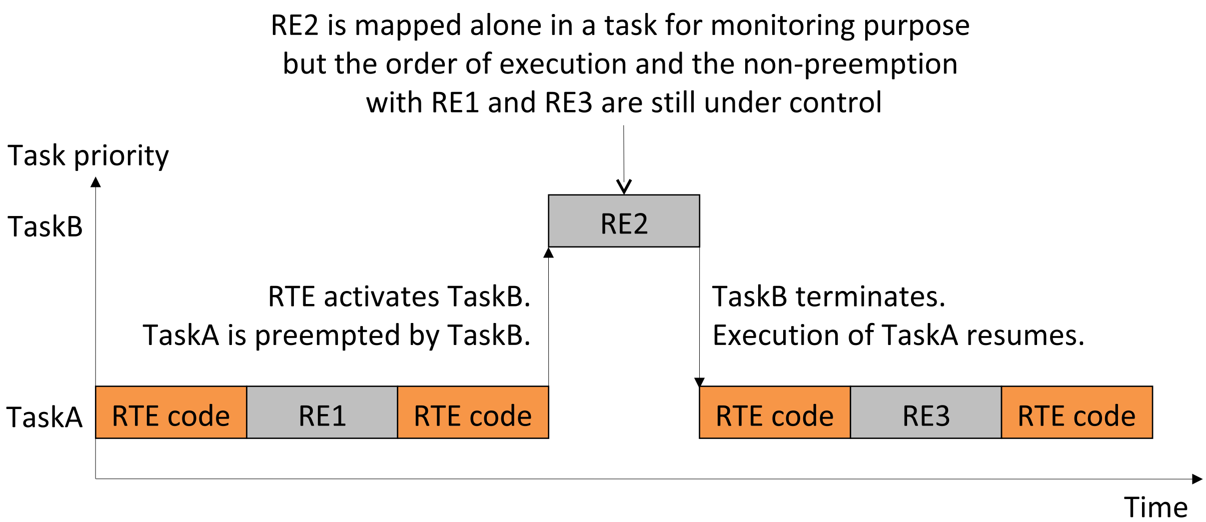


Figure 4.16: Inter task activation and mapping of runnable to individual task for monitoring purpose

This behavior of the RTE can be configured with the attributes RteVirtuallyMappedToTaskRef of the RteEventToTaskMapping. RteVirtuallyMapped-

ToTaskRef references the OsTask in which the execution order of the RunnableEntitys and/or the evaluation order of the RTEEvents are controlled. RteMappedToTaskRef references the individual OsTasks to which the RunnableEntitys are mapped for the purpose of monitoring.

[SWS_Rte_07800] ⌈ The RTE Generator shall respect the configured virtual runnable to task mapping (RteVirtuallyMappedToTaskRef) in the RTE configuration. ⌋*(SRS_Rte_00193)*

Of course this solution requires that the task priorities and scheduling properties are well configured in the OS to allow immediate preemption by the OsTasks to which the monitored RunnableEntitys are mapped. A possible solution is:

- Priority of the OsTask to which the RunnableEntity is mapped is higher than the priority of the OsTask to which the RunnableEntity is virtually mapped and
- the OsTask to which the RunnableEntity is virtually mapped have a full preemptive scheduling or
- the RTE call the OS service Schedule() just after activation of the OsTask to which the RunnableEntity is mapped

Example 1: Without OsEvent

Description of the example:
RunnableEntity RE1 is activated by TimingEvent 100ms T1.
RunnableEntity RE2 is activated by TimingEvent 100ms T2.
RunnableEntity RE3 is activated by TimingEvent 100ms T3.
Execution order of the RunnableEntitys shall be R1, R2 then R3.
RE2 shall be monitored.

Possible RTE configuration:
RE1/T1 is mapped to OsTask TaskA with RtePositionInTask equal to 1.
RE2/T2 is mapped to OsTask TaskB but virtually mapped to TaskA with RtePositionInTask equal to 2.
RE3/T3 is mapped to OsTask TaskA with RtePositionInTask equal to 3.

Possible RTE implementation:
RTE starts cyclic OsAlarm with 100ms period.
This OsAlarm is configured to activate TaskA.
Non preemptive scheduling is configured for Task A.
TaskB priority = TaskA priority + 1

```
void TaskA(void)
{
   RE1();
   ActivateTask(TaskB);
   Schedule();
   RE3();
   TerminateTask();
```

```
}

void TaskB(void)
{
   RE2();
   TerminateTask();
}
```

Example 2: With OsEvent

Description of the example:
RunnableEntity RE1 is activated by DataReceivedEvent DR1.
RunnableEntity RE2 is activated by DataReceivedEvent DR2.
RunnableEntity RE3 is activated by DataReceivedEvent DR3.
Evaluation order of the RTEEvents shall be DR1, DR2 then DR3.
All the runnables shall be monitored.

Possible RTE configuration:
RE1 is mapped to OsTask TaskB but virtually mapped to TaskA with a reference to OsEvent EvtA and RtePositionInTask equal to 1.
RE2 is mapped to OsTask TaskC but virtually mapped to TaskA with a reference to OsEvent EvtB and RtePositionInTask equal to 2.
RE3 is mapped to OsTask TaskD but virtually mapped to TaskA with a reference to OsEvent EvtC and RtePositionInTask equal to 3.

Possible RTE implementation:
RTE set EvtA, EvtB and EvtC according to the callbacks from COM.
Full preemptive scheduling is configured for Task A.
TaskB priority = TaskC priority = TaskD priority = TaskA priority + 1

```
void TaskA(void)
{
   EventMaskType Event;

   while(1)
   {
      WaitEvent(EvtA | EvtB | EvtC);
      GetEvent(TaskA, &Event);
      if (Event & EvtA)
      {
         ClearEvent(EvtA);
         ActivateTask(TaskB);
      }
      else if (Event & EvtB)
      {
         ClearEvent(EvtB);
         ActivateTask(TaskC);
      }
      else if (Event & EvtC)
      {
         ClearEvent(EvtC);
         ActivateTask(TaskD);
      }
   }
```

```
}

void TaskB(void)
{
    RE1();
    TerminateTask();
}

void TaskC(void)
{
    RE2();
    TerminateTask();
}

void TaskD(void)
{
    RE3();
    TerminateTask();
}
```

It is also possible to configure the RTE for the monitoring of group of runnable = monitoring of the sum of the runnable execution times.

Example 3: Monitoring of group of runnables

Description of the example:
RunnableEntity RE1 is activated by TimingEvent 100ms T1.
RunnableEntity RE2 is activated by TimingEvent 100ms T2.
RunnableEntity RE3 is activated by TimingEvent 100ms T3.
RunnableEntity RE4 is activated by DataReceivedEvent DR1.
RunnableEntity RE5 is activated by DataReceivedEvent DR2.
RunnableEntity RE6 is activated by DataReceivedEvent DR3.
RunnableEntity RE7 is activated by DataReceivedEvent DR4.
DataReceivedEvent DR2, DR3 and DR4 references the same dataElement. Evaluation order of the RTEEvents shall be T1, T2, T3, DR1, DR2, DR3 then DR4.
RE2 and RE3 shall be monitored as a group.
RE6 and RE7 shall be monitored as a group.

Possible RTE configuration:
RE1 is mapped to OsTask TaskA with a reference to OsEvent EvtA and RtePositionInTask equal to 1.
RE2 is mapped to OsTask TaskB but virtually mapped to TaskA with a reference to OsEvent EvtA and RtePositionInTask equal to 2.
RE3 is mapped to OsTask TaskB but virtually mapped to TaskA with a reference to OsEvent EvtA and RtePositionInTask equal to 3.
RE4 is mapped to OsTask TaskA with a reference to OsEvent EvtB and RtePositionInTask equal to 4.
RE5 is mapped to OsTask TaskA with a reference to OsEvent EvtC and RtePositionInTask equal to 5.
RE6 is mapped to OsTask TaskC but virtually mapped to TaskA with a reference to OsEvent EvtC and RtePositionInTask equal to 6.

RE7 is mapped to OsTask TaskC but virtually mapped to TaskA with a reference to OsEvent EvtC and RtePositionInTask equal to 7.

Possible RTE implementation:
RTE starts cyclic OsAlarm with 100ms period.
This OsAlarm is configured to set EvtA.
RTE set EvtB and EvtC according to the callbacks from COM.
Full preemptive scheduling is configured for Task A.
TaskB priority = TaskC priority = TaskA priority + 1

```
void TaskA(void)
{
   EventMaskType Event;

   while(1)
   {
      WaitEvent(EvtA | EvtB | EvtC);
      GetEvent(TaskA, &Event);
      if (Event & EvtA)
      {
         ClearEvent(EvtA);
         RE1();
         ActivateTask(TaskB);
      }
      else if (Event & EvtB)
      {
         ClearEvent(EvtB);
         RE4();
      }
      else if (Event & EvtC)
      {
         ClearEvent(EvtC);
         RE5();
         ActivateTask(TaskC);
      }
   }
}

void TaskB(void)
{
   RE2();
   RE3();
   TerminateTask();
}

void TaskC(void)
{
   RE6();
   RE7():
   TerminateTask();
}
```

4.2.2.8 TimingEvent activated runnables

A TimingEvent / BswTimingEvent is a recurring RTEEvent / BswEvent which is used to perform recurrent activities in RunnableEntitys or BswSchedulableEntitys.

[SWS_Rte_06728] ⌈ The RTE shall activate RunnableEntitys triggered by a TimingEvent recurring with the effective period time of an TimingEvent for the component instance. ⌋*(SRS_Rte_00237)*

[SWS_Rte_06729] ⌈ The RTE Generator shall determine the effective period time of a TimingEvent from the period attribute of the TimingEvent if no InstantiationRTEEventProps are defined for the TimingEvent of the component instance. ⌋*(SRS_Rte_00237)*

[SWS_Rte_06730] ⌈ The RTE Generator shall determine the effective period time of a TimingEvent from the period attribute of the InstantiationRTEEventProps if InstantiationRTEEventProps are defined for the TimingEvent of the component instance. ⌋*(SRS_Rte_00237)*

Please note the component instance is defined by RteSoftwareComponentInstanceRef of RteSwComponentInstance referring to the SwComponentPrototype. See figure 7.2.

4.2.2.9 Synchronization of TimingEvent activated runnables

This section describes how the synchronization of TimingEvent activated RunnableEntitys can be done.

The following cases have to be distinguished:

- the RunnableEntitys are mapped to the same OsTask
- the RunnableEntitys are mapped to different OsTasks in the same OsApplication
- the RunnableEntitys are mapped to different OsTasks in different OsApplications on the same core
- the RunnableEntitys are mapped to different OsTasks in different OsApplications on different cores on the same microcontroler
- the RunnableEntitys are mapped to different OsTasks in different OsApplications on different microcontrolers within the same ECU
- the RunnableEntitys are mapped to different OsTasks in different OsApplications on different microcontrolers within different ECUs

As OsAlarms and OsScheduleTableExpiryPoints are used to implement TimingEvents the following different possible solutions exist to synchronize the RunnableEntitys according to the different cases:

- use the same OsAlarm or OsScheduleTableExpiryPoint to implement all the TimingEvents
- use different OsAlarms or OsScheduleTableExpiryPoints in different OsScheduleTables based on the same OsCounter and start them with absolute start offset to control the synchronization between them
- use different OsScheduleTableExpiryPoints in different explicitely synchronized OsScheduleTables based on different OsCounters but with same period and max value

The choice of the OsAlarms or OsScheduleTableExpiryPoints used to implement the TimingEvents can be configured in the RTE with RteUsedOsAlarmRef or RteUsedOsSchTblExpiryPointRef in the RteEventToTaskMapping.

[SWS_Rte_07804] ⌈ The RTE Generator shall respect the configured OsAlarms (RteUsedOsAlarmRef) and OsScheduleTableExpiryPoints (RteUsedOsSchTblExpiryPointRef) for the implementation of the TimingEvents. ⌋*(SRS_Rte_00232)*

The choice of the absolute start offset of the OsAlarms and OsScheduleTables can be configured in the RTE with RteExpectedActivationOffset in the RteUsedOsActivation.

[SWS_Rte_07805] ⌈ The RTE Generator shall respect the configured absolute start offset (RteExpectedActivationOffset) when it starts the OsAlarms and OsScheduleTables used for the implementation of the TimingEvents. ⌋*(SRS_Rte_00232)*

The RTE / *Basic Software Scheduler* is not responsible to synchronize/desynchronize the explicitly synchronized OsScheduleTables. The RTE / *Basic Software Scheduler* is only responsible to start the explicitly synchronized OsScheduleTables. In this case no RteExpectedActivationOffset has to be configured.

4.2.2.10 BackgroundEvent activated Runnable Entities and BasicSoftware Scheduleable Entities

A BackgroundEvent is a recurring RTEEvent / BswEvent which is used to perform background activities in RunnableEntitys or BswSchedulableEntitys. It is similar to a TimingEvent but has no fixed time period and is typically activated only with lowest priority.

A BackgroundEvent triggering can be implemented in two principle ways by the RTE Generator. Either the background activation is done by a real background OS task; or the BackgroundEvents are activated like TimingEvents on a fixed recurrence which is defined by the ECU integrator (see [SWS_Rte_07179] and [SWS_Rte_07180]). The second way might be required to overcome the limitation of a single real background OS task if BackgroundEvents are used in several partitions.

If the background activation is done by a real background OS task, the OS Task has to have the lowest priority on the CPU core (see [SWS_Rte_07181]). If a implementation is used where the OS Task terminates (*BasicTask*) the background OS Task is immediately reactivated after its termination, e.g. by usage of ChainTask call of the OS.

4.2.2.11 InitEvent activated Runnable Entities

An InitEvent which is used to activate RunnableEntitys for initialization purpose in case of start of the RTE or restart of a partition.

[SWS_Rte_06761] ⌈ The RTE shall activate RunnableEntitys triggered by a InitEvent once when Rte_Start is executed. ⌋*(SRS_Rte_00240)*

[SWS_Rte_06762] ⌈ The RTE shall activate RunnableEntitys triggered by a InitEvent once when Rte_RestartPartition is executed for those RunnableEntitys belonging to the restarted partition. ⌋*(SRS_Rte_00240)*

The activation of RunnableEntitys for initialization purpose can basically implemented in two ways. Either the InitEvent is mapped to an OsTask or the InitEvent is mapped to an RteInitializationRunnableBatch.

In case of an OsTask the RunnableEntitys are scheduled once when the related task gets active. In this case the RtePositionInTask decides in which order the RunnableEntitys are scheduled in the whole task. For instance if the InitEvent is mapped after an TimingEvent ans the TimingEvent is already triggered when the OsTask gets active the initialization runnable is called after time periodic runnable. Therefore its in the responsibility of the ECU integrator to ensure the correct and intended order.

In the case the InitEvent is mapped to an RteInitializationRunnableBatch the RunnableEntitys are scheduled when the related Rte_Init function is called. In this case the RtePositionInTask decides in which order in which order the RunnableEntitys are scheduled in the same Rte_Init function.

The triggering of the recurrent RTEEvents is released with the call of Rte_StartTiming.

4.2.3 Activation and Start of **ExecutableEntitys**

This section defines the activation of ExecutableEntity execution-instances by using a state machine (Fig. 4.17).

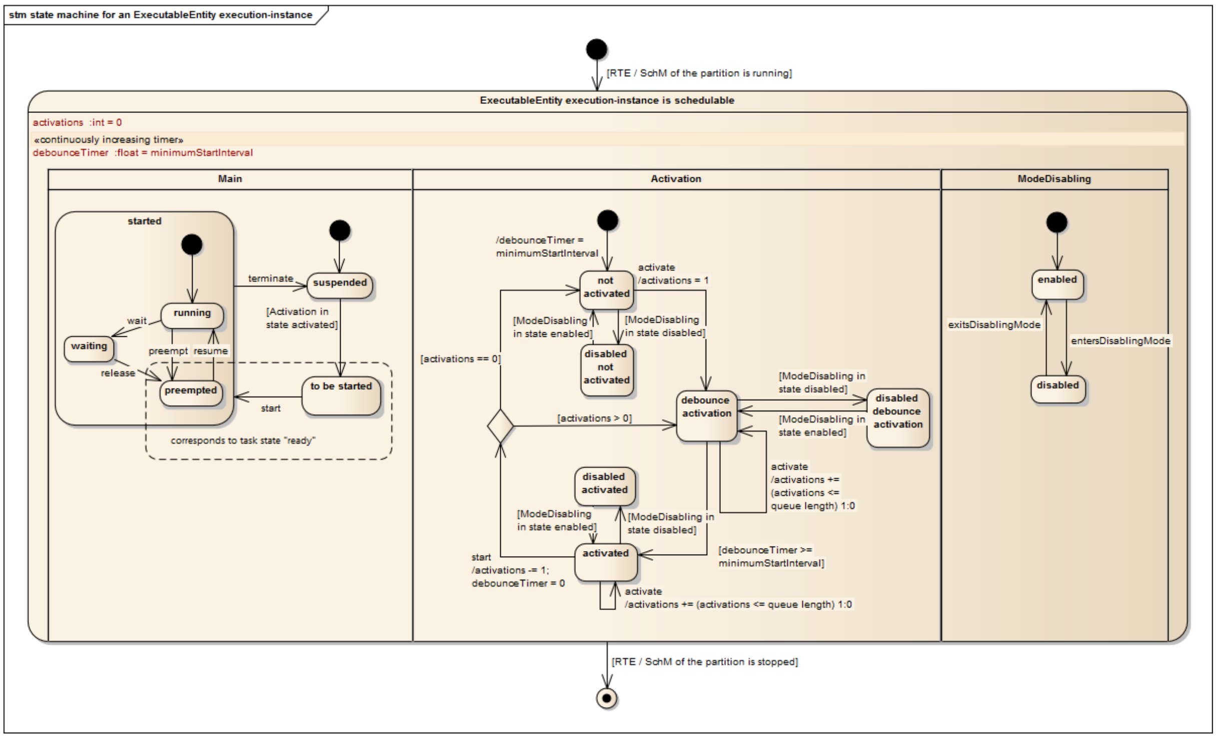


Figure 4.17: General state machine of an ExecutableEntity execution-instance.

An ExecutableEntity execution-instance is one execution-instance of an ExecutableEntity (RunnableEntity or BswSchedulableEntity) with respect to concurrent execution.

For a RunnableEntity with canBeInvokedConcurrently = false or for a BswSchedulableEntity whose referenced BswModuleEntry in the role implementedEntry has a isReentrant attribute set to false, there is only one execution-instance. For a RunnableEntity with canBeInvokedConcurrently = true or for a BswSchedulableEntity whose referenced BswModuleEntry in the role implementedEntry has its isReentrant attribute set to true, there is a well defined number of execution-instances.

E.g., for a server runnable that is executed as direct function call, each ServerCallPoint relates to exactly one ExecutableEntity execution-instance.

The main principles for the activation of runnables are:

- RunnableEntitys are activated by RTEEvents
- BswSchedulableEntitys are activated by BswEvents
- only server runnables (RunnableEntitys activated by an OperationInvokedEvent) are queued. All other ExecutableEntitys are unqueued.

If a RunnableEntity is activated due to several DataReceivedEvents of dataElements with swImplPolicy = queued, it is the responsibility of the RunnableEntity to dequeue all queued data.

- A minimumStartInterval will delay the activation of RunnableEntitys and BswSchedulableEntitys to prevent that a RunnableEntity or a BswSchedulableEntity is started more than once within the minimumStartInterval.

Each ExecutableEntity execution-instance has its own state machine. The full state machine is shown in Fig. 4.17.

Note on Figure 4.17: the debounce timer debounceTimer is an increasing timer. It is local to the ExecutableEntity execution-instance. The activation counter activations is a local integer to count the pending activations. The runnable debounce timer and the activation counter are like the whole state machine just concepts for the specification of the behavior, not for the implementation.

The pending activations are only counted for server runnables when RTE implements a call serialization of their invocation. In all other cases, RTE does not queue activations and the state machine for the activation of ExecutableEntity execution-instances simplifies as shown in Figure 4.18.

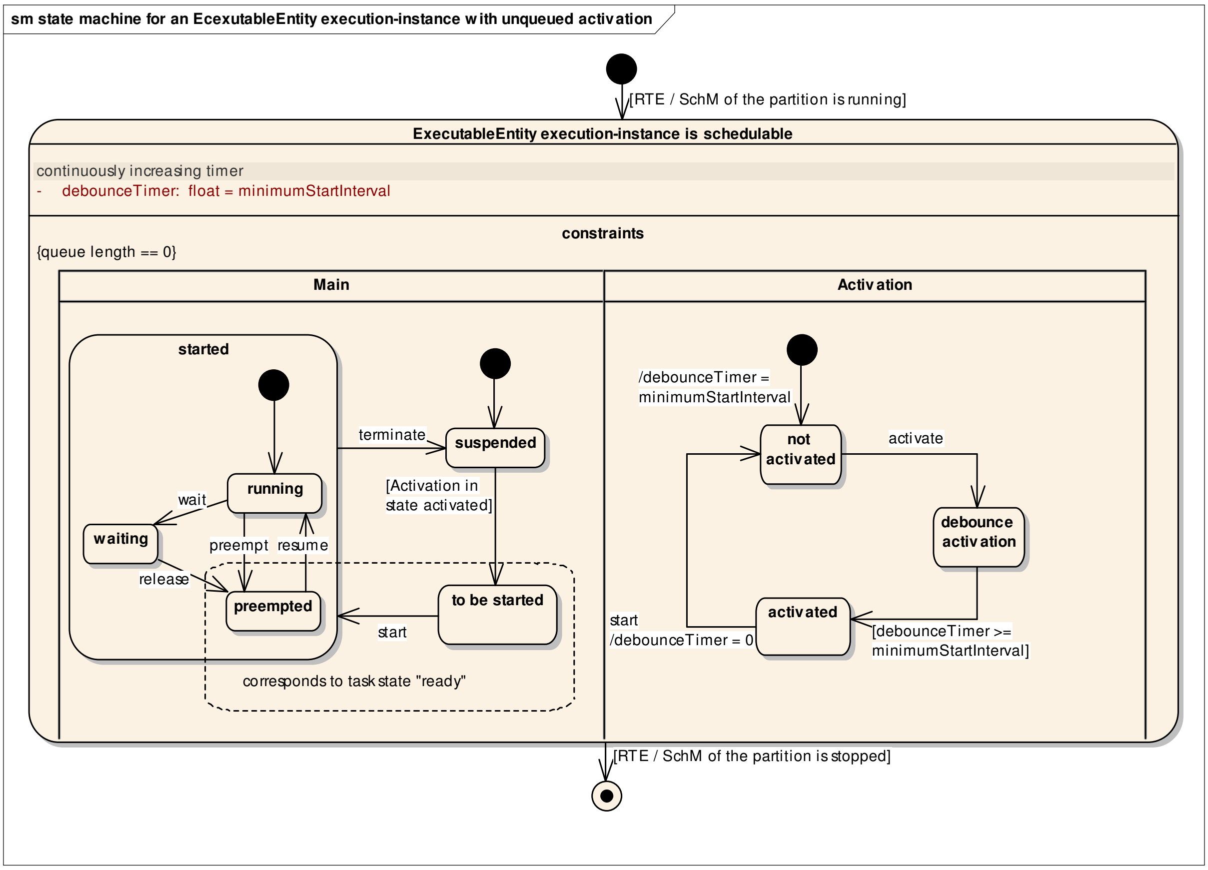


Figure 4.18: Statemachine of an unqueued execution-instance (not a server runnable)

If RTE implements an ExecutableEntity execution-instance by direct function call, as described in section 4.2.3.1, the simplified state machine is shown in Figure 4.21.

The state machine of an ExecutableEntity execution-instance is not identical to that of the task containing the ExecutableEntity execution-instance, but there are dependencies between them. E.g., the ExecutableEntity execution-instance can only be ‘running’ when the corresponding task is ‘running’.

Table 4.1 describes all ExecutableEntity execution-instance states in detail. The ExecutableEntity execution-instance state machine is split in two threads. The Main states describe the real state of the ExecutableEntity execution-instance and the transitions between a suspended and a running ExecutableEntity execution-instance, while the supporting Activation states describe the state of the pending activations by RTEEvents or BswEvents.

ExecutableEntity execution-instance state	**description**

ExecutableEntity execution-instance is schedulable	This super state describes the life time of the state machine. Only when RTE or the SchM that runs the ExecutableEntity execution-instance is started in the corresponding partition, this state machine is active.
ExecutableEntity execution-instance Main states	
suspended	The ExecutableEntity execution-instance is not started and there is no pending request to start the ExecutableEntity execution-instance.
to be started	The ExecutableEntity execution-instance is activated but not yet started. Entering the to be started state, usually implies the activation of a task that starts the ExecutableEntity execution-instance. The ExecutableEntity execution-instance stays in the 'to be started' state, when the task is already running until the gluecode of the task actually calls the function implementing the ExecutableEntity.
running	The function, implementing the ExecutableEntity code is being executed. The task that contains the ExecutableEntity execution-instance is running.
waiting	A task containing the ExecutableEntity execution-instance is waiting at a WaitPoint within the ExecutableEntity.
preempted	A task containing the ExecutableEntity execution-instance is preempted from executing the function that implements the ExecutableEntity.
started	'started' is the super state of 'running', 'waiting' and 'preempted' between start and termination of the ExecutableEntity execution-instance.
ExecutableEntity execution-instance Activation states	
not activated	No RTEEvent / BswEvent requires the activation of the ExecutableEntity execution-instance.
debounce activation	One or more RTEEvents with a startOnEvent relation to the ExecutableEntity execution-instance have occurred [2], but the debounce timer has not yet exceeded the minimumStartInterval. The activation will automatically advance to activated, when the debounce timer reaches the minimumStartInterval.

[2]Note that, e.g., the same OperationInvokedEvent may lead to the activation of different ExecutableEntity execution-instances, depending on the client that caused the event.

activated	One or more RTEEvents or BswEvents with a startOnEvent relation to the ExecutableEntity have occurred, and the debounce timer has exceeded the minimumStartInterval. While the activated state is active, the Main state of the ExecutableEntity execution-instance automatically advances from the suspended to the 'to be started' state. For a server runnable where RTE implements a serialization of server calls, an activation counter counts the number of activations. When the ExecutableEntity execution-instance starts, the activation counter will be decremented. When there is still a pending activation, the Activation state will turn to debounce activation and otherwise to no activation.

Table 4.1: States defined for each ExecutableEntity execution-instance.

Note: For tasks, the equivalent state machine does not distinguish between preempted and to be started. They are subsumed as 'ready'.

ExecutableEntity execution-instance transition	**description of event and actions**
initial transition to 'ExecutableEntity execution-instance is schedulable'	RTE or the SchM that runs the ExecutableEntity execution-instance is being started in the corresponding partition.
termination transition from 'ExecutableEntity execution-instance is schedulable'	RTE or the SchM that runs the ExecutableEntity execution-instance gets stopped in the corresponding partition.
transitions to ExecutableEntity execution-instance Main states	
initial transition to suspended	the suspended state is the initial state of the ExecutableEntity execution-instance Main states.
from started to suspended	The ExecutableEntity execution-instance has run to completion.
from suspended to 'to be started'	This transition is automatically executed, while the Activation state is 'activated'.
from 'to be started' to running	The function implementing the ExecutableEntity is called from the context of this execution-instance.
from preempted to running	A task that is preempted from executing the ExecutableEntity execution-instance changes state from preempted to running.
from running to waiting	The runnable enters a WaitPoint.

from waiting to preempted	The task that contains a runnable waiting at a wait point changes from waiting to preempted.
from running to preempted	A task containing the `ExecutableEntity` execution-instance gets preempted from executing the function that implements the `ExecutableEntity`.
transitions to `ExecutableEntity` execution-instance Activation states	
initial transition to 'not activated'	The 'not activated' state is the initial state of the `ExecutableEntity` execution-instance Activation states. The debounce timer is set to the `minimumStartInterval` value, to prevent a delay for the first activation of the `ExecutableEntity` execution-instance.
from activated to 'not activated'	The function implementing the `ExecutableEntity` is called from the context of this execution-instance and no further activations are pending. The debounce timer is reset to 0.
from 'not activated' to 'debounce activation'	The occurrence of an `RTEEvent` or `BswEvent` requires the activation of the `ExecutableEntity` execution-instance. A local activation counter is set to 1. If no `minimumStartInterval` is configured, or the debounce timer has already exceeded the `minimumStartInterval`, the 'debounce activation' state will be omitted and the transition leads directly to the activated state.
from activated to 'debounce activation'	The function implementing the `ExecutableEntity` is called from the context of this execution-instance (start), and another activation is pending (only for `server runnable`). The activation counter is decremented and the debounce timer reset to 0. If no `minimumStartInterval` is configured, the 'debounce activation' state will be omitted and the transition returns directly at the activated state.
from 'debounce activation' to 'debounce activation'	If RTE implements server call serialization for a `server runnable`, and an `OperationInvokedEvent` occurs for the server runnable. The activation counter is incremented (at most to the queue length).
from 'debounce activation' to activated	The debounce timer is expired, `debounce timer > minimumStartInterval`.

from activated to activated	If RTE implements server call serialization for a server runnable, and an OperationInvokedEvent occurs for the server runnable. The activation counter is incremented (at most to the queue length).

Table 4.2: States defined for each ExecutableEntity execution-instance.

[SWS_Rte_02697] ⌈The activation of ExecutableEntity execution-instances shall behave as described by the state machine in Fig. 4.17, Table 4.1, and Table 4.2. ⌋*(SRS_Rte_00072, SRS_Rte_00160, SRS_Rte_00133, SRS_Rte_00211, SRS_Rte_00214, SRS_Rte_00217, SRS_Rte_00219)*

The RTE will not activate, start or release ExecutableEntity execution-instances of a terminated or restarting partition (see [SWS_Rte_07604]), or when RTE is stopped in that partition (see [SWS_Rte_02538]).

The following examples in Fig. 4.19 and Fig. 4.20 show the different timing situations of the ExecutableEntity execution-instances with or without a minimumStartInterval. The minimumStartInterval can reduce the number of activations by collecting more activating RTEEvents / BswEvents within that interval. No activation will be lost. The activations are just delayed and combined to keep the minimumStartInterval. The started state of the ExecutableEntity execution-instance Main states and the activated state of the Activation states are shown in the figures. Each flash indicates the occurrence of an RTEEvent or BswEvent.

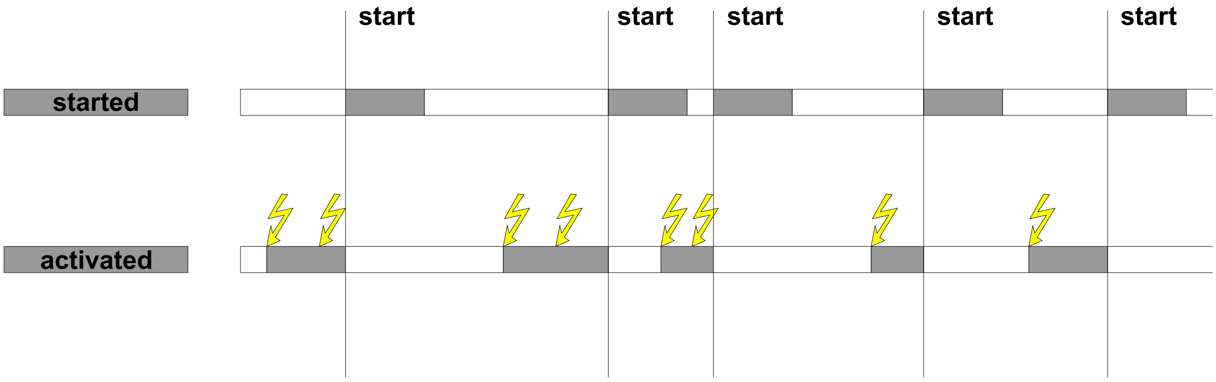


Figure 4.19: Activation of a ExecutableEntity execution-instance without minimumStartInterval

Figure 4.19 illustrates the activation of an ExecutableEntity execution-instance without minimumStartInterval. The execution-instance can only be activated once (does not apply for server runnables). The activation is not queued. The execution-instance can already be activated again when it is still started (see Figure 4.17).

With configuration of the RteEventToTaskMapping such activation can even be used for an immediately restart of the ExecutableEntity before other ExecutableEntitys which are mapped subsequently in the task are getting started.

[SWS_Rte_07061] ⌈ When the parameter RteImmediateRestart / RteBswImmediateRestart is TRUE the RTE shall immediately restart the ExecutableEntity after termination if the ExecutableEntity was activated by this RTEEvent / BswEvent while it was already started. ⌋*(SRS_Rte_00072)*

This can be utilized to spread a long-lasting calculation in several smaller slices with the aim to reduce the maximum blocking time of Tasks in a Cooperative Environment. Typically between each iteration one Schedule Point has to be placed and the number of iteration might depend on operating conditions of the ECU. Further on in a calculation chain the long-lasting calculation shall be completed before consecutive ExecutableEntitys are called.

Example 4.3

Example of RunnableEntity code:

```
LongLastingRunnable()
{
   /* the very long calculation */
   if(!finished)
   {
      /* further call is required to complete the calculation*/
      Rte_IrTrigger_LongLastingCalculation_ProceedCalculation();
   }
}
```

Therefore the ExecutableEntity with a long lasting calculation issues a trigger as long as the calculation is not finished. These trigger activates the ExecutableEntity again. The first activation of the ExecutableEntity might be triggered by another RTEEvent / BswEvent.

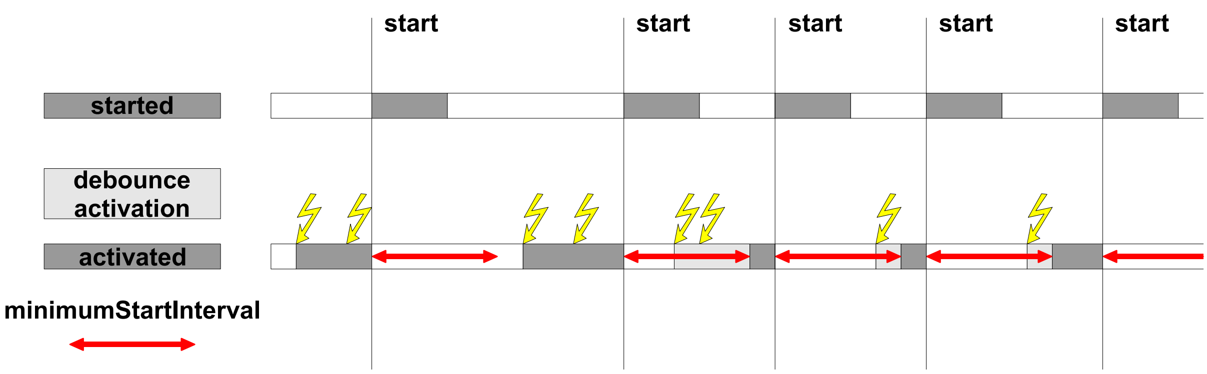


Figure 4.20: Activation of an ExecutableEntity with a minimumStartInterval

Figure 4.20 illustrates the activation of an ExecutableEntity with a minimumStartInterval. (Here no execution-instances have to be distinguished, there is only one.) The red arrows in this figure indicate the minimumStartInterval after each start of the ExecutableEntity. An RTEEvent or BswEventwithin this minimumStartInterval leads to the debounce activation state. When the minimumStartInterval ends, the debounce activation state changes to the activated state.

When a data received event activates a runnable when it is still running, it might be that the data is already dequeued during the current execution of the runnable. Still, the runnable will be started again. So, it is possible that a runnable that is activated by a data received event finds an empty receive queue.

4.2.3.1 Activation by direct function call

In many cases, ExecutableEntity execution-instances can be implemented by RTE by a direct function call if allowed by the canBeInvokedConcurrently. In these cases, the activation and start of the ExecutableEntity execution-instance collapse to one event. The states 'to be started', 'debounce activation', and 'activated' are passed immediately.

Obviously, debounce activation is not possible (see meta model restriction [SWS_Rte_02733]).

There is one ExecutableEntity execution-instance per call point, trigger point, mode switch point, etc.. The state chart simplifies as shown in Figure 4.21.

A triggered ExecutableEntity is activated at least by one ExternalTriggerOccurredEvent or InternalTriggerOccurredEvent. In some cases, the *Trigger Event Communication* or the *Inter Runnable Triggering* is implemented by RTE generator as a direct function call of the triggered ExecutableEntity by the triggering ExecutableEntity.

An OnEntry ExecutableEntity, OnTransition ExecutableEntity, OnExit ExecutableEntity or a mode switch acknowledge ExecutableEntity might be executed in the context of the Rte_Switch API if an asynchronous mode switch procedure is implemented.

A server runnable is exclusively activated by OperationInvokedEvents and implements the server in client server communication. In some cases, the client server communication is implemented by RTE as a direct function call of the server by the client.

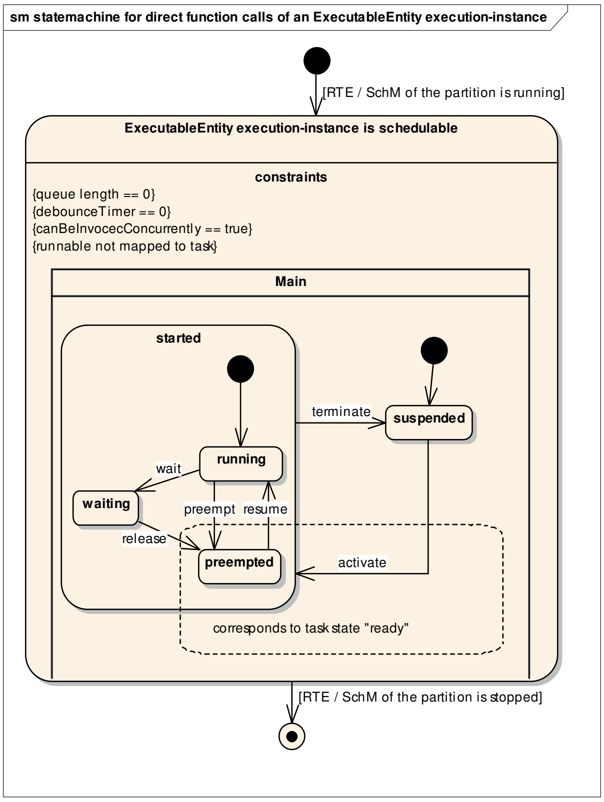


Figure 4.21: State machine of an ExecutableEntity execution-instance that is implemented by direct function calls.

4.2.3.2 Activation Offset for RunnableEntitys and BswSchedulableEntitys

In order to allow optimizations (smooth cpu load, mapping of RunnableEntitys and BswSchedulableEntitys with different periods in the same task to avoid data sharing, etc.), the RTE has to handle the activation offset information from a task shared reference point only for time trigger RunnableEntitys and BswSchedulableEntitys. The maximum period of a task can be calculated automatically as the greatest common divisor (GCD) of all runnables period and offset.It is assumed that the runnables worst case execution is less than the GCD. In case of the worst case execution is greater than the GCD, the behavior becomes undefined.

[SWS_Rte_07000] ⌈ The RTE shall respect the configured activation offset of RunnableEntitys mapped within one OS task. ⌋*(SRS_Rte_00161)*

[SWS_Rte_07520] ⌈ The *Basic Software Scheduler* shall respect the configured activation offset of BswSchedulableEntitys mapped within one OS task. ⌋*(SRS_Rte_00212)*

[constr_9010] Worst case execution time shall be less than the GCD ⌈ The RunnableEntitys or BswSchedulableEntitys worst case execution time shall be less than the GCD of all BswSchedulableEntitys and RunnableEntitys period and offset in activation offset context for RunnableEntitys and BswSchedulableEntitys. ⌋

Note: The following examples are showing RunnableEntitys only. Nevertheless it is applicable for BswSchedulableEntitys or a mixture of RunnableEntitys and BswSchedulableEntitys as well.

Example 1:
This example describes 3 runnables mapped in one task with an activation offset defined for each runnables.

Runnable	Period	Activation Offset
R1	100ms	20ms
R2	100ms	60ms
R3	100ms	100ms

Table 4.3: Runnables timings

The runnables R1, R2 and R3 are mapped in the task T1 at 20 ms which is the GCD of all runnables period and activation offset.

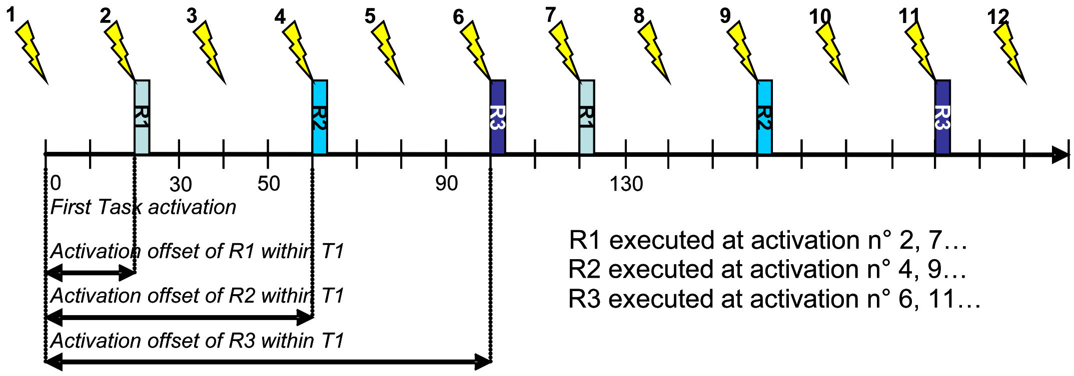


Figure 4.22: Example of activation offset for runnables

Example 2:
This example describes 4 runnables mapped in one task with an activation offset and position in task defined for each runnables.

Runnable	Period	Position in task	Activation Offset
R1	50ms	1	0ms
R2	100ms	2	0ms
R3	100ms	3	70ms
R4	50ms	4	20ms

Table 4.4: Runnables timings with position in task

The runnables R1, R2, R3 and R4 are mapped in the task T1 at 10 ms which is the GCD of all runnables period and activation offset.

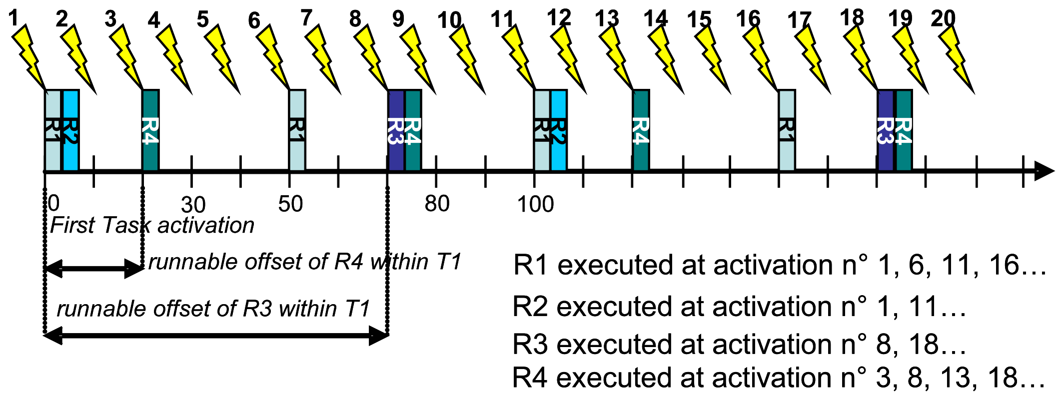


Figure 4.23: Example of activation offset for runnables with position in task

4.2.3.3 Provide activating RTE event

It is possible to define the activation of one runnable entity by several RTE events. But when the runnable entity is invoked by the RTE it is shall be possible to query which of the RTE events actually triggered the execution of this runnable entity run.

Contract Phase:

The provide activating event feature is enabled if the runnable entity has at least one `activationReason` defined.

[SWS_Rte_08051] ⌈ If the provide activating event feature is enabled, the RTE generator in contract phase shall generate the runnable entity signature according to [SWS_Rte_01126] and [SWS_Rte_08071]. ⌋*(SRS_Rte_00238)*

[SWS_Rte_08052] ⌈ If the provide activating event feature is enabled, the RTE generator in contract phase shall generate the type `Rte_ActivatingEvent_<name> (activation vector)`, where `<name>` is the `symbol` describing the runnable entity's entry point, to store the activation bits. Based on the highest value of `ExecutableEntityActivationReason.bitPosition` for this runnable entity the type shall be either `uint8`, `uint16`, or `uint32` so that the highest value of `bitPosition` fits into the data type. ⌋*(SRS_Rte_00238)*

Note that it is considered an invalid configuration if `ExecutableEntityActivationReason.bitPosition` has a value higher than 31 (see [constr_1226] in software component template [2])

[SWS_Rte_08053] ⌈ If the provide activating RTE event feature is enabled, the RTE generator in contract phase shall generate for each `ExecutableEntityActivationReason` of one executable entity a definition to provide the specific bit position in the `Rte_ActivatingEvent_<name>` data type:

```
#define Rte_ActivatingEvent_<name>_<activation> xxU
```

The value of xx is defined by the `bitPosition` xx = 2^bitPosition. ⌋*(SRS_Rte_00238)*

Example: runnable entity symbol = "greek" and has 3 `ExecutableEntityActivationReason`s aggregated. Those are referenced by 4 RTE events:

- RTEEvent: `"alpha"` symbol: `aleph`
- RTEEvent: `"beta"` symbol: `beth`
- RTEEvent: `"gamma"` symbol: `gimel`
- RTEEvent: `"delta"` symbol: `gimel`

This will result in a `unit8 Rte_ActivatingEvent_<name>` data type: `typedef uint8 Rte_ActivatingEvent_greek` and 3 definitions:

- `#define Rte_ActivatingEvent_greek_aleph 01U`
- `#define Rte_ActivatingEvent_greek_beth 02U`

- `#define Rte_ActivatingEvent_greek_gimel  04U`

Generation Phase:

[SWS_Rte_08054] ⌈ If the provide activating RTE event feature is enabled, the RTE shall collect the activating RTE events, which have the `activationReasonRepresentation` reference defined, in the context of the OS task the runnable entity is mapped to in an activation vector at the corresponding bit position as defined in [SWS_Rte_08053]. ⌋*(SRS_Rte_00238)*

[SWS_Rte_08055] ⌈ If the provide activating RTE event feature is enabled, the RTE shall provide the collected activating RTE events (activation vector) to the runnable entity API when the runnable entity is "started". The activation vector shall be reset immediately after it has been provided. ⌋*(SRS_Rte_00238)*

Since it is possible that there is a time gap between the activation and the execution (start) of a runnable entity the subsequent activations are summed up and provided with the start of the runnable entity.

Activations during the execution of a runnable entity are collected for the next start of that runnable entity.

4.2.4 Interrupt decoupling and notifications

4.2.4.1 Basic notification principles

Several BSW modules exist which contain functionality which is not directly activated, triggered or called by AUTOSAR software-components but by other circumstances, like digital input port level changes, complex driver actions, CAN signal reception, etc. In most cases interrupts are a result of those circumstances. For a definition of interrupts, see the VFB [1].

Several of these BSW functionalities create situations, signalled by an interrupt, when AUTOSAR SW-Cs have to be involved. To inform AUTOSAR software components of those situations, runnables in AUTOSAR software components are activated by notifications. So interrupts that occur in the basic software have to be transformed into notifications of the AUTOSAR software components. Such a transformation has to take place at RTE level **at the latest**! Which interrupt is connected to which notification is decided either during system configuration/generation time or as part of the design of Complex Device Drivers or the Microcontroller Abstraction Layer.

This means that runnables in AUTOSAR SW-Cs have to be activated or "waiting" cat2 runnables in extended tasks have to be set to "ready to run" again. In addition some event specific data may have to be passed.

There are two different mechanisms to implement these notifications, depending on the kind of BSW interfaces.

1. **BSW with Standardized interface**. Used with COM and OS.
   Basic-SW modules with Standardized interfaces cannot create RTEEvents. So another mechanism must be chosen: "**callbacks**"
   The typical callback realization in a C/C++ environment is a function call.
2. **BSW with AUTOSAR interface**: Used in all the other BSW modules.
   Basic-SW modules with AUTOSAR-Interfaces have their interface specified in an AUTOSAR BSW description XML file which contains signal specifications according to the AUTOSAR specification. The BSW modules can employ RTE API calls like Rte_Send – see *5.6.5*). `RTEEvent`s may be connected with the RTE API calls, so realizing AUTOSAR SW-C activation.

Note that an AUTOSAR software component can send a notification to another AUTOSAR software component or a BSW module only via an AUTOSAR interface.

4.2.4.2 Interrupts

The AUTOSAR concept as stated in the VFB specification [1] does not allow AUTOSAR software components to run in interrupt context. Only the Microcontroller Abstraction Layer, Complex Device Drivers and the OS are allowed to directly interact with interrupts and implement interrupt service routines (see Requirement [SRS_BSW_00164]. This ensures hardware independence and determinism.

If AUTOSAR software components were allowed to run in interrupt context, one AUTOSAR software component could block the entire system schedule for an unacceptably long period of time. But the main reason is that AUTOSAR software components are supposed to be independent of the underlying hardware so that exchangeability between ECUs can be ensured. The schedule of an ECU is more predictable and better testable if the timing effects of interrupts are restricted to the basic software of that ECU.

Furthermore, AUTOSAR software components are not allowed to explicitly block interrupts as a means to ensure data consistency. They have to use RTE functions for this purpose instead, see Section 4.2.5.

4.2.4.3 Decoupling interrupts on RTE level

Runnables in AUTOSAR SW-Cs may be running as a consequence of an interrupt but **not** in interrupt context, which means not within an interrupt service routine! Between the interrupt service routine and an AUTOSAR SW-C activation there must always be a decoupling instance. AUTOSAR SW-C runnables are only executed in the context of tasks.

The decoupling instance is latest in the RTE. For the RTE there are several options to realize the decoupling of interrupts. Which option is the best depends on the configuration and implementation of the RTE, so only examples are given here.

Example 1:

Situation:

- An interrupt routine calls an RTE callback function

Intention:

- Start a runnable

RTE job:

- RTE starts a task containing the runnable activation code by using the `ActivateTask`()" OS service call.
- Other more sophisticated solutions are possible, e.g. if the task containing the runnable is activated periodically.

Example 2:

Situation:

- An interrupt routine calls an RTE callback function

Intention:

- Make a runnable wake up from a wait point

RTE job:

- RTE sets an OS event

These scenarios described in the examples above not only hold for RTE callback functions but for other RTE API functions as well.

[SWS_Rte_03600] ⌈ The RTE shall prevent runnable entities of AUTOSAR software-components to run in interrupt context. ⌋*(SRS_Rte_00099)*

4.2.4.4 RTE and interrupt categories

Since category 1 interrupts are not under OS control the RTE has absolutely no possibility to influence their execution behavior. So no category 1 interrupt is allowed to reach RTE. This is different for interrupt of category 2.

[SWS_Rte_03594] ⌈ The RTE Generator shall reject the configuration if a `SwcBswRunnableMapping` associates a `BswInterruptEntity` with a `RunnableEntity` and the attribute `interruptCategory` of the `BswInterruptEntity` is equal to `cat 1`. ⌋*(SRS_Rte_00018, SRS_Rte_00099)*

[constr_9012] Category 1 interrupts shall not access the RTE. ⌈ Category 1 interrupts shall not access the RTE. ⌋

4.2.4.5 RTE and Basic Software Scheduler and BswExecutionContext

The RTE and *Basic Software Scheduler* do support the invocation triggered ExecutableEntity via direct function call in some special cases. Nevertheless it shall be prevented that an ExecutableEntity from a particular execution context calls a triggered ExecutableEntity wich requires an execution context with more permissions. The table 4.5 lists the supported combinations.

caller's BswExecutionContext[3]	callee's BswExecutionContext[3]				
	task	**interruptCat2**	**interruptCat1**	**hook**	**unspecified**
task	supported	supported	supported		supported
interruptCat2		supported	supported		supported
interruptCat1			supported		supported
hook					
unspecified	supported				supported

Table 4.5: Possible invocation of ExecutableEntitys by direct function call dependent from BswExecutionContext

For example (fourth column), the invocation of an ExecutableEntity with an interruptCat1 BswExecutionContext can be implemented with a direct function call if the BswExecutionContext of the caller BswModuleEntry is set to task, interruptCat2, or interruptCat1.

This applies to the invocation of a triggered ExecutableEntity by the SchM_Trigger, SchM_ActMain or Rte_Trigger APIs, or to the invocation of an OnEntry ExecutableEntity, OnTransition ExecutableEntity, OnExit ExecutableEntity or mode switch acknowledge ExecutableEntity by the SchM_Switch or Rte_Switch APIs.

4.2.4.5.1 Interrupt decoupling for COM

COM callbacks are used to inform the RTE about something that happened independently of any RTE action. This is often interrupt driven, e.g. when a data item has been received from another ECU or when a S/R transmission is completed.
It is the RTE's job e.g. to create RTEEvents from the interrupt.

[SWS_Rte_03530] ⌈ The RTE shall provide callback functions to allow COM to signal COM events to the RTE. ⌋*(SRS_Rte_00072, SRS_Rte_00099)*

[SWS_Rte_03531] ⌈ The RTE shall support runnable activation by COM callbacks. ⌋*(SRS_Rte_00072, SRS_Rte_00099)*

[SWS_Rte_03532] ⌈ The RTE shall support category 2 runnables to wake up from a wait point as a result of COM callbacks. ⌋*(SRS_Rte_00072, SRS_Rte_00099)*

[3] The execution context of a RunnableEntity is considered as task

See RTE callback API in chapter 5.9.

4.2.5 Data Consistency

4.2.5.1 General

Concurrent accesses to shared data memory can cause data inconsistencies. In general this must be taken into account when several code entities accessing the same data memory are running in different contexts - in other words when systems using parallel (multicore) or concurrent (singlecore) execution of code are designed. More general: Whenever task context-switches occur and data is shared between tasks, data consistency is an issue.

AUTOSAR systems use operating systems according to the AUTOSAR-OS specification which is derived from the OSEK-OS specification. The Autosar OS specification defines a priority based scheduling to allow event driven systems. This means that tasks with higher priority levels are able to interrupt (preempt) tasks with lower priority level.

The "lost update" example in Figure 4.24 illustrates the problem for concurrent read-modify-write accesses:

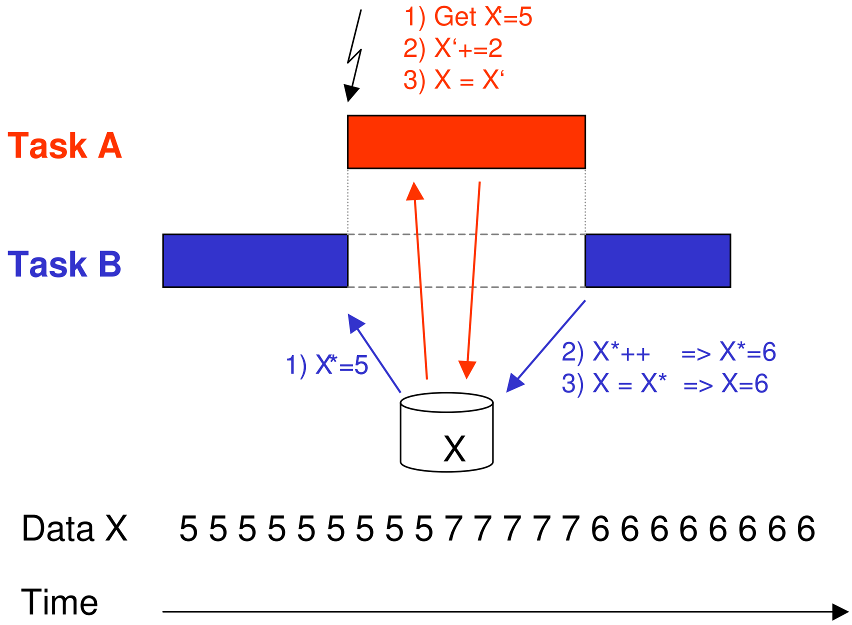


Figure 4.24: Data inconsistency example - lost update

There are two tasks. Task A has higher priority than task B. A increments the commonly accessed counter X by 2, B increments X by 1. So in both tasks there is a read (step1) – modify (step2) – write (step3) sequence. If there are no atomic accesses (fully completed read-modify-write accesses without interruption) the following can happen:

1. Assume X=5.
2. B makes read (step1) access to X and stores value 5 in an intermediate store (e.g. on stack or in a CPU register).
3. B cannot continue because it is preempted by A.
4. A does its read (step1) – modify (step2) – write (step3) sequence, which means that A reads the actual value of X, which is 5, increments it by 2 and writes the new value for X, which is 7. (X=5+2)
5. A is suspended again.
6. B continues where it has been preempted: with its modify (step2) and write (step3) job. This means that it takes the value 5 form its internal store, increments it by one to 6 and writes the value 6 to X (X=5+1).
7. B is suspended again.

The correct result after both Tasks A and B are completed should be X=8, but the update of X performed by task A has been lost.

4.2.5.2 Communication Patterns

In AUTOSAR systems the RTE has to take care that a lot of the communication is not corrupted by data consistency problems. RTE Generator has to apply suitable means if required.

The following communication mechanisms can be distinguished:

- Communication within one atomic AUTOSAR SW-C:
  Communication between Runnables of one atomic AUTOSAR SW-C running in different task contexts where communication between these Runnables takes place via commonly accessed data. If the need to support data consistency by the RTE exists, it must be specified by using the concepts of "ExclusiveAreas" or "InterRunnableVariables" only.
- Intra-partition communication between AUTOSAR SW-Cs:
  Sender/Receiver (S/R) communication between Runnables of different AUTOSAR SW-Cs using *implicit* or *explicit* data exchange can be realized by the RTE through commonly accessed RAM memory areas. Data consistency in Client/Server (C/S) communication can be put down to the same concepts as S/R communication. Data access collisions must be avoided. The RTE is responsible for guaranteeing data consistency.
- Inter-Partition communication
  The RTE has to guarantee data consistency. The different possibilities provided to the RTE for the communication between partitions are discussed in section 4.3.4.

- Intra-ECU communication between AUTOSAR SW-Cs and BSW modules with AUTOSAR interfaces:
  This is a special case of the above two.
- Inter ECU communication
  COM has to guarantee data consistency for communication between ECUs on complete path between the COM modules of different ECUs. The RTE on each ECU has to guarantee that no data inconsistency might occur when it invokes COM send respectively receive calls supplying respectively receiving data items which are concurrently accessed by application via RTE API call, especially when queueing is used since the queues are provided by the RTE and not by COM.

[SWS_Rte_03514] ⌈ The RTE has to guarantee data consistency for communication via AUTOSAR interfaces. ⌋*(SRS_Rte_00032)*

4.2.5.3 Concepts

In the AUTOSAR SW-C Template [2] chapter "Interaction between runnables within one component", the concepts of

1. ExclusiveAreas *(see section 4.2.5.5 below)*
2. InterRunnableVariables *(see section 4.2.5.6 below)*

are introduced to allow the user (SW-Designer) to specify where the RTE shall guarantee data consistency for AUTOSAR SW-C internal communication and execution circumstances. This is discussed in more detail in next sections.

Additionally exclusive areas are also available for *Basic Software Modules* to protect access to module internal data. See [9]. The exclusive areas for *Basic Software Modules* are handled by the *Basic Software Scheduler*.

The AUTOSAR SW-C template specification [2] also states that AUTOSAR SW-Cs may define `PerInstanceMemory` or `arTypedPerInstanceMemory`, allowing reservation of static (permanent) need of global RAM for the SW-C. Nothing is specified about the way Runnables might access this memory. RTE only provides a reference to this memory *(see section 5.6)* but doesn't guarantee data consistency for it.

The implementer of an AUTOSAR SW-C has to take care by himself that accesses to RAM reserved as PerInstanceMemory out of Runnables running in different task contexts don't cause data inconsistencies. On the other hand this provides more freedom in using the memory.

4.2.5.4 Mechanisms to guarantee data consistency

ExclusiveAreas and InterRunnableVariables are only mentioned in association with AUTOSAR SW-C internal communication. Nevertheless the data consistency mechanisms behind can be applied to communication between AUTOSAR SW-Cs or between AUTOSAR SW-Cs and BSW modules too. Everywhere where the RTE has to guarantee data consistency.

The data consistency guaranteeing mechanisms listed here are derived from AUTOSAR SW-C Template and from further discussions. There might be more (see section 4.3.4 for the mechanisms involved for `inter-partition` communication).
The RTE has the responsibility to apply such mechanisms if required. The details how to apply the mechanisms are left open to the RTE supplier.

Mechanisms:

- ***Sequential scheduling strategy***
  The activation code of Runnables is sequentially placed in one task so that no interference between them is possible because one Runnable is only activated after the termination of the other. Data consistency is guaranteed.

- ***Interrupt blocking strategy***
  Interrupt blocking can be an appropriate means if collision avoidance is required for a very short amount of time. This might be done by disabling respectively suspending all interrupts, Os interrupts only or - if hardware supports it - only of some interrupt levels. In general this mechanism must be applied with care because it might influence SW in tasks with higher priority too and the timing of the complete system.

- ***Usage of OS resources***
  Usage of OS resources. Advantage in comparison to Interrupt blocking strategy is that less SW parts with higher priority are blocked. Disadvantage is that implementation might consume more resources (code, runtime) due to the more sophisticated mechanism. Appropriateness of this mechanism may vary depending on the number of OSs/cores and/or the number of available resources.

- ***Task blocking strategy***
  Mutual task preemption is prohibited. This might be reached e.g. by assigning same priorities to affected tasks, by assigning same internal OS resource to affected tasks or by configuring the tasks to be non-preemptive. This mechanism may be inappropriate in multi-partitioned systems.

- ***Cooperative Runnable placement strategy***
  The principle is that tasks containing Runnables to be protected by "Cooperative Runnable placement strategy" are not allowed to preempt other tasks also containing Runnables to be protected by "Cooperative Runnable placement strategy", when one of the Runnables to protect is active - but are allowed between Runnable executions. The RTE's job is to create appropriate task bodies and

use OS services or other mechanisms to achieve the required behavior. This mechanism may be inappropriate in multi-partitioned systems.

To point out the difference to "Task blocking strategy":
In "Task blocking strategy" no task containing Runnables with access to the ExclusiveArea at all is allowed to preempt another task containing Runnables with access to same ExclusiveArea. In "Cooperative Runnable placement strategy" this task blocking mechanism is limited to tasks defined to be within same cooperative context.

Example to explain the cooperative mechanism:

- Runnables R2 and R3a are marked to be protected by cooperative mechanism.
- Runnables R1, R3b and R4 have no cooperative marking.
- R1 is activated in Task T1, R2 is activated in Task T2, R3a is activated in Task T3a, R3b is activated in Task T3b, R4 is activated in Task T4.
- Task priorities are: T4 $>$ T3a $>$ T2 $>$ T1, T3b has same priority as T3a

This setup results in this behavior:

- T4 can always preempt all other tasks (Higher prio than all others).
- T3b can preempt T2 (higher prio of T3b, no cooperative restriction)
- T3a cannot preempt T2 (Higher prio of T3a but same cooperative context). So data access of Runnable R2 to common data cannot interfere with data access by Runnable R3a. Nevertheless if both tasks T3a and T2 are ready to run, it's guaranteed that T3a is running first.
- T1 can never preempt one of the other tasks because of lowest assigned prio.

- ***Copy strategy***
Idea: The RTE creates copies of data items so that concurrent accesses in different task contexts cannot collide because some of the accesses are redirected to the copies.

  How it can work:

  - Application for ***read** conflicts*:
  For all readers with lower priority than the writer a *read copy* is provided.

    Example:
    There exist Runnable R1, Runnable R2, data item X and a copy data item X*. When Runnable R1 is running in higher priority task context than R2, and R1 is the only one writing X and R2 is reading X it is possible to guarantee data consistency by making a copy of data item X to variable X* before activation of R2 and redirecting write access from X to X* or the read

access from X to X* for R2.

– Application for ***write** conflicts*:
  If one or more data item receiver with a higher priority than the sender exist, a *write copy* for the sender is provided.

  Example:
  There exist Runnable R1, Runnable R2, data item X and copy data item X*. When Runnable R1 (running in lower priority task context than R2) is writing X and R2 is reading X, it is possible to guarantee data consistency by making a copy of data item X to data item X* before activation of R1 together with redirecting the write access from X to X* for R1 or the read access from X to X* for R2.

Usage of this copy mechanism may make sense if one or more of the following conditions hold:

– This copy mechanism can handle those cases when only one instance does the data write access.
– R2 is accessing X several times.
– More than one Runnable R2 has read (resp. write) access to X.
– To save runtime is more important than to save code and RAM.
– Additional RAM requirements to hold the copies is acceptable.

Further issues to be taken into account:

– AUTOSAR SW-Cs provided as source code and AUTOSAR SW-Cs provided as object code may or have to be handled in different ways. The redirecting mechanism for source code could use macros for C and C++ very efficiently whereas object-code AUTOSAR SW-Cs most likely are forced to use references.

Note that the copy strategy is used to guarantee data consistency for implicit sender-receiver communication (`VariableAccess`es in the `dataReadAccess` or `dataWriteAccess` role) and for AUTOSAR SW-C internal communication using InterRunnableVariables with implicit behavior.

4.2.5.5 Exclusive Areas

The concept of `ExclusiveArea` is more a working model. It's not a concrete implementation approach, although concrete possible mechanisms are listed in AUTOSAR SW-C template specification [2].

Focus of the ExclusiveArea concept is to block potential concurrent accesses to get data consistency. ExclusiveAreas implement critical section

*ExclusiveArea*s are associated with RunnableEntitys. The RTE is forced to guarantee data consistency when the RunnableEntity runs in an ExclusiveArea. A RunnableEntity can run inside one or several ExclusiveAreas completely or can enter one or several ExclusiveAreas during their execution for one or several times .

- If an AUTOSAR SW-C requests the RTE to look for data consistency for it's internally used data (for a part of it or the complete one) using the ExclusiveArea concept, the SW designer can use the API calls "Rte_Enter()" in 5.6.27 and "Rte_Exit()" in 5.6.28 to specify where he wants to have the protection by RTE applied.
  "Rte_Enter()" defines the begin and "Rte_Exit()" defines the end of the code sequence containing data accesses the RTE has to guarantee data consistency for.
- If the SW designer wants to have the mutual exclusion for complete RunnableEntitys he can specify this by using the *ExclusiveArea* in the role "runsInsideExclusiveArea" in the AUTOSAR SW-C description.

In principle the ExclusiveArea concept can handle the access to single data items as well as the access to several data items realized by a group of instructions. It also doesn't matter if one Runnable is completely running in an ExclusiveArea and another Runnable only temporarily enters the same ExclusiveArea. The RTE has to guarantee data consistency.

[SWS_Rte_03500] ⌈ The RTE has to guarantee data consistency for arbitrary accesses to data items accessed by Runnables marked with the same ExclusiveArea. ⌋*(SRS_Rte_00032, SRS_Rte_00046)*

[SWS_Rte_03515] ⌈ RTE has to provide an API enabling the SW-Cs to access and leave ExclusiveAreas. ⌋*(SRS_Rte_00046)*

If Runnables accessing same ExclusiveArea are assigned to be executing in different task contexts, the RTE can apply suitable mechanisms, e.g. task blocking, to guarantee data consistency for data accesses in the common ExclusiveArea. However, specials attributes can be set that require certain data consistency mechanisms in which case the RTE generator is forced to apply the selected mechanism.

The *Basic Software Scheduler* provides ExclusiveAreas for the *Basic Software Module*s. *Basic Software Module*s have to use the API calls SchM_Enter()" in 6.5.1 and SchM_Exit()" in 6.5.2 to specify where the protection by Basic Software Scheduler has to be applied.

[SWS_Rte_07522] ⌈ The *Basic Software Scheduler* has to guarantee data consistency for arbitrary accesses to data items accessed by BswModuleEntitys marked with the same ExclusiveArea. ⌋*(SRS_Rte_00222, SRS_Rte_00046)*

[SWS_Rte_07523] ⌈ *Basic Software Scheduler* has to provide an API enabling the *Basic Software Module* to access and leave ExclusiveAreas. ⌋*(SRS_Rte_00222, SRS_Rte_00046)*

It is not supported, that a BswModuleEntity which is not a BswSchedulableEntity uses an ExclusiveArea in the role runsInsideExclusiveArea This is not possible, because such BswSchedulableEntity might be called directly by other *Basic Software Modules* and therefore the *Basic Software Scheduler* is not able to enter and exit the ExclusiveArea automatically.

[SWS_Rte_07524] ⌈ The RTE generator shall reject a configuration where a BswModuleEntity which is not a BswSchedulableEntity uses an ExclusiveArea in the role runsInsideExclusiveArea. ⌋*(SRS_Rte_00222, SRS_Rte_00046, SRS_Rte_00018)*

4.2.5.5.1 Assignment of data consistency mechanisms

The data consistency mechanism that has to be applied to anExclusiveArea might be domain, ECU or even project specific. The decision which mechanism has to be applied by RTE / *Basic Software Scheduler* is taken during ECU integration by setting the ExclusiveArea configuration parameter RteExclusiveAreaImplMechanism. This parameter is an input for RTE generator.

As stated in section 4.2.5.4 there might be more mechanisms to realize ExclusiveAreas as mentioned in this specification. So RTE implementations might provide other mechanisms in plus by a vendor specific solutions. This allows further optimizations.

Actually following values for configuration parameter RteExclusiveAreaImplMechanism must be supported:

- ALL_INTERRUPT_BLOCKING
  This value requests enabling and disabling of all Interrupts and is based on the *Interrupt blocking strategy*.
- OS_INTERRUPT_BLOCKING
  This value requests enabling and disabling of Os Interrupts and is based on the *Interrupt blocking strategy*.
- OS_RESOURCE
  This value requests to apply the *Usage of OS resources* mechanism.
- COOPERATIVE_RUNNABLE_PLACEMENT
  This value requires to apply the *Cooperative Runnable Placement Strategy*.

The strategies / mechanisms are described in general in section 4.2.5.4.

[SWS_Rte_03504] ⌈ If the configuration parameter RteExclusiveAreaImplMechanism of an ExclusiveArea is set to value ALL_INTERRUPT_BLOCKING the RTE generator shall use the mechanism of *Interrupt blocking* (blocking all interrupts) to guarantee data consistency if data inconsistency could occur. ⌋*(SRS_Rte_00032)*

[SWS_Rte_05164] ⌈ If the configuration parameter RteExclusiveAreaImplMechanism of an ExclusiveArea is set to value OS_INTERRUPT_BLOCKING the RTE generator shall use the mechanism of *Interrupt blocking* (blocking Os interrupts only) to guarantee data consistency if data inconsistency could occur. ⌋*(SRS_Rte_00032)*

[SWS_Rte_03595] ⌈ If the configuration parameter RteExclusiveAreaImplMechanism of an ExclusiveArea is set to value OS_RESOURCE the RTE generator shall use OS resources to guarantee data consistency if data inconsistency could occur. ⌋*(SRS_Rte_00032)*

The requirements above have the limitation "if data inconsistency could occur" because it makes no sense to apply a data consistency mechanism if no potential data inconsistency can occur. This can be relevant if e.g. the "Sequential scheduling strategy" (described in section 4.2.5.4) still has solved the item by the ECU integrator defining an appropriate runnable-to-task mapping.

[SWS_Rte_03503] ⌈ If the configuration parameter RteExclusiveAreaImplMechanism of an ExclusiveArea is set to value COOPERATIVE_RUNNABLE_PLACEMENT the RTE generator shall generate code according the *Cooperative Runnable Placement Strategy* to guarantee data consistency. ⌋*(SRS_Rte_00032)*

Since the decision to select the *Cooperative Runnable Placement Strategy* to prohibit data access conflicts affects the behavior of several tasks and potentially many ExclusiveAreas the RTE generator is not allowed to override the decision.

In a SWC code, it is not allowed to use WaitPoints inside an ExclusiveArea: The RTE generator might use OSEK services to implement ExclusiveAreas and waiting for an OS event is not allowed when an OSEK resource has been taken for example. For RunnableEntityEntersExclusiveArea, the RTE generator cannot check if WaitPoints are inside an ExclusiveArea. Therefore, it is the responsibility of the SWC Code writer to ensure that no WaitPoints are used inside an exclusive area. But for RunnableEntitys running inside a ExclusiveArea, the RTE generator is able to do the following check.

[SWS_Rte_07005] ⌈ The RTE generator shall reject a configuration with a WaitPoint applied to a RunnableEntity which is using the ExclusiveArea in the role runsInsideExclusiveArea ⌋*(SRS_Rte_00032, SRS_Rte_00018)*

4.2.5.6 InterRunnableVariables

A non-composite AUTOSAR SW-C can reserve InterRunnableVariables which can be accessed by the Runnables of this one AUTOSAR SW-C (also see section 4.3.3.1).

Read and write accesses are possible. There is a separate set of those variables per AUTOSAR SW-C instance.

Again the RTE has to guarantee data consistency. Appropriate means will depend on Runnable placement decisions which are taken during ECU configuration.

[SWS_Rte_03516] ⌈ The RTE has to guarantee data consistency for communication between Runnables of one AUTOSAR software-component instance using the same InterRunnableVariable. ⌋*(SRS_Rte_00142, SRS_Rte_00032)*

Next the two kinds of InterRunnableVariables are treated:

1. InterRunnableVariables with **implicit** behavior

   (`implicitInterRunnableVariable`)

2. InterRunnableVariables with **explicit** behavior

   (`explicitInterRunnableVariable`)

4.2.5.6.1 InterRunnableVariables with implicit behavior

In applications with very high SW-C communication needs and much real time constraints (like in powertrain domain) the usage of a copy mechanism to get data consistency might be a good choice because during `RunnableEntity` execution no data consistency overhead in form of concurrent access blocking code and runtime during its execution exists - independent of the number of data item accesses.
Costs are code overhead in the `RunnableEntity` prologue and epilogue which is often be minimal compared to other solutions. Additional RAM need for the copies comes in plus.

When *InterRunnableVariables with implicit behavior* are used the RTE is required to make the data available to the Runnable using the `semantics of a copy` operation but is not necessarily required to use a unique copy for each `RunnableEntity`.

Focus of *InterRunnableVariable with implicit behavior* is to avoid concurrent accesses by redirecting second, third, .. accesses to data item copies.

[SWS_Rte_03517] ⌈ The RTE shall guarantee data consistency for *InterRunnableVariables with implicit behavior* by avoiding concurrent accesses to data items specified by `implicitInterRunnableVariable` using one or more copies and redirecting accesses to the copies.
⌋*(SRS_Rte_00142, SRS_Rte_00032)*

Compared with Sender/Receiver communication

- Like with `VariableAccess`es in the `dataReadAccess` and `dataWriteAccess` roles, the Runnable IN data is stable during Runnable execution, which means that during an Runnable execution several read accesses to an `implicitInterRunnableVariable` always deliver the same data item value.

- Like with VariableAccesses in the dataReadAccess and dataWriteAccess roles, the Runnable OUT data is forwarded to other Runnables not before Runnable execution has terminated, which means that during an Runnable execution write accesses to implicitInterRunnableVariable are not visible to other Runnables.

This behavior requires that Runnable execution terminates.

[SWS_Rte_03582] ⌈ The value of several read accesses to implicitInterRunnableVariable during a RunnableEntity execution shall only change for write accesses performed within this RunnableEntity to the implicitInterRunnableVariable ⌋*(SRS_Rte_00142)*

[SWS_Rte_03583] ⌈ Several write accesses to implicitInterRunnableVariable during a RunnableEntity execution shall result in only one update of the implicitInterRunnableVariable content visible to other RunnableEntitys with the last written value.
⌋*(SRS_Rte_00142)*

[SWS_Rte_03584] ⌈ The update of implicitInterRunnableVariable done during a RunnableEntity execution shall be made available to other RunnableEntitys after the RunnableEntity execution has terminated.
⌋*(SRS_Rte_00142)*

[SWS_Rte_07022] ⌈ If a RunnableEntity has both read and write access to an implicitInterRunnableVariable the result of the write access shall be immediately visible to subsequent read accesses from within the same runnable entity.
⌋*(SRS_Rte_00142)*

The usage of implicitInterRunnableVariables is permitted for all categories of runnable entities. For runnable entities of category 2, the behavior is guaranteed only if it has a finite execution time. A category 2 runnable that runs forever will not have its data updated.

For API of implicitInterRunnableVariable see sections 5.6.23 and 5.6.24.

For more details how this mechanism could work see "Copy strategy" in section 4.2.5.4.

4.2.5.6.2 InterRunnableVariables with explicit behavior

In many applications saving RAM is more important than saving runtime. Also some application require to have access to the newest data item value without any delay, even several times during execution of a Runnable.

Both requirements can be fulfilled when RTE supports data consistency by blocking second/third/.. concurrent accesses to a signal buffer if data consistency is jeopardized. (Most likely RTE has nothing to do if SW is running on a 16bit machine and making an access to an 16bit value when a 16bit data bus is present.)

Focus of *InterRunnableVariables with explicit behavior* is to block potential concurrent accesses to get data consistency.

The mechanism behind is the same as in the ExclusiveArea concept (see section 4.2.5.5). But although ExclusiveAreas can handle single data item accesses too, their API is made to make the RTE to apply data consistency means for a group of instructions accessing several data items as well. So when using an ExclusiveArea to protect accesses to one single common used data item each time two RTE API calls grouped around are needed. This is very inconvenient and might lead to faults if the calls grouped around might be forgotten.
The solution is to support *InterRunnableVariables with explicit behavior*.

[SWS_Rte_03519] ⌈ The RTE shall guarantee data consistency for *InterRunnableVariables with explicit behavior* by blocking concurrent accesses to data items specified by explicitInterRunnableVariable.
⌋*(SRS_Rte_00142, SRS_Rte_00032)*

The RTE generator is not free to select on it's own if implicit or explicit behavior shall be applied. Behavior must be known at AUTOSAR SW-C design time because in case of *InterRunnableVariables with implicit behavior* the AUTOSAR SW-C designer might rely on the fact that several read accesses always deliver same data item value.

[SWS_Rte_03580] ⌈ The RTE shall supply different APIs for *InterRunnableVariables with implicit* behavior and *InterRunnableVariables with explicit* behavior.
⌋*(SRS_Rte_00142)*

For API of *InterRunnableVariables with explicit behavior* see sections 5.6.25 and 5.6.26.

4.2.6 Multiple trigger of Runnable Entities and Basic Software Schedulable Entities

Concurrent activation

The AUTOSAR SW-C template specification [2] states that runnable entities (further called "runnables") might be invoked concurrently several times if the Runnables attribute canBeInvokedConcurrently is set. It's then in the responsibility of the AUTOSAR SW-C designer that no data might be corrupted when the Runnable is activated several times in parallel.

If a SW-C has multiple instances, they have distinct runnables. Two runnables that use the same RunnableEntity description of the same SwcInternalBehavior description but are instantiated with two different SW-C instances are treated as two distinct runnables in the following. This kind of concurrency is always allowed between SW-Cs, even if the runnables have their canBeInvokedConcurrently attribute set to false.

[SWS_Rte_03523] ⌈ The RTE shall support concurrent activation of the same instance of a runnable entity if the associative attribute canBeInvokedConcurrently is set to TRUE. This includes concurrent activation in several tasks. If the attribute is not set resp. set to FALSE, concurrent activation of the runnable entity is forbidden. (see requirement [SWS_Rte_05083]) ⌋*(SRS_Rte_00072, SRS_Rte_00133)*

The *Basic Software Module Description Template* [9] specifies the possible concurrent activation of BswModuleEntitys by the attribute isReentrant.

[SWS_Rte_07525] ⌈ The *Basic Software Scheduler* shall support concurrent activation of the same instance of a BswSchedulableEntity if the attribute isReentrant of the referenced BswModuleEntry in the role implementedEntry is set to true. This includes concurrent activation in several tasks. If the attribute is set to false concurrent activation of the BswSchedulableEntity is forbidden. (see requirement [SWS_Rte_07588]) ⌋

Concurrent activation of the same instance of a ExecutableEntity results in multiple ExecutableEntity execution-instances. One for each context of activation.

Activation by several RTEEvents and BswEvents

Nevertheless a Runnable whose attribute canBeInvokedConcurrently is NOT set might be still activated by several RTEEvents if activation configuration guarantees that concurrent activation can never occur and the minimumStartInterval condition is kept. This includes activation in different tasks. In this case, the runnable is still considered to have only one ExecutableEntity execution-instances. A standard use case is the activation of same instance of a runnable in different modes.

[SWS_Rte_03520] ⌈ The RTE shall support activation of same instance of a runnable entity by multiple RTEEvents. ⌋*(SRS_Rte_00072)*

RTEEvents are triggering runnable activation and may supply 0..several role parameters, see *section 5.7.3*. Role parameters are not visible in the runnables signature - except in those triggered by an OperationInvokedEvent. With the exception of the RTEEvent OperationInvokedEvent all role parameters can be accessed by user with implicit or explicit Receiver API.

[SWS_Rte_03524] ⌈ The RTE shall support activation of same instance of a runnable entity by RTEEvents of different kinds. ⌋*(SRS_Rte_00072)*

The RTE does NOT support a runnable entity triggered by an RTEEvent OperationInvokedEvent to be triggered by any other RTEEvent except for other OperationInvokedEvents of compatible operations. This limitation is stated in appendix in *section A.2* ([SWS_Rte_03526]).

The similar configuration as mentioned for the RunnableEntitys might be used for BswSchedulableEntitys. Therefore even a BswSchedulableEntity whose referenced BswModuleEntry in the role implementedEntry has its isReentrant attribute set to false can be activated by several BswEvents.

[SWS_Rte_07526] ⌈ The *Basic Software Scheduler* shall support activation of same instance of a BswSchedulableEntity by multiple BswEvents. ⌋

[SWS_Rte_07527] ⌈ The *Basic Software Scheduler* shall support activation of same instance of a BswSchedulableEntity by BswEvents of different kinds. ⌋

4.2.7 Implementation of Parameter and Data elements

4.2.7.1 General

A SWC communicates with other SWCs through ports. A port is characterized by a PortInterface and there are several kinds of PortInterfaces. In this section, we focus on the ParameterInterface, the SenderReceiverInterface, and the NvDataInterface. These three kinds of PortInterfaces aggregate some specific interface elements. For example, a ParameterInterface aggregates 0..* ParameterDataPrototypes.

4.2.7.2 Compatibility rules

A receiver port can only be connected to a compatible provider port. The compatibility rules are explained in the AUTOSAR Software Component Template [2]. The compatibility mainly depends on the attribute swImplPolicy attached to the element of the interface. The table 4.6 below gives an overview of compatibility rules.

Provide Port			Require Port					
Port Interface			Prm			S/R		NvD
Interface Element			PDP			VDP		VDP
		swImplPolicy	fixed	const	standard	standard	queued	standard
Prm	PDP	fixed	yes	yes	yes	yes	no	yes
		const	no	yes	yes	yes	no	yes
		standard	no	no	yes	yes	no	yes
S/R	VDP	standard	no	no	no	yes	no	yes
		queued	no	no	no	no	yes	no
NvD	VDP	standard	no	no	no	yes	no	yes

Table 4.6: Overview of compatibility of ParameterDataPrototype and VariableDataPrototypes

For examples, a Require Port that expects a fixed parameter - i.e produced by a macro #define - can only be connected to a Port that provides a fixed Parameter. This is because this fixed data may be used in a compilation directive like #IF and only macro #define (fixed data) can be compiled in this case. On the other hand, this provided fixed parameter can be connected to almost every require port, except a queued Sender/receiver interface.

The RTE doesn't have to check the compatibility between ports since this task is performed at the VFB level. But it shall provide the right implementation of interface el-

Interface Element		
PDP	:	ParameterDataPrototype
VDP	:	VariableDataPrototype
Port Interface		
Prm	:	ParameterInterface
S/R	:	SenderReceiverInterface
NvD	:	NvDataInterface

Table 4.7: Key to table 4.6

ement and API according the attribute swImplPolicy attached to the interface element.

4.2.7.3 Implementation of an interface element

The implementation of an interface element depends on the attribute swImplPolicy. The attribute swCalibrationAccess determines how the interface element can be accessed by e.g. an external calibration tool. The table 4.8 defines the supported combinations of swImplPolicy and swCalibrationAccess attribute setting and gives the corresponding implementation by the RTE.

4.2.7.4 Initialization of VariableDataPrototypes

Basically the need for initialization of any VariableDataPrototypes is specified by the Software Component Descriptions defining the VariableDataPrototypes. This information is basically defined by the existence of an initValue, the sectionInitializationPolicy of the related SwAddrMethod. As described in section 7.12 additionally the initialization strategy can be adjusted by the integrator of the RTE to adjust the behavior to the start-up code.

[SWS_Rte_07046] ⌈ Variables implementing VariableDataPrototypes shall be initialized if

- an initValue is defined

  AND

- no SwAddrMethod is defined for VariableDataPrototype.

⌋*(SRS_Rte_00052, SRS_Rte_00068, SRS_Rte_00116)*

[SWS_Rte_03852] ⌈ Variables implementing VariableDataPrototypes shall be initialized if

[4] calibration parameter have to be allocated in RAM if data emulation with SW support is required, see 4.2.8.3.5

swImplPolicy	SwCalibrationAccess			
	not Accessible	readOnly	readWrite	Implementation
fixed	yes	not supported	not supported	macro definition or c const declaration dependent from RTE optimization
const	yes	yes	not supported	c const declaration
standard	yes	yes	yes	standard implementation i.e. a variable for VariableDataPrototype in RAM or a calibration parameter in ROM [4]
queued	yes	not supported	not supported	FIFO Queue
measurement Point	not supported	yes	not supported	Variable

Table 4.8: Data implementation according swImplPolicy

- an `initValue` is defined

  AND
- a `SwAddrMethod` is defined for `VariableDataPrototype`

  AND
- the `RteInitializationStrategy` for the `sectionInitializationPolicy` of the related `SwAddrMethod` is NOT configured to `RTE_INITIALIZATION_STRATEGY_NONE`.

⌋(*SRS_Rte_00052*, *SRS_Rte_00068*, *SRS_Rte_00116*)

4.2.7.5 Initial value calculation

Basically the Meta Model defines two different flavors of rule based value specifications:

- ApplicationRuleBasedValueSpecification
- NumericalRuleBasedValueSpecification

The ApplicationRuleBasedValueSpecification defines the values in the physical representation whereas the NumericalRuleBasedValueSpecification defines the values in the numerical representation. (See document [2], section *Data Description*) But both are using the RuleBasedValueSpecification to define a set of values based on a rule and arguments for the rule.

Especially in case of large arrays an high amount of initial values are required. But many arrays are initialized with identical values or at least filled up to the end with identical values. For such use case the RuleBasedValueSpecification of category FILL_UNTIL_END can be used to avoid the creation and maintenance of redundant ValueSpecifications.

[SWS_Rte_06764] ⌈ The RTE Generator shall support ApplicationRuleBased-ValueSpecifications for DataPrototypes typed by ApplicationArray-DataTypes. ⌋*(SRS_Rte_00239)*

[SWS_Rte_06765] ⌈ The RTE Generator shall support NumericalRuleBasedValueSpecifications for DataPrototypes typed by ImplementationDataTypes of category ARRAY and for Compound Primitive Data Types which are mapped to ImplementationDataTypes of category ARRAY. ⌋*(SRS_Rte_00239)*

[SWS_Rte_06733] ⌈ The RTE Generator shall support RuleBasedValueSpecifications with the rule FILL_UNTIL_END. ⌋*(SRS_Rte_00239)*

[SWS_Rte_06734] ⌈ The RTE shall initialize the elements of the array according the values defined by RuleBasedValueSpecification.arguments if a RuleBased-ValueSpecification with the rule FILL_UNTIL_END is applicable.
Thereby the order of arguments corresponds to the order of elements in the array, i.e. the first argument corresponds to the first element of the array, the second argument corresponds to the second element of the array, and so on. ⌋*(SRS_Rte_00239)*
AUTOSAR defines a standardized behavior of RuleBasedValueSpecifications only for the rule FILL_UNTIL_END. RTE vendors are free to add additional, non-standardized rules (see [TPS_SWCT_01495]).

[SWS_Rte_06735] ⌈ The RTE Generator shall apply the value of the last RuleBased-ValueSpecification argument to any following element of the array until the last element of the array if the number of arguments is smaller than the number of elements of the array to which it is applied. ⌋*(SRS_Rte_00239)*

[SWS_Rte_06736] ⌈ The RTE Generator shall ignore arguments that go beyond the last element of the array if the number of arguments exceeds the number of elements of the array to which it is applied. ⌋*(SRS_Rte_00239)*

4.2.8 Measurement and Calibration

4.2.8.1 General

Calibration is the process of adjusting an ECU SW to fulfill its tasks to control physical processes respectively to fit it to special project needs or environments. To do this two different mechanisms are required and have to be distinguished:

1. Measurement
   Measure what's going on in the ECU e.g. by monitoring communication data (Inter-ECU, Inter-Partition, Intra-partition, Intra-SWC). There are several ways to get the monitor data out of the ECU onto external visualization and interpretation tools.

2. Calibration
   Based on the measurement data the ECU behavior is modified by changing parameters like runtime SW switches, process controlling data of primitive or composite data type, interpolation curves or interpolation fields. In the following for such parameters the term calibration parameter is used.

With AUTOSAR, a calibration parameter is instantiated with a `ParameterDataPrototype` class that aggregates a `SwDataDefProps` with properties `swCalibrationAccess` = `readWrite` and `swImplPolicy` = `standard`.

Nevertheless it is supported, that `VariableDataPrototype` is instantiated that aggregates a `SwDataDefProps` with properties `swCalibrationAccess` = `readWrite` and `swImplPolicy` = `standard`. But in this case the implementation of such `VariableDataPrototype` is treated identical to `swCalibrationAccess` = `readOnly` and the RTE Generator has not to implement further measures (for instance "Data emulation with SW support" 4.2.8.3.5).

It's possible that different `SwDataDefProps` settings are specified for a `VariableDataPrototype` and its referenced `AutosarDataType`. In this case the rules specified in the SWC-T shall be applied. See as well [SWS_Rte_07196].

`SwDataDefProps` contain more information how measurement values or characteristics are to be interpreted and presented by external calibration tools. This information is needed for the ASAM2 respectively A2L file generation. Afterwards the A2L file is used by ECU-external measurement and calibration tools so that these tools know e.g. how to interpret raw data received from ECU and how to get them.

4.2.8.1.1 Definition of Calibration Parameters

Calibration parameters can be defined in AUTOSAR SW as well as in Basic-SW. In the *AUTOSAR Architecture* there are two possibilities to define calibration parameters. Which one to choose is not in the focus of this RTE specification.

1. RTE provides the calibration parameter access if they are specified via a ParameterSwComponentType. A ParameterSwComponentType can be defined in order to provide ParameterDataPrototypes (via ports) to other Software Components.

2. Calibration parameter access invisible for RTE
Since multiple instantiation with code sharing is not allowed for Basic-SW and multiple instantiation is not always required for software components it's possible for these software to define own methods how calibration parameters are allocated. Nevertheless these calibration parameters shall be described in the belonging *Basic Software Module Description* respectively *Software Component Description*. In case data emulation with SW-support is used, the whole software and tool chain for calibration and measurement, e.g. Basic-SW (respectively XCP driver) which handles emulation details and data exchange with external calibration tools then has to deal with several emulation methods at once: The one the RTE uses and the other ones each Basic-SW or SWC using local calibration parameters practices.

4.2.8.1.2 Online and offline calibration

The way how measurement and calibration is performed is company, domain and project specific. Nevertheless two different basic situations can be distinguished and are important for understanding:

1. Offline calibration
Measure when ECU is running, change calibration data when ECU is off.
Process might look like this:

   (a) Flash the ECU with current program file

   (b) PowerUp ECU in target (actual or emulated) environment

   (c) Measure running ECU behavior - log or monitor via external tooling

   (d) Switch off ECU

   (e) Change calibration parameters and create a new flashable program file (hex-file) e.g. by performing a new SW make run

   (f) Back to (a).

   Do loop as long as a need for calibration parameter change exists or the Flash survives.

2. Online calibration

   Do measurement and calibration in parallel.
   In this case in principle all steps mentioned in "Offline calibration" above have to be performed in parallel. So other mechanisms are introduced avoiding ECU

flashing when modifying ECU parameters. ECU works temporarily with changed data and when the calibration process is over the result is an updated set of calibration data. In next step this new data set might be merged into the existing program file or the new data set might be an input for a new SW make run. In both cases the output is a new program file to flash into the ECU.

Process might look like this:

(a) Flash the ECU with current program file

(b) PowerUp ECU in target environment

(c) Measure running ECU behavior and temporarily modify calibration parameters. Store set of updated calibration parameters (not on the ECU but on the calibration tool computer). Actions in step c) may be done iteratively.

(d) Switch off ECU

(e) Create a new flashable program file (hex-file) containing the new calibration parameters

Procedure over

4.2.8.2 Measurement

4.2.8.2.1 What can be measured

The AUTOSAR SW-C template specification [2] explains to which AUTOSAR prototypes a measurement pattern can be applied.

RTE provides measurement support for

1. communication between Ports
   Measurable are
   - `VariableDataPrototype`s of a `SenderReceiverInterface` used in a `PortPrototype` (of a `SwComponentPrototype`) to capture sender-receiver communication or between `SwComponentPrototype`s
   - `VariableDataPrototype`s of a `NvDataInterface` used in a `PortPrototype` (of a `SwComponentPrototype`) to capture non volatile data communication or between `SwComponentPrototype`s
   - `ArgumentDataPrototype`s of an `ClientServerOperation` in a `ClientServerInterface` to capture client-server communication between `SwComponentPrototype`s
2. communication inside of AUTOSAR SW-Cs
   Measurable are `implicitInterRunnableVariable`, `explicitInterRunnableVariable` or `arTypedPerInstanceMemory`

3. data structures inside a AUTOSAR NvBlockSwComponent
   Measurable are ramBlocks and romBlocks of a NvBlockSwComponent's *NvBlock*

Further on AUTOSAR SW-Cs and *Basic Software Modules* can define measurables which are not instantiated by RTE. These are described by VariableDataPrototypes in the role staticMemory. Hence those kind of measurables are not described in the generated *McSupportData* of the RTE (see 4.2.8.4).

4.2.8.2.2 RTE support for Measurement

The way how measurement data is read out of the ECU is not focus of the RTE specification. But the RTE structure and behavior must be specified in that way that measurement values can be provided by RTE during ECU program execution.

To avoid synchronization effort it shall be possible to read out measurement data asynchronously to RTE code execution. For this the measurement data must be stable. As a consequence this might forbid direct reuse of RAM locations for implementation of several AUTOSAR communications which are independent of each other but occurring sequentially in time (e.g. usage of same RAM cell to store uint8 data sender receiver communication data between Runnables at positions 3 and 7 and later the same RAM cell for the communication between Runnables at positions 9 and 14 of same periodically triggered task). So applying measurable elements might lead to less optimizations in the generated RTE's code and to increased RAM need.

There are circumstances when RTE will store same communication data in different RAM locations, e.g. when realizing implicit sender receiver communication or Inter Runnable Variables with implicit behavior. In these cases there is only the need to have the content of one of these stores made accessible from outside.

The information that measurement shall be supported by RTE is defined in applied SwDataDefProps:
The value readOnly or readWrite of the property swCalibrationAccess defines that measurement shall be supported, any other value of the property swCalibrationAccess is to be ignored for measurement.

Please note that the definition of [SWS_Rte_03900] and [SWS_Rte_03902] do not have further conditions when the location in memory has to be provided to support the usage of VariableDataPrototype with the swImplPolicy = measurementPoint. In case that the MCD system is permitted to access such a VariableDataPrototype the RTE is not allowed to do optimization which would prevent such measurement even if there is no consuming software component in the input configuration.

The memory locations containing measurement values are initialized according to [SWS_Rte_07046] and [SWS_Rte_03852].

[SWS_Rte_07044] ⌈ The RTE generator shall reject input configurations in which a RunnableEntity defines a read access (VariableAccess in the role readLocalVariable, dataReadAccess, dataReceivePointByValue or dataReceivePointByArgument) to an VariableDataPrototype with a swImplPolicy set to measurementPoint. ⌋*(SRS_Rte_00018)*

For sender-receiver resp. client-server communication same or compatible interfaces are used to specified connected ports. So very often measurement will be demanded two times for same or compatible VariableDataPrototype on provide and require side of a 1:1 communication resp. multiple times in case of 1:N or M:1 communication. In that case providing more than one measurement value for a VariableDataPrototype doesn't make sense and would increase ECU resources need excessively. Instead only one measurement value shall be provided.

Sender-receiver communication

[SWS_Rte_03900] ⌈ If the swCalibrationAccess of a VariableDataPrototype used in an interface of a sender-receiver port of a SwComponentPrototype is set to readOnly or readWrite the RTE generator has to provide one reference to a location in memory where the actual content of the instance specific data of the corresponding VariableDataPrototype of the communication can be accessed. ⌋*(SRS_Rte_00153)*

To prohibit multiple measurement values for same communication:
(Note that affected VariableDataPrototypes might be specified in same or compatible port interfaces.)

[SWS_Rte_03972] ⌈ For 1:1 and 1:N sender-receiver communication the RTE shall provide measurement values taken from sender side if measurement is demanded in provide and require port. ⌋*(SRS_Rte_00153)*

[SWS_Rte_03973] ⌈ For N:1 intra-ECU sender-receiver communication the RTE shall provide measurement values taken from receiver side if measurement is demanded in provide and require ports. ⌋*(SRS_Rte_00153)*

Note:
See further below for support of queued communication.

[SWS_Rte_03974] ⌈ For a VariableDataPrototype with measurement demand associated with received data of inter-ECU sender-receiver communication the RTE shall provide only one measurement store reference containing the actual received data even if several receiver ports demand measurement. ⌋*(SRS_Rte_00153)*

[SWS_Rte_07344] ⌈ For a VariableDataPrototype with measurement demand associated with received data of inter-Partition sender-receiver communication the RTE shall provide only one measurement store reference per partition containing the actual received data even if several receiver ports demand measurement in the Partition. ⌋(SRS_Rte_00153)

Client-Server communication

[SWS_Rte_03901] ⌈ If the swCalibrationAccess of an ArgumentDataPrototype used in an interface of a client-server port of a SwComponentPrototype is set to readOnly the RTE generator has to provide one reference to a location in memory where the actual content of the instance specific argument data of the communication can be read. ⌋(SRS_Rte_00153)

To prohibit multiple measurement values for same communication:
(Note that affected ArgumentDataPrototypes might be specified in same or compatible port interfaces.)

[SWS_Rte_03975] ⌈ For intra-ECU client-server communication the RTE shall provide measurement values taken from client side if measurement of an ArgumentDataPrototypes is demanded by provide and require ports. ⌋(SRS_Rte_00153)

[SWS_Rte_03976] ⌈ For inter-ECU client-server communication with the client being present on same ECU as the RTE, the RTE shall provide measurement values taken from client side. ⌋(SRS_Rte_00153)

[SWS_Rte_03977] ⌈ For inter-ECU client-server communication with the server being present on same ECU as the RTE, the RTE shall provide measurement values taken from server if no client present on same ECU as the server is connected with that server too. ⌋(SRS_Rte_00153)

[SWS_Rte_07349] ⌈ For inter-Partition client-server communication with the server being present on the same ECU as the RTE, the RTE shall provide measurement values taken from server if no client present on the same Partition as the server is connected with that server too. ⌋(SRS_Rte_00153)

Note:
When a measurement is applied to a client-server call additional copy code might be produced so that a zero overhead direct server invocation is no longer possible for this call.

Mode Switch Communication

[SWS_Rte_06700] ⌈ If the swCalibrationAccess of a ModeDeclarationGroupPrototype used in an interface of a mode switch port of a SwComponentPrototype is set to readOnly the RTE generator has to provide three references to locations in memory where the *current mode*, the *previous mode* and the *next mode* of the related mode machine instance can be accessed. ⌋(SRS_Rte_00153)

The affected ModeDeclarationGroupPrototypes might be used at different ports with the same or compatible port interfaces. [SWS_Rte_06701] prohibits the occurrence of multiple measurement values for the same communication:

[SWS_Rte_06701] ⌈ For 1:1 and 1:N mode switch communication the RTE shall provide measurement values taken from mode manager side if measurement is demanded in provide and require port. ⌋*(SRS_Rte_00153)*

Inter Runnable Variables

[SWS_Rte_03902] ⌈ If the swCalibrationAccess of a VariableDataPrototype in the role implicitInterRunnableVariable or explicitInterRunnableVariable is set to readOnly or readWrite the RTE generator has to provide one reference to a location in memory where the actual content of the *Inter Runnable Variable* can be accessed for a specific instantiation of the AUTOSAR SWC.
⌋*(SRS_Rte_00153)*

PerInstanceMemory

[SWS_Rte_07160] ⌈ If the swCalibrationAccess of a VariableDataPrototype in the role arTypedPerInstanceMemory is set to readOnly or readWrite the RTE generator has to provide one reference to a location in memory where the actual content of the arTypedPerInstanceMemory can be accessed for a specific instantiation of the AUTOSAR SWC.
⌋*(SRS_Rte_00153)*

Nv RAM Block

[SWS_Rte_07174] ⌈ If the swCalibrationAccess of a VariableDataPrototype in the role ramBlock of a NvBlockSwComponentType's NvBlockDescriptor is set to readOnly or readWrite the RTE generator has to provide one reference to a location in memory where the actual content of the *Nv RAM Block* can be accessed for a specific instantiation of the AUTOSAR NvBlockSwComponentType.
⌋*(SRS_Rte_00153)*

Non Volatile Data communication

[SWS_Rte_07197] ⌈ If the swCalibrationAccess of a VariableDataPrototype used in an NvDataInterface of a non volatile data port of a SwComponentPrototype is set to readOnly or readWrite the RTE generator has to provide one reference to a location in memory where the actual content of the instance specific data of the corresponding VariableDataPrototype of the communication can be accessed. ⌋*(SRS_Rte_00153)*

To prohibit multiple measurement values for same communication:
(Note that affected VariableDataPrototypes might be specified in same or compatible port interfaces.)

[SWS_Rte_07198] ⌈ For 1:1 and 1:N non volatile data communication the RTE shall provide measurement values taken from ramBlock if measurement is demanded either in provide port, any require port ([SWS_Rte_07197] or ramBlock ([SWS_Rte_07174]). ⌋*(SRS_Rte_00153)*

Unconnected ports or compatible interfaces

As stated in section 5.2.7 RTE supports handling of unconnected ports.

Measurement support for unconnected sender-receiver provide ports makes sense since a port might be intentionally added for monitoring purposes only.

Measurement support for unconnected sender-receiver require ports makes sense since the measurement is specified on the type level of the Software Component and therefore independent of the individual usage of the Software Component. In case of unconnected sender-receiver require ports the measurement shall return the initial value.

Support for unconnected client-server provide port does not make sense since the server cannot be called and with this no data can be passed there.

Support for unconnected client-server require port makes sense since the measurement is specified on the type level of the Software Component and therefore independent of the individual usage of the Software Component. In case of unconnected client-server require ports the measurement shall return the actually provided and returned values.

[SWS_Rte_03978] ⌈ For sender-receiver communication the RTE generator shall respect measurement demands enclosed in unconnected provide ports. ⌋*(SRS_Rte_00139, SRS_Rte_00153)*

[SWS_Rte_05101] ⌈ For sender-receiver communication the RTE generator shall respect measurement demands enclosed in unconnected require ports and deliver the initial value. ⌋*(SRS_Rte_00139, SRS_Rte_00153)*

[SWS_Rte_03980] ⌈ For client-server communication the RTE generator shall ignore measurement demands enclosed in unconnected provide ports. ⌋*(SRS_Rte_00139, SRS_Rte_00153)*

[SWS_Rte_05102] ⌈ For client-server communication the RTE generator shall respect measurement demands enclosed in unconnected require ports. The behavior shall be similar as if the require port would be connected and the server does not respond. ⌋*(SRS_Rte_00139, SRS_Rte_00153)*

[SWS_Rte_05170] ⌈ For client-server communication the RTE generator shall ignore measurement requests for queued client-server communication. ⌋*(SRS_Rte_00139, SRS_Rte_00153)*

In case the measurement of client-server communication is not possible due to requirement [SWS_Rte_05170] the `McSupportData` need to reflect this (see [SWS_Rte_05172]).

In principle the same thoughts as above are applied to unused `VariableDataPrototype`s for sender-receiver communication where ports with compatible but not same interfaces are connected. It's no issue for client-server due to compatibility rules for client-server interfaces since in compatible client-server interfaces all `ClientServerOperation`s have to be present in provide and require port (see AUTOSAR SW-C Template [2]).

[SWS_Rte_03979] ⌈ For sender-receiver communication the RTE generator shall respect measurement demands of those VariableDataPrototypes in connected ports when provide and require port interfaces are not the same (but only compatible) even when a VariableDataPrototype in the provide port has no assigned VariableDataPrototype in the require port.
⌋*(SRS_Rte_00153)*

General measurement disabling switch

To support saving of ECU resources for projects where measurement isn't required at all whereas enclosed AUTOSAR SW-Cs contain SwDataDefProps requiring it, it shall be possible to switch off support for measurement. This shall not influence support for calibration (see 4.2.8.3).

[SWS_Rte_03903] ⌈ The RTE generator shall have the option to switch off support for measurement for generated RTE code. This option shall influence complete RTE code at once. ⌋*(SRS_Rte_00153)*

There also might be projects in which monitoring of ECU internal behavior is required but calibration is not.

[SWS_Rte_03904] ⌈ The enabling of RTE support for measurement shall be independent of the enabling of the RTE support for calibration. ⌋*(SRS_Rte_00153)*

Queued communication

Measurement of queued communication is not supported yet. Reasons are:

- A queue can be empty. What's to measure then?
- Which of the queue entries is the one to take the data from might differ out of user view?
- Only quite inefficient solutions possible because implementation of queues entails storage of information dynamically at different memory locations. So always additional copies are required.

[SWS_Rte_03950] ⌈ RTE generator shall reject configurations where measurement for queued sender-receiver communication is configured. ⌋*(SRS_Rte_00153, SRS_Rte_00018)*

4.2.8.3 Calibration

The RTE and *Basic Software Scheduler* has to support the allocation of calibration parameters and the access to them for SW using them. As seen later on for some calibration methods the RTE and *Basic Software Scheduler* must contain support SW too (see 4.2.8.3.5). But in general the RTE and *Basic Software Scheduler* is not responsible for the exchange of the calibration data values or the transportation of them between the ECU and external calibration tools.

The following sections are mentioning only the RTE but this has to be understood in the context that the support for *Calibration* is a functionality which affects the Basic Software Scheduler part of the RTE as well. In case of the *Basic Software Scheduler Generation Phase* (see 3.4.1) this functionality might even be provided with out any other software component related RTE functionality.

With AUTOSAR, a calibration parameter (which the AUTOSAR SW-C template specification [2] calls `ParameterSwComponentType`) is instantiated with a `ParameterDataPrototype` that aggregates a `SwDataDefProps` with properties `swCalibrationAccess` = `readWrite` and `swImplPolicy` = `standard`. This chapter applies to this kind of `ParameterSwComponentType`s. For other combinations of these properties, consult the section 4.2.7

4.2.8.3.1 Calibration parameters

Calibration parameters can be defined in `ParameterSwComponentType`s, in AUTOSAR SW-Cs, `NvBlockSwComponentType`s and in *Basic Software Module*s.

1. `ParameterSwComponentType`s don't have an internal behavior but contain `ParameterDataPrototype`s and serve to provide calibration parameters used commonly by several AUTOSAR SW-Cs. The use case that one or several of the user SW-Cs are instantiated on different ECUs is supported by instantiation of the `ParameterSwComponentType` on the affected ECUs too.
   Of course several AUTOSAR SW-Cs allocated on one ECU can commonly access the calibration parameters of `ParameterSwComponentType`s too. Also several instances of an AUTOSAR SW-Cs can share the same calibration parameters of a `ParameterSwComponentType`.

2. Calibration parameters defined in AUTOSAR SW-Cs can only be used inside the SW-C and are not visible to other SW-Cs. Instance individual and common calibration parameters accessible by all instances of an AUTOSAR SW-C are possible.

3. For `NvBlockSwComponentType`s it is supported to provide calibration access to the `ParameterDataPrototype` defining the `romBlock`. These values can not be directly accessed by AUTOSAR SW-Cs but are used to serve as *ROM Block* default values for the *Nv Block*.

4. Calibration parameters defined in *Basic Software Module*s can only be used inside the defining *Basic Software Module* and are not visible to other *Basic Software Module*s. In contrast to AUTOSAR SW-Cs, *Basic Software Module*s can only define instance specific calibration parameters.

[SWS_Rte_03958] ⌈ Several AUTOSAR SW-Cs (and also several instances of AUTOSAR SW-Cs) shall be able to share same calibration parameters defined in `ParameterSwComponentType`s. ⌋*(SRS_Rte_00154, SRS_Rte_00159)*

[SWS_Rte_07186] ⌈ The generated RTE shall initialize the memory objects implementing ParameterDataPrototypes in *p-ports* of ParameterSwComponentTypes according the ValueSpecification of the ParameterProvideComSpec referring the ParameterDataPrototype in the *p-port*,

- if such ParameterProvideComSpec exists and
- if no CalibrationParameterValue refers to the FlatInstanceDescriptor associated to the ParameterDataPrototype

This is also applicable if the swImplPolicy = fixed and if the related ParameterDataPrototype is implemented as preprocessor define which does not immediately allocate a memory object. ⌋*(SRS_Rte_00154, SRS_Rte_00159)*

[SWS_Rte_07029] ⌈ The generated RTE shall initialize the memory objects implementing ParameterDataPrototypes in *p-ports* of ParameterSwComponentTypes according the ValueSpecification in the role implInitValue of the CalibrationParameterValue referring the FlatInstanceDescriptor associated to the ParameterDataPrototype if such CalibrationParameterValue is defined. ⌋*(SRS_Rte_00154)*

Note: the initialization according [SWS_Rte_07029] and [SWS_Rte_07030] precedes the initialization values defined in the context of an component type and used in [SWS_Rte_07185] and [SWS_Rte_07186]. This enables to provide initial values for calibration parameter instances to:

- predefine start values for the calibration process
- utilizes the result of the calibration process
- take calibration parameter values from previous projects

[SWS_Rte_03959] ⌈ If the SwcInternalBehavior aggregates an ParameterDataPrototype in the role perInstanceParameter the RTE shall support the access to instance specific calibration parameters of the AUTOSAR SW-C. ⌋*(SRS_Rte_00154, SRS_Rte_00158)*

[SWS_Rte_05112] ⌈ If the SwcInternalBehavior aggregates an ParameterDataPrototype in the role sharedParameter the RTE shall create a common access to the shared calibration parameter. ⌋*(SRS_Rte_00154, SRS_Rte_00159)*

[SWS_Rte_07096] ⌈ If the BswInternalBehavior aggregates an ParameterDataPrototype in the role perInstanceParameter the *Basic Software Scheduler* shall support the access to instance specific calibration parameters of the *Basic Software Module*. ⌋*(SRS_Rte_00154, SRS_Rte_00158)*

[SWS_Rte_07185] ⌈ The generated RTE and *Basic Software Scheduler* shall initialize the memory objects implementing ParameterDataPrototype in the role perInstanceParameter or sharedParameter

- if it has a ValueSpecification in the role initValue according to this ValueSpecification and

- if no CalibrationParameterValue refer to the FlatInstanceDescriptor associated to the ParameterDataPrototype

This is also applicable if the swImplPolicy = fixed and if the related ParameterDataPrototype is implemented as preprocessor define which does not immediately allocate a memory object. ⌋*(SRS_Rte_00154)*

[SWS_Rte_07030] ⌈ The generated RTE and *Basic Software Scheduler* shall initialize the memory objects implementing ParameterDataPrototypes in the role perInstanceParameter or sharedParameter according the ValueSpecification in the role the implInitValue of the CalibrationParameterValue referring the FlatInstanceDescriptor associated to the ParameterDataPrototype if such CalibrationParameterValue is defined. ⌋*(SRS_Rte_00154)*

It might be project specific or even project phase specific which calibration parameters have to be calibrated and which are assumed to be stable. So it shall be selectable on ParameterSwComponentTypes and AUTOSAR SW-C granularity level for which calibration parameters RTE shall support calibration.

If an r-port contains a ParameterDataPrototype, the following requirements specify its behavior if the port is unconnected.

[SWS_Rte_02749] ⌈ In case of an unconnected parameter r-port, the RTE shall set the values of the ParameterDataPrototypes of the r-port according to the initValue of the r-port's ParameterRequireComSpec referring to the ParameterDataPrototype. ⌋*(SRS_Rte_00139, SRS_Rte_00159)*

If the port is unconnected, RTE expects an init value, see [SWS_Rte_02750].

ParameterDataPrototypes in role romBlock

[SWS_Rte_07033] ⌈ If the swCalibrationAccess of a ParameterDataPrototype in the role romBlock is set to readWrite the RTE generator has to provide one reference to a location in memory where the actual content of the *romBlock* can be accessed. ⌋*(SRS_Rte_00154)*

[SWS_Rte_07034] ⌈ The generated RTE shall initialize any ParameterDataPrototype in the role romBlock

- if it has a ValueSpecification in the role initValue according to this ValueSpecification and
- if no CalibrationParameterValue refer to the FlatInstanceDescriptor associated to the ParameterDataPrototype

⌋*(SRS_Rte_00154)*

[SWS_Rte_07035] ⌈ The generated RTE shall initialize the memory objects implementing ParameterDataPrototypes in the role romBlock according the ValueSpecification in the role the implInitValue of the CalibrationParameterValue referring the FlatInstanceDescriptor associated to the ParameterDataPrototype if such CalibrationParameterValue is defined. ⌋*(SRS_Rte_00154)*

ParameterDataPrototype used as romBlock are instantiated according to [SWS_Rte_07693].

Configuration of calibration support

[SWS_Rte_03905] ⌈ It shall be configurable for each ParameterSwComponentType if RTE calibration support for the enclosed ParameterDataPrototypes is enabled or not. ⌋*(SRS_Rte_00154, SRS_Rte_00156)*

[SWS_Rte_03906] ⌈ It shall be configurable for each AUTOSAR SW-C if RTE calibration support for the enclosed ParameterDataPrototypes is enabled or not. ⌋*(SRS_Rte_00154, SRS_Rte_00156)*

RTE calibration support means the creation of SW as specified in section 4.2.8.3.5 "Data emulation with SW support".

Require ports on ParameterSwComponentTypes don't make sense. ParameterSwComponentTypes only have to provide calibration parameters to other Component types. So the RTE generator shall reject configurations containing require ports attached to ParameterSwComponentTypes. (see section A.13)

4.2.8.3.1.1 Separation of calibration parameters

Sometimes it is required that one or more calibration parameters out of the mass of calibration parameters of an ParameterSwComponentType respectively an AUTOSAR SW-C shall be placed in another memory location than the other parameters of the ParameterSwComponentType respectively the AUTOSAR SW-C. This might be due to security reasons (separate normal operation from monitoring calibration data in memory) or the possibility to change calibration data during a diagnosis session (which the calibration parameter located in NVRAM).

[SWS_Rte_03907] ⌈ The RTE generator shall support separation of calibration parameters from ParameterSwComponentTypes, AUTOSAR SW-Cs and *Basic Software Modules* depending on the ParameterDataPrototype property swAddrMethod. ⌋*(SRS_Rte_00154, SRS_Rte_00158)*

4.2.8.3.2 Support for offline calibration

As described in section 4.2.8.1 when using an offline calibration process measurement is decoupled from providing new calibration parameters to the ECUs SW. During measurement phase information is collected needed to define to which values the calibration parameters are to be set best. Afterwards the new calibration parameter set is brought into the ECU e.g. by using a bootloader.

[SWS_Rte_03971] ⌈ The RTE generator shall have the option to switch off all *data emulation* support for generated RTE code. This option shall influence complete RTE code at once. ⌋*(SRS_Rte_00154, SRS_Rte_00156)*

The term *data emulation* is related to mechanisms described in section 4.2.8.3.3.

Out of view of RTE the situation is same as when *data emulation without SW support* (described in section 4.2.8.3.4) is used:
The RTE is only responsible to provide access to the calibration parameters via the RTE API as specified in section 5.6. Exchange of `ParameterDataPrototype` content is done invisibly for ECU program flow and with this for RTE too.

When no *data emulation support* is required calibration parameter accesses to parameters stored in FLASH could be performed by direct memory read accesses without any indirection for those cases when accesses are coming out of single instantiated AUTOSAR SW-Cs or from *Basic Software Module*s. Nevertheless it's not goal of this specification to require direct accesses since this touches implementation. It might be ECU HW dependent or even be project dependent if other accesses are more efficient or provide other significant advantages or not.

4.2.8.3.3 Support for online calibration: Data emulation

To allow **online calibration** it must be possible to provide alternative calibration parameters invisible for application. The mechanisms behind are described here. We talk of *data emulation*.

In the following several calibration methods are described:

1. Data emulation without SW support and
2. several methods of data emulation with SW-support.

The term **data emulation** is used because the change of calibration parameters is emulated for the ECU SW which uses the calibration data. This change is invisible for the user-SW in the ECU.

RTE is significantly involved when SW support is required and has to create calibration method specific SW. Different calibration methods means different support in Basic SW which typically is ECU integrator specific. So it does not make sense to support DIFFERENT data emulation with SW support methods in ANY one RTE build. But it makes sense that the RTE supports direct access (see section 4.2.8.3.4) for some AUTOSAR SW-Cs resp. `ParameterSwComponentType`s resp. *Basic Software Modules* and one of the data emulation with SW support methods (see section 4.2.8.3.5) for all the other AUTOSAR SW-Cs resp. `ParameterSwComponentType`s resp. *Basic Software Modules* at the same time.

[SWS_Rte_03909] ⌈ The RTE shall support only one of the data emulation with SW support methods at once. ⌋*(SRS_Rte_00154, SRS_Rte_00156)*

4.2.8.3.4 Data emulation without SW support (direct access)

For "online calibration" (see section 4.2.8.1) the ECU is provided with additional hardware which consists of control logic and memory to store modified calibration parameters in. During ECU execution the brought in control logic redirects memory accesses to new bought in memory whose content is modified by external tooling without disturbing normal ECU program flow. Some microcontrollers contain features supporting this. A lot of smaller microcontrollers don't. So this methods is highly HW dependent.

To support these cases the RTE doesn't have to provide e.g. a reference table like described in section 4.2.8.3.5. Exchange of `ParameterDataPrototype` content is done invisibly for program flow and for RTE too.

[SWS_Rte_03942] ⌈ The RTE generator shall have the option to switch off *data emulation with SW support* for generated RTE code. This option shall influence complete RTE code at once. ⌋*(SRS_Rte_00154, SRS_Rte_00156)*

4.2.8.3.5 Data emulation with SW support

In case "online calibration" (see section 4.2.8.1) is required, quite often data emulation without support by special SW constructs isn't possible. Several methods exist, all have the consequence that additional need of ECU resources like RAM, ROM/FLASH and runtime is required.

Data emulation with SW support is possible in different manners. During calibration process in each of these methods modified calibration data values are kept typically in RAM. Modification is controlled by ECU external tooling and supported by ECU internal SW located in AUTOSAR basic SW or in complex driver.

If calibration process isn't active the accessed calibration data is originated in ROM/FLASH respectively in NVRAM in special circumstances (as seen later on).

Since multiple instantiation is to be supported several instances of the same `ParameterDataPrototype`s have to be allocated. Because the RTE is the only one SW in an AUTOSAR ECU able to handle the different instances the access to these calibration parameters can only be handled by the RTE. So the RTE has to provide additional SW constructs required for data emulation with SW support for calibration.

However the RTE doesn't know which of the ECU functionality shall be calibrated during a calibration session. To allow expensive RAM to be reused to calibrate different ECU functionalities in one or several online calibration sessions (see 4.2.8.1) in case of the single and double pointered methods for data emulation with SW support described below the RTE has only to provide the access to `ParameterDataPrototype`s during runtime but allowing other SW (a BSW module or a complex driver) to redirect the access to alternative calibration parameter values (e.g. located in RAM) invisibly for application.

The RTE is neither the instance to supply the alternative values for ParameterDataPrototypes nor in case of the pointered methods for data emulation with SW support to do the redirection to the alternative values.

[SWS_Rte_03910] ⌈ The RTE shall support *data emulation with SW support* for calibration. ⌋*(SRS_Rte_00154, SRS_Rte_00156)*

[SWS_Rte_03943] ⌈ The RTE shall support these data emulation methods with SW support:

- Single pointered calibration parameter access
  further called "single pointered method"
- Double pointered calibration parameter access further called "double pointered method"
- Initialized RAM parameters further called "initRAM parameter method"

⌋*(SRS_Rte_00154, SRS_Rte_00156)*

Please note that the support data emulation methods is applicable for calibration parameters provided for software components as well as calibration parameters provided for basic software modules.

ParameterElementGroup

To save RAM/ROM/FLASH resources in single pointered method and double pointered method ParameterDataPrototype allocation is done in groups. One entry of the calibration reference table references the begin of a group of ParameterDataPrototypes. For better understanding of the following, this group is called ParameterElementGroup (which is no term out of the AUTOSAR SW-C template specification [2]). One ParameterElementGroup can contain one or several ParameterDataPrototypes.

[SWS_Rte_03911] ⌈ If data emulation with SW support is enabled, the RTE generator shall allocate all ParameterDataPrototypes marked with same property swAddrMethod of one instance of a ParameterSwComponentType consecutively. Together they build a separate ParameterElementGroup. ⌋*(SRS_Rte_00154, SRS_Rte_00156, SRS_Rte_00158)*

[SWS_Rte_03912] ⌈ If data emulation with SW support is enabled, the RTE shall guarantee that all non-shared ParameterDataPrototypes marked with same property swAddrMethod of an AUTOSAR SWC instance are allocated consecutively. Together they build a separate ParameterElementGroup. ⌋*(SRS_Rte_00154, SRS_Rte_00158)*

[SWS_Rte_05194] ⌈ If data emulation with SW support is enabled, the RTE shall guarantee that all shared ParameterDataPrototypes marked with same property swAddrMethod of an AUTOSAR SWC type are allocated consecutively. Together they build a separate ParameterElementGroup. ⌋*(SRS_Rte_00154, SRS_Rte_00158)*

It is not possible to access same calibration parameter inside of a ParameterSwComponentType via several ports. This is a consequence of the need to support the use case that a ParameterSwComponentType shall be able to contain several calibration parameters derived from one ParameterDataPrototype which is contained in one interface applied to several ports of the ParameterSwComponentType. Using only the ParameterDataPrototype names for the names of the elements of a ParameterElementGroup would lead to a name clash since then several elements with same name would have to created. So port prototype and ParameterDataPrototype name are concatenated to specify the ParameterElementGroup member names.
This use case cannot be applied to AUTOSAR SW-C internal calibration parameters since they cannot be accessed via AUTOSAR ports.

[SWS_Rte_03968] ⌈ The names of the elements of a ParameterElementGroup derived from a ParameterSwComponentType shall be <port>_<element> where <port> is the short-name of the provided AUTOSAR port prototype and <element> the short-name of the ParameterDataPrototype within the ParameterInterface categorizing the PPort. ⌋*(SRS_Rte_00154, SRS_Rte_00156)*

4.2.8.3.5.1 Single pointered method

There is one calibration reference table in RAM with references to one or several ParameterElementGroups. Accesses to calibration parameters are indirectly performed via this reference table.

Action during calibration procedure e.g. calibration parameter value exchange is not focus of this specification. Nevertheless an example is given for better understanding.

Example how the exchange of calibration parameters could be done for single pointered method:

1. Fill a RAM buffer with the modified calibration parameter values for complete ParameterElementGroup
2. Modify the corresponding entry in the calibration reference table so that a redirection to new ParameterElementGroup is setup

Now calibration parameter accesses deliver the modified values.

Figure 4.25 illustrates the method.

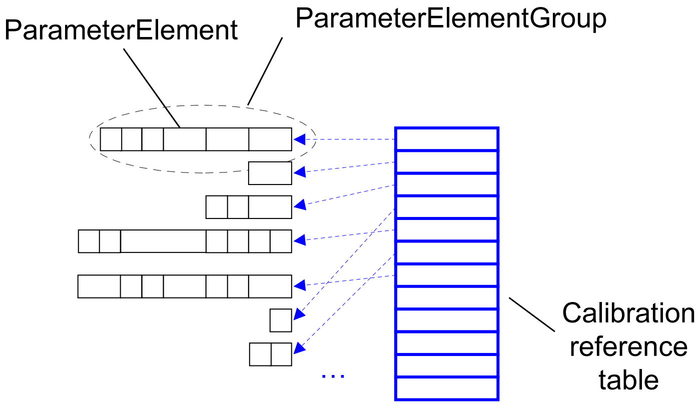


Figure 4.25: ParameterElementGroup in single pointered method context

[SWS_Rte_03913] ⌈ If data emulation with SW support with single pointered method is enabled, the RTE generator shall create a table located in RAM with references to `ParameterElementGroups`. The type of the table is an array of void pointers. ⌋*(SRS_Rte_00154, SRS_Rte_00156)*

One reason why in this approach the calibration reference table is realized as an array is to make ECU internal reference allocation traceable for external tooling. Another is to allow a Basic-SW respectively a complex driver to emulate other calibration parameters which requires the standardization of the calibration reference table too.

[SWS_Rte_03947] ⌈ If data emulation with SW support with single method is enabled the name (the label) of the calibration reference table shall be `<RteParameterRefTab>`. ⌋*(SRS_Rte_00154, SRS_Rte_00156)*

Calibration parameters located in NVRAM are handled same way (also see section 4.2.8.3.6).

[SWS_Rte_03936] ⌈ If data emulation with SW support with single or double pointered method is enabled and calibration parameter respectively a `ParameterElementGroups` is located in NVRAM the corresponding calibration reference table entry shall reference the PerInstanceMemory working as the NVRAM RAM buffer. ⌋*(SRS_Rte_00154, SRS_Rte_00156, SRS_Rte_00157)*

4.2.8.3.5.2 Double pointered method

There is one calibration reference table in ROM respectively Flash with references to one or several `ParameterElementGroups`. Accesses to calibration parameters are performed through a double indirection access. During system startup the base reference is initially filled with a reference to the calibration reference table.

Action during calibration procedure e.g. calibration parameter value exchange is not focus of this specification. Nevertheless an example is given for better understanding.

Example how the exchange of calibration parameters could be done for double pointered method:

1. Copy the calibration reference table into RAM
2. Fill a RAM buffer with modified calibration parameter values for complete `ParameterElementGroup`
3. Modify the corresponding entry in the RAM copy of the reference table so that a redirection to new `ParameterElementGroup` is setup
4. Change the content of the base reference so that it references the calibration reference table copy in RAM.

Now calibration parameter accesses deliver the modified values.

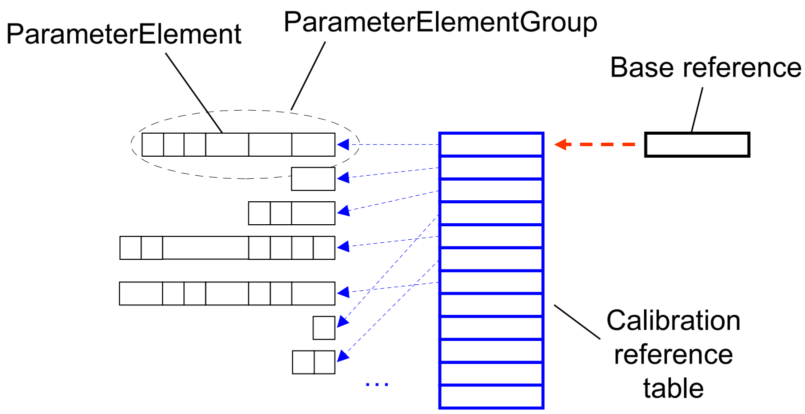


Figure 4.26: ParameterElementGroup in double pointered method context

[SWS_Rte_03914] ⌈ If data emulation with SW support with double pointered method is enabled, the RTE generator shall create a table located in ROM respectively FLASH with references to `ParameterElementGroups`. The type of the table is an array of void pointers. ⌋*(SRS_Rte_00154, SRS_Rte_00156)*

Figure 4.26 illustrates the method.

To allow a Basic-SW respectively a complex driver to emulate other calibration parameters the standardization of the base reference is required.

[SWS_Rte_03948] ⌈ If data emulation with SW support with double method is enabled the name (the label) of the calibration base reference shall be `<RteParameterBase>`. This label and the base reference type shall be exported and made available to other SW on same ECU.
⌋*(SRS_Rte_00154, SRS_Rte_00156)*

Calibration parameters located in NVRAM are handled same way (also see section 4.2.8.3.6).

For handling of calibration parameters located in NVRAM with single or double pointered method see [SWS_Rte_03936] in section 4.2.8.3.5.1. General information is found in section 4.2.8.3.6).

4.2.8.3.5.3 InitRam parameter method

For each instance of a ParameterDataPrototype the RTE generator creates a calibration parameter in RAM and a corresponding value in ROM/FLASH. During startup of RTE the calibration parameter values of ROM/FLASH are copied into RAM. Accesses to calibration parameters are performed through a direct access to RAM without any indirection.

Action during calibration procedure e.g. calibration parameter value exchange is not focus of this specification. Nevertheless an example is given for better understanding: An implementation simply would have to exchange the content of the RAM cells during runtime.

[SWS_Rte_03915] ⌈ If data emulation with SW support with initRam parameter method is enabled, the RTE generator shall create code guaranteeing that

1. calibration parameters are allocated in ROM/Flash and
2. a copy of them is allocated in RAM made available latest during RTE startup

for those ParameterDataPrototypes for which calibration support is enabled. ⌋*(SRS_Rte_00154, SRS_Rte_00156)*

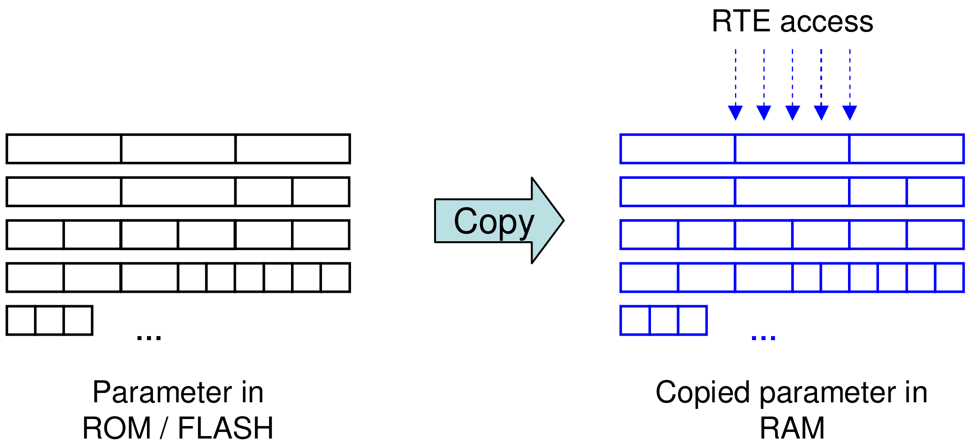


Figure 4.27: initRam Parameter method setup

Figure 4.27 illustrates the method.

A special case is the access of ParameterDataPrototypes instantiated in NVRAM (also see section 4.2.8.3.6). In this no extra RAM copy is required because a RAM location containing the calibration parameter value still exists.

[SWS_Rte_03935] ⌈ If data emulation with SW support with initRam parameter method is enabled, the RTE generator shall create direct accesses to the PerInstanceMemory working as RAM buffer for the calibration parameters defined to be in NVRAM. ⌋*(SRS_Rte_00154, SRS_Rte_00156)*

4.2.8.3.5.4 Arrangement of a ParameterElementGroup for pointered methods

For data emulation with SW support with single or double pointered methods the RTE has to guarantee access to each single member of a ParameterElementGroup for source code and object code delivery independent if the member is a primitive or a composite data type. For this the creation of a record type for a ParameterElementGroup was chosen.

[SWS_Rte_03916] ⌈ One ParameterElementGroup shall be realized as one record type. ⌋*(SRS_Rte_00154, SRS_Rte_00156)*

The sequence order of ParameterDataPrototype in a ParameterElementGroup and the order of ParameterElementGroups in the reference table will be documented by the RTE Generator by the means of the McSwEmulationMethodSupport, see 4.2.8.4.4.

4.2.8.3.5.5 Further definitions for pointered methods

As stated in section 4.2.8.3.1.1, dependent of the value of property swAddrMethod calibration parameters shall be separated in different memory locations.

[SWS_Rte_03908] ⌈ If data emulation with SW support with single or double pointered method is enabled the RTE shall create a separate instance specific ParameterElementGroup for all those ParameterDataPrototypes with a common value of the appended property swAddrMethod. Those ParameterDataPrototypes which have no property swAddrMethod appended, shall be grouped together too. ⌋*(SRS_Rte_00154, SRS_Rte_00156, SRS_Rte_00158)*

To allow traceability for external tooling the sequence order of ParameterDataPrototype in a ParameterElementGroup and the order of ParameterElementGroups in the reference table will be documented by the RTE Generator by the means of the McSwEmulationMethodSupport, see 4.2.8.4.4.

4.2.8.3.5.6 Calibration parameter access

Calibration parameters are derived from ParameterDataPrototypes. The RTE has to provide access to each calibration parameter via a separate API call.

API is specified in 5.6.

[SWS_Rte_03922] ⌈ If data emulation with SW support and single or double pointered method is enabled the RTE generator shall export the label of the calibration reference table. ⌋*(SRS_Rte_00154, SRS_Rte_00156)*

[SWS_Rte_03960] ⌈ If data emulation with SW support and double pointered method is enabled the RTE generator shall export the label and the type of the calibration base reference. ⌋*(SRS_Rte_00154, SRS_Rte_00156)*

[SWS_Rte_03932] ⌈ If data emulation with SW support with single pointered method is enabled the RTE generator shall create API calls using single indirect access via the calibration reference table for those ParameterDataPrototypes which are in a ParameterElementGroup for which calibration is enabled. ⌋*(SRS_Rte_00154, SRS_Rte_00156)*

[SWS_Rte_03933] ⌈ If data emulation with SW support with double pointered method is enabled the RTE generator shall create API calls using double indirection access via the calibration base reference and the calibration reference table for those ParameterDataPrototypes which are in a ParameterElementGroup for which calibration is enabled. ⌋*(SRS_Rte_00154, SRS_Rte_00156)*

[SWS_Rte_03934] ⌈ If data emulation with SW support with double pointered method is enabled, the calibration base reference shall be located in RAM. ⌋*(SRS_Rte_00154, SRS_Rte_00156)*

4.2.8.3.5.7 Calibration parameter allocation

Since only the RTE knows which instances of AUTOSAR SW-Cs, ParameterSwComponentTypes and *Basic Software Modules* are present on the ECU the RTE has to allocate the calibration parameters and reserve memory for them. This approach is also covering multiple instantiated object code integration needs. So memory for instantiated ParameterDataPrototypes is neither provided by ParameterSwComponentTypes nor by AUTOSAR SW-C.

Nevertheless AUTOSAR SW-Cs and *Basic Software Module*s can define calibration parameters which are not instantiated by RTE. These are described by ParameterDataPrototypes in the role constantMemory. Further on the RTE can not implement any software support for data emulation for such calibration parameters. Hence those kind of calibration parameters are not described in the generated *McSupportData* of the RTE (see 4.2.8.4).

[SWS_Rte_03961] ⌈ The RTE shall allocate the memory for calibration parameters. ⌋*(SRS_Rte_00154, SRS_Rte_00156)*

A ParameterDataPrototype can be defined to be instance specific or can be shared over all instances of an AUTOSAR SW-C or a ParameterSwComponentType. The input for the RTE generator contains the values the RTE shall apply to the calibration parameters.

To support online and offline calibration (see section 4.2.8.1) all parameter values for all instances have to be provided.
Background:

- For online calibration often initially the same default values for calibration parameters can be applied. Variation is then handled later by post link tools. Initial ECU startup is not jeopardized. This allows the usage of a default value e.g. by

AUTOSAR SW-C or ParameterSwComponentType supplier for all instances of a ParameterDataPrototype.

- On the other hand applying separate default values for the different instances of a ParameterDataPrototype will be required often for online calibration too, to make a vehicle run initially. This requires additional configuration work e.g. for integrator.
- Offline calibration based on new SW build including new RTE build and compilation process requires all calibration parameter values for all instances to be available for RTE.

Shared ParameterDataPrototypes

[SWS_Rte_03962] ⌈ For accesses to a shared ParameterDataPrototype the RTE API shall deliver the same one value independent of the instance the calibration parameter is assigned to. ⌋*(SRS_Rte_00154, SRS_Rte_00156)*

[SWS_Rte_03963] ⌈ The calibration parameter of a shared ParameterDataPrototype shall be stored in one memory location only. ⌋*(SRS_Rte_00154, SRS_Rte_00156)*

Requirements [SWS_Rte_03962] and [SWS_Rte_03963] are to guarantee that only one physical location in memory has to be modified for a change of a shared ParameterDataPrototype. Otherwise this could lead to unforeseeable confusion.
Multiple locations are possible for calibration parameters stored in NVRAM. But there a shared ParameterDataPrototype is allowed to have only one logical data too.

Instance specific ParameterDataPrototypes

[SWS_Rte_03964] ⌈ For accesses to an instance specific ParameterDataPrototype the RTE API shall deliver a separate calibration parameter value for each instance of a ParameterDataPrototype. ⌋*(SRS_Rte_00154, SRS_Rte_00156)*

[SWS_Rte_03965] ⌈ For an instance specific ParameterDataPrototype the calibration parameter value of each instance of the ParameterDataPrototype shall be stored in a separate memory location. ⌋*(SRS_Rte_00154, SRS_Rte_00156)*

Usage of swAddrMethod

SwDataDefProps contain the optional property *swAddrMethod*. It contains meta information about the memory section in which a measurement data store resp. a calibration parameter shall be allocated in. This abstraction is needed to support the reuse of unmodified AUTOSAR SW-Cs resp. ParameterSwComponentTypes in different projects but allowing allocation of measurement data stores resp. calibration parameters in different sections.
Section usage typically depends on availability of HW resources. In one project the micro controller might have less internal RAM than in another project, requiring that most measurement data have to be placed in external RAM. In another project one addressing method (e.g. indexed addressing) might be more efficient for most of the

measurement data - but not for all. Or some calibration parameters are accessed less often than others and could be - depending on project specific FLASH availability - placed in FLASH with slower access speed, others in FLASH with higher access speed.

[SWS_Rte_03981] ⌈ The memory section used to store measurement values in shall be the memory sections associated with the `swAddrMethod` enclosed in the `SwDataDefProps` of a measurement definition. ⌋ *(SRS_Rte_00153)*

Since it's measurement data obviously this must be in RAM.

[SWS_Rte_03982] ⌈ The memory section used to store calibration parameters in shall be the memory sections associated with the `swAddrMethod` enclosed in the `SwDataDefProps` of a calibration parameter definition. ⌋ *(SRS_Rte_00153)*

4.2.8.3.6 Calibration parameters in NVRAM

Calibration parameters can be located in NVRAM too. One use case for this is to have the possibility to modify calibration parameters via a diagnosis service without need for special calibration tool.

To allow NVRAM calibration parameters to be accessed, NVRAM with statically allocated RAM buffer in form of PIM memory for the calibration parameters has to be defined or the `ramBlock` of a `NvBlockSwComponentType` defines `readWrite` access for the MCD system. Please see as well [SWS_Rte_07174] and [SWS_Rte_07160].

Note:

As the NVRAM Manager might not be able to access the `PerInstanceMemory` across core boundaries in a multi core environment, the support of Calibration parameters in NVRAM for multi core controllers is limited. See also note in 4.2.9.1.

4.2.8.4 Generation of *McSupportData*

The RTE Generator supports the definition, allocation and access to measurement and calibration data for Software Components as well as for Basic Software. The specific support of measurement and calibration tools however is neither in the focus of the RTE Generator nor AUTOSAR. This would require the generation of an "A2L"-file (like specified in [20]) which is the standard in this domain – but out of the focus of AUTOSAR.

The RTE Generator however shall support an intermediate exchange format called `McSupportData` which is building the bridge between the ECU software and the final "A2L"-file needed by the measurement and calibration tools. The details about the `McSupportData` format and the involved methodology are described in the Basic Software Module Description Template document [9].

In this section the requirements on the RTE Generator are collected which elements shall be provided in the McSupportData element.

4.2.8.4.1 Export of the *McSupportData*

Figure 4.28 shows the structure of the McSupportData element. The McSupportData element and its sub-content is part of the Implementation element. In case of the RTE this is the BswImplementation element which is generated / updated by the RTE Generator in the Generation Phase (see [SWS_Rte_05086] in chapter 3.4.2).

[SWS_Rte_05118] ⌈ The RTE Generator in Generation Phase shall create the McSupportData element as part of the BswImplementation description of the generated RTE. ⌋ *(SRS_Rte_00189)*

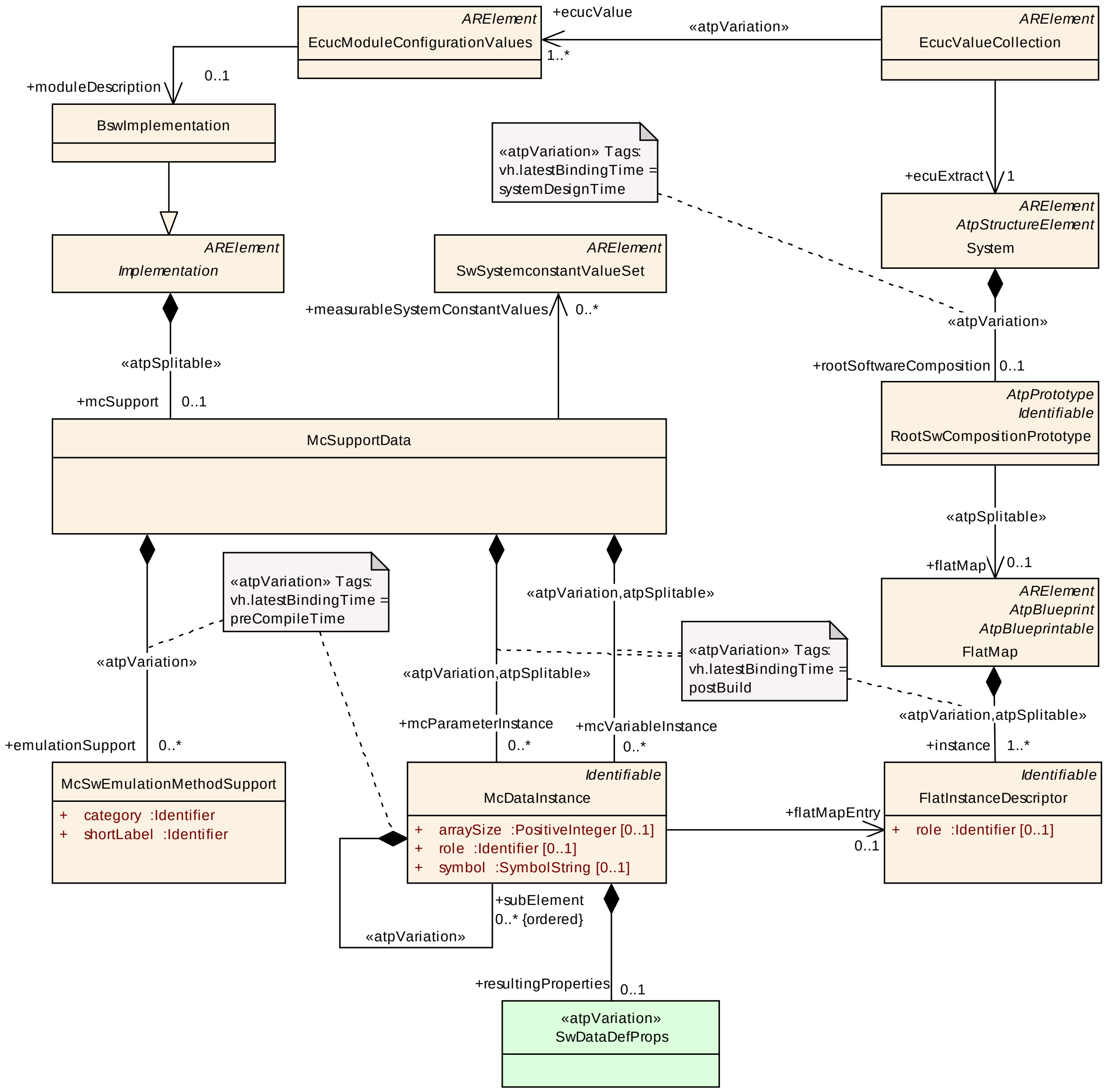


Figure 4.28: Overview of the McSupportData element

The individual measurable and calibratable data is described using the element McDataInstance. This is aggregated from McSupportData in the role mcVariableInstance (for measurement) or mcParameterInstance (for calibration).

Usage of the *FlatMap*

The FlatMap is part of the *Ecu Extract of System Description* and contains a collection of FlatInstanceDescriptor elements. The details of the FlatMap are described in the *Specification of the System Template* [8].

Common attributes of McDataInstance

The element McDataInstance specifies one element of the McSupportData. The following requirement specify common attributes which shall to be filled in a harmonized way.

[SWS_Rte_05130] ⌈ The RTE Generator shall use the shortName of the FlatInstanceDescriptor as the shortName of the McDataInstance. ⌋*(SRS_Rte_00189)*

[SWS_Rte_05131] ⌈ If the input element (e.g. ApplicationDataType or ImplementationDataType) has a category specified the category value shall be copied to the McDataInstance element. ⌋*(SRS_Rte_00189)*

[SWS_Rte_05132] ⌈ If the input element (e.g. ApplicationDataType or ImplementationDataType) specifies an array, the attribute arraySize of McDataInstance shall be set to the size of the array. ⌋*(SRS_Rte_00189)*

[SWS_Rte_05133] ⌈ If the input element (e.g. ApplicationDataType or ImplementationDataType) specifies a record, the McDataInstance shall aggregate the record element's parts as subElements of type McDataInstance. ⌋*(SRS_Rte_00189)*

[SWS_Rte_05119] ⌈ The McSupportData element and its sub-structure shall be self-contained in the sense that there is no need to deliver the whole upstream descriptions of the ECU (including the ECU Extract, Software Component descriptions, Basic Software Module descriptions, ECU Configuration Values descriptions, Flat Map, etc.) in order to later generate the final "A2L"-file. This means that the RTE Generator has to copy the required information from the upstream descriptions into the McSupportData element. ⌋*(SRS_Rte_00189)*

[SWS_Rte_05129] ⌈ The RTE Generator in Generation Phase shall export the effective SwDataDefProps (including all of the referenced and aggregated sub-elements like e.g. CompuMethod or SwRecordLayout) in the role resultingProperties for each McDataInstance after resolving the precedence rules defined in the SW-Component Template [2] chapter *Properties of Data Definitions*. Thereby the ImplementationDataType properties compuMethod and dataConstraint are not taken in consideration for effective SwDataDefProps of the McDataInstance due to their refinement nature of **C** and **AI**. ⌋*(SRS_Rte_00189)*

[SWS_Rte_05135] ⌈ If a ParameterDataPrototype is associated with a ParameterAccess the corresponding SwDataDefProps and their sub-structure shall be exported. ⌋ *(SRS_Rte_00189)*

For each flatMapEntry referencing to measurable or calibratible data prototype or measureable ModeDeclarationGroupPrototype the McDataInstance shall be generated in the McSupportData. Thereby the effected SwDataDefProps shall be taken from the data prototype according the precedence rules defined in the SWCT.

[SWS_Rte_08313] ⌈ The RTE Generator shall create McDataInstance element(s) in the McSupportData for each measurable or calibratible DataPrototype / ModeDeclarationGroupPrototype referenced by a FlatInstanceDescriptor. ⌋ *(SRS_Rte_00189)*

Explanation: In case of connected ports it may occur that the DataPrototype in the DataInterface of the PPortPrototype and the DataPrototype in the DataInterface of the RPortPrototype are referenced by FlatInstanceDescriptors. In this case its intended to get two McDataInstance in order to access the value by MCD system with two different names and may be with two different scaling (typically offset and resolution).

In case of composite data FlatInstanceDescriptors may point to one or several ApplicationCompositeElementDataPrototypes in order to define a individual name for each record or array element. Thereby it is even possible that a FlatInstanceDescriptor exists for the "whole" DataPrototype typed by an ApplicationCompositeDataType and additional FlatInstanceDescriptors exist for the ApplicationCompositeElementDataPrototypes of such DataPrototype.

In this case a McDataInstance as child of McSupportData exists due to the FlatInstanceDescriptors for the "whole" DataPrototype and additional McDataInstances as child of McSupportData exists for each FlatInstanceDescriptor pointing to a ApplicationCompositeElementDataPrototypes in the "whole" DataPrototypes type.

[SWS_Rte_08314] ⌈ If the input element is typed by an ApplicationDataType the subElements structure of the McDataInstance is determined by the ApplicationDataType. This means

- in case of ApplicationRecordDataType the number and shortName of the subElement is determined by the ApplicationRecordElement if [SWS_Rte_05133] and [SWS_Rte_08316] is applied,
- in case of ApplicationArrayDataType the number of the subElements is determined by the ApplicationArrayElement if [SWS_Rte_08315] is applied,
- in case of a ApplicationPrimitiveDataType, inclusive compound primitives, no subElements are applicable.

⌋ *(SRS_Rte_00189)*

[SWS_Rte_08315] ⌈ If the input element (e.g. ApplicationDataType or ImplementationDataType) specifies an array, the McDataInstance shall aggregate subElementss for each array element. The McDataInstance.subElements.symbol shall express the array index in the C-notation. (e.g. [0], [4]). ⌋*(SRS_Rte_00189)*

[SWS_Rte_08316] ⌈ If the input element (e.g. ApplicationDataType or ImplementationDataType) specifies a record and no FlatInstanceDescriptor is defined for the record element, the McDataInstance.subElement shortName shall be set copied either from the related ApplicationRecordElement. Or from the ImplementationDataTypeElement if no ApplicationDataType is typing the DataPrototype. The McDataInstance.subElement.symbol is set to the related ImplementationDataTypeElement.shortName ⌋*(SRS_Rte_00189)*

General handling of the symbol attribute: The concatenation of all symbol strings starting from the root element over the hierarchy of McDataInstances shall represent the full combined symbol in the programming language for all hierarchy levels in the McDataInstance tree. When the concatenation is applied the subElements of McDataInstances of category STRUCTURE are separated by a dot.

[SWS_Rte_08317] ⌈ The RTE Generator shall document the Rte internal grouping of measurement and calibration data in composite data datatypes in each symbol attribute of the McDataInstances representing the data which is grouped.

This means the RTE Generator has to document the insertion of structures for Rte internal purpose in the symbol attribute of the related McDataInstance. For instance if the Rte groups a set of measurable inside a Rte internal structure (here called RteInternalBuffer) the McDataInstance.symbol of the first measurable child element carries the information about the internal structure element. e.g. McDataInstance.shortName: "MyMeasurable" McDataInstance.symbol: "RteInternalBuffer.measurable1"
⌋*(SRS_Rte_00189)*

4.2.8.4.2 Export of Measurement information

Sender-Receiver communication

[SWS_Rte_05120] ⌈ If the swCalibrationAccess of a VariableDataPrototype used in an interface of a sender-receiver port of a SwComponentPrototype is set to readOnly or readWrite and RteMeasurementSupport is set to *true* the RTE Generator shall create a McDataInstance element with

- symbol set to the C-symbol name used for the allocation (see also [SWS_Rte_03900])
- flatMapEntry referencing to the corresponding FlatInstanceDescriptor element of the VariableDataPrototype

⌋*(SRS_Rte_00153, SRS_Rte_00189)*

Client-Server communication

[SWS_Rte_05121] ⌈ If the swCalibrationAccess of an ArgumentDataPrototype used in an interface of a client-server port of a SwComponentPrototype is set to readOnly and RteMeasurementSupport is set to *true* the RTE Generator shall create a McDataInstance element with

- symbol set to the C-symbol name used for the allocation (see also [SWS_Rte_03901])
- flatMapEntry referencing to the corresponding FlatInstanceDescriptor element of the ArgumentDataPrototype

⌋*(SRS_Rte_00153, SRS_Rte_00189)*

[SWS_Rte_05172] ⌈ If the measurement of client-server communication is ignored due to requirement [SWS_Rte_05170] the corresponding McDataInstance in the McSupportData shall have a resultingProperties swCalibrationAccess set to notAccessible. ⌋*(SRS_Rte_00153)*

Mode Switch Communication

[SWS_Rte_06702] ⌈ If the swCalibrationAccess of a ModeDeclarationGroupPrototype used in an interface of a mode switch port of a SwComponentPrototype is set to readOnly and RteMeasurementSupport is set to *true* the RTE Generator shall create three McDataInstance elements with

- symbol set to the C-symbol name used for the allocation (see also [SWS_Rte_06700])
- flatMapEntry referencing to the corresponding FlatInstanceDescriptor element of the ModeDeclarationGroupPrototype

Thereby the McDataInstance element corresponding to the

- *current mode* has to reference the FlatInstanceDescriptor which role attribute is set to CURRENT_MODE,
- *previous mode* has to reference the FlatInstanceDescriptor which role attribute is set to PREVIOUS_MODE and
- *next mode* has to reference the FlatInstanceDescriptor which role attribute is set to NEXT_MODE

⌋*(SRS_Rte_00153, SRS_Rte_00189)*

Please note that the resultingProperties of the McDataInstance elements corresponding to the ModeDeclarationGroupPrototype may get associated with a CompuMethod if a CompuMethod is defined at the FlatInstanceDescriptor due to [SWS_Rte_05129]. Those CompuMethod may specify a literal display of the measured modes.

InterRunnableVariable

[SWS_Rte_05122] ⌈ If the swCalibrationAccess of a VariableDataPrototype in the role implicitInterRunnableVariable or explicitInterRunnableVariable is set to readOnly or readWrite and RteMeasurementSupport is set to *true* the RTE Generator shall create a McDataInstance element with

- symbol set to the C-symbol name used for the allocation (see also [SWS_Rte_03902])
- flatMapEntry referencing to the corresponding FlatInstanceDescriptor element of the VariableDataPrototype

⌋*(SRS_Rte_00153, SRS_Rte_00189)*

PerInstanceMemory

[SWS_Rte_05123] ⌈ If the swCalibrationAccess of a VariableDataPrototype in the role arTypedPerInstanceMemory is set to readOnly or readWrite and RteMeasurementSupport is set to *true* the RTE Generator shall create a McDataInstance element with

- symbol set to the C-symbol name used for the allocation (see also [SWS_Rte_07160])
- flatMapEntry referencing to the corresponding FlatInstanceDescriptor element of the VariableDataPrototype

⌋*(SRS_Rte_00153, SRS_Rte_00189)*

Nv RAM Block

[SWS_Rte_05124] ⌈ If the swCalibrationAccess of a VariableDataPrototype in the role ramBlock of a NvBlockSwComponentType's NvBlockDescriptor is set to readOnly or readWrite and RteMeasurementSupport is set to *true* the RTE Generator shall create a McDataInstance element with

- symbol set to the C-symbol name used for the allocation (see also [SWS_Rte_07174])
- flatMapEntry referencing to the corresponding FlatInstanceDescriptor element of the NvBlockSwComponentType

⌋*(SRS_Rte_00153, SRS_Rte_00189)*

Non Volatile Data communication

[SWS_Rte_05125] ⌈ If the swCalibrationAccess of a VariableDataPrototype used in an NvDataInterface of a non volatile data port of a SwComponentPrototype is set to readOnly or readWrite and RteMeasurementSupport is set to *true* the RTE Generator shall create a McDataInstance element with

- symbol set to the C-symbol name used for the allocation (see also [SWS_Rte_07197])

- flatMapEntry referencing to the corresponding FlatInstanceDescriptor element of the VariableDataPrototype

⌋(*SRS_Rte_00153, SRS_Rte_00189*)

4.2.8.4.3 Export Calibration information

Calibration can be either actively supported by the RTE using the pre-defined calibration mechanisms of section 4.2.8.3.5 or calibration can be transparent to the RTE. In both cases the location and attributes of the calibratable data has to be provided by the RTE Generator in the Generation Phase in order to support the setup of the measurement and calibration tools.

ParameterDataPrototypes of ParameterSwComponentType

[SWS_Rte_05126] ⌈ For each ParameterDataPrototype in a PortPrototype of a ParameterSwComponentType with the swCalibrationAccess set to readOnly or readWrite an entry in the McSupportData with the role mcParameterInstance shall be created with the following attributes:

- symbol set to the C-symbol name used for the allocation
- flatMapEntry referencing to the corresponding FlatInstanceDescriptor element of the ParameterDataPrototype

⌋(*SRS_Rte_00189*)

Shared ParameterDataPrototypes

[SWS_Rte_05127] ⌈ For each ParameterDataPrototype of a AtomicSwComponentType's SwcInternalBehavior aggregated in the role sharedParameter with the swCalibrationAccess set to readOnly or readWrite an entry in the McSupportData with the role mcParameterInstance shall be created with the following attributes:

- symbol set to the C-symbol name used for the allocation
- flatMapEntry referencing to the corresponding FlatInstanceDescriptor element of the ParameterDataPrototype

⌋(*SRS_Rte_00189*)

Instance specific ParameterDataPrototypes

[SWS_Rte_05128] ⌈ For each ParameterDataPrototype of a AtomicSwComponentType's SwcInternalBehavior aggregated in the role perInstanceParameter with the swCalibrationAccess set to readOnly or readWrite an entry in the McSupportData with the role mcParameterInstance shall be created with the following attributes:

- symbol set to the C-symbol name used for the allocation

- flatMapEntry referencing to the corresponding FlatInstanceDescriptor element of the ParameterDataPrototype

⌋(SRS_Rte_00189)

[SWS_Rte_07097] ⌈ For each ParameterDataPrototype of a BswModuleDescription's BswInternalBehavior aggregated in the role perInstanceParameter with the swCalibrationAccess set to readOnly or readWrite an entry in the McSupportData with the role mcParameterInstance shall be created with the following attributes:

- symbol set to the C-symbol name used for the allocation
- flatMapEntry referencing to the corresponding FlatInstanceDescriptor element of the ParameterDataPrototype

⌋(SRS_Rte_00189)

NvRom Block

[SWS_Rte_05136] ⌈ If the swCalibrationAccess of a ParameterDataPrototype in the role romBlock is set to readOnly or readWrite an entry in the McSupportData with the role mcParameterInstance shall be created with the following attributes:

- symbol set to the C-symbol name used for the allocation in [SWS_Rte_07033]
- flatMapEntry referencing to the corresponding FlatInstanceDescriptor element of the ParameterDataPrototype

⌋(SRS_Rte_00153, SRS_Rte_00189)

4.2.8.4.4 Export of the Calibration Method

The RTE does provide several Software Emulation Methods which can be selected in the Ecu Configuration of the RTE (see section 7.3).

Which Software Emulation Method has been used for a particular RTE Generation shall be documented in the McSupportData in order to allow measurement and calibration tools to support the RTE's Software Emulation Methods. Additionally it is also possible for an RTE Vendor to add custom Software Emulation Methods which needs to be documented as well. The structure of the McSwEmulationMethodSupport is shown in figure 4.29.

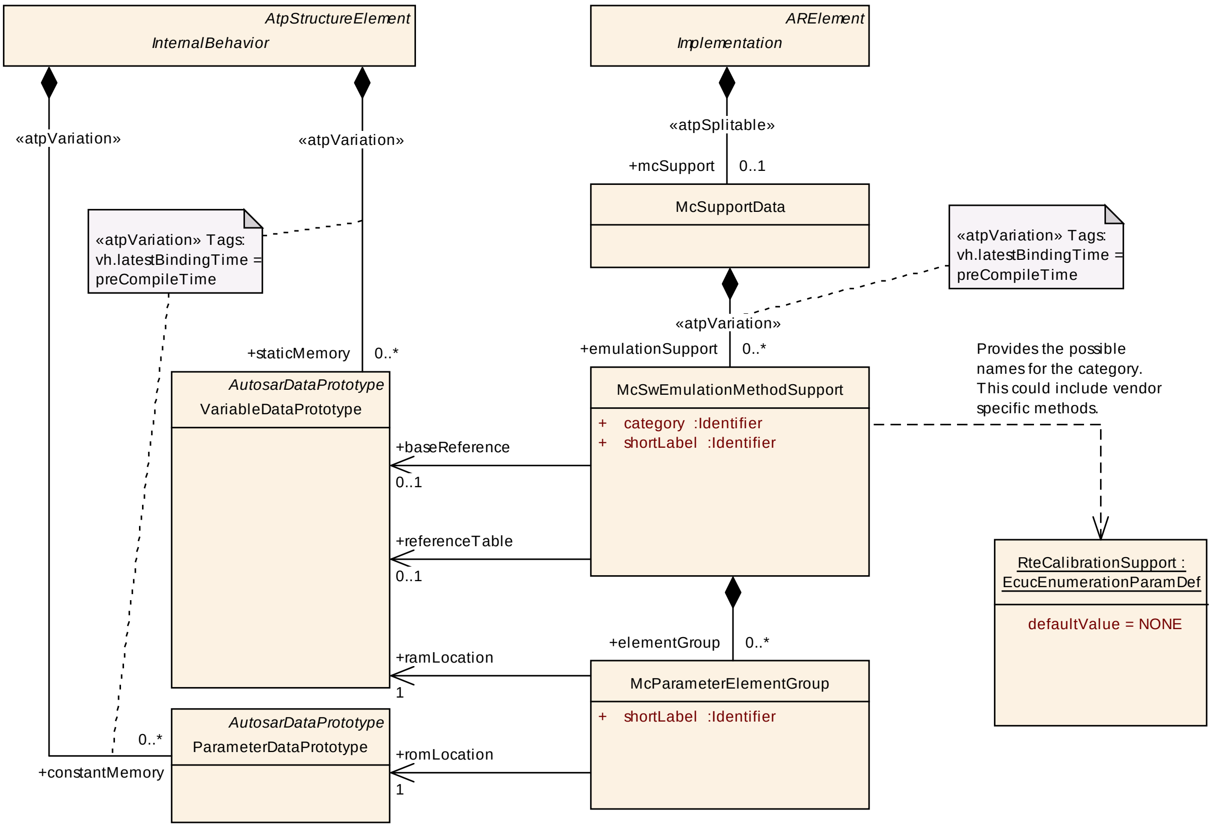


Figure 4.29: Structure of the `McSwEmulationMethodSupport` element

[SWS_Rte_05137] ⌈ The RTE Generator in Generation Phase shall create the `McSwEmulationMethodSupport` element as part of the `McSupportData` description of the generated RTE. ⌋ *(SRS_Rte_00189)*

[SWS_Rte_05138] ⌈ The RTE Generator in Generation Phase shall set the value of the `category` attribute of `McSwEmulationMethodSupport` element according to the implemented Software Emulation Method based on the Ecu configuration parameter `RteCalibrationSupport`:

- `NONE`
- `SINGLE_POINTERED`
- `DOUBLE_POINTERED`
- `INITIALIZED_RAM`
- *custom category name*: vendor specific Software Emulation Method

⌋ *(SRS_Rte_00189)*

The description of the generated structures is using the existing mechanisms already available in the Basic Software Module Description Template [9].

Description of ParameterElementGroup

For the description of the ParameterElementGroup an ImplementationDataType representing a structure of the group is created ([SWS_Rte_05139]).

[SWS_Rte_05139] ⌈ For each generated ParameterElementGroup an ImplementationDataType shall be created. The contained ParameterDataPrototypes are aggregated with the role subElement as ImplementationDataTypeElement.
⌋ *(SRS_Rte_00189)*

In the example figure 4.30 the ImplementationDataTypes are called RteMcSupportGroupType1 and RteMcSupportGroupType2.

McSupport description of the InitRam parameter method

For the description of the InitRam parameter method the specific ParameterElementGroups allocated in ram and rom are specified ([SWS_Rte_05140] and [SWS_Rte_05141]). Then the collection and correspondence of these groups is specified (in [SWS_Rte_05142]).

[SWS_Rte_05140] ⌈ If the RTE Generator is configured to support the (INITIALIZED_RAM) method the RTE Generator in generation phase shall generate for each ParameterElementGroup a ParameterDataPrototype with the role constantMemory in the InternalBehavior of the RTE's Basic Software Module Description. The ParameterDataPrototype shall have a reference to the corresponding ImplementationDataType from [SWS_Rte_05139] with the role type.
⌋ *(SRS_Rte_00189)*

[SWS_Rte_05141] ⌈ If the RTE Generator is configured to support the (INITIALIZED_RAM) method the RTE Generator in generation phase shall generate for each ParameterElementGroup a VariableDataPrototype with the role staticMemory in the InternalBehavior of the RTE's Basic Software Module Description. The VariableDataPrototype shall have a reference to the corresponding ImplementationDataType from [SWS_Rte_05139] with the role type.
⌋ *(SRS_Rte_00189)*

[SWS_Rte_05142] ⌈ If the RTE Generator is configured to support the (INITIALIZED_RAM) method the RTE Generator in generation phase shall generate for each ParameterElementGroup a McParameterElementGroup with the role elementGroup in the McSwEmulationMethodSupport [SWS_Rte_05137] element.

- The McParameterElementGroup shall have a reference to the corresponding ParameterDataPrototype from [SWS_Rte_05140] with the role romLocation.
- The McParameterElementGroup shall have a reference to the corresponding VariableDataPrototype from [SWS_Rte_05141] with the role ramLocation.

⌋ *(SRS_Rte_00189)*

McSupport description of the Single pointered method

For the description of the Single pointered method the specific ParameterElementGroups allocated in rom are specified ([SWS_Rte_05143]). Then an array data type is specified which contains as many number of elements (void pointers) as there are ParameterElementGroups ([SWS_Rte_05144]). Then the instance of this array is specified in ram ([SWS_Rte_05152]) and referenced from the McSwEmulationMethodSupport ([SWS_Rte_05153]). The actual values for each array element are specified as references to the ParameterElementGroup prototypes ([SWS_Rte_05154]).

[SWS_Rte_05143] ⌈ If the RTE Generator is configured to support the (SINGLE_POINTERED) method the RTE Generator in generation phase shall generate for each ParameterElementGroup a ParameterDataPrototype with the role constantMemory in the InternalBehavior of the RTE's Basic Software Module Description. The ParameterDataPrototype shall have a reference to the corresponding ImplementationDataType from [SWS_Rte_05139] with the role type. ⌋*(SRS_Rte_00189)*

[SWS_Rte_05144] ⌈ If the RTE Generator is configured to support the (SINGLE_POINTERED) method the RTE Generator in generation phase shall generate an ImplementationDataType with one ImplementationDataTypeElement in the role subElement.

- The ImplementationDataTypeElement shall have the attribute arraySize set to the number of ParameterElementGroups from [SWS_Rte_05139].
- The ImplementationDataTypeElement shall have a SwDataDefProps element with a reference to an ImplementationDataType representing a *void pointer*, in the role implementationDataType.

⌋*(SRS_Rte_00189)*

[SWS_Rte_05152] ⌈ If the RTE Generator is configured to support the (SINGLE_POINTERED) method the RTE Generator in generation phase shall generate a VariableDataPrototype with the role staticMemory in the InternalBehavior of the RTE's Basic Software Module Description. The VariableDataPrototype shall have a reference to the ImplementationDataType from [SWS_Rte_05144] with the role type. ⌋*(SRS_Rte_00189)*

[SWS_Rte_05153] ⌈ If the RTE Generator is configured to support the (SINGLE_POINTERED) method the RTE Generator in generation phase shall generate a reference from the McSwEmulationMethodSupport [SWS_Rte_05137] element to the VariableDataPrototype [SWS_Rte_05152] in the role referenceTable. ⌋*(SRS_Rte_00189)*

[SWS_Rte_05154] ⌈ If the RTE Generator is configured to support the (SINGLE_POINTERED) method the RTE Generator in generation phase shall generate an ArrayValueSpecification as the initValue of the array [SWS_Rte_05152] and for each ParameterElementGroup a ReferenceValueSpecification element in the ArrayValueSpecification defining the references to the individual ParameterElementGroup prototypes [SWS_Rte_05143]. ⌋*(SRS_Rte_00189)*

McSupport description of the Double pointered method

The description of the Double pointered method is quite similar to the Single pointered method, but the allocation to ram and rom is different and it allocates the additional pointer parameter. The specific ParameterElementGroups allocated in rom are specified ([SWS_Rte_05155]). Then an array data type is specified which contains as many number of elements (void pointers) as there are ParameterElementGroups ([SWS_Rte_05156]). Then the instance of this array is specified in rom ([SWS_Rte_05157]) and referenced from the McSwEmulationMethodSupport ([SWS_Rte_05158]). The actual values for each array element are specified as references to the ParameterElementGroup prototypes ([SWS_Rte_05159]). Then the type of the base pointer is then created ([SWS_Rte_05160]) and an instance is allocated in ram ([SWS_Rte_05161]). The reference is initialized to the array in rom ([SWS_Rte_05162]).

[SWS_Rte_05155] ⌈ If the RTE Generator is configured to support the (DOUBLE_POINTERED) method the RTE Generator in generation phase shall generate for each ParameterElementGroup a ParameterDataPrototype with the role constantMemory in the InternalBehavior of the RTE's Basic Software Module Description. The ParameterDataPrototype shall have a reference to the corresponding ImplementationDataType from [SWS_Rte_05139] with the role type. ⌋*(SRS_Rte_00189)*

In the example figure 4.30 the ParameterDataPrototypes are called RteMcSupportParamGroup1 and RteMcSupportParamGroup1.

[SWS_Rte_05156] ⌈ If the RTE Generator is configured to support the (DOUBLE_POINTERED) method the RTE Generator in generation phase shall generate an ImplementationDataType with one ImplementationDataTypeElement in the role subElement.

- The ImplementationDataTypeElement shall be of category ARRAY with the attribute arraySize set to the number of ParameterElementGroups from [SWS_Rte_05139].
- The ImplementationDataTypeElement shall have a SwDataDefProps element with a reference to an ImplementationDataType representing a *void pointer*, in the role implementationDataType.

⌋*(SRS_Rte_00189)*

In the example figure 4.30 the ImplementationDataType is called RteMcSupportPointerTableType.

[SWS_Rte_05157] ⌈ If the RTE Generator is configured to support the (DOUBLE_POINTERED) method the RTE Generator in generation phase shall generate a ParameterDataPrototype with the role constantMemory in the InternalBehavior of the RTE's Basic Software Module Description. The ParameterDataPrototype shall have a reference to the ImplementationDataType from [SWS_Rte_05156] with the role type. ⌋*(SRS_Rte_00189)*

In the example figure 4.30 the ParameterDataPrototype is called RteMcSupportPointerTable.

[SWS_Rte_05158] ⌈ If the RTE Generator is configured to support the (DOUBLE_POINTERED) method the RTE Generator in generation phase shall generate a reference from the McSwEmulationMethodSupport [SWS_Rte_05137] element to the ParameterDataPrototype [SWS_Rte_05157] in the role referenceTable. ⌋*(SRS_Rte_00189)*

[SWS_Rte_05159] ⌈ If the RTE Generator is configured to support the (DOUBLE_POINTERED) method the RTE Generator in generation phase shall generate an ArrayValueSpecification as the initValue of the array [SWS_Rte_05157] and for each ParameterElementGroup a ReferenceValueSpecification element in the ArrayValueSpecification defining the references to the individual ParameterElementGroup prototypes [SWS_Rte_05155]. ⌋*(SRS_Rte_00189)*

In the example figure 4.30 the ArrayValueSpecification is called RteMcSupportPointerTableInit. The ReferenceValueSpecifications are called RteMcSupportParamGroup1Ref and RteMcSupportParamGroup2Ref.

[SWS_Rte_05160] ⌈ If the RTE Generator is configured to support the (DOUBLE_POINTERED) method the RTE Generator in generation phase shall generate an ImplementationDataType with one ImplementationDataTypeElement being a reference to the array type from [SWS_Rte_05156]. ⌋*(SRS_Rte_00189)*

In the example figure 4.30 the ImplementationDataType is called RteMcSupportBasePointerType.

[SWS_Rte_05161] ⌈ If the RTE Generator is configured to support the (DOUBLE_POINTERED) method the RTE Generator in generation phase shall generate a VariableDataPrototype with the role staticMemory in the InternalBehavior of the RTE's Basic Software Module Description. The VariableDataPrototype shall have a reference to the ImplementationDataType from [SWS_Rte_05160] with the role type. ⌋*(SRS_Rte_00189)*

In the example figure 4.30 the VariableDataPrototype is called RteMcSupportBasePointer.

[SWS_Rte_05162] ⌈ If the RTE Generator is configured to support the (DOUBLE_POINTERED) method the RTE Generator in generation phase shall generate a ReferenceValueSpecification to the array from [SWS_Rte_05157] as the initValue of the reference [SWS_Rte_05161]. ⌋*(SRS_Rte_00189)*

In the example figure 4.30 the ReferenceValueSpecification is called RteMcSupportBasePointerInit.

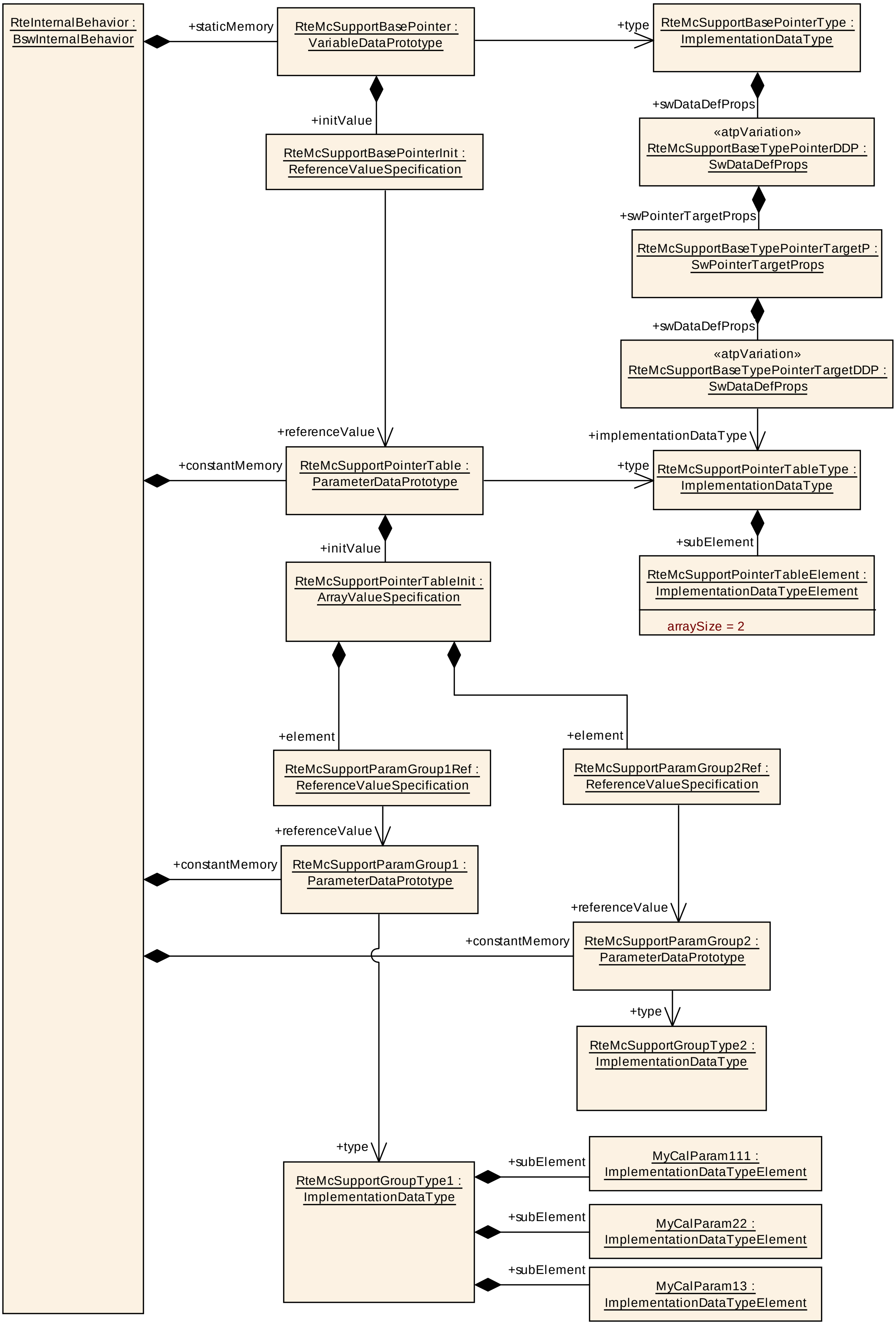

RteInternalBehavior : BswInternalBehavior
+staticMemory
RteMcSupportBasePointer : VariableDataPrototype
+type
RteMcSupportBasePointerType : ImplementationDataType
+initValue
RteMcSupportBasePointerInit : ReferenceValueSpecification
+swDataDefProps
«atpVariation» RteMcSupportBaseTypePointerDDP : SwDataDefProps
+swPointerTargetProps
RteMcSupportBaseTypePointerTargetP : SwPointerTargetProps
+swDataDefProps
«atpVariation» RteMcSupportBaseTypePointerTargetDDP : SwDataDefProps
+referenceValue
+implementationDataType
+constantMemory
RteMcSupportPointerTable : ParameterDataPrototype
+type
RteMcSupportPointerTableType : ImplementationDataType
+initValue
+subElement
RteMcSupportPointerTableInit : ArrayValueSpecification
RteMcSupportPointerTableElement : ImplementationDataTypeElement
arraySize = 2
+element
+element
RteMcSupportParamGroup1Ref : ReferenceValueSpecification
RteMcSupportParamGroup2Ref : ReferenceValueSpecification
+referenceValue
+constantMemory
RteMcSupportParamGroup1 : ParameterDataPrototype
+referenceValue
+constantMemory
RteMcSupportParamGroup2 : ParameterDataPrototype
+type
RteMcSupportGroupType2 : ImplementationDataType
+type
+subElement
MyCalParam111 : ImplementationDataTypeElement
RteMcSupportGroupType1 : ImplementationDataType
+subElement
MyCalParam22 : ImplementationDataTypeElement
+subElement
MyCalParam13 : ImplementationDataTypeElement

Figure 4.30: Example of the structure for Double Pointered Method

4.2.8.4.5 Export of Variant Handling

The Rte Generator shall provide information on values of system constants. The values are part of the input information and need to be collected and copied into a dedicated artifact to be delivered with the McSupportData.

[SWS_Rte_05168] ⌈ The Rte Generator in generation phase shall create an elements of type SwSystemconstantValueSet and create copies of all system constant values found in the input information of type SwSystemconstValue where the referenced SwSystemconst element has the swCalibrationAccess set to readOnly. ⌋*(SRS_Rte_00153, SRS_Rte_00191)*

In case the SwSystemconstValue is subject to variability and the variability can be resolved during Rte generation phase

[SWS_Rte_05176] ⌈ If a SwSystemconst with swCalibrationAccess set to readOnly has an assigned SwSystemconstValue which is subject to variability with the latest binding time SystemDesignTime or CodeGenerationTime the related SwSystemconstValue copy in the SwSystemconstantValueSet according to [SWS_Rte_05168] shall contain the resolved value. ⌋*(SRS_Rte_00153, SRS_Rte_00191)*

[SWS_Rte_05174] ⌈ If a SwSystemconst with swCalibrationAccess set to readOnly has an assigned SwSystemconstValue which is subject to variability with the latest binding time PreCompileTime the related SwSystemconstValue copy in the SwSystemconstantValueSet according to [SWS_Rte_05168] shall have an AttributeValueVariationPoint. The *PreBuild* conditions of the AttributeValueVariationPoint shall correspond to the *PreBuild* conditions of the input SwSystemconstValue's conditions. ⌋*(SRS_Rte_00153, SRS_Rte_00191)*

[SWS_Rte_05169] ⌈ The Rte Generator in generation phase shall create a reference from the McSupportData element ([SWS_Rte_05118]) to the SwSystemconstantValueSet element ([SWS_Rte_05168]). ⌋*(SRS_Rte_00153, SRS_Rte_00191)*

In case the RTE Generator implements variability on a element which is accessible by a MCD system the related existence condition has to be documented in the McSupportData structure as well.

[SWS_Rte_05175] ⌈ If an element in the McSupportData is related to an element in the input configuration which is subject to variability with the latest binding time PreCompileTime or *PostBuild* the RTE Generator shall add a VariationPoint for such element. The *PreBuild* and *PostBuild* conditions of the VariationPoint shall correspond to the *PreBuild* and *PostBuild* conditions of the input element's conditions. ⌋*(SRS_Rte_00153, SRS_Rte_00191)*

4.2.9 Access to NVRAM data

4.2.9.1 General

There are different methods available for AUTOSAR SW-Cs to access data stored in NVRAM:

- **"Calibration data"** – Calibrations can be stored in NVRAM, but are not modified during a "normal" execution of the ECU. Calibrations are usually directly read from their memory location, but can also be read from a RAM buffer when the access time needs to be optimized (e.g. for interpolation tables). They are described in section 4.2.8.
- **"Access to NVM blocks"** – This method uses `PerInstanceMemory` as a RAM mirror for the NVRAM blocks. While this method is efficient, its use is restricted.

  The NVRAM Manager [21] is a BSW module which provides services for SW-C to access NVRAM blocks during runtime. The NVM block data is not accessed directly, but through a RAM mirror, which can be a `PerInstanceMemory` instantiated by the RTE, or a SW-C internal buffer. When this method is used, the RTE does not provide any data consistency mechanisms (i.e. different runnables from the SW-C and the NVM can access the RAM mirror concurrently without being protected by the RTE).

  Note:

  This mechanism permits efficient usage of NVRAM data, but requires the SW-C designer to take care that accesses to the `PerInstanceMemory` from different task contexts don't cause data inconsistencies. The "Access to NVM blocks" should not be used in multi core environments. In AUTOSAR release 4.0, it can not be expected that the NVRAM Manager can access the `PerInstanceMemory` of another core. The presence of a shared memory section is not required by AUTOSAR. Only in the case of `arTypedPerInstanceMemory`, a `SwDataDefProps` item is available to assign the `PerInstanceMemory` to a shared memory section.
- **"Access to NVRAM data with a `NvBlockSwComponentType`** – The data is accessed through a `NvDataInterface` connected to a `NvBlockSwComponentType`s. This access is modeled at the VFB level, and, when necessary, protected by the RTE against concurrent accesses. It will be described further in this section.

4.2.9.2 Usage of the NvBlockSwComponentType

The code of NvBlock `SwComponentPrototype`s is implemented by the RTE Generator. `NvBlockSwComponentType`s provide a port interface for the access and management of data stored in NVRAM.

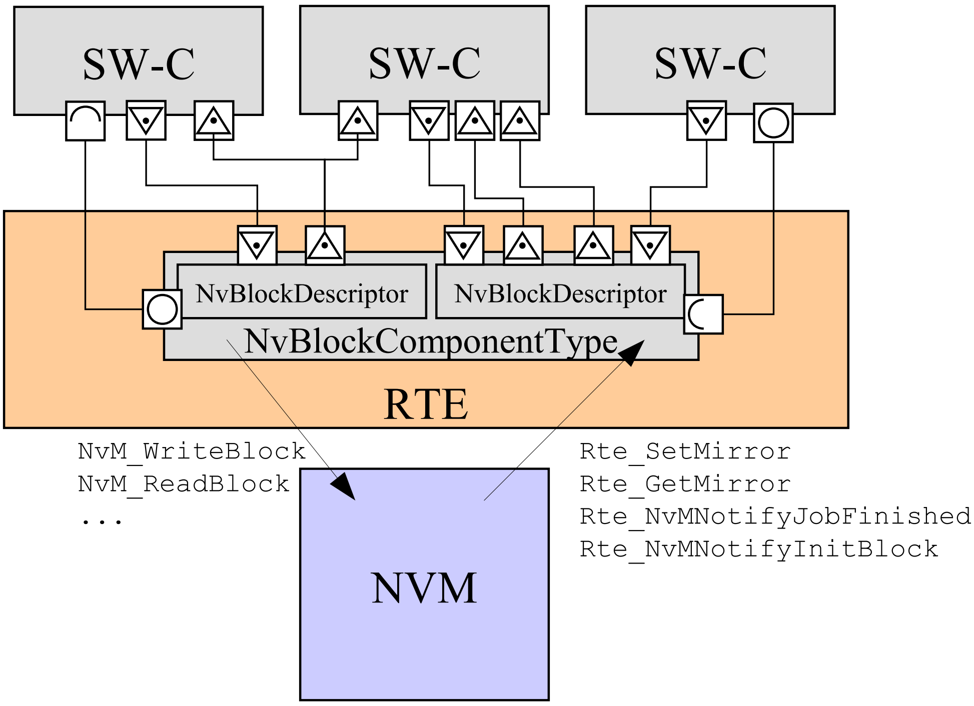


Figure 4.31: Connection to the NvBlockSwComponentType

Figure 4.31 illustrates the usage of a NvBlockSwComponentType. Depending on the use-case SW-Cs can be connected to a NvBlockSwComponentType in different ways. For example by S/R communication only or by S/R and C/S communication. S/R communication is used to provide access to NV data and C/S communication is used for the management of NV data. Managing NV data by SW-Cs is useful in order to copy data of the RAM mirror to NVM blocks and vice versa at certain points in time (SW-Cs are clients). Additionally SW-Cs can get notifications from NVM (SW-Cs are servers).

In the following sections the requirements for the usage of NvBlockSwComponentType will be given.

[SWS_Rte_07301] ⌈ Several AUTOSAR SW-Cs (and also several instances of a AUTOSAR SW-C) shall be able to read the same VariableDataPrototypes of a NvBlockSwComponentType. ⌋*(SRS_Rte_00176)*

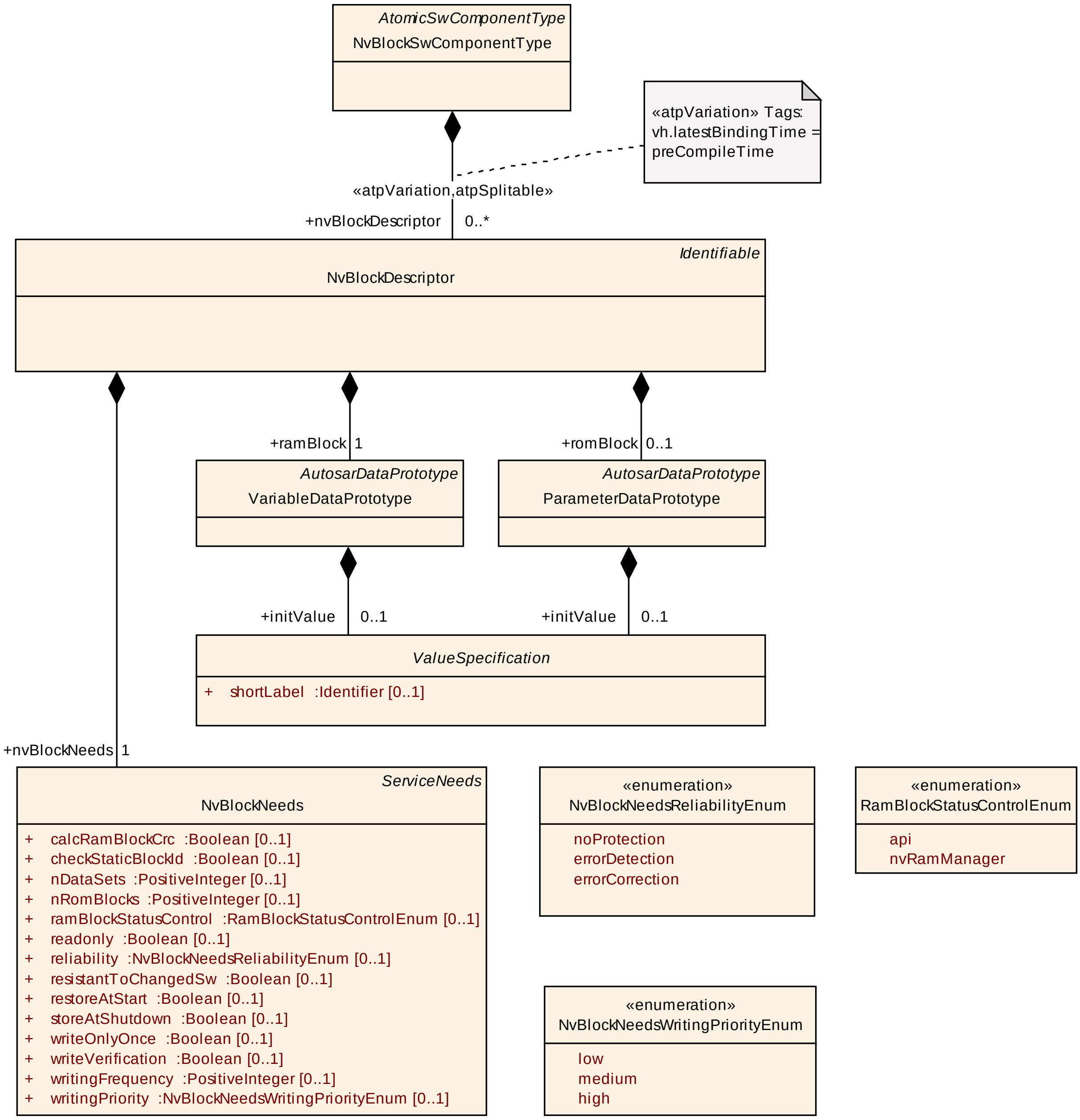


Figure 4.32: `NvBlockSwComponentType` and `NvBlockDescriptor`

A `NvBlockSwComponentType` contains multiple `NvBlockDescriptor`s. Each of these `NvBlockDescriptor` is associated to exactly one NVM block.

A `NvBlockDescriptor` contains a `VariableDataPrototype` which acts as a RAM mirror for the NVM block, and possibly a `ParameterDataPrototype` to act as the default ROM value for the NVM block.

[SWS_Rte_07353] ⌈ The RTE Generator shall reject configurations where a `NvBlockDescriptor` of a `NvBlockSwComponentType` contains a `romBlock` whose data type is not compatible with the type of the `ramBlock`. ⌋*(SRS_Rte_00177, SRS_Rte_00018)*

[SWS_Rte_07303] ⌈ The RTE shall allocate memory for the ramBlock VariableDataPrototype of the NvBlockDescriptor instances. ⌋*(SRS_Rte_00177)*

[SWS_Rte_07632] ⌈ The variables allocated for the ramBlocks shall be initialized if the general initialization conditions in [SWS_Rte_07046] are fulfilled. The initialization as to be applied during Rte_Start and Rte_RestartPartition depending from the configured RteInitializationStrategy. ⌋*(SRS_Rte_00177)*

Note: When blocks are configured to be read by NvM_ReadAll, the initialization may erase the value read by the NVM. These blocks should not have an initValue.

[SWS_Rte_07355] ⌈ For each NvBlockDescriptor with a romBlock ParameterDataPrototype, the RTE shall allocate a constant ROM block. ⌋*(SRS_Rte_00177)*

[SWS_Rte_07633] ⌈ The constants allocated for the romBlocks shall be initialized to the value of the initValue, if they have an initValue. ⌋*(SRS_Rte_00177)*

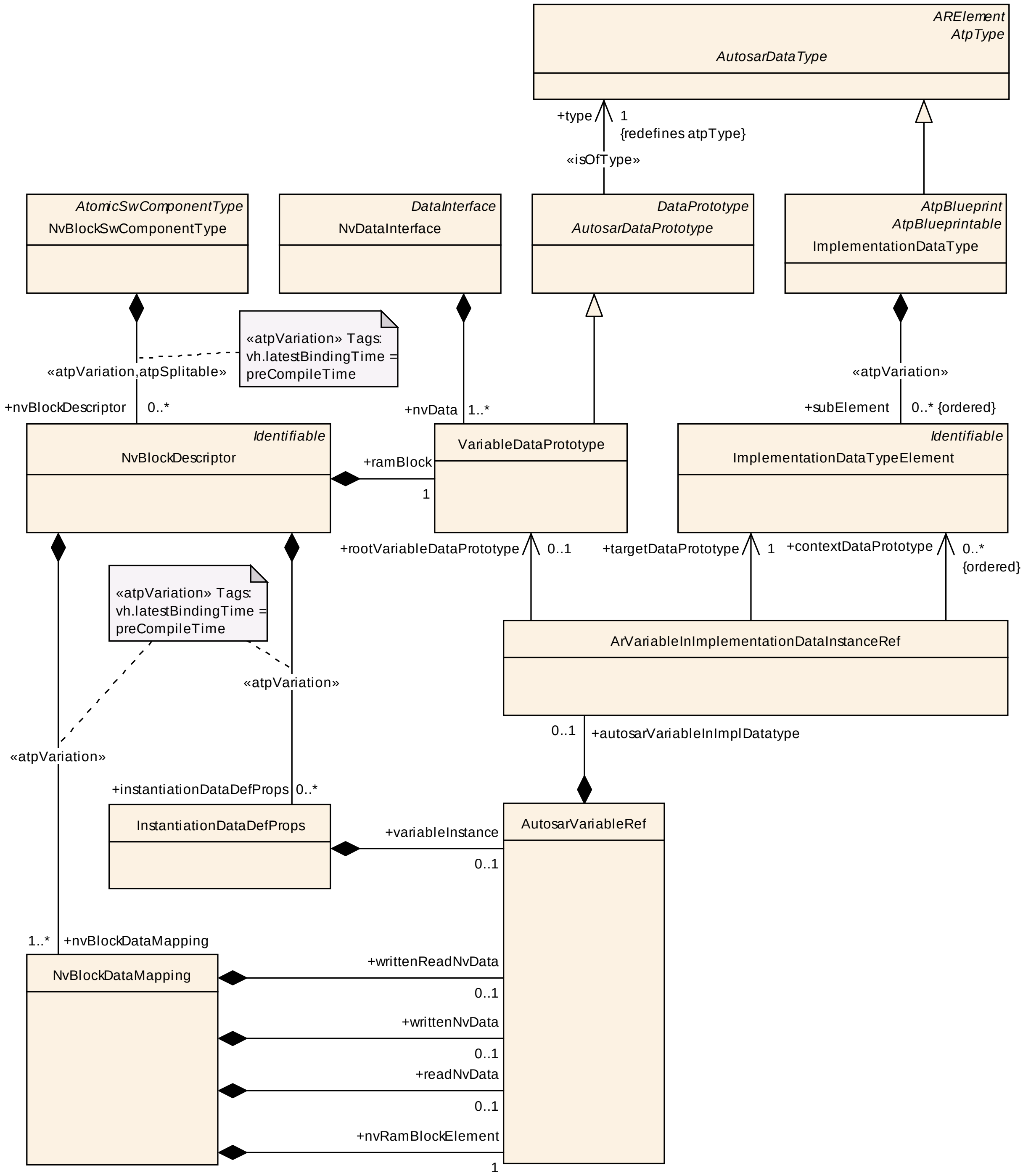


Figure 4.33: `NvBlockDataMapping`

For each element stored in the NvM block of a `NvBlockDescriptor`, there should be one `NvBlockDataMapping` to associate the `VariableDataPrototype`s of the ports used for read and write access and the `VariableDataPrototype` defining the location of the element in the `ramBlock`. Thereby the `ImplementationDataType`s of the `VariableDataPrototype`s have to compatible.

[SWS_Rte_07621] ⌈ The RTE Generator shall reject configurations where [constr_2013] or [constr_1285] is violated. ⌋*(SRS_Rte_00018)*

Note: This is required to ensure that the default values in romBlock are structurally matching data in the ramBlock and therefore can be copied to the ramBlock in case that the callback Rte_NvMNotifyInitBlock of the related NvBlock is called.

[SWS_Rte_07343] ⌈ The RTE Generator shall reject configurations where a VariableDataPrototype instance in the role ramBlock is accessed by SW-C instances of different partitions. ⌋*(SRS_Rte_00177, SRS_Rte_00018)*

The rational for [SWS_Rte_07343] is to allow the implementation of cleanup activities in case of termination or restart of a partition. These cleanup activities may require to invalidate the RAM mirror or reload data from the NVRAM device, which would impact other partitions if a the ramBlock is shared by SW-Cs of different partitions.

A NvBlockSwComponentType can be used to reduce the quantity of NVRAM blocks needed on an ECU:

- the same block can be used to store different flags or other small DataElements;
- the same DataElement can be used by different SW-Cs or different instances of a SW-C.

It also permits to simplify processes and algorithms when it must be guaranteed that two SW-Cs of an ECU use the same NVRAM data.

Note: this feature can increase the RAM usage of the ECU because it forces the NVRAM Manager to instantiate an additional RAM buffer. However, when the same DataElements have to be shared between SW-Cs, it reduces the number of RAM mirrors needed to be instantiated by the RTE, and can reduce the overall RAM usage of the ECU.

[SWS_Rte_07356] ⌈ The RTE Generator shall reject configurations where a VariableDataPrototype referenced by a NvDataInterface has a *queued* swImplPolicy. ⌋*(SRS_Rte_00018)*

[SWS_Rte_07357] ⌈ The RTE Generator shall reject configurations where a DataReceivedEvent is referenced by a WaitPoint and references a VariableDataPrototype referenced by a NvDataInterface. ⌋*(SRS_Rte_00018)*

[constr_9011] NvMBlockDescriptor related to a RAM Block of a NvBlockSwComponentType shall use NvmBlockUseSyncMechanism ⌈ The NVM block associated to the NvBlockDescriptors of a NvBlockSwComponentType shall be configured with the NvMBlockUseSyncMechanism feature enabled, and the NvMWriteRamBlockToNvCallback and NvMReadRamBlockFromNvCallback parameters set to the Rte_GetMirror and Rte_SetMirror API of the NvBlockDescriptor. ⌋

An NvBlockSwComponentType may have unconnected p-ports or r-ports (see [SWS_Rte_01329]).

[SWS_Rte_07669] ⌈ An NvBlockSwComponentType with an unconnected r-port shall behave as if no updated data were received for VariableDataPrototypes this unconnected r-port. ⌋*(SRS_Rte_00139)*

4.2.9.3 Interface of the NvBlockSwComponentType

4.2.9.3.1 Access to the NVRAM data

The NvBlockSwComponentType provides PPortPrototypes and RPortPrototypes with an NvDataInterface data Sender-Receiver semantic to read the value of the NVRAM data or write the new value.

Like the SenderReceiverInterfaces, each of these NvDataInterfaces can provide access to multiple VariableDataPrototypes.

The same Rte_Read, Rte_IRead, Rte_DRead, Rte_Write, Rte_IWrite, Rte_IWriteRef APIs are used to access these VariableDataPrototypes as for SenderReceiverInterfaces.

[SWS_Rte_07667] ⌈ The RTE Generator shall reject configurations where an r-port typed with an NvDataInterface is not connected and no NvRequireComSpec with a initValue are provided for each VariableDataPrototype of this NvDataInterface. This requirement does not apply if the r-port belongs to a NvBlockSwComponentType. ⌋*(SRS_Rte_00018, SRS_Rte_00139)*

[SWS_Rte_07667] is required to avoid unconnected r-port without a defined initValue. Please note that for NvBlockSwComponent unconnected r-ports without init values are not a fault because the init values are defined in the NvBlockDescriptors ramBlock (see as well [SWS_Rte_07632], [SWS_Rte_07669])

[SWS_Rte_07668] ⌈ The RTE shall initialize the VariableDataPrototypes of an r-port according to the initValue of the r-port's NvRequireComSpec referring to the VariableDataPrototype. ⌋*(SRS_Rte_00139, SRS_Rte_00108, SRS_Rte_00068)*

4.2.9.3.2 NVM interfaces

The NvBlockSwComponentType can also have ports used for NV data management and typed by Client-Server interfaces compatible to the NVRAM Manager [21] standardized one. Note that these ports shall always have a PortInterface with the attribute isService set to FALSE.

The standardized NvM Client-Server interfaces are composed as follows:

- NvMService

  This interface is used to send commands to the NVM. The NvBlockSwComponentType provides a server port intended to be used by the SW-C users of this NvBlockSwComponentType.

- NvMNotifyJobFinished

  This interface is used by the NVM to notify the end of job. The NvBlockSwComponentType provides a server port intended to be used by the NVM, and client ports intended to be connected to the SW-C users of this NvBlockSwComponentType.

- NvMNotifyInitBlock

  This interface is used by the NVM to request users to provide the default values in the RAM mirror. The NvBlockSwComponentType provides a server port intended to be used by the NVM, and client ports intended to be connected to the SW-C users of this NvBlockSwComponentType.

- NvMAdmin

  This interface is used to order some administrative operations to the NVM. The NvBlockSwComponentType provides a server port intended to be used by the SW-C users of this NvBlockSwComponentType.

For the implementation of NvBlockSwComponentTypes that have NvM service ports the RTE has to call the API of NvM. In order to access NvM API the NvM.h file has to be included.

[SWS_Rte_08063] ⌈ The RTE shall include the NvM.h file, if it has to access NvM API. ⌋*(SRS_Rte_00177)*

Note: no restrictions have been added to the NVM interfaces. However, some operations of the NVM might require cooperation between the different users of the NvBlockSwComponentType. For example, a ReadBlock operation will erase the RAM mirror, which might affect multiple SW-Cs.

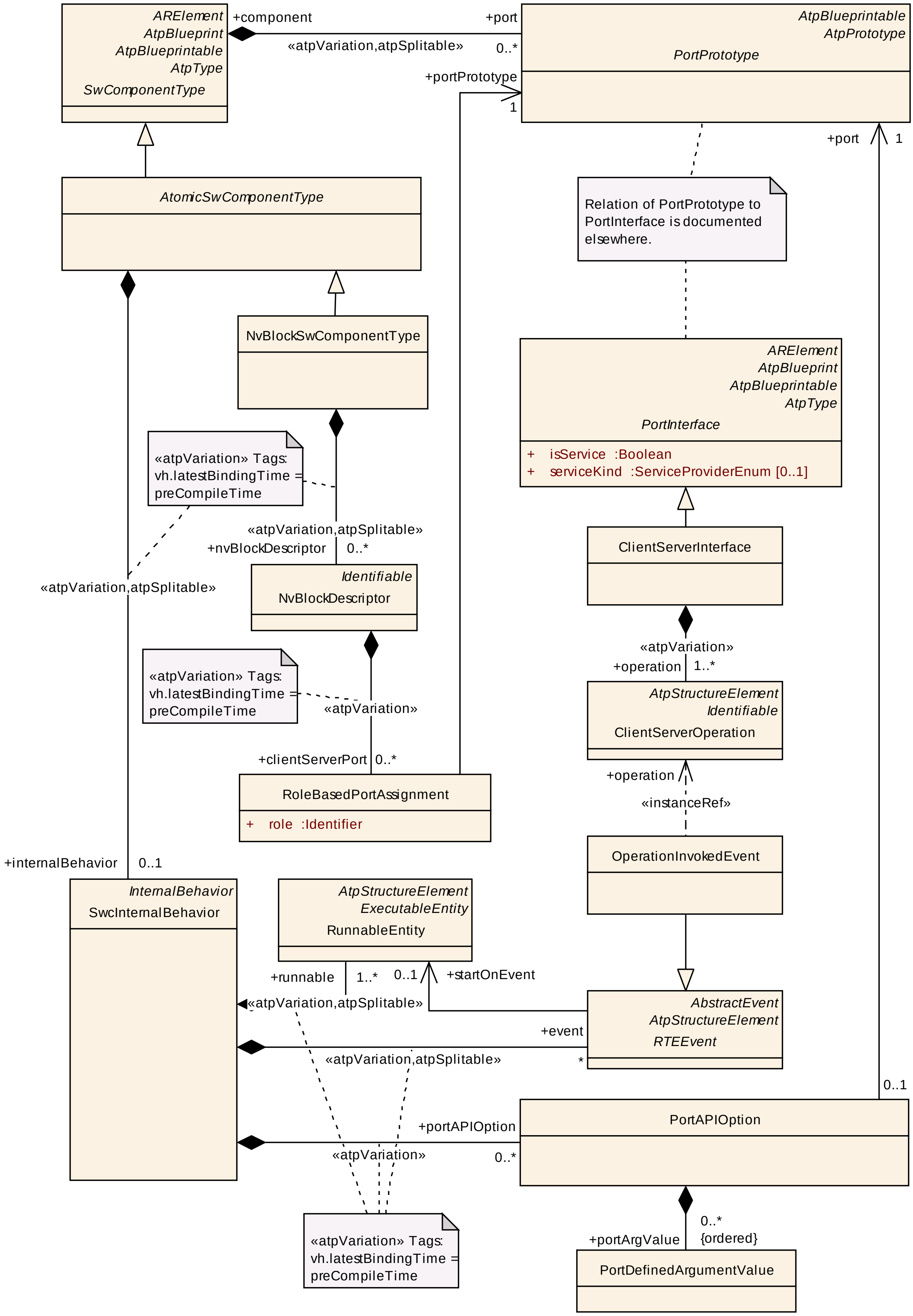


Figure 4.34: SwcInternalBehavior of NvBlockSwComponentTypes

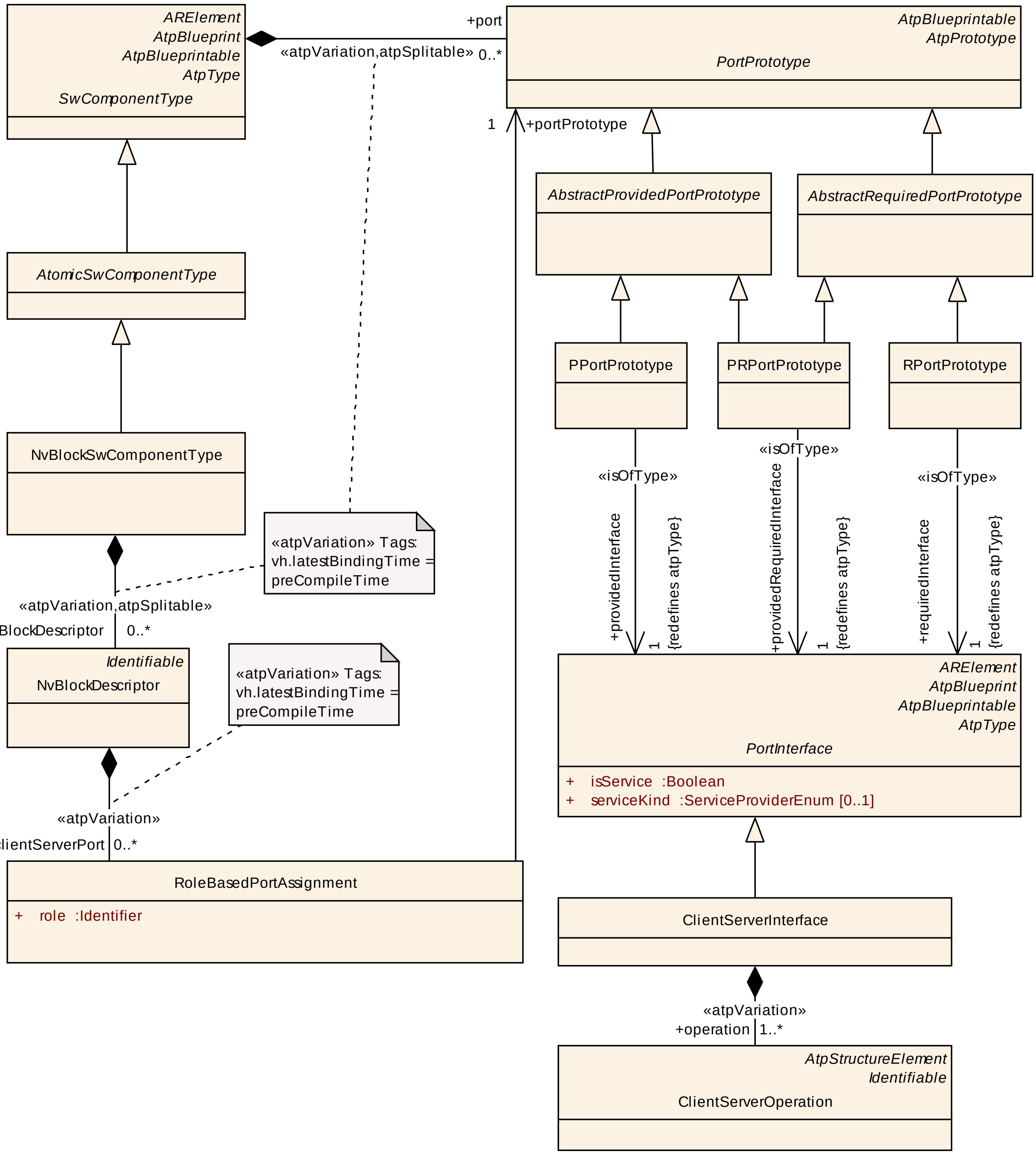


Figure 4.35: NVM notifications

The requests received from the SW-C side are forwarded by the NvBlockSwComponentType's runnables to the NVM module, using the NVM C API indicated by the RoleBasedPortAssignment. See figure 4.34.

Notifications received from the NVM are forwarded to all the SW-C connected to the notification interfaces of the NvBlockSwComponentType with a RoleBasedPortAssignment of the corresponding type. See figure 4.35.

[SWS_Rte_07398] ⌈ The RTE Generator shall implement runnables for each connected server port of a NvBlockSwComponentType. ⌋ *(SRS_Rte_00177)*

[SWS_Rte_07399] ⌈ The NvBlockSwComponentType's runnables used as servers connected to the SW-C shall forward the request to the NVM by calling the associated NVM API. ⌋*(SRS_Rte_00177)*

[SWS_Rte_08064] ⌈ The symbol attribute of RunnableEntitys of NvBlockSwComponentTypes shall be used by the RTE generator to identify the to be called NvM API function (see [constr_1234] in software component template [2]). ⌋*(SRS_Rte_00177)*

Note: A BlockId PortDefinedArgumentValue is also provided to runnables and used as a first argument in the NVM APIs.

4.2.9.4 Data Consistency

A VariableDataPrototype contained in a NvBlockSwComponentType is accessed when SW-Cs read the value or write a new value. It is also accessed by the NVM when read or write requests are processed by the NVM for the associated block.

The NVM does not access directly the VariableDataPrototypes, but shall use the Rte_GetMirror, and Rte_SetMirror APIs specified in section 5.9.4

The RTE has to ensure the data consistency of the VariableDataPrototypes, with any of the data consistency mechanisms defined in section 4.2.5. Depending on the user's input, an efficient scheduling with the use of implicit APIs should permit a low resources (OS resources, RAM, and code) implementation.

4.3 Communication Paradigms

AUTOSAR supports two basic communication paradigms: Client-Server and Sender-Receiver. AUTOSAR software-components communicate through well defined ports and the behavior is statically defined by attributes. Some attributes are defined on the modeling level and others are closely related to the network topology and must be defined on the implementation level.

The RTE provides the implementation of these communication paradigms. For inter-ECU communication the RTE uses the functionalities provided by COM. For inter-Partition communication (within the same ECU) the RTE uses functionalities provided by the IOC module. For intra-Partition the RTE provides the functionality on its own.

With Sender-Receiver communication there are two main principles: Data Distribution and Event Distribution. When data is distributed, the last received value is of interest (last-is-best semantics). When events are distributed the whole history of received events is of interest, hence they must be queued on receiver side. Therefore the software implementation policy can be queued or non queued. This is stated in the swImplPolicy attribute of the SwDataDefProps, which can have the value queued (corresponding to event distribution with a queue) or standard (corresponding to last-is-best data distribution). If a data element has event semantics, the swImplPolicy is set to queued. The other possible values of this attribute correspond to data semantics.

[SWS_Rte_07192] ⌈ The RTE generator shall reject the configuration when an r-port is connected to an r-port or a p-port is connected to a p-port with an AssemblySwConnector⌋*(SRS_Rte_00018)*

For example, a require port (r-port) of a component typed by an AUTOSAR sender-receiver interface can read data elements of this interface. A provide port (p-port) of a component typed by an AUTOSAR sender-receiver interface can write data elements of this interface.

[SWS_Rte_07006] ⌈ The RTE generator shall reject the configuration violating the [constr_1032], so when an r-port is connected to a p-port or a p-port is connected to an r-port with a DelegationSwConnector. ⌋*(SRS_Rte_00018)*

[SWS_Rte_08767] ⌈ In case of functionality depending on attributes of ComSpecs the RTE Generator shall consider only the ComSpecs defined in the context of AtomicSwComponentTypes or ParameterSwComponentTypes. ⌋*(SRS_Rte_00018)*

4.3.1 Sender-Receiver

4.3.1.1 Introduction

Sender-receiver communication involves the transmission and reception of signals consisting of atomic data elements that are sent by one component and received by one or more components. A sender-receiver interface can contain multiple data elements. Sender-receiver communication is one-way - any reply sent by the receiver is sent as a separate sender-receiver communication.

A require port (r-port) of a component typed by an AUTOSAR sender-receiver interface can read data elements of this interface. A provide port (p-port) of a component typed by an AUTOSAR sender-receiver interface can write data elements of this interface.

4.3.1.2 Receive Modes

The RTE supports multiple receive modes for passing data to receivers. The four possible receive modes are:

- **"Implicit data read access"** – when the receiver's runnable executes it shall have access to a "copy" of the data that remains unchanged during the execution of the runnable.

  [SWS_Rte_06000] ⌈ For data elements specified with implicit data read access, the RTE shall make the receive data available to the runnable through the semantics of a copy. ⌋*(SRS_Rte_00128, SRS_Rte_00019)*

[SWS_Rte_06001] ⌈ For data elements specified with implicit data read access the receive data shall not change during execution of the runnable. ⌋*(SRS_Rte_00128)*

When “implicit data read access” is used the RTE is required to make the data available as a “copy”. It is not necessarily required to use a unique copy for each runnable. Thus the RTE may use a unique copy of the data for each runnable entity or may, if several runnables (even from different components) need the same data, share the same copy between runnables. Runnable entities can only share a copy of the same data when the scheduling structure can make sure the contents of the data is protected from modification by any other party.

[SWS_Rte_06004] ⌈ The RTE shall read the data elements specified with implicit data read access before the associated runnable entity is invoked. ⌋*(SRS_Rte_00128)*

Composite data types shall be handled in the same way as primitive data types, i.e. RTE shall make a “copy” available for the `RunnableEntity`.

[SWS_Rte_06003] ⌈ The “implicit data read access” receive mode shall be valid for all categories of runnable entity (i.e. 1A, 1B and 2). ⌋*(SRS_Rte_00134)*

- **“Explicit data read access”** – the RTE generator creates a non-blocking API call to enable a receiver to poll (and read) data. This receive mode is an “explicit” mode since an explicit API call is invoked by the receiver.

  The explicit “data read access” receive mode is only valid for category 1B or 2 runnable entities [SRS_Rte_00134].

- **“wake up of wait point”** – the RTE generator creates a blocking API call that the receiver invokes to read data.

  [SWS_Rte_06002] ⌈ The “wake up of wait point” receive mode shall support a timeout to prevent infinite blocking if no data is available. ⌋*(SRS_Rte_00109, SRS_Rte_00069)*

  The “wake up of wait point” receive mode is inherently only valid for a category 2 runnable entity.

  A category 2 runnable entity is required since the implementation may need to suspend execution of the caller if no data is available.

- **“activation of runnable entity”** – the receiving runnable entity is invoked automatically by the RTE whenever new data is available. To access the new data, the runnable entity either has to use “implicit data read access” or “explicit data read access”, i.e. invoke an `Rte_IRead`, `Rte_Read`, `Rte_DRead` or `Rte_Receive` call, depending on the input configuration. This receive mode differs from “implicit data read access” since the receiver is invoked by the RTE in response to a `DataReceivedEvent`.

  [SWS_Rte_06007] ⌈ The “activation of runnable entity” receive mode shall be valid for category 1A, 1B and 2 runnable entities. ⌋*(SRS_Rte_00134)*

The validity of receive modes in conjunction with different categories of runnable entity is summarized in Table 4.9.

Receive Mode	Cat 1A	Cat 1B	Cat 2
Implicit Data Read Access	Yes	Yes	Yes
Explicit Data Read Access	No	Yes	Yes
Wake up of wait point	No	No	Yes
Activation of runnable entity	Yes	Yes	Yes

Table 4.9: Receive mode validity

The category of a runnable entity is not an inherent property but is instead determined by the features of the runnable. Thus the presence of explicit API calls makes the runnable at least category 1B and the presence of a `WaitPoint` forces the runnable to be category 2.

4.3.1.2.1 Applicability

The different receive modes are not just used for receivers in sender-receiver communication. The same semantics are also applied in the following situations:

- **Success feedback** – The mechanism used to return transmission acknowledgments to a component. See Section 5.2.6.9.
- **Asynchronous client-server result** – The mechanism used to return the result of an asynchronous client-server call to a component. See Section 5.7.5.4.

4.3.1.2.2 Representation in the Software Component Template

The following list serves as a reference for how the RTE Generator determines the Receive Mode from its input [SRS_Rte_00109]. Note that references to "the `VariableDataPrototype`" within this sub-section will implicitly mean "the `VariableDataPrototype` for which the API is being generated".

- **"wake up of wait point"** – A `VariableAccess` in the `dataReceivePointByValue` or `dataReceivePointByArgument` role references a `VariableDataPrototype` and a `WaitPoint` references a `DataReceivedEvent` which in turn references the same `VariableDataPrototype`.
- **"activation of runnable entity"** – a *DataReceivedEvent* references the `VariableDataPrototype` and a runnable entity to start when the data is received.
- **"explicit data read access"** – A `VariableAccess` in the `dataReceivePointByValue` or `dataReceivePointByArgument` role references the `VariableDataPrototype`.

- **"implicit data read access"** – A VariableAccess in the dataReadAccess role references the VariableDataPrototype.

It is possible to combine certain access methods; for example 'activation of runnable entity' can be combined with 'explicit' or 'implicit' data read access (indeed, one of these pairings is necessary to cause API generation to actually *read* the datum) but it is an input error if 'activation of runnable entity' and 'wakeup of wait point' are combined (i.e. a WaitPoint references a DataReceivedEvent that references a runnable entity). It is also possible to specify both implicit and explicit data read access simultaneously.

For details of the semantics of "implicit data read access" and "explicit data read access" see Section 4.3.1.5.

4.3.1.3 Multiple Data Elements

A sender-receiver interface can contain one or more data elements. The transmission and reception of elements is independent – each data element, e.g. AUTOSAR signal, can be considered to form a separate logical data channel between the "provide" port and a "require" port.

[SWS_Rte_06008] ⌈ Each data element in a sender-receiver interface shall be sent separately. ⌋*(SRS_Rte_00089)*

Example 4.4

Consider an interface that has two data elements, `speed` and `freq` and that a component template defines a provide port that is typed by the interface. The RTE generator will then create two API calls; one to transmit `speed` and another to transmit `freq`.

Where it is important that multiple data elements are sent simultaneously they should be combined into a composite data structure (Section 4.3.1.11.1). The sender then creates an instance of the data structure which is filled with the required data before the RTE is invoked to transmit the data.

4.3.1.3.1 Initial Values

[SWS_Rte_06009] ⌈ For each data element in an interface specified with data semantics, the RTE shall support the `initValue` attribute. ⌋*(SRS_Rte_00108)*

The `initValue` attribute is used to ensure that AUTOSAR software-components always access valid data even if no value has yet been received. This information is required for inter-ECU, inter-Partition, and intra-Partition communication. For inter-ECU communication initial values can be handled by COM but for intra-ECU communication RTE has to guarantee that `initValue` is handled.

In general, the specification of an initValue is mandatory for each data element prototype with data semantics, see [SWS_Rte_07642]. If all senders and receivers are located in the same partition, this restriction is relaxed, see [SWS_Rte_04501].

[SWS_Rte_06010] ⌈ The RTE shall use any specified initial value to prevent the receiver performing calculations based on invalid (i.e. uninitialized) values when the swImplPolicy is not queued and if the general initialization conditions in [SWS_Rte_07046] are fulfilled. ⌋*(SRS_Rte_00107)*

The above requirement ensures that RTE API calls return the initialized value until a "real" value has been received, possibly via the communication service. The requirement does *not* apply when "event" semantics are used since the implied state change when the event data is received will mean that the receiver will not start to process invalid data and would therefore never see the initialized value.

[SWS_Rte_04500] ⌈ An initial value cannot be specified when the implementation policy is set to 'queued' attribute is specified as true. ⌋*(SRS_Rte_00107)*

For senders, an initial value is not used directly by the RTE (since an AUTOSAR SW-C must supply a value using Rte_Send) however it may be needed to configure the communication service - for example, an un-initialised signal can be transmitted if multiple signals are mapped to a single frame and the communication service transmits the whole frame when any contained signal is sent by the application. Note that it is not the responsibility of the RTE generator to configure the communication service.

It is permitted for an initial value to be specified for either the sender or receiver. In this case the same value is used for both sides of the communication.

[SWS_Rte_04501] ⌈ If in context of one partition a sender specifies an initial value and the receiver does not (or *vice versa*) the same initial value is used for both sides of the communication. ⌋*(SRS_Rte_00108)*

It is also permitted for both sender and receiver to specify an initial value. In this case it is defined that the receiver's initial value is used by the RTE generator for both sides of the communication.

[SWS_Rte_04502] ⌈ If in context of one partition both receiver and sender specify an initial value the specification for the *receiver* takes priority. ⌋*(SRS_Rte_00108)*

4.3.1.4 Multiple Receivers and Senders

Sender-receiver communication is not restricted to communication connections between a single sender and a single receiver. Instead, sender receiver communication connection can have multiple senders ('n:1' communication) or multiple receivers ('1:m' communication) with the restrictions that multiple senders are not allowed for mode switch notifications, see metamodel restriction [SWS_Rte_02670].

The RTE does not impose any co-ordination on senders – the behavior of senders is independent of the behavior of other senders. For example, consider two senders A and B that both transmit data to the same receiver (i.e. 'n:1' communication). Transmissions by either sender can be made at any time and there is no requirement that the senders co-ordinate their transmission. However, while the RTE does not impose any co-ordination on the senders it does ensure that simultaneous transmissions do not conflict.

In the same way that the RTE does not impose any co-ordination on senders there is no co-ordination imposed on receivers. For example, consider two receivers P and Q that both receive the same data transmitted by a single sender (i.e. '1:m' communication). The RTE does not guarantee that multiple receivers see the data simultaneously even when all receivers are on the same ECU.

4.3.1.5 Implicit and Explicit Data Reception and Transmission

[SWS_Rte_06011] ⌈ The RTE shall support 'explicit' and 'implicit' data reception and transmission. ⌋*(SRS_Rte_00019, SRS_Rte_00098, SRS_Rte_00129, SRS_Rte_00128, SRS_Rte_00141)*

Implicit data access transmission means that a runnable does not actively initiate the reception or transmission of data. Instead, the required data is received automatically when the runnable starts and is made available for other runnables at the earliest when it terminates.

Explicit data reception and transmission means that a runnable employs an explicit API call to send or receive certain data elements. Depending on the category of the runnable and on the configuration of the according ports, these API calls can be either blocking or non-blocking.

4.3.1.5.1 Implicit

Implicit Read

For the implicit reading of data, `VariableAccess`es aggregated with a `dataReadAccess` role [SRS_Rte_00128], the data is made available when the runnable starts using the `semantics of a copy` operation and the RTE ensures that the 'copy' will not be modified until the runnable terminates.

When a runnable *R* is started, the RTE reads all `VariableDataPrototype`s referenced by a `VariableAccess` in the `dataReadAccess` role, if the data elements may be changed by other runnables a copy is created that will be available to runnable *R*. The runnable *R* can read the data element by using the RTE APIs for implicit read (see the API description in Section 5.6.18). That way, the data is guaranteed not to change (e.g. by write operations of other runnables) during the entire lifetime of *R*. If several runnables (even from different components) need the data, they can share the *same* buffer. This is only applicable when the scheduling structure can make sure the contents of the data is protected from modification by any other party.

Note that this concept implies that the runnable does in fact terminate. Therefore, while implicit read is allowed for category 1A and 1B runnable entities as well as category 2 only the former are guaranteed to have a finite execution time. A category 2 runnable that runs forever will not see any updated data.

VariableAccess in the dataReadAccess role is only allowed for VariableDataPrototypes with their swImplPolicy different from 'queued' ([constr_2020]).

Implicit Write

Implicit writing, VariableAccesses aggregated with a dataWriteAccess role [SRS_Rte_00129], is the opposite concept. VariableDataPrototypes referenced by a VariableAccess in the dataWriteAccess role are sent by the RTE after the runnable terminates. The runnable can write the data element by using the RTE APIs for implicit write (see the API description in Sect. 5.6.19 and 5.6.20). The sending is independent from the position in the execution flow in which the Rte_IWrite is performed inside the Runnable. When performing several write accesses during runnable execution to the same data element, only the last one will be recognized. Here we have a last-is-best semantics.

Note:
If a VariableDataPrototype is referenced by a VariableAccess in the dataWriteAccess role, but no RTE API for implicit write of this VariableDataPrototype is called during an execution of the runnable, an undefined value is written back when the runnable terminates.

[SWS_Rte_03570] ⌈ For VariableAccesses in the dataWriteAccess role the RTE shall make the sent data available to others (other runnables, other AUTOSAR SWCs, Basic SW, ..) with the semantics of a copy. ⌋*(SRS_Rte_00129)*

[SWS_Rte_03571] ⌈ For VariableAccesses in the dataWriteAccess role the RTE shall make the sent data available to others (other runnables, other AUTOSAR SWCs, Basic SW, ..) at the earliest when the runnable has terminated. ⌋*(SRS_Rte_00129)*

[SWS_Rte_03572] ⌈ For VariableAccesses in the dataWriteAccess role several accesses to the same VariableDataPrototype performed inside a runnable during one runnable execution shall lead to only one transmission of the VariableDataPrototype. ⌋*(SRS_Rte_00129)*

[SWS_Rte_03573] ⌈ If several VariableAccesses in the dataWriteAccess role referencing the same VariableDataPrototype are performed inside a runnable during the runnable execution, the RTE shall use the last value written. (last-is-best semantics) ⌋*(SRS_Rte_00129)*

A VariableAccess in the dataWriteAccess role is only sensible for runnable entities that are guaranteed to terminate, i.e. category 1A and 1B. If it is used for a category 2 runnable which does not terminate then no data write-back will occur.

[SWS_Rte_03574] ⌈ VariableAccess in the dataWriteAccess role shall be valid for all categories of runnable entity. ⌋*(SRS_Rte_00129, SRS_Rte_00134)*

To get common behavior in RTEs from different suppliers further requirements defining the semantic of implicit communication exist:

Please note that the behavior of Implicit Communication can be adjusted with ECU Configuration. For further information see section 7.8.

Implicit Communication Behavior in case of Incoherent Implicit Data Access

[SWS_Rte_03954] ⌈ The RTE generator shall use exactly one buffer to contain data copies of the same `VariableDataPrototype` per `Preemption Area` for the implementation of the `copy semantic` of `Incoherent Implicit Data Access`. ⌋*(SRS_Rte_00128, SRS_Rte_00129, SRS_Rte_00134)*

Requirement [SWS_Rte_03954] means that all runnable entities mapped to tasks of a `Preemption Area` with a `Incoherent Implicit Read Access` or `Incoherent Implicit Write Access` access the same buffers.

[SWS_Rte_03598] ⌈ For implicit communication, the RTE shall provide a single shared read/write buffer when no runnable entity mapped to tasks of the `Preemption Area` has `VariableAccess` in both `Incoherent Implicit Read Access` and `Incoherent Implicit Write Access` referencing the same `VariableDataPrototype`. ⌋*(SRS_Rte_00128, SRS_Rte_00129)*

If either the sender or the receiver uses a `Data Element with Status` and the other uses a `Data Element without Status`, a `Data Element with Status` can be implemented and casted in the component data structure when a pointer to a `Data Element without Status` is needed.

[SWS_Rte_03955] ⌈ For implicit communication, in case that dedicated `RPortPrototype` and `PPortPrototype` are used, separate read and write buffers shall be used when at least one `RunnableEntity` mapped to tasks of the `Preemption Area` has `Implicit Read Access` and `Implicit Write Access` referencing the same `VariableDataPrototype`. ⌋*(SRS_Rte_00128, SRS_Rte_00129)*

In the case that a `RunnableEntity` defines `dataWriteAccess` and `dataReadAccess` to the same `VariableDataPrototype` in the context of a `PRPortPrototype` [SWS_Rte_03955] does not apply. In such configuration the writing `RunnableEntity` immediately sees its own updates of the data values even before the `RunnableEntity` has terminated.

[SWS_Rte_08408] ⌈ If a `RunnableEntity` has both `dataWriteAccess` and `dataReadAccess` to a `VariableDataPrototype` in the context of a `PRPortPrototype` the result of the write access shall be immediately visible to subsequent read accesses from within the same `RunnableEntity`. ⌋*(SRS_Rte_00128, SRS_Rte_00129)*

Please note that the content of the write buffers are copied into the read buffer of the `Preemption Area` after the `RunnableEntity` with the write access terminates (see [SWS_Rte_07041]). Therefore the write buffer might be implemented as temporary buffer.

[SWS_Rte_03599] ⌈ For implicit communication with Incoherent Implicit Data Access all readers within a Preemption Area shall access the same buffer. ⌋*(SRS_Rte_00128)*

[SWS_Rte_03953] ⌈ For implicit communication with Incoherent Implicit Data Access all writers within a Preemption Area shall access the same buffer. ⌋*(SRS_Rte_00129)*

The content of a shared buffer (see [SWS_Rte_03598]) is not guaranteed to stay constant during the whole task since a writer will change the shared copy and hence readers mapped in the task after the writer will access the updated copy. When buffers are shared, written data is visible to other RunnableEntitys within the same execution of the task. However since no runnable within the task will both read and write the same buffer ([SWS_Rte_03598] and [SWS_Rte_03955]) consistency *within a runnable* is ensured.

When separate buffers used for implicit communication (see [SWS_Rte_03955]) any data written by a runnable is not visible (to either other RunnableEntitys or to the writing runnable) until the data is written back after the runnable has terminated.

Implicit Communication Behavior in case of Coherent Implicit Data Access

[SWS_Rte_07062] ⌈ The RTE generator shall use exactly one buffer to contain data copies of the same VariableDataPrototype per Coherency Group for the implementation of the copy semantic of Coherent Implicit Data Access. ⌋*(SRS_Rte_00128, SRS_Rte_00129, SRS_Rte_00134)*

Requirement [SWS_Rte_07062] means that all runnable entities with Coherent Implicit Data Accesses access the same buffers. Please note that it is only supported to group Implicit Read Accesses or Implicit Write Accesses of RunnableEntitys executed in the same OS Task. Therefore a Coherent Implicit Data Access results in a task local buffer as it was specified in previous AUTOSAR releases. With this means a backward compatible bahavior of the RTE can be ensured.

Please note that [SWS_Rte_03955] applies as well for Coherent Implicit Data Access. [SWS_Rte_07062] includes already that a single shared read/write buffer shall be used when no runnable entity has Coherent Implicit Read Access and Coherent Implicit Write Access belonging to the same Coherency Group.

Implicit Communication buffer handling

The Preemption Area specific buffer should not be updated or made available more often than required. The following requirements detail how to obtain that for read and write access.

[SWS_Rte_03956] ⌈ The content of a Preemption Area specific buffer used for an Incoherent Implicit Read Access to a VariableDataElement shall be filled with actual data by a copy action between the beginning of the task and the execution of the first RunnableEntity with access to this VariableDataElement in the task. ⌋*(SRS_Rte_00128)*

[SWS_Rte_07020] ⌈ If the RteImmediateBufferUpdate = TRUE is configured for a Incoherent Implicit Read Access to a VariableDataElement the content of a Preemption Area specific buffer used for that VariableAccess shall be filled with actual data by a copy action immediately before the RunnableEntity with the related implicit read access to the VariableDataElement starts. ⌋ *(SRS_Rte_00128)*

[SWS_Rte_07041] ⌈ The content of a separate write buffer (see [SWS_Rte_03955]) modified by a Incoherent Implicit Write Access of a RunnableEntity shall be made available to RunnableEntitys using a Implicit Read Access allocated in the **same** Preemption Area immediately after the execution of the RunnableEntity with the related Implicit Write Access to the VariableDataElement. ⌋ *(SRS_Rte_00129)*

[SWS_Rte_03957] ⌈ The content of a Preemption Area specific buffer modified by a Incoherent Implicit Write Access in one task shall be made available to RunnableEntitys using an Implicit Read Access allocated in **other** Preemption Areas at latest after the execution of the last RunnableEntity mapped to the task. ⌋ *(SRS_Rte_00129)*

[SWS_Rte_07021] ⌈ If the RteImmediateBufferUpdate = TRUE is configured for a Incoherent Implicit Write Access the content of a Preemption Area specific buffer shall be made available to RunnableEntitys using a Implicit Read Access allocated in **other** Preemption Areas immediately after the execution of the RunnableEntity with the related Implicit Write Access to the VariableDataElement. ⌋ *(SRS_Rte_00129)*

Note:
It's the semantic of implicit communication that a VariableAccess in the dataWriteAccess role is interpreted as writing the whole dataElement.

Explicit Schedule Points defined by RteOsSchedulePoints are placed between RunnableEntitys after the data written with implicit write access by the RunnableEntity are propagated to other RunnableEntitys and before the Preemption Area specific buffer used for a implicit read access of the successor RunnableEntity are filled with actual data by a copy action according [SWS_Rte_07020]. This ensures that the data produced by one RunnableEntity is propagated before RunnableEntitys assigned to other Os Tasks are activated due to Task scheduling caused by the explicit Schedule Point. See as well [SWS_Rte_07042] and [SWS_Rte_07043].

The requirements regarding buffer handling for implicit communication do not apply in case of filters. Buffer handling of RTE for filters is specified in chapter 4.3.1.9 (requirements: [SWS_Rte_08077], [SWS_Rte_08078] and [SWS_Rte_08079]).

Implicit Communication buffer handling for Coherent Implicit Data Access

[SWS_Rte_07063] ⌈ The content of a Coherency Group specific buffer used for an Coherent Implicit Read Access to one or more VariableDataElements shall be filled with actual data by a copy action between the beginning of the task and the execution of the first RunnableEntity in the task with a Coherent Implicit Read Access belonging to the Coherency Group. ⌋(SRS_Rte_00128)

[SWS_Rte_07064] ⌈ If the RteImmediateBufferUpdate = TRUE is configured for Coherent Implicit Read Accesses the content of a Coherency Group specific buffer used for these VariableAccesses shall be filled with actual data by a copy action immediately before the first RunnableEntity in the task with a Coherent Implicit Read Access belonging to the Coherency Group starts. ⌋(SRS_Rte_00128)

[SWS_Rte_07065] ⌈ The content of a separate write buffer (see [SWS_Rte_03955]) modified by a Coherent Implicit Write Access of a RunnableEntity shall be made available to RunnableEntitys using a Coherent Implicit Read Access belonging to the same Coherency Group immediately after the execution of the RunnableEntity with the related Coherent Implicit Write Access. ⌋(SRS_Rte_00129)

[SWS_Rte_07066] ⌈ The content of a Coherency Group specific buffer modified by Coherent Implicit Write Accesses in one task shall be made available to other RunnableEntitys at earliest after the execution of the last RunnableEntity with a Coherent Implicit Write Access belonging to this Coherency Group. ⌋(SRS_Rte_00129)

[SWS_Rte_07067] ⌈ The content of a Coherency Group specific buffer modified by Coherent Implicit Write Accesses in one task shall be made available to other RunnableEntitys at latest after the execution of the last RunnableEntity mapped to the task. ⌋(SRS_Rte_00129)

[SWS_Rte_07068] ⌈ If the RteImmediateBufferUpdate = TRUE is configured for a Coherent Implicit Write Accesses the content of a Coherency Group specific buffer modified by Coherent Implicit Write Accesses in one task shall be made available to other readers not belonging to this Coherency Group immediately after the execution of the last RunnableEntity with a Coherent Implicit Write Access belonging to this Coherency Group ⌋(SRS_Rte_00129)

Handling of ConsistencyNeeds

ConsistencyNeeds are not directly processed by the RTE Generator but providing an important information for the correct configuration of the RTE and OS with respect to preemption, RteEventToTaskMapping and RteImplicitCommunication. Therefore following constraints apply:

[constr_9001] Whole DataPrototypeGroup in role dpgRequiresCoherency shall be propagated coherently ⌈

All RunnableEntitys in a RunnableEntityGroup with dataWriteAccess to data belonging to the same DataPrototypeGroup in the role dpgRequiresCoherency shall

- Be mapped to the same OS Task

AND shall

- A) either be scheduled in a way that these RunnableEntitys can not be interrupted by RunnableEntitys with dataReadAccess to (more than one) data belonging to the DataPrototypeGroup.
- B) or the RteImplicitCommunication shall be configured to ensure a coherent propagation (RteCoherentAccess == true) for reading RunnableEntitys [5].

⌋

Please note that the interruption of RunnableEntitys and between RunnableEntitys depends from many factors like the configuration of the OS and the configuration of the RTE (e.g. RteOsSchedulePoint).

[constr_9002] The whole DataPrototypeGroup shall be read stable for the whole RunnableEntityGroup in the role regRequiresStability ⌈.

All RunnableEntitys with dataReadAccess to data belonging to the same DataPrototypeGroup and which are belonging to the same RunnableEntityGroup in the role regRequiresStability shall

- either be configured in a way that the chain of RunnableEntitys with dataReadAccess to the data of the DataPrototypeGroup can not be interrupted by any of the RunnableEntity(s) with dataWriteAccess to data of the DataPrototypeGroup
- or the RteImplicitCommunication shall be configured to ensure stable data values (RteCoherentAccess == true) for reading RunnableEntitys belonging to the RunnableEntityGroup.

⌋

Examples

Following examples shall illustrate how ConsistencyNeeds can be implemented with either scheduling or Coherency Groups.

Example 4.5

Common definition of PortInterfaces

In order to simplify the examples all PortInterfaces are of type SenderReceiverInterface and contain exactly one VariableDataPrototype with identical shortName. For example SenderReceiverInterface "A" contains VariableDataPrototype "A"

[5] RunnableEntitys with have as well dataWriteAccess to data belonging to the DataPrototypeGroup are excluded because inside the calculation chain the latest data values are visible

Additionally the shortName of the SenderReceiverInterface is identical to the shortName of the PortPrototype. For example PPortPrototype "A" is typed by SenderReceiverInterface "A".

Example 4.6

Stability need for received data

Setup of SWCs

ApplicationSwComponentType "ASWC_A" with the PPortPrototypes: "A","B"

and the RunnableEntity "ASWC_A_RUN1" which in turn has following dataWriteAccesses

- "DWP_ASWC_A_RUN1_A_A" referencing VariableDataPrototype "A" in PPortPrototype "A"
- "DWP_ASWC_A_RUN1_B_B" referencing VariableDataPrototype "B" in PPortPrototype "B"

ApplicationSwComponentType "ASWC_B" with the RPortPrototypes: "A","B"

and the RunnableEntity "ASWC_B_RUN1" which in turn has dataReadAccesses

- "DRP_ASWC_B_RUN1_A_A" referencing VariableDataPrototype "A" in RPortPrototype "A"
- "DRP_ASWC_B_RUN1_B_B" referencing VariableDataPrototype "B" in RPortPrototype "B"

ApplicationSwComponentType "ASWC_C" with the RPortPrototypes: "A","B"

and the RunnableEntity "ASWC_C_RUN1" which in turn has dataReadAccesses

- "DRP_ASWC_C_RUN1_A_A" referencing VariableDataPrototype "A" in RPortPrototype "A"
- "DRP_ASWC_C_RUN1_B_B" referencing VariableDataPrototype "B" in RPortPrototype "B"

The ConsistencyNeeds "CN_BC" defines a RunnableEntityGroup in the role regRequiresStability with the members "ASWC_B_RUN1", "ASWC_C_RUN1" In addition the ConsistencyNeeds "CN_BC" defines a DataPrototypeGroup in the role dpgDoesNotRequireCoherency to the VariableDataPrototypes ASWC_B.A.A.A, ASWC_C.A.A.A, ASWC_B.B.B.B and ASWC_C.B.B.B The complete example is listed as ARXML in Appendix F.2.

Assuming now a configuration:

ASWC_A_RUN1 is mapped to OsTask T10MS

ASWC_B_RUN1 is mapped to OsTask T100MS

ASWC_C_RUN1 is mapped to OsTask T100MS

where T10MS can **NOT** interrupt T100MS during the execution of ASWC_B_RUN1 and ASWC_C_RUN1. This configuration fulfills [constr_9002] with respect to "CN_BC" due the scheduling conditions. Since the producer of "A" and "B" can **NOT** interrupt the RunnableEntitys with the dataReadAccesses it is guaranteed that the value for all accesses of ASWC_B_RUN1 and ASWC_C_RUN1 to the same data is identical (and therefore stable) during one execution of OsTask T100MS.

Assuming now a configuration:

ASWC_A_RUN1 is mapped to OsTask T10MS

ASWC_B_RUN1 is mapped to OsTask T100MS + RteOsSchedulePoint == UNCONDITIONAL

ASWC_C_RUN1 is mapped to OsTask T100MS

where T10MS can interrupt T100MS after the execution of ASWC_B_RUN1. Without further means this configuration would violate [constr_9002] due the scheduling conditions. Since the producer of "A" and "B" can interrupt the RunnableEntitys with the dataReadAccesse it is not guaranteed that the value for all accesses of ASWC_B_RUN1 and ASWC_C_RUN1 to the same data is kept stable during one execution of OsTask T100MS.

With the additional configuration RteImplicitCommunication "CN_BC_A":

- RteVariableReadAccessRef referencing "DRP_ASWC_B_RUN1_A_A"
- RteVariableReadAccessRef referencing "DRP_ASWC_C_RUN1_A_A"
- RteCoherentAccess = true

and

RteImplicitCommunication "CN_BC_B":

- RteVariableReadAccessRef referencing "DRP_ASWC_B_RUN1_B_B"
- RteVariableReadAccessRef referencing "DRP_ASWC_C_RUN1_B_B"
- RteCoherentAccess = true

"ASWC_B_RUN1_A_A" and "ASWC_C_RUN1_A_A" as well as "ASWC_B_RUN1_B_B" and "ASWC_C_RUN1_B_B" are in the same Coherency Group. Therefore the read data values for "A" and "B" are from the same age in one execution of OsTask T100MS for ASWC_B_RUN1 and ASWC_C_RUN1.

Please note, since it is not requested that data "A" and "B" are communicated coherently the setup of RteImplicitCommunication for "A" and "B" can be handled independently from each other. In particular if there a further RunnableEntitys with dataReadAccesses to "A" or "B" mapped to the OsTask T100MS the buffers for "A" and "B" can be loaded at different points in the execution sequence. Further on it is not requested that "A" and "B" is produced in the same recurrence as it is show in this example.

Example 4.7

Coherency need and stability need for received data

Setup of SWCs

ApplicationSwComponentType "ASWC_H" with the PPortPrototype: "X"

and the RunnableEntity "ASWC_H_RUN1" which in turn has following dataWriteAccesses

- "DWP_ASWC_H_RUN1_X_X" referencing VariableDataPrototype "X" in PPortPrototype "X"

ApplicationSwComponentType "ASWC_I" with the RPortPrototype: "Y"

and the RunnableEntity "ASWC_I_RUN1" which in turn has following dataWriteAccesses

- "DWP_ASWC_I_RUN1_Y_Y" referencing VariableDataPrototype "Y" in RPortPrototype "Y"

ApplicationSwComponentType "ASWC_J" with the RPortPrototypes: "X","Y"

and the RunnableEntity "ASWC_J_RUN1" which in turn has following dataReadAccesses

- "DRP_ASWC_J_RUN1_X_X" referencing VariableDataPrototype "X" in RPortPrototype "X"
- "DRP_ASWC_J_RUN1_Y_Y" referencing VariableDataPrototype "Y" in RPortPrototype "Y"

ApplicationSwComponentType "ASWC_K" with the RPortPrototype: "X"

and the RunnableEntity "ASWC_K_RUN1" which in turn has following dataReadAccesses

- "DRP_ASWC_K_RUN1_X_X" referencing VariableDataPrototype "X" in RPortPrototype "X"

The ConsistencyNeeds "CN_J" defines a RunnableEntityGroup in the role regDoesNotRequireStability with the member "ASWC_I_RUN1" In addition the ConsistencyNeeds "CN_J" defines a DataPrototypeGroup in the role dpgRequiresCoherency to the VariableDataPrototypes ASWC_J.X.X.X, ASWC_K.Y.Y.Y

The ConsistencyNeeds "CN_JK" defines a RunnableEntityGroup in the role regRequiresStability with the member "ASWC_I_RUN1", "ASWC_J_RUN1" In addition the ConsistencyNeeds "CN_JK" defines a DataPrototypeGroup in the role dpgDoesNotRequireCoherency to the VariableDataPrototypes ASWC_J.X.X.X, ASWC_K.X.X.X

Assuming now a configuration:

ASWC_H_RUN1 is mapped to OsTask T100MS + RteOsSchedulePoint == UNCONDITIONAL

ASWC_I_RUN1 is mapped to OsTask T100MS

ASWC_J_RUN1 is mapped to OsTask T10MS

ASWC_K_RUN1 is mapped to OsTask T10MS

where T10MS can interrupt T100MS Without further means this configuration would violate [constr_9001] with respect to "CN_J" due to the scheduling conditions. Since the consumer of "X" and "Y" can interrupt the RunnableEntitys witch are producing "X" and "Y"it is not guaranteed that the value for all accesses of ASWC_J_RUN1 and ASWC_K_RUN1 returning data of the same age during one execution of OsTask T10MS. The ConsistencyNeeds "CN_JK" is already fulfilled since the consumers "ASWC_J_RUN1" and "ASWC_K_RUN1" can't be interrupted by the producing RunnableEntity ASWC_H_RUN1

With the additional configuration RteImplicitCommunication "CN_J":

- RteVariableWriteAccessRef referencing "DWP_ASWC_H_RUN1_X_X"
- RteVariableReadAccessRef referencing "DWP_ASWC_I_RUN1_Y_Y"
- RteCoherentAccess = true

the write accesses to "X" and "Y" are in the same Coherency Group. Due to this "CN_J" is fulfilled since the propagation of "X" and "Y" is delayed until the termination of ASWC_I_RUN1.

4.3.1.5.2 Explicit

The behavior of explicit reception depends on the category of the runnable and on the configuration of the according ports.

An explicit API call can be either non-blocking or blocking. If the call is non-blocking (i.e. there is a VariableAccess in the dataReceivePointByValue or dataReceivePointByArgument role referencing the VariableDataPrototype for which the API is being generated, but no WaitPoint referencing a DataReceivedEvent which references the VariableDataPrototype for which the API is being generated), the API call immediately returns the next value to be read and, if the communication is queued (event reception), it removes the data from the receiver-side queue, see Section 4.3.1.10

[SWS_Rte_06012] ⌈ A non-blocking RTE API "read" call shall indicate if no data is available. ⌋*(SRS_Rte_00109)*

In contrast, a blocking call (i.e. the VariableDataPrototype, referenced by a VariableAccess in the role dataReceivePointByArgument, and for which the API is being generated, is referenced by a DataReceivedEvent which is itself referenced by a WaitPoint) will suspend execution of the caller until new data arrives (or a timeout occurs) at the according port. When new data is received, the RTE resumes the execution of the waiting runnable. ([SRS_Rte_00092])

To prevent infinite waiting, a blocking RTE API call can have a timeout applied. The RTE monitors the timeout and if it expires without data being received returns a particular error status.

[SWS_Rte_06013] ⌈ A blocking RTE API "read" call shall indicate the expiry of a timeout. ⌋*(SRS_Rte_00069)*

The "timeout expired" indication also indicates that no data was received before the timeout expired.

Blocking reception of data ("wake up of wait point" receive mode as described in Section 4.3.1.2) is only applicable for category 2 runnables whereas non-blocking reception ("explicit data read access" receive mode) can be employed by runnables of category 2 or 1B. Neither blocking nor non-blocking explicit reception is applicable for category 1A runnable because they must not invoke functions with unknown execution time (see table 4.9).

[SWS_Rte_06016] ⌈ The RTE API call for explicit sending (VariableAccessin the dataSendPoint role, [SRS_Rte_00098]) shall be non-blocking. ⌋*(SRS_Rte_00098)*

Using this API call, the runnable can explicitly send new values of the VariableDataPrototype.

Explicit writing is valid for runnables of category 1b and 2 only. Explicit writing is not allowed for a category 1A runnable since these require API calls with constant execution time (i.e. macros).

Although the API call for explicit sending is non-blocking, it is possible for a category 2 runnable to block waiting for a notification whether the (explicit) send operation was successful. This is specified by the AcknowledgementRequest attribute and occurs by a separate API call Rte_Feedback. If the feedback method is 'wake_up_of_wait_point', the runnable will block and be resumed by the RTE either when a positive or negative acknowledgment arrives or when the timeout associated with the WaitPoint expires.

4.3.1.5.3 Concepts of data access

Tables 4.10 and 4.11 summarize the characteristics of implicit versus explicit data reception and transmission.

Implicit Read	Explicit Read
Receiving of data element values is performed only once when runnable starts	Runnable decides when and how often a data element value is received
Values of data elements do not change while runnable is running.	Runnable can always decide to receive the latest value
Several API calls to the same signal always yield the same data element value	Several API calls to the same signal may yield different data element values
Runnable must terminate (all categories)	Runnable is of cat. 1B or 2

Table 4.10: Implicit vs. explicit read

Implicit Write	Explicit Write
Sending of data element values is only done once after runnable returns	Runnable can decide when sending of data element values is done via the API call
Several usages of the API call inside the runnable cause only one data element transmission	Several usages of the API call inside the runnable cause several transmissions of the data element content. (Depending on the behavior of COM, the number of API calls and the number of transmissions are not necessarily equal.)
Runnable must terminate (all categories)	Runnable is cat. 1B or 2

Table 4.11: Implicit vs. explicit write

4.3.1.6 Transmission Acknowledgement

When `TransmissionAcknowledgementRequest` is specified, the RTE will inform the sending component if the signal has been sent correctly or not. Note that there is no insurance that the signal has actually been *received* correctly by the corresponding receiver AUTOSAR software-component. Thus, only the RTE on the sender side is involved in supporting `TransmissionAcknowledgementRequest`.

[SWS_Rte_05504] ⌈ The RTE shall support the use of `TransmissionAcknowledgementRequest` independently for each data item of an AUTOSAR software-component's AUTOSAR interface. ⌋*(SRS_Rte_00122)*

[SWS_Rte_08076] ⌈ The RTE generator shall reject configurations violating [constr_3074] in System Template [8]. ⌋*(SRS_Rte_00122, SRS_Rte_00018)*

[SWS_Rte_07927] ⌈ The RTE generator shall reject configurations violating [constr_1256] in Software Component Template [2]. ⌋*(SRS_Rte_00122, SRS_Rte_00018)*

The result of the feedback can be collected using “wake up of wait point”, “explicit data read access”, “implicit data read access” or “activation of runnable entity”.

The `TransmissionAcknowledgementRequest` allows to specify a timeout.

[SWS_Rte_03754] ⌈ If `TransmissionAcknowledgementRequest` is specified, the RTE shall ensure that timeout monitoring is performed, regardless of the receive mode of the acknowledgment. ⌋*(SRS_Rte_00069, SRS_Rte_00122)*

For inter-ECU communication, AUTOSAR COM provides the necessary functionality, for intra-ECU communication, the RTE has to implement the timeout monitoring.

If a `WaitPoint` is specified to collect the acknowledgment, two timeout values have to be specified, one for the `TransmissionAcknowledgementRequest` and one for the `WaitPoint`.

[SWS_Rte_03755] ⌈ The RTE generator shall reject the configuration, violating the [constr_2033]. ⌋*(SRS_Rte_00018)* The `DataSendCompletedEvent` associated with the `VariableAccess` in the `dataSendPoint` role for a `VariableDataPrototype` shall indicate that the transmission was successful or that the transmission was not successful. The status information about the success of the transmission shall be available as the return value of the generated RTE API call.

[SWS_Rte_03756] ⌈ For each transmission of a `VariableDataPrototype` only one acknowledgment shall be passed to the sending component by the RTE. The acknowledgment indicates either that the transmission was successful or that the transmission was not successful. ⌋*(SRS_Rte_00122)*

[SWS_Rte_03757] ⌈ The status information about the success or failure of the transmission shall be available as the return value of the RTE API call to retrieve the acknowledgment. ⌋*(SRS_Rte_00122)*

[SWS_Rte_03604] ⌈ The status information about the success or failure of the transmission shall be buffered with last-is-best semantics. When a data item is sent, the status information is reset. ⌋*(SRS_Rte_00122)*

[SWS_Rte_03604] implies that once the `DataSendCompletedEvent` has occurred, repeated API calls to retrieve the acknowledgment shall always return the same result until the next data item is sent.

[SWS_Rte_03758] ⌈ If the timeout value of the `TransmissionAcknowledgementRequest` is 0, no timeout monitoring shall be performed. ⌋*(SRS_Rte_00069, SRS_Rte_00122)*

4.3.1.7 Communication Time-out

When sender-receiver communication is performed using some physical network there is a chance this communication may fail and the receiver does not get an update of data (in time or at all). To allow the receiver of a data element to react appropriately to such a condition the SW-C template allows the specification of a time-out which the infrastructure shall monitor and indicate to the interested software components.

A “data element” is the actual information exchanged in case of sender-receiver communication. In the COM specification this is represented by a ComSignal. In the SW-C template a data element is represented by the instance of a VariableDataPrototype.

When present, the aliveTimeout attribute[6] enables the monitoring of the timely reception of the data element with data semantics transmitted over the network.

[SWS_Rte_08061] ⌈ If the aliveTimeout attribute is present the RTE shall provide the RTE COM Rx timeout callback (Rte_COMCbkRxTOut_<sg> or Rte_COMCbkRxTOut_<sn>). ⌋*(SRS_Rte_00147)*

The monitoring functionality is provided by the COM module, the RTE transports the event of reception time-outs to software components as “data element outdated”. The software components can either subscribe to that event (activation of runnable entity) or get that situation passed by the implicit and explicit status information (using API calls).

[SWS_Rte_08062] ⌈ If COM indicates a reception timeout (via RTE COM Rx timeout callback) the RTE shall raise an event of reception timeout to software components as ”data element outdated”. ⌋*(SRS_Rte_00147)*

[SWS_Rte_05021] ⌈ The RTE shall have timeout monitoring disabled for communications local to the partition, independently of the presence of aliveTimeout. ⌋*(SRS_Rte_00147)*

In such case, The RTE does not raise events of reception timeout to software components.

Therefore the Software Component shall not rely in its functionality on the time-out notification, because for local communication the notification will never occur. Time-out notification is intended as pure error reporting.

[SWS_Rte_02710] ⌈ If aliveTimeout is present, and the communication is between different partitions of the same ECU, time-out monitoring is disabled. Instead, a time-out notification of the receiver will occur immediately, when the partition of the sender is stopped and the last correctly received value shall be provided to the software components. ⌋*(SRS_Rte_00147)*

[6] This attribute is called “LIVELIHOOD” in the VFB specification

Therefore the Software Component shall not rely in its functionality on the time-out notification, because for local communication the notification will never occur. Time-out notification is intended as pure error reporting.

[SWS_Rte_03759] ⌈ If the `aliveTimeout` attribute is 0, no timeout monitoring shall be performed. ⌋*(SRS_Rte_00069, SRS_Rte_00147)*

[SWS_Rte_05022] ⌈ If a time-out has been detected in inter ECU communication, the value provided from COM shall be provided to the software components. ⌋*(SRS_Rte_00147)*

[SWS_Rte_08004] ⌈ If a signal is received, even if the signal is marked as invalid, the time-out for the same signal shall be restarted. ⌋*(SRS_Rte_00078, SRS_Rte_00147)*

Note: time-out detection may already be implemented by COM. Nevertheless this is the expected behavior towards the software components.

The time-out support (called "deadline monitoring" in COM) provided by COM has some restrictions which have to be respected when using this mechanism. Since the COM module is configured based on the System Description the restrictions mainly arise from the `data element` to I-PDU mapping. This already has to be considered when developing the System Description and the RTE Generator can only provide warnings when inconsistencies are detected. Therefore the RTE Generator needs to have access to the configuration information of COM.

In case time-out is enabled on a `data element` with update bit, there shall be a separate time-out monitoring for each `data element` with an update bit [SWS_Com_00292].

There shall be an I-PDU based time-out for `data elements` without an update bit [SWS_Com_00290]. For all data elements without update bits within the same I-PDU, the smallest configured time-out of the associated data elements is chosen as time-out for the I-PDU [SWS_Com_00291]. The notification from COM to RTE is performed per data element.

In case one `data element` coming from COM needs to be distributed to several AUTOSAR software-components the AUTOSAR Software Component Template allows to specify different `aliveTimeout` values at each Port. Therefore the configurator of COM has to decide about the `aliveTimeout` value to be used for the notification of the time-out to several software-components. This is typically the smallest timeout.

4.3.1.8 Data Element Invalidation

The Software Component template allows to specify whether a `data element`, defined in an AUTOSAR Interface, can be invalidated by the sender. The communication infrastructure shall provide means to set a data element to invalid and also indicate an invalid data element to the receiving software components. This functionality is called "data element invalidation". For an overview see figure 4.44.

[SWS_Rte_05024] ⌈ If the handleInvalid attribute of the InvalidationPolicy (when present) is set to keep or replace the invalidation support for this dataElement is enabled on sender side. The actual value used to represent the invalid data element shall be specified in the Data Semantics part of the data element definition defined in invalidValue[7]. ⌋*(SRS_Rte_00078)*

[SWS_Rte_05032] ⌈ On receiver side the handleInvalid attribute of the associated InvalidationPolicy specifies how to handle the reception of the invalid value. ⌋*(SRS_Rte_00078)*

Data element invalidation is only supported for data elements with a swImplPolicy different from 'queued'. Configurations violating this constraint are rejected by the RTE generator, see [SWS_Rte_06727].

[SWS_Rte_06727] ⌈ The RTE generator shall reject configurations which are violating [constr_1219]. ⌋*(SRS_Rte_00078)*

The API to set a dataElement to invalid shall be provided to the RunnableEntitys on data element level.

In case an invalidated data element is received a software component can be notified using the activation of runnable entity. If an invalidated data element is read by the SW-C the invalid status shall be indicated in the status code of the API.

[SWS_Rte_08005] ⌈ If the initValue of an unqueued data element equals the invalidValue and handleInvalid is set to keep and the handleNeverReceived is set to FALSE, the RTE APIs Rte_Read() and Rte_IStatus() shall return RTE_E_INVALID until first reception of data element. In this case the APIs Rte_Read() and Rte_IRead() shall provide the invalidValue. ⌋*(SRS_Rte_00078, SRS_Rte_00184)*

[SWS_Rte_08008] ⌈ If the initValue of an unqueued data element equals the invalidValue and handleInvalid is set to keep and the handleNeverReceived is not defined, the RTE APIs Rte_Read() and Rte_IStatus() shall return RTE_E_INVALID until first reception of data element. In this case the APIs Rte_Read() and Rte_IRead() shall provide the invalidValue. ⌋*(SRS_Rte_00078, SRS_Rte_00184)*

[SWS_Rte_08009] ⌈ If the initValue of an unqueued data element equals the invalidValue and handleInvalid is set to keep and the handleNeverReceived is set to TRUE, the RTE APIs Rte_Read() and Rte_IStatus() shall return RTE_E_NEVER_RECEIVED until first reception of data element. In this case the APIs Rte_Read() and Rte_IRead() shall provide the initValue. ⌋*(SRS_Rte_00078, SRS_Rte_00184)*

[SWS_Rte_08007] ⌈ The RTE Generator shall reject configurations in which the initValue of an unqueued data element equals the invalidValue and handleInvalid is set to replace. ⌋*(SRS_Rte_00078)*

[7] When InvalidationPolicy is set to keep or replace but there is no invalidValue specified it is considered as an invalid configuration.

[SWS_Rte_08046] ⌈ If the initValue of an unqueued data element equals the invalidValue and handleInvalid is set to dontInvalidate and the handleNeverReceived is set to FALSE, the RTE APIs Rte_Read() and Rte_IStatus() shall return RTE_E_OK until first reception of data element. In this case the APIs Rte_Read() and Rte_IRead() shall provide the initValue. ⌋*(SRS_Rte_00078, SRS_Rte_00184)*

[SWS_Rte_08047] ⌈ If the initValue of an unqueued data element equals the invalidValue and handleInvalid is set to dontInvalidate and the handleNeverReceived is not defined, the RTE APIs Rte_Read() and Rte_IStatus() shall return RTE_E_OK until first reception of data element. In this case the APIs Rte_Read() and Rte_IRead() shall provide the initValue. ⌋*(SRS_Rte_00078, SRS_Rte_00184)*

[SWS_Rte_08048] ⌈ If the initValue of an unqueued data element equals the invalidValue and handleInvalid is set to dontInvalidate and the handleNeverReceived is set to TRUE, the RTE APIs Rte_Read() and Rte_IStatus() shall return RTE_E_NEVER_RECEIVED until first reception of data element. In this case the APIs Rte_Read() and Rte_IRead() shall provide the initValue. ⌋*(SRS_Rte_00078, SRS_Rte_00184)*

4.3.1.8.1 Data Element Invalidation in case of Inter-ECU communication

Sender:

If data element invalidation is enabled and the communication is Inter-ECU:

- explicit data transmission: data element invalidation will be performed by COM (COM needs to be configured properly).
- implicit data transmission: data element invalidation will be performed by RTE.

Receiver:

If data element invalidation is enabled and the communication is Inter-ECU and:

- if all receiving software components requesting the same value for handleInvalid attribute of the InvalidationPolicy associated to one dataElement: data element invalidation will be performed by COM (COM needs to be configured properly), see [SWS_Rte_05026], [SWS_Rte_05048].
- if the receiving software components requesting different values for handleInvalid attribute of the InvalidationPolicy associated to one dataElement: data element invalidation will be performed by RTE, see [SWS_Rte_07031], [SWS_Rte_07032]. This can occur in case of 1:n communication where for one connector a VariableAndParameterInterfaceMapping is applied to two SenderReceiverInterfaces with different InvalidationPolicys for the mapped VariableDataPrototype.

[SWS_Rte_05026] ⌈ If a data element has been received invalidated in case of Inter-ECU communication and the attribute handleInvalid is set to keep for all receiving software components – the query of the value shall return the value provided by COM together with an indication of the invalid case. ⌋*(SRS_Rte_00078)*

[SWS_Rte_08405] ⌈ In case of Inter-ECU communication with the attribute handleInvalid set to keep for all receiving software components, the RTE shall raise a DataReceiveErrorEvent in case of reception of a data element invalid. ⌋*(SRS_Rte_00078)*

[SWS_Rte_05048] ⌈ If a data element has been received invalidated in case of Inter-ECU communication and the attribute handleInvalid is set to replace for all receiving software components – the query of the value shall return the initValue (ComDataInvalidAction is *REPLACE* [SWS_Com_00314]). ⌋*(SRS_Rte_00078)*

[SWS_Rte_08406] ⌈ In case of Inter-ECU communication with the attribute handleInvalid set to replace for all receiving software components, in case of reception of a data element invalid, the RTE shall raise a DataReceivedEvent as if a valid value would have been received. ⌋*(SRS_Rte_00078)*

[SWS_Rte_07031] ⌈ If a data element has been invalidated in case of Inter-ECU communication where receiving software components requesting different values for handleInvalid and the attribute handleInvalid is set to keep for a particular r-port – the query of the value shall return for the r-port the same value as if COM would have handled the invalidation (copy COM behavior). ⌋*(SRS_Rte_00078)*

[SWS_Rte_08407] ⌈ In case of Inter-ECU communication where receiving software components requesting different values for the attribute handleInvalid and this attribute is set to keep for a particular R-Port, in case of reception of a data element invalid, the RTE shall raise a DataReceiveErrorEvent. ⌋*(SRS_Rte_00078)*

[SWS_Rte_07032] ⌈ If a data element has been received invalidated in case of Inter-ECU communication where receiving software components requesting different values for handleInvalid and the attribute handleInvalid is set to replace for an particular r-port – RTE shall perform the “invalid value substitution” with the initValue for the r-port. Then the reception will be handled as if a valid value would have been received (activation of runnable entities using the DataReceivedEvent). ⌋*(SRS_Rte_00078)*

[SWS_Rte_08049] ⌈ If a data element has been received invalidated in case of Inter-ECU communication and the attribute handleInvalid is set to dontInvalidate – the query of the value shall return the value provided by COM. Then the reception will be handled as if a valid value would have been received (activation of runnable entities using the DataReceivedEvent). ⌋*(SRS_Rte_00078)*

4.3.1.8.2 Data Element Invalidation in case of Intra-ECU communication

Sender:

[SWS_Rte_05025] ⌈ If data element invalidation is enabled, and the communication is Intra-ECU, data element invalidation can be implemented by the RTE ⌋*(SRS_Rte_00078)*

In case of implicit data transmission the RTE shall always implement the data element invalidation and therefore provide an API to set the data element's value to the invalid value. The actual invalid value is specified in the SW-C template invalidValue.

Receiver:

[SWS_Rte_05030] ⌈ If a data element has been invalidated in case of Intra-ECU communication and the attribute handleInvalid is set to keep – the query of the value shall return the same value as if COM would have handled the invalidation (copy COM behavior). Then the reception of the invalid value will be handled as an error and the activation of runnable entities can be performed using the DataReceiveErrorEvent. ⌋*(SRS_Rte_00078)*

[SWS_Rte_05049] ⌈ If a data element has been received invalidated in case of Intra-ECU communication and the attribute handleInvalid is set to replace – RTE shall perform the "invalid value substitution" with the initValue. Then the reception will be handled as if a valid value would have been received (activation of runnable entities using the DataReceivedEvent). ⌋*(SRS_Rte_00078)*

[SWS_Rte_08050] ⌈ If a data element has been received invalidated in case of Intra-ECU communication and the attribute handleInvalid is set to dontInvalidate – the query of the value shall return the received value. Then the reception will be handled as if a valid value would have been received (activation of runnable entities using the DataReceivedEvent). ⌋*(SRS_Rte_00078)*

4.3.1.9 Filters

By means of the filter attribute [SRS_Rte_00121] an additional filter layer can be added on the receiver side of unqueued S/R-Communication. *Value-based* filters can be defined, i.e. only signal values fulfilling certain conditions are made available for the receiving component. The possible filter algorithms are taken from OSEK COM version 3.0.2. They are listed in the meta model (see [2]. According to the SW-C template [2], filters are only allowed for signals that are compatible to C language unsigned integer types (i.e. characters, unsigned integers and enumerations). Thus, filters cannot be applied to composite data types like for instance ApplicationRecordDataType or ApplicationArrayDataType.

[SWS_Rte_05503] ⌈ The RTE shall provide value-based filters on the receiver-side of unqueued S/R-Communication as specified in the SW-C template [2]. ⌋*(SRS_Rte_00121)*

[SWS_Rte_05500] ⌈ For inter-ECU communication, the RTE shall use the filter implementation of the COM layer [SRS_Rte_00121]. For intra-ECU and inter-Partition communication, the RTE can use the filter implementation of COM, but may also implement the filters itself for efficiency reasons, without using COM. ⌋*(SRS_Rte_00019, SRS_Rte_00121)*

[SWS_Rte_05501] ⌈ The RTE shall support a different filter specification for each `dataElement` in a component's AUTOSAR interface. ⌋*(SRS_Rte_00121)*

[SWS_Rte_08077] ⌈ In case that filtering applies the input value shall be calculated from the "unfiltered buffer" before the `RunnableEntity` starts, the result of the filter calculation shall be stored in a "filtered buffer" and the `RunnableEntity` accessing a `dataElement` in a Receiver Port with a filter shall get access to the "filtered buffer" instead of the "unfiltered buffer". ⌋*(SRS_Rte_00121)*

[SWS_Rte_08078] ⌈ For optimization reasons no "filtered buffer" should be provided, if filtering applies for a `dataElement` and the "unfiltered buffer" is not used at all. The "unfiltered buffer" should be used for filtering instead. ⌋*(SRS_Rte_00121)*

[SWS_Rte_08079] ⌈ Separate "filtered buffers" shall be provided, if the same `dataElement` is accessed by `RunnableEntity`s via different Receiver Ports and filters with different semantics are applied in each Port. ⌋*(SRS_Rte_00121)*

4.3.1.10 Buffering

[SWS_Rte_02515] ⌈ The buffering of sender-receiver communication shall be done on the receiver side. This does not imply that COM does no buffering on the sender side. On the receiver side, two different approaches are taken for the buffering of 'data' and of 'events', depending on the value of the software implementation policy. ⌋*(SRS_Rte_00110)*

4.3.1.10.1 Last-is-Best-Semantics for 'data' Reception

[SWS_Rte_02516] ⌈ On the receiver side, the buffering of 'data' (`swImplPolicy` not `queued`) shall be realized by the RTE by a single data set for each data element instance. ⌋*(SRS_Rte_00107)*

The use of a single data set provides the required semantics of a single element queue with overwrite semantics (new data replaces old). Since the RTE is required to ensure data consistency, the generated RTE should ensure that non-atomic reads and writes of the data set (e.g. for composite data types) are protected from conflicting concurrent access. RTE may use lower layers like COM to implement the buffer.

[SWS_Rte_02517] ⌈ The RTE shall initialize this data set [SWS_Rte_02516] with a startup value depending on the ports attributes and if the general initialization conditions in [SWS_Rte_07046] are fulfilled. ⌋*(SRS_Rte_00068, SRS_Rte_00108)*

[SWS_Rte_02518] ⌈ Implicit or explicit read access shall always return the last received data. ⌋*(SRS_Rte_00107)*

Requirement [SWS_Rte_02518] applies whether or not there is a DataReceivedEvent referencing the VariableDataPrototype for which the API is being generated.

[SWS_Rte_02519] ⌈ Explicit read access shall be non blocking in the sense that it does not wait for new data to arrive. The RTE shall provide mutual exclusion of read and write accesses to this data, e.g., by ExclusiveAreas. ⌋*(SRS_Rte_00109)*

[SWS_Rte_02520] ⌈ When new data is received, the RTE shall silently discard the previous value of the data, regardless of whether it was read or not. ⌋*(SRS_Rte_00107)*

4.3.1.10.2 Queueing for 'event' Reception

The application of event semantics implies a state change. Events usually have to be handled. In many cases, a loss of events can not be tolerated. Hence the swImplPolicy is set to queued to indicate that the received 'events' have to be buffered in a queue.

[SWS_Rte_02521] ⌈ The RTE shall implement a receive queue for each event-like data element (swImplPolicy = queued) of a receive port. ⌋*(SRS_Rte_00107)*

The queueLength attribute of the QueuedReceiverComSpec referencing the event assigns a constant length to the receive queue.

[SWS_Rte_02522] ⌈ The events shall be written to the end of the queue and read (consuming) from the front of the queue (i.e. the queue is first-in-first-out). ⌋*(SRS_Rte_00107, SRS_Rte_00110)*

[SWS_Rte_02523] ⌈ If a new event is received when the queue is already filled, the RTE shall discard the received event and set an error flag. ⌋*(SRS_Rte_00107, SRS_Rte_00110)*

[SWS_Rte_02524] ⌈ The error flag described in [SWS_Rte_02523] shall be reset during the next explicit read access on the queue. In this case, the status value RTE_E_LOST_DATA shall be presented to the application together with the data. ⌋*(SRS_Rte_00107, SRS_Rte_00110, SRS_Rte_00094)*

[SWS_Rte_02525] ⌈ If an empty queue is polled, the RTE shall return with a status RTE_E_NO_DATA to the polling function, (see chap. 5.5.1). ⌋*(SRS_Rte_00107, SRS_Rte_00110, SRS_Rte_00094)*

The minimum size of the queue is 1.

[SWS_Rte_02526] ⌈ The RTE generator shall reject a queueLength attribute of an QueuedReceiverComSpec with a queue length ≤ 0. ⌋*(SRS_Rte_00110, SRS_Rte_00018)*

4.3.1.10.3 Queueing of mode switches

The communication of mode switch notifications is typically event driven. Accordingly, RTE offers a similar queueing mechanism as for the 'queued' sender receiver communication, described above.

[SWS_Rte_02718] ⌈ The RTE shall implement a receive queue for the mode switch notifications of each mode machine instance. ⌋*(SRS_Rte_00107)*

The queueLength attribute of the ModeSwitchSenderComSpec referencing the mode machine instance, assigns a constant length to the receive queue. In contrast to the event communication, for mode switch communication, the length is associated with the sender side, the mode manager, because it is unique for the mode machine instance.

[SWS_Rte_02719] ⌈ The mode switch notification shall be written to the end of the queue and read (consuming) from the front of the queue (i.e. the queue is first-in-first-out). ⌋*(SRS_Rte_00107, SRS_Rte_00110)*

[SWS_Rte_02720] ⌈ If a new mode switch notification is received when the queue is already filled, the RTE shall discard the received notification. ⌋*(SRS_Rte_00107, SRS_Rte_00110)* In this case, Rte_Switch will return an error, see [SWS_Rte_02675].

[SWS_Rte_02721] ⌈ RTE shall dequeue a mode switch notification, when the mode switch is completed. ⌋*(SRS_Rte_00107, SRS_Rte_00110, SRS_Rte_00094)*

The minimum size of the queue is 1.

[SWS_Rte_02723] ⌈ The RTE generator shall reject a queueLength attribute of an ModeSwitchSenderComSpec with a queue length ≤ 0. ⌋*(SRS_Rte_00110, SRS_Rte_00018)*

In case of a queue length of 1, RTE will reject new mode switch notifications during the mode transition.

4.3.1.11 Operation

4.3.1.11.1 Inter-ECU Mapping

This section describes the mapping from VariableDataPrototypes to COM signals or COM signal groups for sender-receiver communication. The mapping is described in the input of the RTE generator, in the DataMapping section of the System Template [8].

If a VariableDataPrototype is mapped to a COM signal or COM signal group but the communication is local, the RTE generator can use the COM signal/COM signal group for the transmission or it can use its own direct implementation of the communication for the transmission.

4.3.1.11.1.1 Primitive Data Types

[SWS_Rte_04504] ⌈ If a data element is a primitive type and the communication is inter-ECU, the DataMappings element shall contain a mapping of the data element to at least one COM signal, else the missing data mapping shall be interpreted as an unconnected port. ⌋*(SRS_Rte_00091)*

The mapping defines all aspects of the signal necessary to configure the communication service, for example, the network signal endianess and the communication bus. The RTE generator only requires the COM signal handle id since this is necessary for invoking the COM API.

[SWS_Rte_04505] ⌈ The RTE shall use the `ComHandleId` of the corresponding `ComSignal` when invoking the COM API for signal. ⌋*(SRS_Rte_00091)*

The actual COM handle id has to be gathered from the ECU configuration of the COM module. The input information `ComSignalHandleId` is used to establish the link between the `ComSignal` of the COM module's configuration and the corresponding `ISignal` of the System Template.

4.3.1.11.1.2 Composite Data Types

When a data element is a composed type the RTE is required to perform more complex actions to marshall the data [SRS_Rte_00091] than is the case for primitive data types.

The DataMappings element of the ECU configuration contains (or reference) sufficient information to allow the data item or operation parameters to be transmitted. The mapping indicates the COM signals or signal groups to be used when transmitting a given data item of a given port of a given software component instance within the composition.

[SWS_Rte_04506] ⌈ If a data element is a composite data type and the communication is inter-ECU, the DataMappings element shall contain a mapping of the data element to COM signals such that each element of the composite data type that is a primitive data type is mapped to a separate COM signal(s), else the missing data mapping shall be interpreted as an unconnected port. ⌋*(SRS_Rte_00091)*

[SWS_Rte_04507] ⌈ If a data element is typed by a composite data type and the communication is inter-ECU, the DataMappings element shall contain a mapping of the data element to COM signals such that each element of the composite data type that is itself a composite data type shall be recursively mapped to a primitive type and hence to a separate COM signal(s). ⌋*(SRS_Rte_00091)*

The above requirements have two key features; firstly, COM is responsible for endianness conversion (if any is required) of primitive types and, secondly, differing structure member alignment between sender and receiver is irrelevant since the COM signals are packed into I-PDUs by the COM configuration.

The DataMappings shall contain sufficient COM signals to map each primitive element[8] of the AUTOSAR signal.

[SWS_Rte_04508] ⌈ The RTE generator shall reject configuration violating the constraint [constr_3059]. ⌋*(SRS_Rte_00091)*

[SWS_Rte_02557] ⌈

1. Each signal that is mapped to an element of the same composite data item shall be mapped to the same signal group.
2. If two signals are not mapped to an element of the same composite data item, they shall not be mapped to the same signal group.
3. If a signal is not mapped to an element of a composite data item, it shall not be mapped to a signal group.

⌋*(SRS_Rte_00091)*

[SWS_Rte_05081] ⌈ The RTE shall use the `ComHandleId` of the corresponding `ComSignalGroup` when invoking the COM API for signal groups. ⌋*(SRS_Rte_00091)*

[SWS_Rte_05173] ⌈ The RTE shall use the `ComHandleId` of the corresponding `ComGroupSignal` when invoking the COM API for shadow signals. ⌋*(SRS_Rte_00091)*

The actual COM handle id has to be gathered from the ECU configuration of the COM module. The input information `ComHandleId` is used to establish the link between the `ComSignalGroup` of the COM module's configuration and the corresponding `ISignalGroup` of the System Template.

The input information `ComHandleId` of shadow signals is used to establish the link between the `ComGroupSignal` of the COM module's configuration and the corresponding `ISignal` of the System Template.

4.3.1.11.2 Atomicity

[SWS_Rte_04527] ⌈ The RTE is required to treat AUTOSAR signals transmitted using sender-receiver communication atomically [SRS_Rte_00073]. To achieve this the "signal group" mechanisms provided by COM shall be utilized. See [SWS_Rte_02557] for the mapping. ⌋*(SRS_Rte_00019, SRS_Rte_00073, SRS_Rte_00091)*

The RTE decomposes the composite data type into single signals as described above and passes them to the COM module by using the COM API call `Com_SendSignal` (if parameter `RteUseComShadowSignalApi` is `FALSE`) or `Com_UpdateShadowSignal` (if parameter `RteUseComShadowSignalApi` is `TRUE`). As this set of single signals has to be treated as atomic, it is placed in a "signal group". A signal group has to be placed always in a single I-PDU. Thus, atomicity is estab-

[8] An AUTOSAR signal that is a primitive data type contains exactly one one primitive element whereas a signal that is a composite data type one or more primitive elements.

lished. When all signals have been updated, the RTE initiates transmission of the signal group by using the COM API call Com_SendSignalGroup.

As would be expected, the receiver side is the exact reverse of the transmission side: the RTE must first call Com_ReceiveSignalGroup precisely once for the signal group and then call Com_ReceiveSignal (if parameter RteUseComShadowSignalApi is FALSE) or Com_ReceiveShadowSignal (if parameter RteUseComShadowSignalApi is TRUE) to extract the value of each signal within the signal group.

A signal group has the additional property that COM guarantees to inform the receiver by invoking a call-back about its arrival only after all signals belonging to the signal group have been unpacked into a shadow buffer.

4.3.1.11.3 Fan-out

Fan-out can be divided into two scenarios; *PDU fanout* where the same I-PDU is sent to multiple destinations and *signal fan-out* where the same signal, i.e. data element is sent in different I-PDUs to multiple receivers.

For Inter-ECU communication, the RTE does not perform PDU fan-out. Instead, the RTE invokes Com_SendSignal once for a primitive data element and expects the fan-out to multiple PDU destinations to occur lower down in the AUTOSAR communication stack. However, it is necessary for the RTE to support *signal fan-out* since this cannot be performed by any lower level layer of the AUTOSAR communication stack.

The data mapping in the System Template[8] is based on the SystemSignal and SystemSignalGroup. The COM module however uses the ISignal and ISignalGroup counterparts (ComSignal, ComSignalGroup, ComGroupSignal) to define the COM API. The RTE Generator needs to identify whether there are several ISignal or ISignalGroup elements defined for the SystemSignal or SystemSignalGroup and implement the fan-out accordingly. Then the corresponding elements in the COM ecu configuration (ComSignal, ComSignalGroup, ComGroupSignal) are required to establish the interaction between Rte and COM.

[SWS_Rte_06023] ⌈ For inter-ECU transmission of a primitive data type, the RTE shall invoke Com_SendSignal for each ISignal to which the primitive data element is mapped. ⌋ *(SRS_Rte_00019, SRS_Rte_00028)*

For the invocation the ComHandleId from the ComSignal of COM's ecu configuration shall be used (see [SWS_Rte_04505]).

If the data element is typed by a composite data type, RTE invokes Com_SendSignal (if parameter RteUseComShadowSignalApi is FALSE) or Com_UpdateShadowSignal (if parameter RteUseComShadowSignalApi is TRUE) for each primitive element (ISignal) in the composite data type and each COM signal to which that primitive element is mapped, and Com_SendSignalGroup for each ISignalGroup to which the data element is mapped.

[SWS_Rte_04526] ⌈ For inter-ECU transmission of composite data type, the RTE shall invoke `Com_SendSignal` (if parameter `RteUseComShadowSignalApi` is `FALSE`) or `Com_UpdateShadowSignal` (if parameter `RteUseComShadowSignalApi` is `TRUE`) for each `ISignal` to which an element in the composite data type is mapped and `Com_SendSignalGroup` for each `ISignalGroup` to which the composite data element is mapped. ⌋*(SRS_Rte_00019, SRS_Rte_00028)*

For the invocation the `ComHandleId` from the `ComGroupSignal` and `ComSignalGroup` of COM's ecu configuration shall be used (see [SWS_Rte_05173] and [SWS_Rte_05081]).

For intra-ECU transmission of data elements, the situation is slightly different; the RTE handles the communication (the lower layers of the AUTOSAR communication stack are not used) and therefore must ensure that the data elements are routed to all receivers. For `inter-partition` communication, RTE may use the IOC.

[SWS_Rte_06024] ⌈ For `inter-partition` transmission of data elements, the RTE shall perform the fan-out to each receiver. ⌋*(SRS_Rte_00019, SRS_Rte_00028)*

4.3.1.11.4 Fan-in

When receiving data from multiple senders in inter-ECU communication, either the RTE on the receiver side has to collect data received in different COM signals or COM signal groups and pass it to one receiver or the RTE on the sender side has to provide shared access to a COM signal or COM signal group to multiple senders. The receiver RTE, which has to handle multiple COM signals or signal groups, is notified about incoming data for each COM signal or COM signal group separately but has to ensure data consistency when passing the data to the receiver. The sender RTE, which has to handle multiple senders sharing COM signals or signal groups, has to ensure consistent access to the COM API, since COM API calls for the same signal are not reentrant.

[SWS_Rte_03760] ⌈ If multiple senders use different COM signals or signal groups for inter-ECU transmission of a data element prototype with `swImplPolicy` different from `queued` to a receiver, the RTE on the receiver side has to pass the last received value to the receiver component while ensuring data consistency. ⌋*(SRS_Rte_00019, SRS_Rte_00131)*

[SWS_Rte_03761] ⌈ If multiple senders use different COM signals or signal groups for inter-ECU transmission of a data element prototype with `event semantic`s to a receiver, the RTE on the receiver side has to queue all incoming values while ensuring data consistency. ⌋*(SRS_Rte_00019, SRS_Rte_00131)*

[SWS_Rte_03762] ⌈ If multiple senders share COM signals or signal groups for inter-ECU transmission of a data element prototype to a receiver, the RTE on the sender side shall ensure that the COM API for those signals is not invoked concurrently. ⌋*(SRS_Rte_00019, SRS_Rte_00131)*

4.3.1.11.5 Sequence diagrams of Sender Receiver communication

Figure 4.36 shows a sequence diagram of how Sender Receiver communication for data transmission and non-blocking reception may be implemented by RTE. The sequence diagram also shows the `Rte_Read` API behavior if an `initValue` is specified.

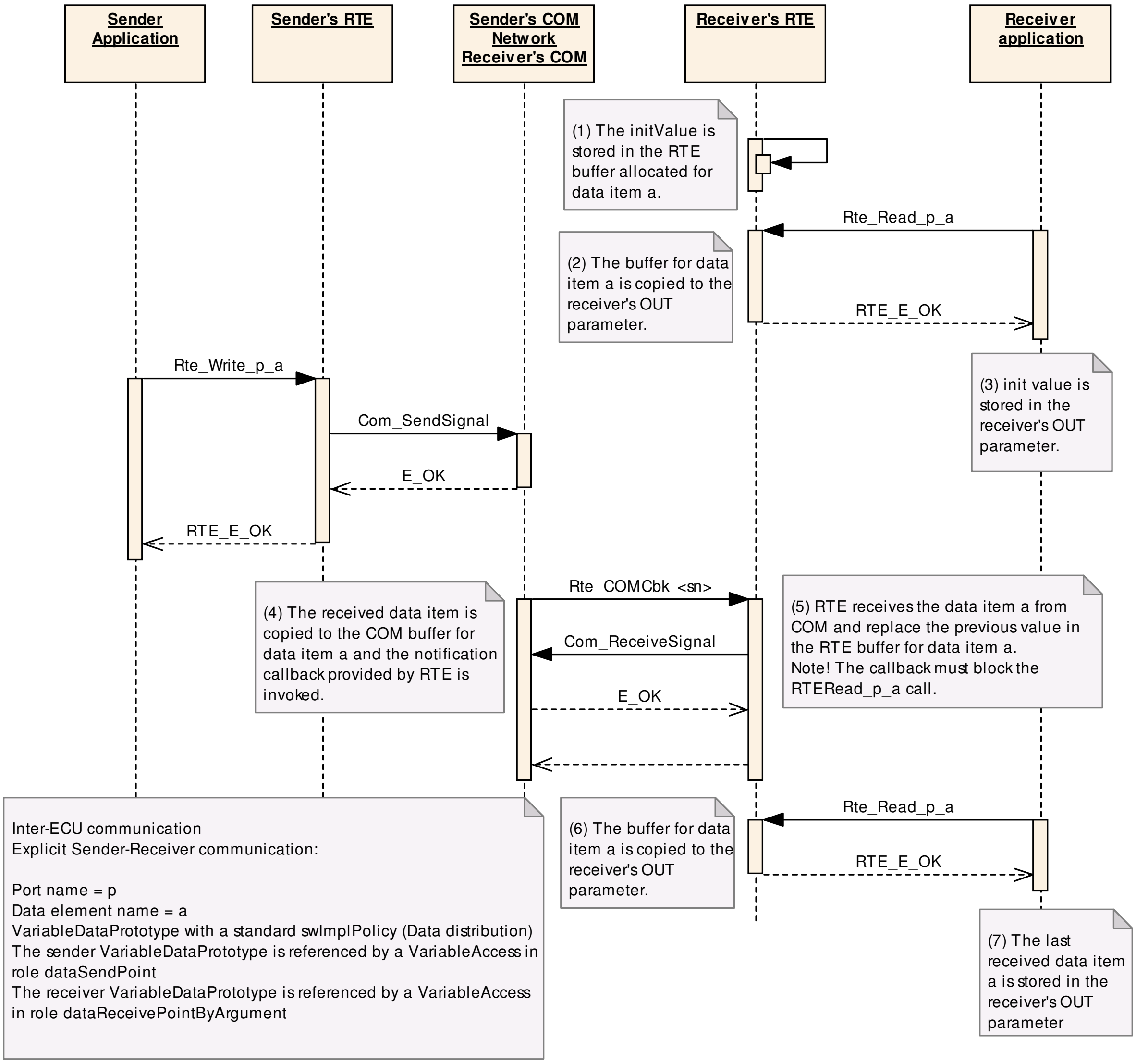


Figure 4.36: Sender Receiver communication with `data semantics` and dataReceivePointByArgument as reception mechanism

Figure 4.37 shows a sequence diagram of how Sender Receiver communication for event transmission and non-blocking reception may be implemented by RTE. The sequence diagram shows the `Rte_Receive` API behavior when the queue is empty.

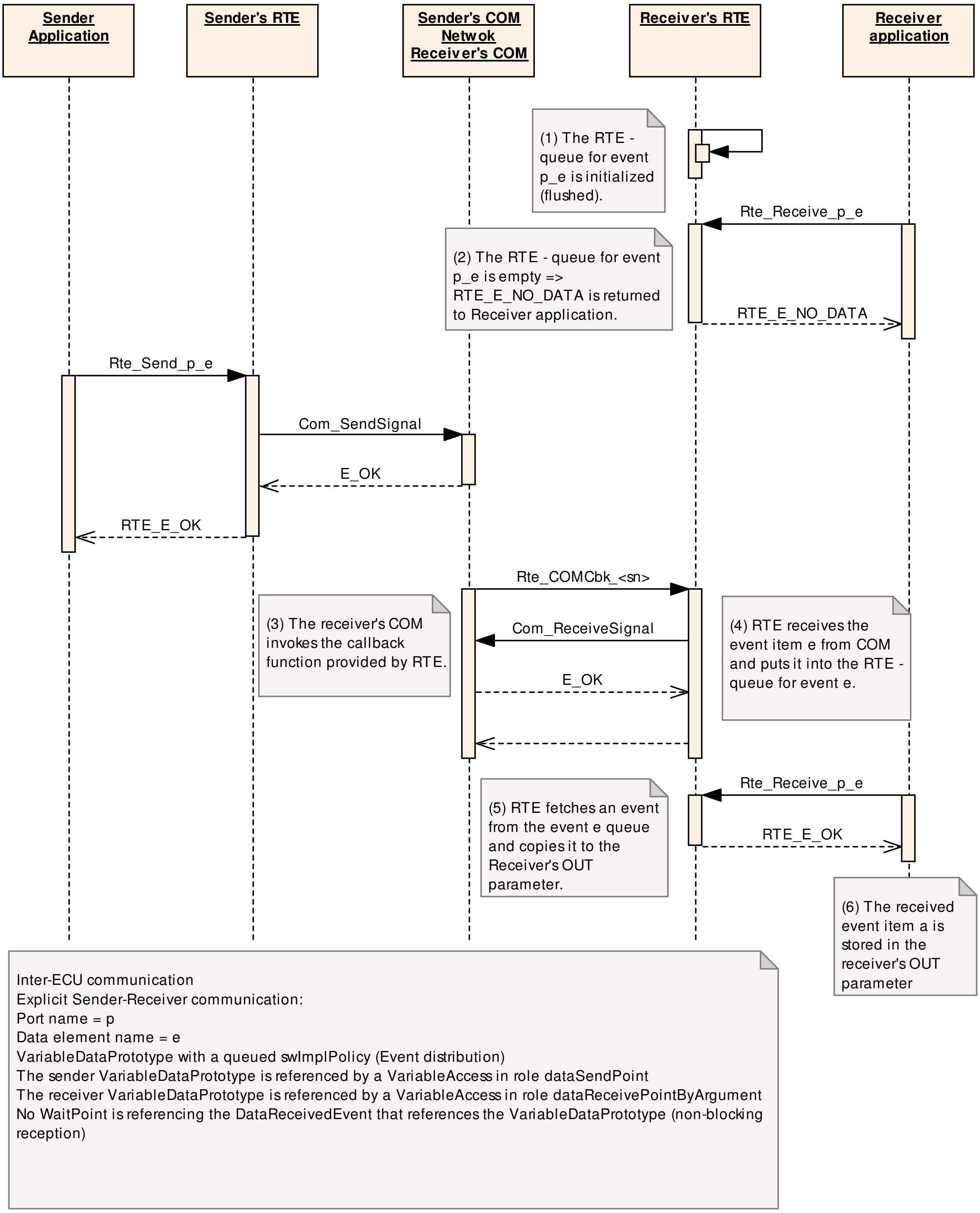


Figure 4.37: Sender Receiver communication with `event semantics` and dataReceivePointByArgument as reception mechanism

Figure 4.38 shows a sequence diagram of how Sender Receiver communication for event transmission and activation of runnable entity on the receiver side may be implemented by RTE.

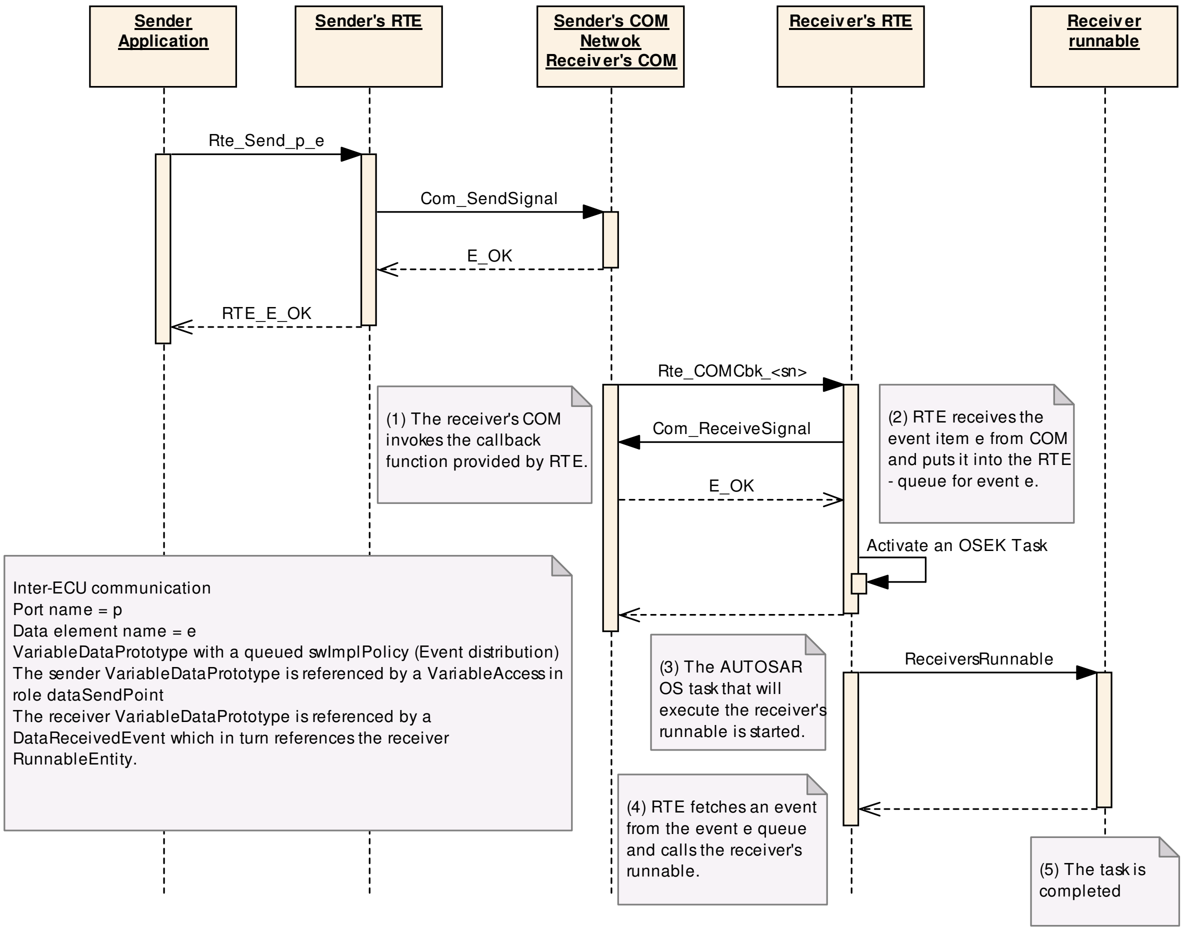


Figure 4.38: Sender Receiver communication with `event semantic`s and activation of runnable entity as reception mechanism

4.3.1.12 “Never received status” for Data Element

The Software Component template allows specifying whether an unqueued data, defined in an AUTOSAR Interface, has been updated since system start (or partition restart) or not. This additional optional status establishes the possibility to check whether a data element has been changed since system start (or partition restart).

[SWS_Rte_07381] ⌈ On receiver side the `handleNeverReceived` attribute of the `NonqueuedReceiverComSpec` shall specify the handling of the never received status. ⌋*(SRS_Rte_00184)*

[SWS_Rte_07382] ⌈ The initial status of the data elements with the attribute `handleNeverReceived` set to TRUE shall be `RTE_E_NEVER_RECEIVED`. ⌋*(SRS_Rte_00184)*

[SWS_Rte_07383] ⌈ The initial status of the data elements with the attribute handleNeverReceived set to TRUE shall be cleared when the first reception occurs. ⌋*(SRS_Rte_00184)*

[SWS_Rte_07645] ⌈ The status of data elements shall be reset on the receiver side to RTE_E_NEVER_RECEIVED when the receiver's partition is restarted. ⌋*(SRS_Rte_00184, SRS_Rte_00224)*

4.3.1.13 "Update flag" for Data Element

The Software Component template allows specifying whether an unqueued data, defined in an AUTOSAR Interface, has been updated since last read or not. This additional optional status establishes the possibility to check, whether a data element has been updated since last read.

On receiver side the "enableUpdate" attribute of the NonqueuedReceiverComSpec has to activate the handling of the update flag.

[SWS_Rte_07385] ⌈ The RTE shall provide one update flag per dataElement in a RPortPrototype where the "enableUpdate" attribute of the NonqueuedReceiverComSpec is set to true and where at least one RunnableEntity defines a VariableAccess in the dataReceivePointByArgument or dataReceivePointByValue role. ⌋*(SRS_Rte_00179)*

[SWS_Rte_07386] ⌈ The update flag of the data elements configured with the "enableUpdate" attribute shall be set by receiving new data from COM or from a local software-conponent. ⌋*(SRS_Rte_00179)*

[SWS_Rte_07387] ⌈ The update flag of a particular dataElement in a RPortPrototype shall be cleared after each read by Rte_Read or Rte_DRead of the data element. ⌋*(SRS_Rte_00179)*

Please note that the "UpdateFlag" for dataElements is only available for explicit communication, see see [SWS_Rte_07391].

[SWS_Rte_07689] ⌈ The update flag shall be cleared when the RTE is started or when the partition of the software-component is restarted. ⌋*(SRS_Rte_00179)*

The update flag can be queried by the Rte_IsUpdated API, see 5.6.34.

4.3.1.14 Dynamic data type

Dynamic data are data whose length varies at runtime.

This includes:

- arrays with variable number of elements
- structures including arrays with variable number of elements

This excludes:

- structures including variable number of elements

The length of the dynamic data is accessed thanks to the optional parameter `<length>` of the RTE APIs for communication. See chapter 5 for more information.

In case of inter-ECU communication, dynamic data are mapped to dynamic signals and received/transmitted through the TP by the COM stack.

With the current release of SWS_COM, COM limits the dynamic signals to the `ComSignalType` UINT_8DYN (see the requirement COM569).

In order to respect the VFB concept the capability of inter-ECU and intra-ECU communication should be equal. So it has been decided to extend these limitation from COM also to the intra-ECU communication. As a consequence the only one dynamic data type supported by the RTE is the type uint8[n] whatever the communication is intra or inter-ECU. See [SWS_Rte_07810].

4.3.1.15 Inter-ECU communication through TP

Inter-ECU communication can be configured in COM to be supported by the TP. This is especially necessary if:

- Size of the signal exceed the size of the L-PDU (large signals)
- Size of the signal group exceed the size of the L-PDU
- Size of the signal varies at runtime (dynamic signals)

In the current release of SWS_COM, COM APIs to access signal values might return the error code COM_BUSY for the signals mapped to N-PDU. This error code indicates that the access to the signal value has failed (internally rejected by COM) and should be retried later. This situation might only be possible when the transmission or the reception of the corresponding PDU is in progress in COM at the time the access to the signal value is requested.

This is a problem for the handling of data with `data semantic` (last is best behavior) because:

- "COM_BUSY like" errors are not compatible with real time systems that should have predictable response time.
- Forwarding this error code to the application implies that every applications should handle it (implement a retry) even if it will never comes (data is not be mapped to N-PDU).
- Error code can not be forwarded to the application in case of direct read or implicit write.

This is not a problem for the handling of data with `event semantic` (queued behavior) because:

- The COM_BUSY error should not be possible during the execution of COM callbacks (Rx indication and Tx confirmation) that can be used by the RTE to handle the queue.
- Data are queued internally by RTE and accessible at any time by the application.

Note: First point is especially true if the `ComIPduSignalProcessing` is configured as `IMMEDIATE`. But if the `ComIPduSignalProcessing` is configured as `DEFFERED` and 2 events are closely received, it is possible that at the time the RTE tries to access the corresponding COM signal the second event reception has already started. In this case the RTE will received COM_BUSY and the event will be lost but it is more a problem of configuration than a limitation from COM.

As a consequence it has been decided to limit the data mapped to N-PDU to the `event semantic` (queued behavior). See [SWS_Rte_07811].

Note: As the data mapping is not mandatory for the RTE contract phase, it is possible that a configuration is accepted at contract phase but rejected at generation phase when the data mapping is known.

Dynamic data are always mapped to N-PDU in case of inter-ECU communication. So in order to avoid such situation (late rejection at generation phase) and in order to respect the VFB concept (intra and inter-ECU should be equal) it has been decided to extend this limitation to every dynamic data whatever the communication is intra or inter-ECU. See [SWS_Rte_07812].

4.3.1.16 Inter-ECU communication of arrays of bytes

Generally the communication of arrays in the case of inter-ECU communication must make use of the signal group mechanisms to send an array to COM. This implies sending each array element to a shadow buffer in COM (with `Com_SendSignal()` API, if parameter `RteUseComShadowSignalApi` is `FALSE` or `Com_UpdateShadowSignal()` API, if parameter `RteUseComShadowSignalApi` is `TRUE`), and in the end send the signal group (with `Com_SendSignalGroup()` API).

An exception to this general rule is for arrays of bytes. In this case, the RTE shall use the native COM interface to send directly the data.

[SWS_Rte_07408] ⌈ The RTE shall use the `Com_SendSignal` or `Com_ReceiveSignal` APIs to send or receive fixed-length arrays of bytes. ⌋*(SRS_Rte_00231)*

[SWS_Rte_07817] ⌈ The RTE shall use the `Com_SendDynSignal` or `Com_ReceiveDynSignal` APIs to send or receive variable-length arrays of bytes. ⌋*(SRS_Rte_00231)*

4.3.1.17 Handling of acknowledgment events

As a general rule, the acknowledgment events DataWriteCompletedEvent and DataSendCompletedEvent shall be raised immediately after the sending to all receivers has been performed and in case of Inter-ECU communication all acknowledgments from COM have been received. As part of the implementation detailed rules for the following communication scenarios have to be considered:

Intra-Partition communication

[SWS_Rte_08017] ⌈ For intra-partition communication with implicit dataWriteAccess the DataWriteCompletedEvent shall be fired if and only if a task terminates and the write-back copy actions to the global RTE-buffer are completed. The transmission status shall be RTE_E_TRANSMIT_ACK and can be collected with Rte_IFeedback API. ⌋*(SRS_Rte_00122)*

[SWS_Rte_08043] ⌈ For intra-partition communication with incoherent implicit dataWriteAccess no write-back copy actions to a global RTE-buffer will be performed, if the involved runnables are all running in one preemption area. In this case the DataWriteCompletedEvent shall be fired after the termination of the last sending runnable in the sending task. The transmission status shall be RTE_E_TRANSMIT_ACK and can be collected with Rte_IFeedback API. ⌋*(SRS_Rte_00122)*

[SWS_Rte_08018] ⌈ For intra-partition communication with explicit dataSendPoint the DataSendCompletedEvent shall be fired if and only if the sending to all receivers has been performed. The transmission status shall be RTE_E_TRANSMIT_ACK and can be collected with Rte_Feedback API. ⌋*(SRS_Rte_00122)*

Inter-Partition communication

[SWS_Rte_08020] ⌈ For inter-partition communication with implicit dataWriteAccess the DataWriteCompletedEvent shall be fired if and only if a task terminates and the write-back copy actions to the global RTE-buffer are completed. In addition the execution of the data write operations at the data receiver partitions must have taken place. Thereby the return status of the IOC for the different write operations can be neglected. The transmission status shall be RTE_E_TRANSMIT_ACK and can be collected with Rte_IFeedback API. ⌋*(SRS_Rte_00122)*

[SWS_Rte_08044] ⌈ For inter-partition communication with incoherent implicit dataWriteAccess no write-back copy actions to a global RTE-buffer will be performed, if the involved runnables are all running in one preemption area. In this case the DataWriteCompletedEvent shall be fired after the termination of the last sending runnable in the sending task and after the execution of the data write operations at the data receiver partitions have taken place. Thereby the return status of the IOC for the different write operations can be neglected. The transmission status shall be RTE_E_TRANSMIT_ACK and can be collected with Rte_IFeedback API. ⌋*(SRS_Rte_00122)*

[SWS_Rte_08021] ⌈ For inter-partition communication with explicit dataSendPoint the DataSendCompletedEvent shall be fired if and only if the sending to all receivers has been performed and the execution of the data write operations at the data receiver partitions have taken place. Thereby the return status of the IOC for the different write operations can be neglected. The transmission status shall be RTE_E_TRANSMIT_ACK and can be collected with Rte_Feedback API. ⌋*(SRS_Rte_00122)*

Inter-ECU communication

[SWS_Rte_08022] ⌈ For inter-ECU communication with implicit dataWriteAccess the DataWriteCompletedEvent shall be fired if and only if a task terminates and the write-back copy actions to the global RTE-buffer are completed. In addition the transmission acknowledgment from COM must be complete, i.e. the acknowledgment has been received and in case of RTE-fanout all acknowledgments have been received. The transmission status shall be RTE_E_TRANSMIT_ACK and can be collected with Rte_IFeedback API. ⌋*(SRS_Rte_00122)*

[SWS_Rte_08045] ⌈ For inter-ECU communication with incoherent implicit dataWriteAccess no write-back copy actions to a global RTE-buffer will be performed, if the involved runnables are all running in one preemption area. In this case the DataWriteCompletedEvent shall be fired after the termination of the last sending runnable in the sending task and after the transmission acknowledgment from COM is complete, i.e. the acknowledgment has been received and in case of RTE-fanout all acknowledgments have been received. The transmission status shall be RTE_E_TRANSMIT_ACK and can be collected with Rte_IFeedback API. ⌋*(SRS_Rte_00122)*

[SWS_Rte_08023] ⌈ For inter-ECU communication with explicit dataSendPoint the DataSendCompletedEvent shall be fired if and only if the sending to all receivers has been performed and the transmission acknowledgment from COM is complete, i.e. the acknowledgment has been received and in case of RTE-fanout all acknowledgments have been received. The transmission status shall be RTE_E_TRANSMIT_ACK and can be collected with Rte_Feedback API. ⌋*(SRS_Rte_00122)*

4.3.2 Client-Server

4.3.2.1 Introduction

Client-server communication involves two entities, the client which is the requirer (or user) of a service and the server that provides the service.

The client initiates the communication, requesting that the server performs a service, transferring a parameter set if necessary. The server, in the form of the RTE, waits for incoming communication requests from a client, performs the requested service and dispatches a response to the client's request. So, the direction of initiation is used to categorize whether a AUTOSAR software-component is a client or a server.

A single component can be both a client and a server depending on the software realization.

The invocation of a server is performed by the RTE itself when a request is made by a client. The invocation occurs synchronously with respect to the RTE (typically via a function call) however the client's invocation can be either synchronous (wait for server to complete) or asynchronous with respect to the server.

Note: servers which have an asynchronous operation (i.e. they accept a request and another provide a feedback by invoking a server of the caller) should be avoided as the RTE does not know the link between these 2 client-server communications. In particular, the server should have no OUT (or INOUT) parameters because the RTE cannot perform the copy of the result in the caller's environment when the request was processed.

[SWS_Rte_06019] ⌈ The only mechanism through which a server can be invoked is through a client-server invocation request from a client. ⌋*(SRS_Rte_00029)*

The above requirement means that *direct invocation* of the function implementing the server outside the scope of the RTE is not permitted.

A server has a dedicated provide port and a client has a dedicated require port. To be able to connect a client and a server, both ports must be categorized by the same interface.

The client can be blocked (synchronous communication) respectively non-blocked (asynchronous communication) after the service request is initiated until the response of the server is received.

A server implemented by a RunnableEntity with attribute canBeInvokedConcurrently set to FALSE is not allowed to be invoked concurrently and since a server can have one or more clients the server may have to handle concurrent service calls (n:1 communication) the RTE must ensure that concurrent calls do not interfere.

[SWS_Rte_04515] ⌈The RTE shall ensure that call serialization[9] of the operation is enforced when the server runnable attribute canBeInvokedConcurrently is FALSE.
⌋*(SRS_Rte_00019, SRS_Rte_00033)*

Note that the same server may be called using both synchronous and asynchronous communication.

Note also that even when canBeInvokedConcurrently is FALSE, an AtomicSwComponentType might be instantiated multiple times. In this case, the implementation of the RunnableEntity can still be invoked concurrently from several tasks. However, there will be no concurrent invocations of the implementation with the same instance handle.

[SWS_Rte_04516] ⌈ The RTE's implementation of the client-server communication shall ensure that a service result is dispatched to the correct client if more than one client uses a service. ⌋*(SRS_Rte_00019, SRS_Rte_00080)*

The result of the client/server operation can be collected using "wake up of wait point", "explicit data read access" or "activation of runnable entity".

[SWS_Rte_07409] ⌈ The RTE generator shall support the optimization of a client-server call to a direct function call without interaction with the RTE or the communication services, at least when the following conditions are true:

- the server runnable's property canBeInvokedConcurrently is set to TRUE
- the client and server execute in the same partition, i.e. intra-partition Client-Server communication
- the ServerCallPoint is Synchronous
- the OperationInvokedEvent is not mapped to an OsTask

⌋

Note: In case the conditions in [SWS_Rte_04522] are fulfilled the RTE Generator may implement a client-server call with a direct function call, even when the server runnable's property canBeInvokedConcurrently is set to FALSE.

Since the communication occurs conceptually via the RTE (it is initiated via an RTE API call) the optimization does not violate the requirement that servers are only invoked via client-server requests (see Sect. 5.6.13, [SWS_Rte_06019]).

[SWS_Rte_07662] ⌈ The RTE Generator shall reject configurations where an ClientServerOperation has an ArgumentDataPrototype whose Implemen-

[9] Call Serialization ensures at most one thread of control is executing an instance of a runnable entity at any one time. An AUTOSAR software-component can have multiple instances (and therefore a runnable entity can also have multiple instances). Each instance represents a different server and can be executed in parallel by different threads of control thus serialization only applies to an individual instance of a runnable entity – multiple runnable entities within the same component instance may also be executed in parallel.

tationDataType is of category DATA_REFERENCE and whose direction is INOUT. ⌋*(SRS_Rte_00018, SRS_Rte_00019)*

[SWS_Rte_08731] ⌈ If the return value of the serialization call is not equal to E_OK the RTE shall not call Com_SendSignal ⌋*(SRS_Rte_00091)*

4.3.2.2 Multiplicity

Client-server interfaces contain two dimensions of multiplicity; multiple clients invoking a single server and multiple operations within a client-server interface.

4.3.2.2.1 Multiple Clients Single Server

Client-server communication involves an AUTOSAR software-component invoking a defined “server” operation in another AUTOSAR software-component which may or may not return a reply.

[SWS_Rte_04519] ⌈ The RTE shall support multiple clients invoking the same server operation (’n:1’ communication where n $\geq$ 1). ⌋*(SRS_Rte_00029)*

4.3.2.2.2 Multiple operations

A client-server interface contains one or more operations. A port of a AUTOSAR software-component that *requires* an AUTOSAR client-server interface to the component can independently invoke any of the operations defined in the interface [SRS_Rte_00089].

[SWS_Rte_04517] ⌈ The RTE API shall support independent access to operations in a client-server interface. ⌋*(SRS_Rte_00029)*

Example 4.8

Consider a client-server interface that has two operations, op1 and op2 and that an AUTOSAR software-component definition requires a port typed by the interface. As a result, the RTE generator will create two API calls; one to invoke op1 and another to invoke op2. The calls can invoke the server operations either synchronously or asynchronously depending on the configuration.

Recall that each data element in a sender-receiver interface is transmitted independently (see Section 4.3.1.3) and that the coherent transmission of multiple data items is achieved through combining multiple items into a single composite data type. The transmission of the parameters of an operation in a client-server interface is similar to a record since the RTE guarantees that all parameters are handled atomically [SRS_Rte_00073].

[SWS_Rte_04518] ⌈ The RTE shall treat the parameters and the results of a client-server operation atomically. ⌋*(SRS_Rte_00033)*

However, unlike a sender-receiver interface, there is no facility to combine multiple client-server operations so that they are invoked as a group.

4.3.2.2.3 Single Client Multiple Server

The RTE is *not* required to support multiple server operations invoked by a single client component request ('1:n' communication where n $>$ 1).

4.3.2.2.4 Call Serialization

Each client can invoke the server simultaneously and therefore the RTE is required to support multiple requests of servers. If the server requires call serialization, the RTE has to ensure it.

[SWS_Rte_04520] ⌈ The RTE shall support simultaneous invocation requests of a server operation. ⌋*(SRS_Rte_00019, SRS_Rte_00080)*

[SWS_Rte_04522] ⌈ The RTE shall ensure that the `RunnableEntity` implementing a server operation has completed the processing of a request before it begins processing the next request, if serialization is required by the server operation, i.e `canBeInvokedConcurrently` attribute of the server is set to FALSE and client `RunnableEntity`s to `OsTask` mapping (`RteEventToTaskMapping`) may lead to concurrent invocations of the server. ⌋*(SRS_Rte_00019, SRS_Rte_00033)*

When this requirement is met the operation is said to be "call serialized". A call serialized server only accepts and processes requests atomically and thus avoids the potential for conflicting concurrent access.

Client requests that cannot be serviced immediately due to a server operation being "busy" are required to be queued pending processing. The presence and depth of the queue is configurable.

If the `RunnableEntity` implementing the server operation is reentrant , i.e. `canBeInvokedConcurrently` attribute set to TRUE, no serialization is necessary. This allows to implement invocations of reentrant server operations as direct function calls without involving the RTE.

But even when the `canBeInvokedConcurrently` attribute is set to FALSE the RTE Generator still can utilize a direct function call, if the mapping of the client `RunnableEntity`s to `OsTask`s will not imply a concurrent execution of the server.

[SWS_Rte_08001] ⌈ If two operations are mapped to the same RunnableEntity, and [SWS_Rte_04522] requires a call serialization, then the operation invoked events shall be mapped to same task and they shall have the same position in task. Otherwise the RTE Generator shall reject configuration. ⌋*(SRS_Rte_00019, SRS_Rte_00033)*

[SWS_Rte_08002] ⌈ If two operations are mapped to the same RunnableEntity, and [SWS_Rte_04522] requires a call serialization, then a single queue is implemented for invocations coming from any of the operations. ⌋*(SRS_Rte_00019, SRS_Rte_00033)*

4.3.2.3 Communication Time-out

The ServerCallPoint allows to specify a timeout so that the client can be notified that the server is not responding and can react accordingly. If the client invokes the server synchronously, the RTE API call to invoke the server reports the timeout. If the client invokes the server asynchronously, the timeout notification is passed to the client by the RTE as a return value of the API call that collects the result of the server operation.

[SWS_Rte_03763] ⌈ The RTE shall ensure that timeout monitoring is performed for client-server communication, regardless of the receive mode for the result. ⌋*(SRS_Rte_00069, SRS_Rte_00029)*

If the server is invoked asynchronously and a WaitPoint is specified to collect the result, two timeout values have to be specified, one for the ServerCallPoint and one for the WaitPoint.

[SWS_Rte_03764] ⌈ The RTE generator shall reject the configuration if different timeout values are specified for the AsynchronousServerCallPoint and for the WaitPoint associated with the AsynchronousServerCallReturnsEvent for this AsynchronousServerCallPoint. ⌋*(SRS_Rte_00018)*

In asynchronous client-server communication the AsynchronousServerCallReturnsEvent associated with the AsynchronousServerCallPoint for an ClientServerOperation indicates that the server communication is finished or that a timeout occurred. The status information about the success of the server operation is available as the return value of the RTE API call generated to collect the result.

[SWS_Rte_03765] ⌈ For each asynchronous invocation of an operation prototype only one AsynchronousServerCallReturnsEvent shall be passed to the client component by the RTE. The AsynchronousServerCallReturnsEvent shall indicate either that the transmission was successful or that the transmission was not successful. ⌋*(SRS_Rte_00079)*

[SWS_Rte_03766] ⌈ The status information about the success or failure of the asynchronous server invocation shall be available as the return value of the RTE API call to retrieve the result. ⌋*(SRS_Rte_00079)*

After a timeout was detected, no result shall be passed to the client.

[SWS_Rte_03770] ⌈ In case Rte_Call API returns RTE_E_LIMIT, RTE_E_SERIALIZATION_LIMIT, RTE_E_SERIALIZATION_ERROR, RTE_E_COM_STOPPED, RTE_E_TIMEOUT, RTE_E_UNCONNECTED, RTE_E_IN_EXCLUSIVE_AREA or RTE_E_SEG_FAULT, the RTE shall not modify the OUT and INOUT parameters. ⌋*(SRS_Rte_00069, SRS_Rte_00029)*

[SWS_Rte_08310] ⌈ In case Rte_Result API returns RTE_E_NO_DATA, RTE_E_SERIALIZATION_ERROR, RTE_E_COM_STOPPED, RTE_E_TIMEOUT, RTE_E_UNCONNECTED, RTE_E_IN_EXCLUSIVE_AREA or RTE_E_SEG_FAULT, the RTE shall not modify the OUT and INOUT parameters. ⌋*(SRS_Rte_00069, SRS_Rte_00029)*

Since an asynchronous client can have only one outstanding server invocation at a time, the RTE has to monitor when the server can be safely invoked again. In normal operation, the server can be invoked again when the result of the previous invocation was collected by the client.

[SWS_Rte_03773] ⌈ If a server is invoked asynchronously and no timeout occurred, the RTE shall ensure that the server can be invoked again by the same client, after the result was successfully passed to the client. ⌋*(SRS_Rte_00069)*

In intra-partition client-server communication, the RTE can determine whether the server runnable is still running or not.

[SWS_Rte_03771] ⌈ If a timeout was detected in asynchronous intra-partition client-server communication, the RTE shall ensure that the server is not invoked again by the same client until the server runnable has terminated. ⌋*(SRS_Rte_00069, SRS_Rte_00079)*

In inter-ECU communication, the client RTE has no knowledge about the actual status of the server. The response of the server could have been lost because of a communication error or because the server itself did not respond. Since the client-side RTE cannot distinguish the two cases, the client must be able to invoke the server again after a timeout expired. As partitions in one ECU are decoupled in a similar way like separate ECUs, and can be restarted separately, client server communication should behave similar for inter-ECU and intra-partition communication.

[SWS_Rte_03772] ⌈ If a timeout was detected in asynchronous inter-ECU or inter-partition client-server communication, the RTE shall ensure that the server can be invoked again by the same client after the timeout notification was passed to the client. ⌋*(SRS_Rte_00069, SRS_Rte_00079)*

Note that this might lead to client and server running out of sync, i.e. the response of the server belongs to the previous, timed-out invocation of the client. The application has to handle the synchronization of client and server after a timeout occurred.

[SWS_Rte_03767] ⌈ If the timeout value of the ServerCallPoint is 0, no timeout monitoring shall be performed. ⌋*(SRS_Rte_00069, SRS_Rte_00029)*

[SWS_Rte_03768] ⌈ If the canBeInvokedConcurrently attribute of the server runnable is set to TRUE, no timeout monitoring shall be performed if the RTE API call to invoke the server is implemented as a direct function call. ⌋*(SRS_Rte_00069, SRS_Rte_00029)*

[SWS_Rte_02709] ⌈ In case of inter partition communication, if the partition of the server is stopped or restarting at the invocation time of the server call or during the operation of the server call, the RTE shall immediately provide a timeout indication to the client. ⌋

Note: In case of inter-ECU or interpartition client-server communication it is recommended to always specify a timeout>0. Otherwise in case of a full server queue the client would wait for the server response infinitely.

4.3.2.4 Port-Defined argument values

Port-defined argument values exist in order to support interaction between Application Software Components and Basic Software Modules.

Several Basic Software Modules use an integer identifier to represent an object that should be acted upon. For instance, the NVRAM Manager uses an integer identifier to represent the NVRAM block to access. This identifier is not known to the client, as the client must be location independent, and the NVRAM block to access for a given application software component cannot be identified until components have been mapped onto ECUs.

There is therefore a mismatch between the information available to the client and that required by the server. Port-defined argument values bridge that gap.

The required port-defined arguments (the fact that they are required, their data type and their values) are specified within the input to the RTE generator.

[SWS_Rte_01360] ⌈ When invoking the runnable entity specified for an OperationInvokedEvent, the RTE shall include the port-defined argument values between the instance handle (if it is included) and the operation-specific parameters, in the order they are given in the Software Component Template Specification [2]. ⌋*(SRS_Rte_00152)*

Requirement [SWS_Rte_01360] means that a client will make a request for an operation on a require (Client-Server) port including only its instance handle (if required) and the explicit operation parameters, yet the server will be passed the implicit parameters as it requires.

Note that the values of implicit parameters are constant for a particular server runnable entity; it is therefore expected that using port-defined argument values imposes no RAM overhead (beyond any extra stack required to store the additional parameters).

4.3.2.5 Buffering

Client-Server-Communication is a two-way-communication. A request is sent from the client to the server and a response is sent back.

Unless a server call is implemented as direct function call, the RTE has to store or buffer the communication on the corresponding receiving sides, requests on server side and responses on client side, respectively:

- **[SWS_Rte_02527]** ⌈ Unless a server call is implemented as a direct function call, the RTE shall buffer a request on the server side in a first-in-first-out queue as described in chapter 4.3.1.10.2 for queued data elements.

  Note: The data that shall be buffered is implementation specific but at least RTE should store the IN parameters, the IN/OUT parameters and a client identifer. ⌋*(SRS_Rte_00019, SRS_Rte_00033, SRS_Rte_00110)*

- **[SWS_Rte_02528]** ⌈ Unless a server call is implemented as a direct function call, RTE shall keep the response on the client side in a queue with queue length 1.

  Note: The data that shall be buffered is implementation specific but at least RTE should store the IN/OUT parameters, the OUT parameters and the error code. ⌋*(SRS_Rte_00019, SRS_Rte_00033)*

For the server side, the `queueLength` attribute of `ServerComSpec` specifies the length of the queue.

[SWS_Rte_02529] ⌈ The RTE generator shall reject a `queueLength` attribute of a `ServerComSpec` with a queue length ≤ 0. ⌋*(SRS_Rte_00033, SRS_Rte_00110, SRS_Rte_00018)*

[SWS_Rte_02530] ⌈The RTE shall use the queue of requests to call serialise access to a server. ⌋*(SRS_Rte_00033, SRS_Rte_00110)*

A buffer overflow of the server is not reported to the client. The client will receive a time out.

[SWS_Rte_07008] ⌈ If a server call is implemented by direct function call the RTE shall not create any copy for parameters passed by reference. ⌋*(SRS_Rte_00033, SRS_Rte_00110)*

Therefore, it is the responsibility of the application to provide consistency mechanisms for referenced parameters if necessary.

4.3.2.6 Inter-ECU and Inter-Partition Response to Request Mapping

RTE is responsible to map a response to the corresponding request. With this mapping, RTE can activate or resume the corresponding runnable and provide the response to the correct client. The following situations can be distinguished:

- Mapping of a response to the correct request within one ECU. In general, this is solved already by the call stack. The details are implementation specific and will not be discussed in this document.
- Mapping of a response coming from a different partition or a different ECU.

The problem of request to response mapping in inter-ECU and inter-Partition communication can be split into:

- Mapping of a response to the correct client. This is discussed in 4.3.2.6.1.
- Mapping of a response to the correct request within of one client. This is discussed in 4.3.2.6.2.

The general approach for the inter-ECU and inter-Partition request response mapping is to use transaction handles.

[SWS_Rte_02649] ⌈ In case of inter-ECU client-server communication, the transaction handle shall contain two parts of unsigned integer type:

- Client Identifier
- Client Sequence Counter

⌋*(SRS_Rte_00027, SRS_Rte_00082)*

[SWS_Rte_08711] ⌈ The Client Identifier of the transaction handle used for a inter-ECU client server communication shall be of type `uint8`, `uint16` or `uint32`. It shall have at least the size given by the maximum `lengthClientId` attribute of all `ClientServerToSignalMapping`s. See [constr_9065]. ⌋*(SRS_Rte_00082, SRS_Rte_00091)*

[SWS_Rte_08712] ⌈ The Client Sequence Counter part of the transaction handle used for a inter-ECU client server communication is optional. If existing it shall be of type `uint8`, `uint16` or `uint32` and have at least the size given by the maximum `lengthSequenceCounter` attribute of all `ClientServerToSignalMapping`s. See [constr_9065]. ⌋*(SRS_Rte_00082, SRS_Rte_00091)*

[SWS_Rte_08713] ⌈ If the `lengthClientId` attribute of the all `ClientServerToSignalMapping` is zero the client identifier part of the transaction handle shall be omitted. See [constr_9065]. ⌋*(SRS_Rte_00082, SRS_Rte_00091)*

The presence of an part of the transaction handle is an input to the RTE Generator and up to the system design.

[SWS_Rte_07346] ⌈ In case of inter-Partition client-server communication, the RTE shall not communicate any response to the client if the client is part of a partition that was restarted since the request was sent. ⌋*(SRS_Rte_00027, SRS_Rte_00082)*

[SWS_Rte_07346] could be implemented with a transaction handle that contains a sequence counter.

[SWS_Rte_02651] ⌈ In case of inter-ECU client-server communication, the transaction handle shall be used for the identification of client server transactions communicated via COM. ⌋*(SRS_Rte_00027, SRS_Rte_00082)*

[SWS_Rte_02653] ⌈ The RTE on the server side shall return the transaction handle of the request without modification together with the response. ⌋*(SRS_Rte_00027, SRS_Rte_00082)*

Since there is always at most one open request per client (see [SWS_Rte_02658]), the transaction handle can be kept within the RTE and does not have to be exposed to the *AUTOSAR SW-C*.

4.3.2.6.1 Client Identifier

In case of a server on one ECU with clients on other ECUs, the inter-ECU client-server communication has to use different unique `SystemSignal`s for each client-ECU to allow the identification of the client-ECU associated with each client call.

[SWS_Rte_02579] ⌈ The RTE Generator shall reject configurations where there is inter-ECU client-server communication from several client-ECUs using the same `SystemSignal`s. ⌋*(SRS_Rte_00029, SRS_Rte_00082, SRS_Rte_00018)*

With this mechanism, the server-side RTE must handle the fan-in. This is done in the same way as for sender-receiver communication.

However it is allowed to have several clients in one client-ECU communicating using inter-ECU client-server communication with a server on a different ECU, if the client identifier is used to distinguish the different clients.

[SWS_Rte_05111] ⌈ The RTE Generator shall reject configurations where there is inter-ECU client-server communication from several clients on the same client-ECU and no client identifier is configured for at least one of these inter-ECU client-server communications. ⌋*(SRS_Rte_00018, SRS_Rte_00029, SRS_Rte_00082)*

[SWS_Rte_03769] ⌈ If multiple clients have access to one server, the RTE on the server side has to queue all incoming server invocations while ensuring data consistency. ⌋*(SRS_Rte_00019, SRS_Rte_00029, SRS_Rte_00080)*

4.3.2.6.2 SequenceCounter

The purpose of sequence counters is to map a response to the correct request of a known client.

[SWS_Rte_02658] ⌈ In case of inter-ECU and inter-Partition communication, RTE shall allow only one request per client and server operation at any time. ⌋*(SRS_Rte_00079)*

[SWS_Rte_02658] does not apply to intra-partition communication because there can be several `execution-instances`.

[SWS_Rte_02658] implies under normal operation that a response can be mapped to the previous request. But, when a request or response is lost or delayed, this order can get out of phase. To allow a recovery from lost or delayed signals, a sequence counter is used. The sequence counter can also be used to detect stale responses after a restart of the client side RTE and SW-C.

[SWS_Rte_02654] ⌈ RTE shall support a sequence counter for the inter ECU client server connection where configured in the input information. ⌋*(SRS_Rte_00027, SRS_Rte_00082)*

[SWS_Rte_02655] ⌈ RTE shall initialize all sequence counters with zero during `Rte_Start`. ⌋*(SRS_Rte_00082)*

[SWS_Rte_02656] ⌈ RTE shall increase each sequence counter in a cyclic manner after a client server operation has finished successfully or with a timeout. ⌋*(SRS_Rte_00082)*

[SWS_Rte_02657] ⌈ RTE shall ignore incoming responses that do not match the sequence counter. ⌋*(SRS_Rte_00027, SRS_Rte_00082)*

4.3.2.7 Parameter Serialization

[SWS_Rte_08710] ⌈ For inter-ECU client-server communication, the RTE of the server ECU shall serialize the server's results using the `BswModuleEntry` referenced by the `RteSerializedSignal` of the `returnSignal`. ⌋*(SRS_Rte_00027, SRS_Rte_00082, SRS_Rte_00091, SRS_Rte_00123)*

[SWS_Rte_08700] ⌈ The RTE generator shall reject an input configuration where a configured inter-ECU client-server communication of a client ECU (a `ClientServerOperation` of a `PortPrototype` of one Software Component instance) is not referenced by one and only one `ClientServerToSignalMapping`. ⌋*(SRS_Rte_00018, SRS_Rte_00027, SRS_Rte_00082, SRS_Rte_00091)*

[SWS_Rte_08701] ⌈ The RTE generator shall reject an input configuration where a configured inter-ECU client-server communication of a server ECU (a `ClientServerOperation` of a `PortPrototype` of one Software Component instance) is not referenced by at least one `ClientServerToSignalMapping`. ⌋*(SRS_Rte_00018, SRS_Rte_00027, SRS_Rte_00082, SRS_Rte_00091)*

[SWS_Rte_08702] ⌈ The RTE generator shall reject an input configuration where the `callSignal` or `returnSignal` of a `ClientServerToSignalMapping` is not referenced by a `RteSerializedSignal`. ⌋*(SRS_Rte_00018, SRS_Rte_00027, SRS_Rte_00082, SRS_Rte_00091)*

[SWS_Rte_08703] ⌈ For an inter-ECU client-server communication, the RTE of the client ECU shall communicate the request to a remote server using the `callSignal` of the `ClientServerToSignalMapping` which references the operation instance. ⌋*(SRS_Rte_00027, SRS_Rte_00082, SRS_Rte_00091)*

[SWS_Rte_08704] ⌈ For an inter-ECU client-server communication, the RTE of the client ECU shall serialize the client's request using the `BswModuleEntry` referenced by the `RteSerializedSignal` of the `callSignal`. ⌋*(SRS_Rte_00027, SRS_Rte_00082, SRS_Rte_00091)*

[SWS_Rte_08705] ⌈ For an inter-ECU client-server communication, the RTE of the client ECU shall receive the results of a remote server using the `returnSignal` of the `ClientServerToSignalMapping` which references the operation instance. ⌋*(SRS_Rte_00027, SRS_Rte_00082, SRS_Rte_00091, SRS_Rte_00123)*

[SWS_Rte_08706] ⌈ For an inter-ECU client-server communication, the RTE of the client ECU shall deserialize the server's results using the `BswModuleEntry` referenced by the `RteSerializedSignal` of the `returnSignal`. ⌋*(SRS_Rte_00027, SRS_Rte_00082, SRS_Rte_00091, SRS_Rte_00123)*

[SWS_Rte_08730] ⌈ If the return value of the deserializer call is not equal to `E_OK` the RTE on the server side shall not invoke the server runnable. ⌋*(SRS_Rte_00027, SRS_Rte_00082, SRS_Rte_00091)*

[SWS_Rte_08707] ⌈ For an inter-ECU client-server communication, the RTE of the server ECU shall receive a request of a remote client using the `callSignal` of the `ClientServerToSignalMapping` which references the operation instance. ⌋*(SRS_Rte_00027, SRS_Rte_00082, SRS_Rte_00091)*

[SWS_Rte_08708] ⌈ For an inter-ECU client-server communication, the RTE of the server ECU shall deserialize a client's request using the `BswModuleEntry` referenced by the `RteSerializedSignal` of the `callSignal`. ⌋*(SRS_Rte_00027, SRS_Rte_00082, SRS_Rte_00091)*

[SWS_Rte_08709] ⌈ For inter-ECU client-server communication, the RTE of the server ECU shall communicate the results to a remote client using the `returnSignal` of the `ClientServerToSignalMapping` which references the operation instance. ⌋*(SRS_Rte_00027, SRS_Rte_00082, SRS_Rte_00091, SRS_Rte_00123)*

4.3.2.8 Operation

4.3.2.8.1 Inter-ECU Mapping

The client server protocol defines how a client call and the server response are mapped onto the communication infrastructure of AUTOSAR in case of inter-ECU communication. This allows RTE implementations from different vendors to interpret the client server communication in the same way.

The AUTOSAR System Template [8] does specify a protocol for the client server communication in AUTOSAR.

4.3.2.8.2 Atomicity

The requirements for atomicity from Section 4.3.1.11.2 also apply for the composite data types described in Section 4.3.2.8.1.

4.3.2.8.3 Fault detection and reporting

Client Server communication may encounter interruption like:

- Buffer overflow at serialization
- Buffer overflow at the server side.
- Communication interruption.
- Server might be inaccessible for some reason.

The client specifies a timeout that will expire in case the server or communication fails to complete within the specified time. The reporting method of an expired timeout depends on the communication attributes:

- If the C/S communication is synchronous the RTE returns `RTE_E_TIMEOUT` on the `Rte_Call` function (see section 5.6.13).
- If the C/S communication is asynchronous the RTE returns `RTE_E_TIMEOUT` on the `Rte_Result` function (see section 5.6.14).

In the case that RTE detects that the COM service is not available when forwarding signals to COM, the RTE returns `RTE_E_COM_STOPPED` on the `Rte_Call` (see section 5.6.13).

If the client still has an outstanding server invocation when the server is invoked again, the RTE returns `RTE_E_LIMIT` on the `Rte_Call` (see chapter 5.6.13).

In the absence of structural errors, application errors will be reported if present.

4.3.2.8.4 Asynchronous Client Server communication

Figure 4.39 shows a sequence diagram of how asynchronous client server communication may be implemented by RTE.

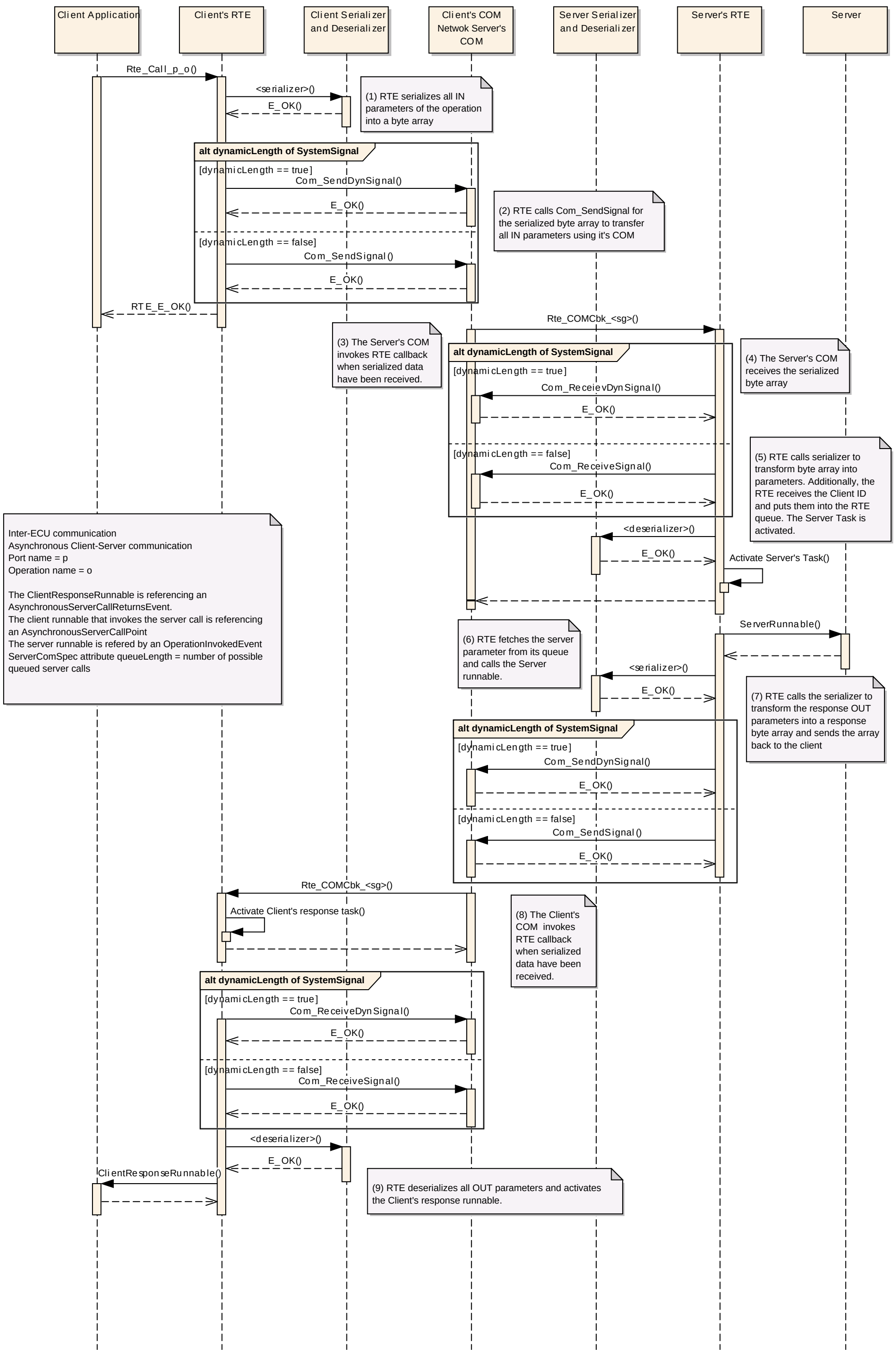


Figure 4.39: Client Server asynchronous

4.3.2.8.5 Synchronous Client Server communication

Figure 4.40 shows a sequence diagram of how synchronous client server communication may be implemented by RTE.

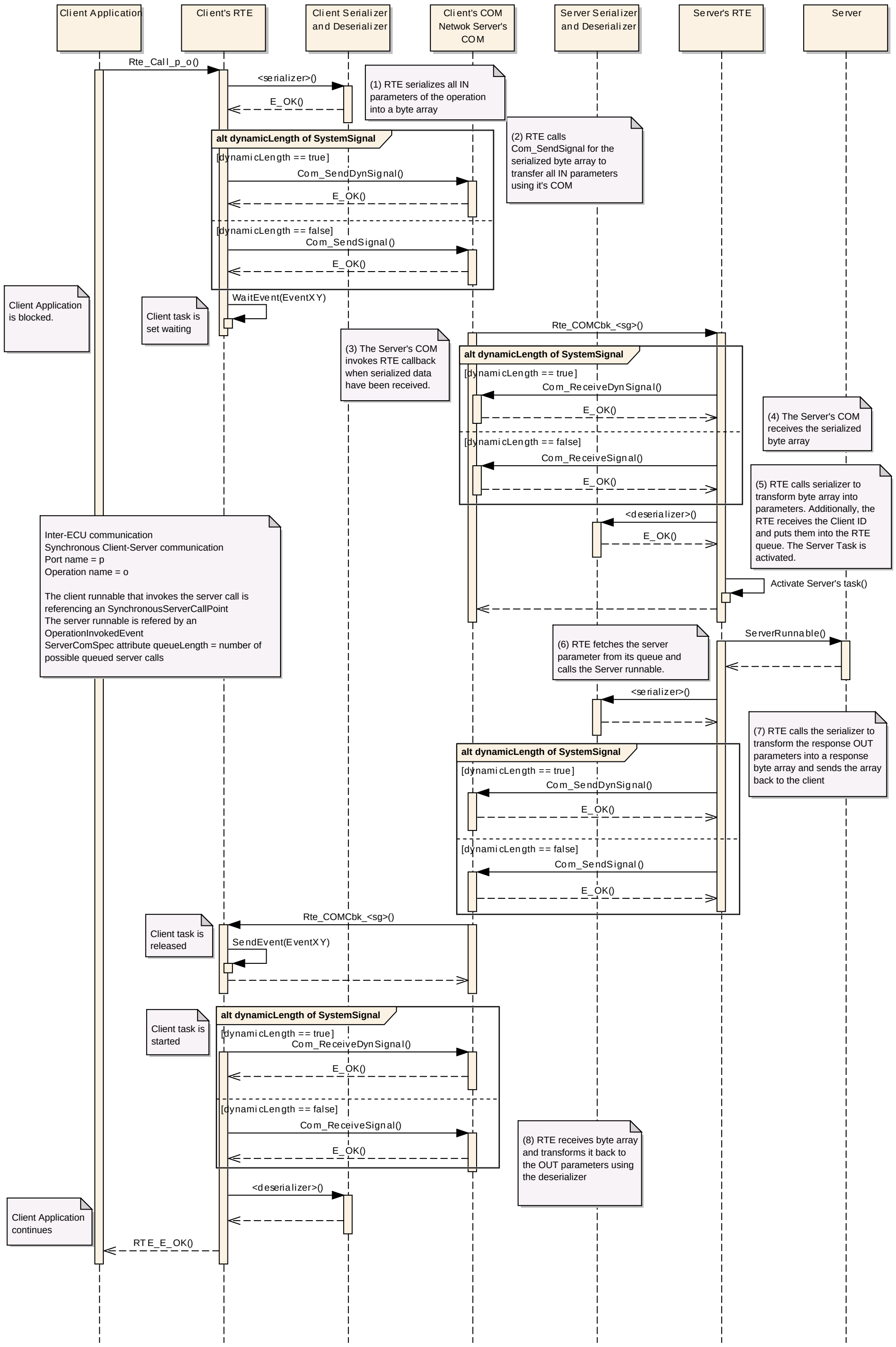


Figure 4.40: Client Server synchronous

4.3.3 SWC internal communication

4.3.3.1 Inter Runnable Variables

Sender/Receiver and Client/Server communication through AUTOSAR ports are the model for communication between AUTOSAR SW-Cs.

For communication between Runnables inside of an AUTOSAR SW-C the AUTOSAR SW-C Template [2] establishes a separate mechanism. Non-composite AUTOSAR SW-C can reserve InterRunnableVariables which can only be accessed by the Runnables of this one AUTOSAR SW-C. The Runnables might be running in the same or in different task contexts. Read and write accesses are possible.

[SWS_Rte_03589] ⌈ The RTE shall support *Inter Runnable Variable*s for single and multiple instances of AUTOSAR SW-Cs. ⌋*(SRS_Rte_00142)*

[SWS_Rte_07187] ⌈ The generated RTE shall initialize a defined `implicitInterRunnableVariable` and `explicitInterRunnableVariable` according to the `ValueSpecification` of the `VariableDataPrototype` defining the `implicitInterRunnableVariable` respectively `explicitInterRunnableVariable` if the general initialization conditions in [SWS_Rte_07046] and [SWS_Rte_03852] are fulfilled. ⌋*(SRS_Rte_00142)*

InterRunnableVariables have a behavior corresponding to Sender/Receiver communication *between* AUTOSAR SW-Cs (or rather between Runnables of different AUTOSAR SW-Cs).

But why not use Sender/Receiver communication directly instead? Purpose is data encapsulation / data hiding. Access to InterRunnableVariables of an AUTOSAR SW-C from other AUTOSAR SWCs is not possible and not supported by RTE. InterRunnableVariable content stays SW-C internal and so no other SW-C can use it. Especially not misuse it without understanding how the data behaves.

Like in Sender/Receiver (S/R) communication between AUTOSAR SW-Cs two different behaviors exist:

1. *Inter Runnable Variable*s with *implicit* behavior (`implicitInterRunnableVariable`)
   This behavior corresponds with `VariableAccess`es in the `dataReadAccess` and `dataWriteAccess` roles of Sender/Receiver communication and is supported by *implicit S/R API* in this specification.

   Note:
   If a `VariableAccess` in the `writtenLocalVariable` role referring to a `VariableDataPrototype` in the `implicitInterRunnableVariable` role is specified for a certain interrunnable variable, but no RTE API for implicit write of this interrunnable variable is called during an execution of the runnable, an undefined value is written back when the runnable terminates.

For more details see section 4.2.5.6.1.
For APIs see sections 5.6.23 and 5.6.24.

Note 2:
As for the Implicit Sender/Receiver communication, the implicit concept for InterRunnableVariables implies that the runnable does terminate. For runnable entities of category 2, the behavior is guaranteed only if it has a finite execution time. A category 2 runnable that runs forever will not have its data updated.

2. *Inter Runnable Variable*s with *explicit* behavior (`explicitInterRunnableVariable`)
   This behavior corresponds with `VariableAccess`es in the `dataSendPoint`, `dataReceivePointByValue`, or `dataReceivePointByArgument` roles of Sender/Receiver communication and is supported by *explicit S/R API* in this specification.

   For more details see section 4.2.5.6.2
   For APIs see sections 5.6.25 and 5.6.26.

4.3.4 Inter-Partition communication

`Partition`s are used to decompose an ECU into functional units. Partitions can contain both SW-Cs and BSW modules. The partitioning is done to protect the software contained in the partitions against each other or to increase the performance by running the partitions on different cores of a multi core controller.

Since the partitions may be separated by core boundaries or memory boundaries and since the partitions can be stopped and restarted independently, the observable behavior to the SW-Cs for the communication between different partitions is rather similar to the inter ECU communication than to the intra partition communication. The RTE needs to use special mechanisms to communicate from one partition to another.

Like for the inter ECU communication, inter partition communication uses the connectionless communication paradigm. This means, that a send operation is successful for the sender, even if the receiving partition is stopped. A receiver will only, by means of a timeout, be notified if the partition of the sender is stopped.

Unlike most basic software, the RTE does not have a main processing function. The execution logic of the RTE is contained in the generated task bodies, the wrapper code around the runnables whose execution RTE manages.

As the tasks that contain the SW-Cs runnables are uniquely assigned to partitions (see page 11EER of [15]), the execution logic of the RTE is split among the partitions. It can not be expected that the RTE generated wrapper code running in one partition can directly access the memory objects assigned to the RTE part of another partition.

In this sense, there is one RTE per partition, that contains runnable entities.

Still, RTE is responsible to support the communication between SW-Cs allocated to the different partitions. According to the AUTOSAR software layered architecture [], RTE has to be independent of the micro controller architecture. AUTOSAR supports a wide variety of multi core and memory protection architectures.

[SWS_Rte_02734] ⌈ The RTE generator shall support a mode in which the generated code is independent of the micro controller. ⌋*(SRS_BSW_00161)*

It can not be generally assumed that a cache coherent, shared memory is available for the communication between partitions. Direct memory access and function calls across partition boundaries are generally not possible. In the extreme case, communication might even be limited to a message passing interface.

To allow memory protection and multi core support in spite of [SWS_Rte_02734], the AUTOSAR OS provides a list of mechanisms, that can be used for the communication across cores (see [4]). Especially, the IOC has been designed to support the communication needs of RTE in a way that should not introduce additional run time overhead.

If a communication between Basic Software Modules is necessary for which the IOC does not suffice, for example Sender-Receiver or Client-Server communication, there are also mechanisms provided by the Basic Software Scheduler. These mechanisms follow the Client-Server communication pattern or the Sender-Receiver communication pattern of the VFB but cannot be used for inter-ECU communication. The Basic Software Scheduler can internally use the IOC to cross the partition boundaries. See [22].

The following sections describe the use of some OS mechanisms that are designed for inter partition communication.

4.3.4.1 Inter partition data communication using IOC

The general idea to allow the data communication between partitions in a most efficient way and still be independent of the micro controller implementation is to take the buffers and queues from the intra partition communication case and replace them with so called IOC communication objects in the inter partition communication case.

In the ideal case, the access macros to the IOC communication object resolve to a direct access to shared memory.

The IOC (Inter OS-Application Communication) is a feature of the AUTOSAR OS, which provides a data oriented communication mechanism between partitions. The IOC provides communication buffers, queues, and protected access functions/macros to these buffers that can be used from any pre-configured partitions concurrently.

The IOC offers communication of data to another core or between memory protected partitions with guarantee of data consistency.

All data communications including the passing of parameters and return values in client server communication, can be implemented by using the IOC. The basic principle for using the IOC is to replace the RTE internal communication buffers by IOC buffers.

The IOC supports 1:1 and N:1 communication. For 1:N communication, N IOC communication objects have to be used. The IOC is configured and provides generated APIs for each IOC communication object. In case of N:1 communication, each sender has a separate API.

The IOC API is not reentrant.

[SWS_Rte_02737] ⌈ RTE shall prevent concurrent access to the same IOC API from different `ExecutableEntity` `execution-instances`. ⌋

The IOC will use the appropriate mechanism to communicate between the partitions, whether it requires communicating with another core, communicating with a partition with a different level of trust, or communicating with another memory partition.

The IOC channels are configured in the OS Configuration. Their configurations has to be provided as inputs for the RTE generator when the external configuration switch `strictConfigurationCheck` [SWS_Rte_05148] is set to true, and can be provided by the RTE Generator or RTE Configuration Editor when `strictConfigurationCheck` is set to false (see [SWS_Rte_05150]).

The IOC APIs use:

1. types declared by user on input to RTE (sender-receiver communication across OsApplication boudaries).
2. types created by RTE to collect client-server operation arguments into single data structure.

For the second item, RTE uses internal types that have to be described as ImplementationDataTypes (see [SWS_Rte_08400]).

The signaling between partitions is not covered by the IOC. The callbacks of IOC are in interrupt context and are mainly intended for direct use by BSW. For the signaling between partitions, RTE can use the activation of tasks or setting of events, see section 4.3.4.4.

[SWS_Rte_02736] ⌈ The RTE shall not execute ExecutableEntitys in the context of IOC callbacks. ⌋

This is necessary to ensure that ExecutableEntitys will not be executed in interrupt context or when a partition is terminated or restarted.

4.3.4.2 Inter partition data communication using Basic Software Scheduler

The Basic Software Scheduler provides Sender-Receiver and Client-Server communications mechanisms for communication between Basic Software Modules in different partitions. Therefore these communication paradigms can be used by Basic Software Modules in a multi core environment.

The usage is described in [9].

For Sender-Receiver communication currently only "explicit" transmission of data elements with "event" semantic (queued) is supported.

[SWS_Rte_08763] ⌈ For inter-ECU Sender-Receiver communication the length of the queue is specified by the attribute queueLength of the BswQueuedDataReceptionPolicy which references through receivedData the VariableDataPrototype of the Sender-Receiver communication. ⌋*(SRS_Rte_00243)*

[SWS_Rte_08764] ⌈ The RTE generator shall reject a queueLength attribute of a BswQueuedDataReceptionPolicy with a queue length ≤ 0. ⌋*(SRS_Rte_00243)*

4.3.4.3 Accessing COM from slave core in multicore configuration

In case of a multi core configuration, if a software component on the slave core wants to send data to a software component on another ECU, the RTE has to send data from the slave core through the IOC to the master core which in turn calls the send API of COM. The same behavior is required for receive case where the master core is responsible for forwarding received COM data to slave core through IOC.

[SWS_Rte_08306] ⌈ It is the RTEs responsibility to interact with COM whenever it is needed. ⌋

This requires some special handling by the RTE since it implies, at least in the send case, the need of a scheduable entity to do the actual call of COM send API.

[SWS_Rte_08307] ⌈ The RTE shall generate two (*BswSchedulableEntity*'s):

- `Rte_ComSendSignalProxyPeriodic`.
- `Rte_ComSendSignalProxyImmediate`.

`Rte_ComSendSignalProxyPeriodic` shall handle the sending of periodic signals and `Rte_ComSendSignalProxyImmediate` shall handle the sending of immediate signals. ⌋

[SWS_Rte_08308] ⌈ It shall be a possible to configure whether the return value of RTE APIs is based on RTE-IOC interaction or RTE-COM interaction using the configuration parameter `RteIocInteractionReturnValue`. A warning should preferably be issued in case RTE-COM interaction return value is chosen since that will cause major performance decrease. ⌋

4.3.4.3.1 Example sequence diagrams of accessing COM from Slave core

Figure 4.41 shows a sequence diagram of how receive data through COM from slave core may be implemented by RTE.

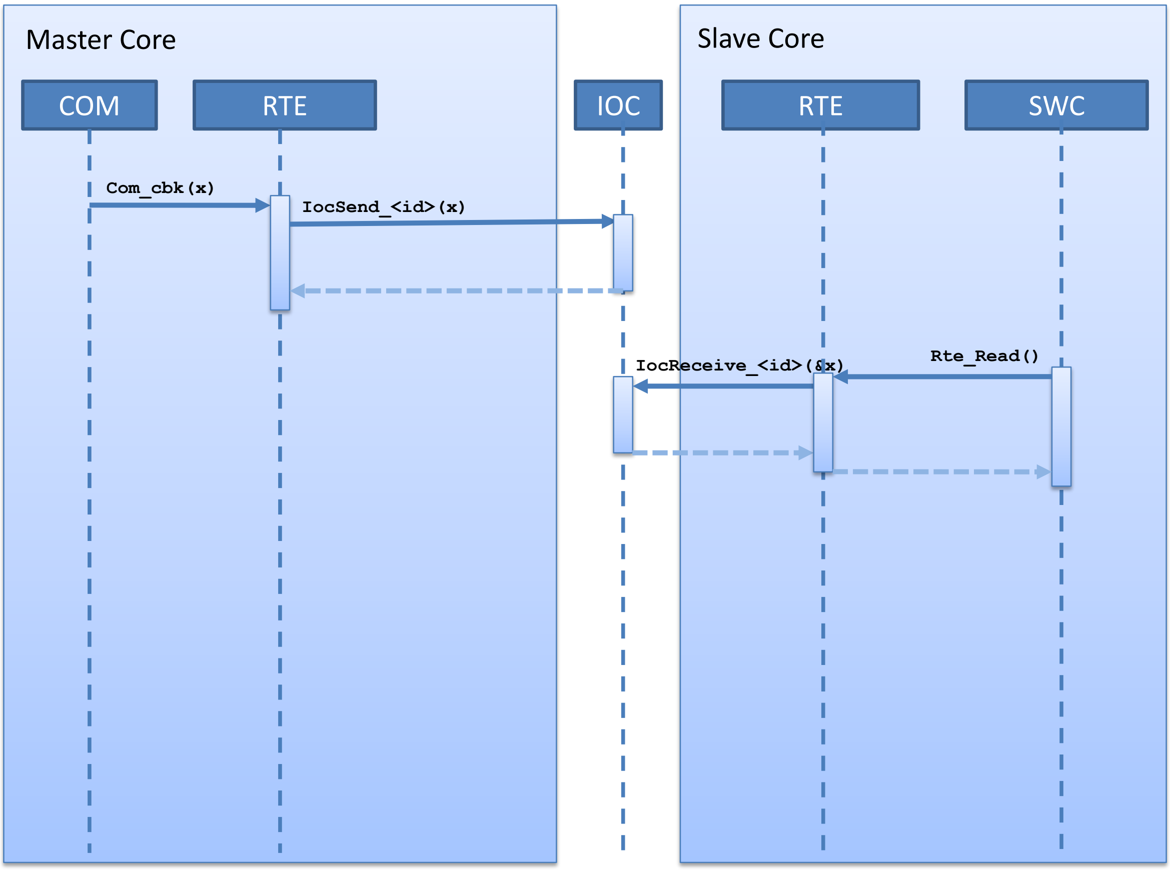


Figure 4.41: Receive data through COM from slave core

Figure 4.42 shows a sequence diagram of how send from COM to slave core may be implemented by RTE.

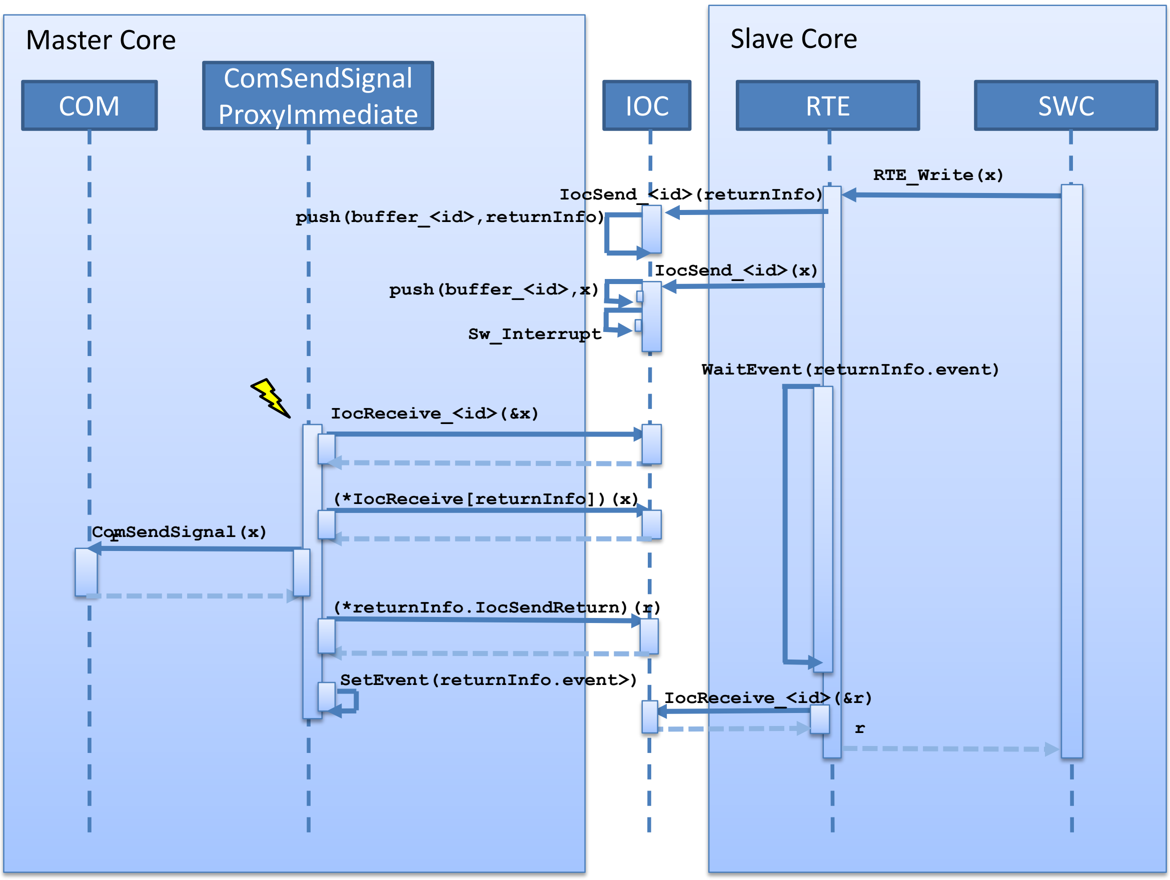


Figure 4.42: Send data through COM from slave core

Figure 4.43 shows a sequence diagram of how send from COM to slave core using return value based on RTE-IOC interaction may be implemented by RTE.

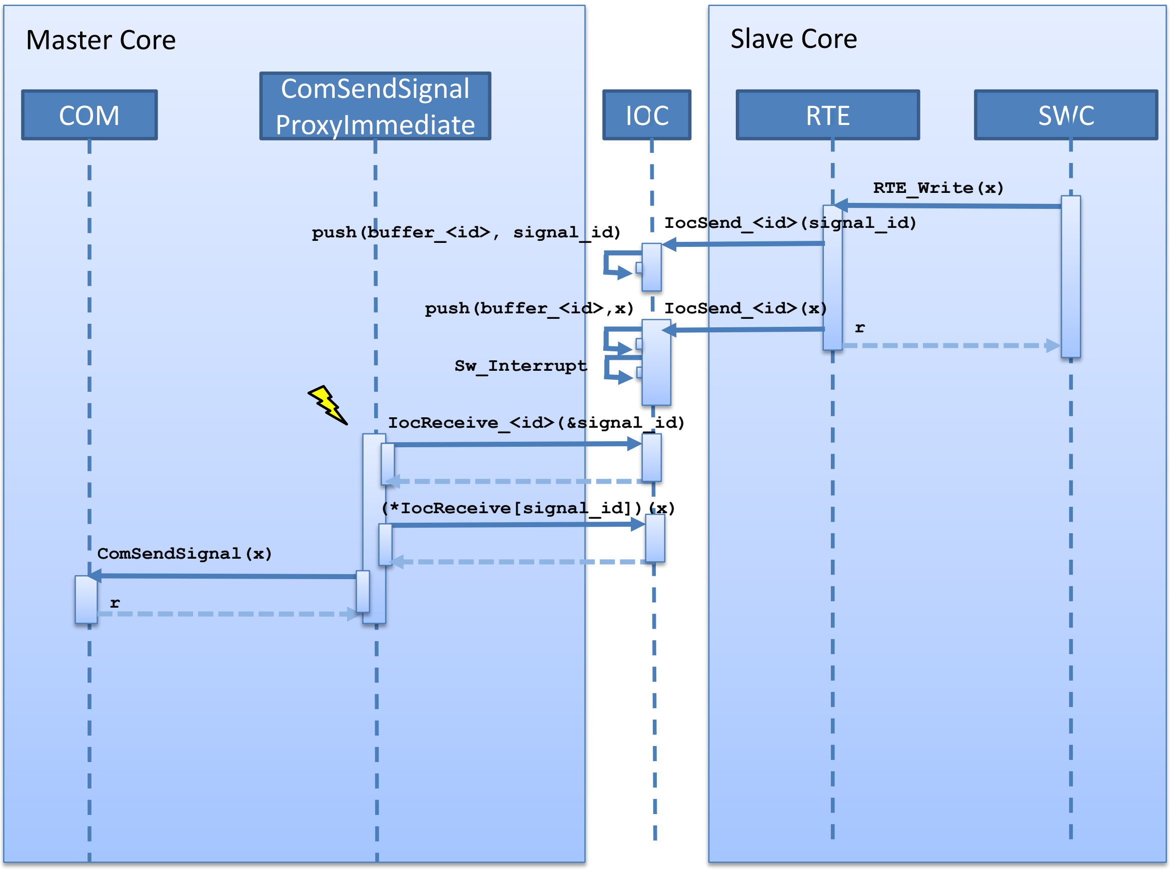


Figure 4.43: Send data through COM from slave core using return value based on RTE-IOC interaction

4.3.4.4 Signaling and control flow support for inter partition communication

The OS representation of a partition is an OS Application.

This is a (non-exhaustive) summary of OS features that can be used for signaling and control flow across partition boundaries:

- activation of tasks
- start and stop of schedule tables
- event signaling
- alarms
- spin locks (for inter core synchronization)

The following are not available for inter core signaling:

- OS Resource
- DisableAllInterrupts

For inter core synchronization, spin locks are provided. But, for efficiency reasons they should be used with care.

4.3.4.5 Trusted Functions

The call-trusted-function mechanism of AUTOSAR OS can be used in a memory protected controller to implement a function call from an untrusted to a trusted partition.

This Trusted Partition is a partition that has full access to the OS objects of other partitions on the same core. The Basic Software is assumed to reside in a trusted partition. It is assumed that the trusted partition cannot be terminated or restarted.

The typical use case for the call-trusted-function mechanism are AUTOSAR services which are usually provided by a client/server interface where the service side resides together with the basic software in the trusted partition.

Beware that this mechanism can not be used between two untrusted partitions or between cores.

The trusted functions are configured in the OS Configuration. Their configurations shall be provided as inputs for the RTE generator when the external configuration switch `strictConfigurationCheck` [SWS_Rte_05148] is set to true, and can be provided by the RTE Generator or RTE Configuration Editor when `strictConfigurationCheck` is set to false (see [SWS_Rte_05150]).

[SWS_Rte_07606] ⌈ Direct start of an `ExecutableEntity` execution-instance by the mean of a trusted function shall only be used for the start of an `ExecutableEntity` in the Trusted Partition. ⌋ *(SRS_Rte_00195, SRS_Rte_00210)*

The OS ensures that the partition of the caller is not terminated or restarted when a trusted function is executed. If needed, the termination or restart of the caller's partition is delayed after the trusted function returns.

RTE has to ensure, that the OS does not kill an RTE-generated task due to stopping or restarting a partition while this task is executing a function call to BSW or to the software component of another partition when this call is not a pure function.

For this purpose, RTE can use either the OS mechanism of trusted function call, or it can allocate the server to a different task than the client.

[SWS_Rte_02761] ⌈ In a partitioned system that supports stop or restart of partitions, the RTE shall not use a direct function call (without use of OS call trusted function) from a task of an untrusted partition to BSW or to the SW-C of another partition unless this is a pure function. ⌋*(SRS_Rte_00196)*

Please note that [SWS_Rte_02761] might require the use of OS call trusted function for a partitioned system even without memory protection.

4.3.4.6 Memory Protection and Pointer Type Parameters in RTE API

In a memory protected ECU, a SW-C from an untrusted partition might misuse the transition to the trusted context to modify memory in another partition. This can occur when a pointer to a different memory partition is passed from the untrusted partition to the trusted context. The RTE shall avoid this misuse by at least checking the validity of the address of the pointer, and, where possible, also checking the integrity of the associated memory object.

[SWS_Rte_02752] ⌈ When a SW-C in an untrusted partition receives (OUT parameter) or provides (IN parameter with composite data type) an `ArgumentDataPrototype` or `VariableDataPrototype`, it hands over a pointer to a memory object to an RTE API. The RTE shall only forward this pointer to a trusted SW-C after it has checked that the whole memory object is owned by the caller's partition. ⌋*(SRS_Rte_00210)*

[SWS_Rte_02753] ⌈ When a SW-C in an untrusted partition passes an `ArgumentDataPrototype` or `VariableDataPrototype`, as a reference type to a SW-C in a trusted partition (`DATA_REFERENCE` as an IN parameter), the RTE shall only check that the caller's partition owns the start address of the referenced memory. ⌋*(SRS_Rte_00210)*

Note to [SWS_Rte_02753]: The RTE only checks whether the start address referenced directly by the `DataPrototype`s belongs to the calling partition. Because the RTE is not aware of the semantic of the pointed reference, it cannot check if the referenced object is completely contained in the calling partition (e.g. the RTE does not know the size and does not know if the referenced object also contains references to other objects). The BSW is responsible to make sure that the referenced memory object does not cross memory section boundaries.

The OS API CheckTaskMemoryAccess can be used to fulfill [SWS_Rte_02752] and [SWS_Rte_02753].

4.3.5 PortInterface Element Mapping and Data Conversion

AUTOSAR supports the connection of an R-port to a P-port with an interface that is not compatible in the sense of the AUTOSAR compatibility rules. In addition, for sender-receiver communication it is possible to specify how data elements are represented given that the communication requires the usage of a dedicated communication bus. In these cases the generated RTE has to support the conversion and re-scaling of data.

4.3.5.1 PortInterface Element Mapping

Per default the shortNames of PortInterface elements are used to identify the matching element pairs of connected ports. In case of non fitting names — might be caused due to distributed development, off-the-shelf development, or re-use of software components — it is required to explicitly specify which PortInterface elements shall correlate. This is modelled with PortInterfaceMappings. A connection of two ports can be associated with a set of PortInterfaceMappings. If two ports are connected and a PortInterfaceMapping for the pair of interfaces of the two ports is associated with the connection, the interface elements are mapped and converted as specified in the PortInterfaceMapping. If no PortInterfaceMapping for the respective pair of interfaces is associated with the connection, the ordinary interface compatibility rules are applied.

The general approach is to perform the data conversion in the RTE of the ECU implementing the R-port. The reason for this design decision is that in case of 1:n sender-receiver communication it is inefficient to perform all the data conversions for the multiple receivers on the sender side and then send multiple sets of the same data just in different representations over the communication bus.

[SWS_Rte_03815] ⌈ The RTE shall support the mapping of sender-receiver interfaces, parameter interfaces and non-volatile data interface elements. ⌋*(SRS_Rte_00182)*

[SWS_Rte_03816] ⌈ If a P-port specified by a SenderReceiverInterface or NvDataInterface is connected to an R-port with an incompatible interface and a VariableAndParameterInterfaceMapping for both interfaces is associated with the connection, the RTE of the ECU implementing the R-port shall map and convert the data elements of the sender's interface to the data elements of the receiver's interface. ⌋*(SRS_Rte_00182)*

[SWS_Rte_07091] ⌈ The RTE shall support the *Mapping of elements of composite data types* in the context of a mapping of SenderReceiverInterface, NvDataInterface or ParameterInterface elements. ⌋*(SRS_Rte_00182, SRS_Rte_00234)*

[SWS_Rte_07092] ⌈ The RTE of the ECU implementing the R-port shall map and convert the composite data type elements of DataPrototypes of the sender's interface to the composite data type elements of DataPrototypes of the receiver's interface according the SubElementMapping
if a P-port specified by a SenderReceiverInterface, NvDataInterface or ParameterInterface is connected to an R-port with an incompatible interface and
a VariableAndParameterInterfaceMapping exists for both interfaces and is associated with the connection and
the SubElementMapping maps composite data type elements of the provided interface to composite data type elements of the required interface. ⌋*(SRS_Rte_00182, SRS_Rte_00234)*

[SWS_Rte_07099] ⌈ The RTE of the ECU implementing the R-port shall map and convert the composite data type elements of DataPrototype of the sender's interface to the primitive DataPrototype of the receiver's interface according the SubElementMapping
if a P-port specified by a SenderReceiverInterface, NvDataInterface or ParameterInterface is connected to a R-port with an incompatible interface and
a VariableAndParameterInterfaceMapping exists for both interfaces and is associated with the connection and the SubElementMapping exclusively maps one composite data type element of the provided interface ⌋*(SRS_Rte_00182, SRS_Rte_00234)*

According to [TPS_SWCT_01551], incomplete SubElementMappings are allowed for unqueued communication, when unmapped dataElements on the receiver side have an initValue.

Please note that the DataPrototypes of the provide port and DataPrototypes of the require port might use exclusively ApplicationDataTypes, exclusively ImplementationDataTypes or both kinds of AutosarDataTypes in a mixed manner.

[SWS_Rte_03817] ⌈ If a P-port specified by a SenderReceiverInterface or NvDataInterface is connected to an R-port with an incompatible interface and no VariableAndParameterInterfaceMapping for this pair of interfaces is associated with the connection, the RTE generator shall reject the input as an invalid configuration. ⌋*(SRS_Rte_00182, SRS_Rte_00018)*

[SWS_Rte_03818] ⌈ The RTE shall support the mapping of client-server interface elements. ⌋*(SRS_Rte_00182)*

[SWS_Rte_03819] ⌈ If a P-port specified by a ClientServerInterface is connected to an R-port with an incompatible interface and a ClientServerInterfaceMapping for both interfaces is associated with the connection, the RTE of the ECU implementing the R-port, i. e. the client, shall map the operation and map and convert the operation arguments of the client's interface to the operation arguments of the server's interface. ⌋*(SRS_Rte_00182)*

[SWS_Rte_07925] ⌈ If a ClientServerApplicationErrorMapping exists, the RTE shall translate the error codes of the server into the corresponding error codes described by the mapping. ⌋*(SRS_Rte_00182, SRS_Rte_00123)*

[SWS_Rte_07926] ⌈ If a ClientServerApplicationErrorMapping exists and a particular error of the server is not mapped, this error shall be translated to RTE_E_OK. ⌋*(SRS_Rte_00182, SRS_Rte_00123)*

[SWS_Rte_03820] ⌈ If a P-port specified by a ClientServerInterface is connected to an R-port with an incompatible interface and no ClientServerInterfaceMapping for this pair of interfaces is associated with the connection, the RTE generator shall reject the input as an invalid configuration. ⌋*(SRS_Rte_00182, SRS_Rte_00018)*

[SWS_Rte_03821] ⌈ The RTE shall support the mapping of ModeSwitchInterface elements. ⌋*(SRS_Rte_00182)*

[SWS_Rte_03822] ⌈ If a P-port specified by a ModeSwitchInterface is connected to an R-port with an incompatible interface and a ModeInterfaceMapping for both interfaces is associated with the connection, the RTE of the ECU implementing the R-port shall map and convert the mode elements of the sender's interface to the mode elements of the receiver's interface. ⌋*(SRS_Rte_00182)*

[SWS_Rte_03823] ⌈ If a P-port specified by a ModeSwitchInterface is connected to an R-port with an incompatible interface and no ModeInterfaceMapping for this pair of interfaces is associated with the connection, the RTE generator shall reject the input as an invalid configuration. ⌋*(SRS_Rte_00182, SRS_Rte_00018)*

[SWS_Rte_03824] ⌈ The RTE shall support the mapping of trigger interface elements. ⌋

[SWS_Rte_03825] ⌈ If a P-port specified by a TriggerInterface is connected to an R-port with an incompatible interface and a TriggerInterfaceMapping for both interfaces is associated with the connection, the RTE of the ECU implementing the R-port shall map the trigger of the sender's interface to the trigger of the receiver's interface. ⌋

[SWS_Rte_03826] ⌈ If a P-port specified by a TriggerInterface is connected to an R-port with an incompatible interface and no TriggerInterfaceMapping for this pair of interfaces is associated with the connection, the RTE generator shall reject the input as an invalid configuration. ⌋*(SRS_Rte_00018)*

In order to generate the RTE for the ECU implementing the R-ports, the RTE generator has to know the interfaces of the P-ports that are connected over the bus. This information is provided in the ECU extract via the networkRepresentationProps (see section 4.3.5.2) specified at the ISignal representing the data element.

4.3.5.2 Network Representation

For sender-receiver communication it is possible to specify how data elements are represented given that the communication requires the usage of a dedicated communication bus. For this purpose networkRepresentationProps can be specified at the ISignal, describing the representation of the data element on the communication bus via the attributes compuMethod and baseType.

[SWS_Rte_07842] ⌈ The RTE generator shall reject any input that violates [TPS_SYST_02001] as an invalid configuration. ⌋*(SRS_Rte_00018)*

[SWS_Rte_03827] ⌈ The RTE of the transmitting ECU shall perform the conversion of the data element that has to be sent over a communication bus to the representation specified by the baseType and compuMethod of the networkRepresentationProps of the respective ISignal if the dataTypePolicy of the ISignal is set to override or legacy. The converted data shall be passed to COM. ⌋*(SRS_Rte_00181)*

[SWS_Rte_06737] ⌈ If the dataTypePolicy of the respective ISignal is set to networkRepresentationFromComSpec and the networkRepresentation of the respective SenderComSpec is defined, the RTE of the transmitting ECU shall perform the conversion of the data element that has to be sent over a communication bus to the representation specified by the baseType and compuMethod of the networkRepresentation of the respective SenderComSpec. The converted data shall then be passed to COM. ⌋*(SRS_Rte_00181)*

[SWS_Rte_03828] ⌈ The RTE of the receiving ECU shall perform the conversion of the data element that is received over a communication bus from the representation specified by the baseType and compuMethod of the networkRepresentationProps of the respective ISignal to the data element's application data type if the dataTypePolicy of the ISignal is set to override or legacy. In this case [SWS_Rte_03816] shall not be applied. ⌋*(SRS_Rte_00181)*

[SWS_Rte_06738] ⌈ If the dataTypePolicy of the respective ISignal is set to networkRepresentationFromComSpec and the networkRepresentation of the respective ReceiverComSpec is defined, the RTE of the receiving ECU shall perform the conversion of the data element that is received over a communication bus from the representation specified by the baseType and compuMethod of the networkRepresentation of the respective ReceiverComSpec. In this case [SWS_Rte_03816] shall not be applied. ⌋*(SRS_Rte_00181)*

[SWS_Rte_07844] ⌈ If the dataTypePolicy of the respective ISignal is set to networkRepresentationFromComSpec but there is no networkRepresentation defined by the ReceiverComSpec (respectively SenderComSpec) then no conversion shall be performed by RTE. ⌋*(SRS_Rte_00181)*

As an alternative to networkRepresentationProps the representation of the VariableDataPrototypes and ArgumentDataPrototypes on the communication bus can be expressed by the used DataTypes in the PortInterfaces on the outerPorts of the CompositionSwComponentType describing the ecu extract. In this case the conversion between the network representation and the representation for the software components on the ecu are described by a PortInterfaceMapping which in turn is referenced by the DelegationSwConnector connecting the innerPort of the software component and the outerPort. These supports especially conversions of texttable data representation where a TextTableMapping is needed to describe the particular conversion rule.

[SWS_Rte_07828] ⌈ If a PortInterfaceMapping is specified at the DelegationSwConnector of a P-port, the RTE of the transmitting ECU shall perform the conversion of the VariableDataPrototypes or ArgumentDataPrototypes that has to be sent over a communication bus to the representation specified by the outerPort. The converted data shall be passed to COM. ⌋*(SRS_Rte_00181)*

[SWS_Rte_07829] ⌈ d If a PortInterfaceMapping is specified at the DelegationSwConnector of a R-port, the RTE of the receiving ECU shall perform the conversion of the VariableDataPrototypes or ArgumentDataPrototypes that is received over a communication bus from the representation specified by the outerPort to the representation specified by the innerPort. In this case [SWS_Rte_03816] shall not be applied. ⌋*(SRS_Rte_00181)*.

4.3.5.3 Data Conversion

[SWS_Rte_03829] ⌈ The RTE shall support the conversion of an identical or linear scaled data representation to another identical or linear scaled data representation. In this context, the term "linear scaled data representation" also includes floating-point data representations. ⌋*(SRS_Rte_00182)*

[SWS_Rte_08801] ⌈ The RTE shall support the conversion integer-to-float and float-to-integer. ⌋*(SRS_Rte_00182)*

Today the RTE Specification does not define any specific behavior supporting float to integer and integer to float conversions. This enables the RTE implementers to develop the most efficient, stable and robust solution.

[SWS_Rte_03830] ⌈ The RTE shall support the conversion of a texttable data representation (enumeration) to another texttable data representation. ⌋*(SRS_Rte_00182)*

[SWS_Rte_03855] ⌈ The RTE shall support the conversion of a mixed linear scaled and texttable data representation to another mixed linear scaled and texttable data representation. ⌋*(SRS_Rte_00182)*

[SWS_Rte_03856] ⌈ The RTE shall support the conversion between a texttable data representation (enumeration) and a mixed linear scaled and texttable data representation. In this case only the enumeration part of the data representation shall be converted, the linear scaled part shall be handled as out of range data. ⌋*(SRS_Rte_00182)*

[SWS_Rte_03857] ⌈ The RTE shall support the conversion between an identical or linear scaled data representation and a mixed linear scaled and texttable data representation. A scale with a compuConst shall be handled as out of range data if the mapping to a value is not defined by a TextTableMapping. ⌋(SRS_Rte_00182)

[SWS_Rte_03860] ⌈ The RTE shall support the conversion of composite data representations. In this case, the respective requirements [SWS_Rte_03829], [SWS_Rte_03830], [SWS_Rte_03855], [SWS_Rte_03856], [SWS_Rte_03857], [SWS_Rte_03831], [SWS_Rte_03832], and [SWS_Rte_03833] are applicable to the individual composite elements. ⌋(SRS_Rte_00182)

[SWS_Rte_03831] ⌈ The RTE generator shall reject any input that requires a conversion which is not supported according to [SWS_Rte_03829], [SWS_Rte_03830], [SWS_Rte_03855], [SWS_Rte_03856], or [SWS_Rte_03860] as an invalid configuration. ⌋(SRS_Rte_00182, SRS_Rte_00018)

[SWS_Rte_07928] ⌈ The data conversion shall be supported for data types that refer to CompuMethods of category LINEAR, IDENTICAL, SCALE_LINEAR_AND_TEXTTABLE, and TEXTTABLE. ⌋(SRS_Rte_00182)

[SWS_Rte_03832] ⌈ For the conversion between two data representations with linear scaling described either by an ApplicationDataType or a combination of BaseType and CompuMethod (used for the specification of the network representation at the ComSpec respectively the SystemSignal) the RTE generator shall derive the data conversion code automatically from the referred CompuMethods of the two representations. In this context the scaling of a data representation is linear if the referred CompuMethod is of category IDENTICAL, LINEAR, or SCALE_LINEAR_AND_TEXTTABLE. In case of a CompuMethod of category SCALE_LINEAR_AND_TEXTTABLE this requirement applies to the linear scaled part only. The conversion shall only be performed if it does not violate [constr_1042] in Software Component Template [2]. ⌋(SRS_Rte_00182)

For a linear conversion the linear conversion factor can be calculated out of the factorSiToUnit and offsetSiToUnit attributes of the referred Units and the CompuRationalCoeffs of a compuInternalToPhys of the referred CompuMethods.

Example 4.9

A software component SwcA on an ECU EcuA sends a data element u of an uint16 type t_VoltageAtSender via its port SenderPort. The referenced CompuMethod is cm_VoltageAtSender, describing a fixpoint representation with offset 0 and LSB $\frac{1}{4} = 2^{-2}$. The port SenderPort is connected to the port ReceiverPort of a software component SwcB that is deployed on a different ECU EcuB. The sent data element u is mapped to a data element u of an uint16 type t_VoltageAtReceiver on the receiving side that references a CompuMethod named cm_VoltageAtReceiver. cm_VoltageAtReceiver describes a fixpoint representation with offset $\frac{16}{8} = 2$ and LSB $\frac{1}{8} = 2^{-3}$. For transportation over the bus a networkRepresentation that references an uint8 type t_VoltageOnNetwork is specified, using a fixpoint representation described by the CompuMethod cm_VoltageOnNetwork with offset $\frac{1}{2} = 0.5$ and LSB $\frac{1}{2} = 2^{-1}$.

Definition of the CompuMethods in *XML*:

```xml
<COMPU-METHOD>
  <SHORT-NAME>cm_VoltageAtSender</SHORT-NAME>
  <CATEGORY>LINEAR</CATEGORY>
  <COMPU-INTERNAL-TO-PHYS>
    <COMPU-SCALES>
      <COMPU-SCALE>
        <COMPU-RATIONAL-COEFFS>
          <COMPU-NUMERATOR><V>0</V><V>1</V></COMPU-NUMERATOR>
          <COMPU-DENOMINATOR><V>4</V></COMPU-DENOMINATOR>
        </COMPU-RATIONAL-COEFFS>
      </COMPU-SCALE>
    </COMPU-SCALES>
  </COMPU-INTERNAL-TO-PHYS>
</COMPU-METHOD>

<COMPU-METHOD>
  <SHORT-NAME>cm_VoltageAtReceiver</SHORT-NAME>
  <CATEGORY>LINEAR</CATEGORY>
  <COMPU-INTERNAL-TO-PHYS>
    <COMPU-SCALES>
      <COMPU-SCALE>
        <COMPU-RATIONAL-COEFFS>
          <COMPU-NUMERATOR><V>16</V><V>1</V></COMPU-NUMERATOR>
          <COMPU-DENOMINATOR><V>8</V></COMPU-DENOMINATOR>
        </COMPU-RATIONAL-COEFFS>
      </COMPU-SCALE>
    </COMPU-SCALES>
  </COMPU-INTERNAL-TO-PHYS>
</COMPU-METHOD>

<COMPU-METHOD>
  <SHORT-NAME>cm_VoltageOnNetwork</SHORT-NAME>
  <CATEGORY>LINEAR</CATEGORY>
  <COMPU-INTERNAL-TO-PHYS>
    <COMPU-SCALES>
      <COMPU-SCALE>
        <COMPU-RATIONAL-COEFFS>
          <COMPU-NUMERATOR><V>1</V><V>1</V></COMPU-NUMERATOR>
          <COMPU-DENOMINATOR><V>2</V></COMPU-DENOMINATOR>
        </COMPU-RATIONAL-COEFFS>
      </COMPU-SCALE>
    </COMPU-SCALES>
  </COMPU-INTERNAL-TO-PHYS>
</COMPU-METHOD>
```

Implementation of Rte_Send on the sending ECU EcuA:

```c
Std_ReturnType
Rte_Send_SwcA_SenderPort_u(t_voltageAtSender u)
{
  ...
  /*
  u_NetworkRepresentation
   = ((u * LSB_sender + off_sender) - off_network) / LSB_network
   = ((u / 4          + 0         ) - 0.5        ) * 2
```

```
    =  (u / 2                      ) - 1
  */
  u_NetworkRepresentation = (uint8) ((u >> 1) - 1);
  ...
}
```

Implementation of Rte_Receive on the receiving ECU EcuB:

```
Std_ReturnType
Rte_Receive_SwcB_ReceiverPort_u(t_voltageAtReceiver * u)
{
  ...
  /*
  *u
  *u = ((u_NetworkRepresentation * LSB_network + off_network)
        - off_receiver) / LSB_receiver
     = ((u_NetworkRepresentation / 2           + 0.5         )
        - 2             ) * 8
     =  (u_NetworkRepresentation * 4           + 4           )
        - 16
     =   u_NetworkRepresentation * 4           - 12
  */
  *u = (uint16) ((u_NetworkRepresentation << 2) - 12);
  ...
}
```

The intention of this specification is not to describe any mechanism that supports the generation of identical conversion code for each implementation of an RTE generator. Even if the generated C code for the conversion would be the same, the numerical result of the conversion still depends on the microcontroller target and the compiler.

Strategies how to handle the conversion of values that are out of range of the target representation are described in section 4.3.5.4.

[SWS_Rte_03833] ⌈ For the conversion between two texttable data representations (enumerations) described either by an ApplicationDataType or an ImplementationDataType (used for the specification of the network representation) the RTE generator shall generate the data conversion code according to the TextTableMapping. This requirement also applies to the texttable part of a mixed linear scaled and texttable data representation. ⌋*(SRS_Rte_00182)*

4.3.5.4 Range Checks during Runtime

A software component might try to send a value that is outside the range that is specified at a dataElement or ISignal. In case of different ranges the result of a data conversion might also be a value that is out of range of the target representation. For a safe handling of these use cases the RTE provides range checks during runtime. For an overview see figure 4.44.

[SWS_Rte_08024] ⌈ Range checks during runtime shall occur after data invalidation, i.e. first the handleNeverReceived check, then the invalidation check and lastly the range check shall be effected. ⌋*(SRS_Rte_00180)*

[SWS_Rte_03861] ⌈ The range check is intended to be performed according to the following rule: If a upper/lower limit is specified at the DataConstr, this value shall be taken for the range check. If it is not specified at the DataConstr, the highest/lowest representable value of the datatype shall be used. ⌋*(SRS_Rte_00180)*

Whether a range check is required is specified in case of intra ECU communication at the handleOutOfRange attribute of the respective SenderComSpec or ReceiverComSpec and in case of inter ECU communication at the handleOutOfRange attribute of ISignalProps of the sending or receiving ISignal.

Range checks at sender's side

Range checks during runtime for intra ECU communication at the sender's side are described in the following requirements:

[SWS_Rte_08026] ⌈ The RTE shall implement a range check of sent data in the sending path of a particular component if the handleOutOfRange is defined at the SenderComSpec and has any value other than none. In this case all receivers receive the value after the range check was applied. ⌋*(SRS_Rte_00180)*

[SWS_Rte_08039] ⌈ The RTE shall use the preceding limits ([SWS_Rte_07196]) from the DataPrototype in the PPortPrototype or PRPortPrototype for the range check of sent data in the sending path of a particular component if the handleOutOfRange is defined at the SenderComSpec. ⌋*(SRS_Rte_00180)*

[SWS_Rte_03839] ⌈ If for a dataElement to be sent a SenderComSpec with handleOutOfRange=ignore is provided, a range check shall be implemented in the sending component. If the value is out of bounds, the sending of the dataElement shall not be propagated. This means for a non-queued communication that the last valid value will be propagated and for a queued communication that no value will be enqueued.

In case of a composite datatype the sending of the whole dataElement shall not be propagated, if any of the composite elements is out of bounds. ⌋*(SRS_Rte_00180)*

[SWS_Rte_03840] ⌈ If for a dataElement to be sent a SenderComSpec with handleOutOfRange=saturate is provided, a range check shall be implemented in the sending component. If the value is out of bounds, the value actually sent shall be set to the lower respectively the upper limit.

In case of a composite datatype each composite element whose actual value is out of bounds shall be saturated. ⌋*(SRS_Rte_00180)*

[SWS_Rte_03841] ⌈ If for a dataElement to be sent a NonqueuedSenderComSpec with handleOutOfRange=default is provided, a range check shall be implemented in the sending component. If the value is out of bounds and the initValue is not equal to the invalidValue, the value actually sent shall be set to the initValue.

In case of a composite datatype each composite element whose actual value is out of bounds shall be set to the initValue. ⌋*(SRS_Rte_00180)*

[SWS_Rte_03842] ⌈ If for a dataElement to be sent a NonqueuedSenderComSpec with handleOutOfRange=invalid is provided, a range check shall be implemented in the sending component. If the value is out of bounds, the value actually sent shall be set to the invalidValue.

In case of a composite datatype each composite element whose actual value is out of bounds shall be set to the invalidValue. ⌋*(SRS_Rte_00180)*

[SWS_Rte_03843] ⌈ If for a dataElement to be sent a QueuedSenderComSpec with handleOutOfRange set to default or invalid is provided, the RTE generator shall reject the input as an invalid configuration, since for a QueuedSenderComSpec the attribute initValue is not defined (see SW-C Template [2]) and data invalidation is not supported (see [SWS_Rte_06727]). ⌋*(SRS_Rte_00180)*

Range checks during runtime for inter ECU communication at the sender's side are described in the following requirements:

[SWS_Rte_08027] ⌈ The RTE shall implement a range check of sent data in the sending path of a particular signal if the handleOutOfRange is defined at the ISignalProps and has any value other than none. In this case only receivers of the specific ISignal receive the value after the range check was applied. ⌋*(SRS_Rte_00180)*

[SWS_Rte_08040] ⌈ The RTE shall use the limits from the ISignal for the range check of sent data in the sending path of a particular signal if the handleOutOfRange is defined at the ISignalProps. ⌋*(SRS_Rte_00180)*

[SWS_Rte_08030] ⌈ If for an ISignal to be sent an ISignalProps with handleOutOfRange=ignore is provided, a range check shall be implemented in the sending signal. If the value is out of bounds, the sending of the ISignal shall not be propagated. In this case the RTE shall behave as if no sending occurred. ⌋*(SRS_Rte_00180)*

[SWS_Rte_08031] ⌈ If for an ISignal to be sent an ISignalProps with handleOutOfRange=saturate is provided, a range check shall be implemented in the sending signal. If the value is out of bounds, the value actually sent shall be set to the lower respectively the upper limit. ⌋*(SRS_Rte_00180)*

[SWS_Rte_08032] ⌈ If for an ISignal to be sent an ISignalProps with handleOutOfRange=default is provided, a range check shall be implemented in the sending signal. If the value is out of bounds and the initValue is not equal to the invalidValue, the value actually sent shall be set to the initValue. ⌋*(SRS_Rte_00180)*

[SWS_Rte_08033] ⌈ If for an ISignal to be sent an ISignalProps with handleOutOfRange=invalid is provided, a range check shall be implemented in the sending signal. If the value is out of bounds, the value actually sent shall be set to the invalidValue. ⌋(SRS_Rte_00180)

Range checks at receiver's side

Range checks during runtime for intra ECU communication at the receiver's side are described in the following requirements:

[SWS_Rte_08028] ⌈ The RTE shall implement a range check in the receiving path of a particular component if the handleOutOfRange is defined at the ReceiverComSpec and has any value other than none. In this case the range check applies only for data received by the particular component. ⌋(SRS_Rte_00180)

[SWS_Rte_08041] ⌈ The RTE shall use the preceding limits ([SWS_Rte_07196]) from the DataPrototype in the rPort for the range check of received data in the receiving path of a particular component if the handleOutOfRange is defined at the ReceiverComSpec. ⌋(SRS_Rte_00180)

[SWS_Rte_03845] ⌈ If for a dataElement to be received a ReceiverComSpec with handleOutOfRange=ignore is provided, a range check shall be implemented in the receiving component. If the value is out of bounds, the reception of the dataElement shall not be propagated. This means for a non-queued communication that the last valid value will be propagated and for a queued communication that no value will be enqueued.

If the value of the received dataElement is out of bounds and a NonqueuedReceiverComSpec with handleOutOfRangeStatus=indicate is provided, the return value of the RTE shall be RTE_E_OUT_OF_RANGE.

In case of a composite datatype the reception of the whole dataElement shall not be propagated, if any of the composite elements is out of bounds. If the handleOutOfRangeStatus attribute is set to indicate, the return value of the RTE shall be RTE_E_OUT_OF_RANGE. ⌋(SRS_Rte_00180)

[SWS_Rte_03846] ⌈ If for a dataElement to be received a ReceiverComSpec with handleOutOfRange=saturate is provided, a range check shall be implemented in the receiving component. If the value is out of bounds, the value actually received shall be set to the lower respectively the upper limit.

If the value of the received dataElement is out of bounds and a NonqueuedReceiverComSpec with handleOutOfRangeStatus=indicate is provided, the return value of the RTE shall be RTE_E_OUT_OF_RANGE.

In case of a composite datatype each composite element whose actual value is out of bounds shall be saturated. If the handleOutOfRangeStatus attribute is set to indicate, the return value of the RTE shall be RTE_E_OUT_OF_RANGE, if any of the composite elements is out of bounds. ⌋(SRS_Rte_00180)

[SWS_Rte_03847] ⌈ If for a dataElement to be received a NonqueuedReceiverComSpec with handleOutOfRange=default is provided, a range check shall be implemented in the receiving component. If the value is out of bounds and the initValue is not equal to the invalidValue, the value actually received shall be set to the initValue.

If the value of the received dataElement is out of bounds and a NonqueuedReceiverComSpec with handleOutOfRangeStatus=indicate is provided, the return value of the RTE shall be RTE_E_OUT_OF_RANGE.

In case of a composite datatype each composite element whose actual value is out of bounds shall be set to the initValue. If the handleOutOfRangeStatus attribute is set to indicate, the return value of the RTE shall be RTE_E_OUT_OF_RANGE, if any of the composite elements is out of bounds. ⌋*(SRS_Rte_00180)*

[SWS_Rte_03848] ⌈ If for a dataElement to be received a NonqueuedReceiverComSpec with handleOutOfRange=invalid is provided, a range check shall be implemented in the receiving component. If the value is out of bounds, the value actually received shall be set to the invalidValue.

If the value of the received dataElement is out of bounds and a ReceiverComSpec with handleOutOfRangeStatus=indicate is provided, the return value of the RTE shall be RTE_E_INVALID.

In case of a composite datatype each composite element whose actual value is out of bounds shall be set to the invalidValue. If the handleOutOfRangeStatus attribute is set to indicate, the return value of the RTE shall be RTE_E_INVALID, if any of the composite elements is out of bounds. ⌋*(SRS_Rte_00180)*

[SWS_Rte_08016] ⌈ If for a dataElement to be received a ReceiverComSpec with handleOutOfRange=externalReplacement is provided, a range check shall be implemented in the receiving component. If the value is out of bounds, the value actually received shall be replaced by the value sourced from the ReceiverComSpec.handleOutOfRange (e.g. constant, NVRAM parameter).

If the value of the received dataElement is out of bounds and a NonqueuedReceiverComSpec with handleOutOfRangeStatus=indicate is provided, the return value of the RTE shall be RTE_E_OUT_OF_RANGE.

In case of a composite datatype the value actually received shall be completely replaced by the external value, if any of the composite elements is out of bounds. If the handleOutOfRangeStatus attribute is set to indicate, the return value of the RTE shall be RTE_E_OUT_OF_RANGE. ⌋*(SRS_Rte_00180)*

[SWS_Rte_03849] ⌈ If for a dataElement to be received a QueuedReceiverComSpec with handleOutOfRange set to default or invalid is provided, the RTE generator shall reject the input as an invalid configuration, since for a QueuedReceiverComSpec the attribute initValue is not defined (see SW-C Template [2]) and data invalidation is not supported (see [SWS_Rte_06727]). ⌋*(SRS_Rte_00180)*

[SWS_Rte_08025] ⌈ If for a dataElement to be received a QueuedReceiverComSpec is provided and the handleOutOfRangeStatus attribute is set to indicate, the RTE generator shall reject the input as an invalid configuration. ⌋*(SRS_Rte_00180)*

Range checks during runtime for inter ECU communication at the receiver's side are described in the following requirements:

[SWS_Rte_08029] ⌈ The RTE shall implement a range check in the receiving path of a particular signal if the handleOutOfRange is defined at the ISignalProps and has any value other than none. In this case all receivers of the specific ISignal on that ECU receive the value after the range check was applied. ⌋*(SRS_Rte_00180)*

[SWS_Rte_08042] ⌈ The RTE shall use the limits from the ISignal for the range check of received data in the receiving path of a particular signal if the handleOutOfRange is defined at the ISignalProps. ⌋*(SRS_Rte_00180)*

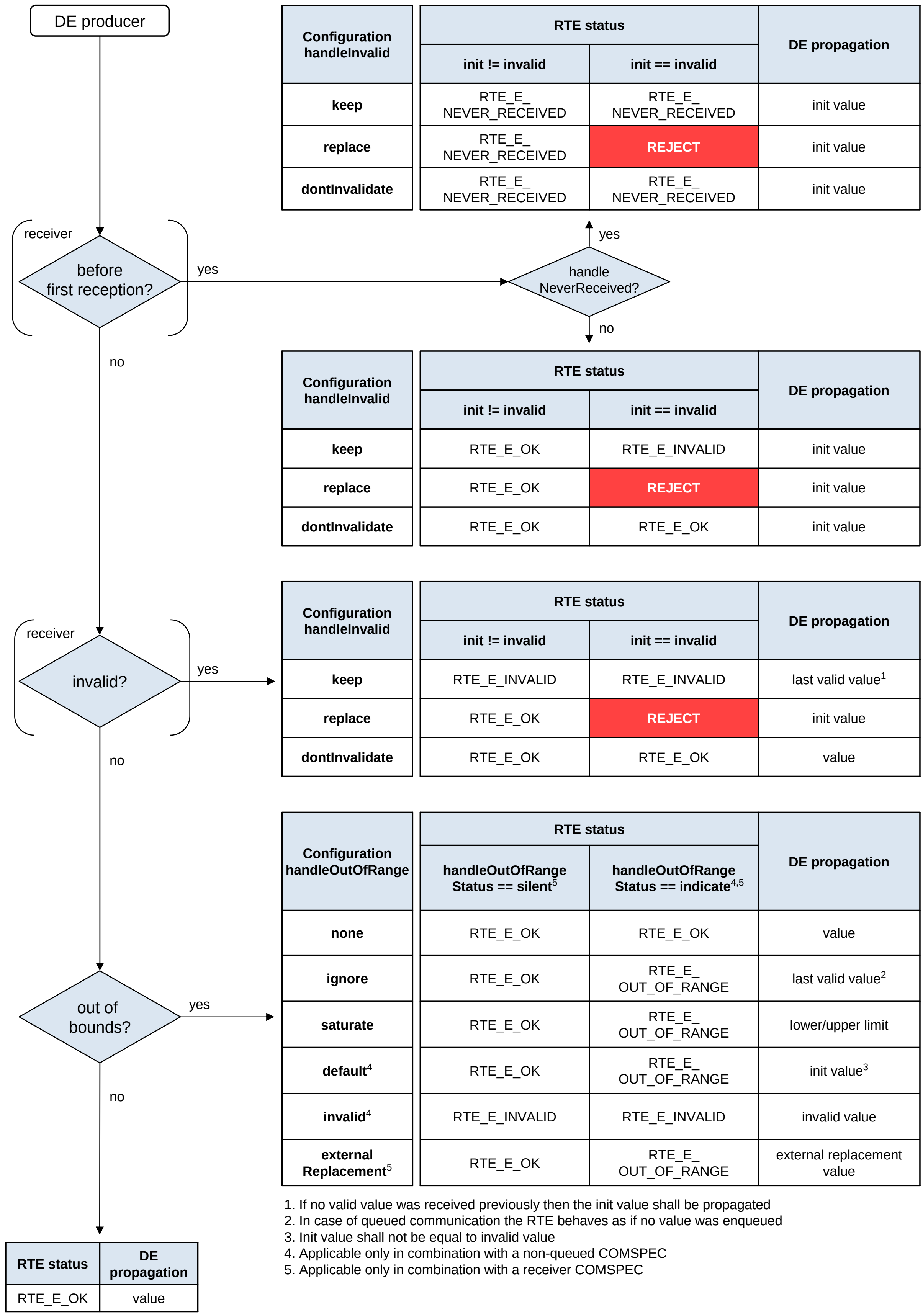
[SWS_Rte_08034] ⌈ If for an ISignal to be received an ISignalProps with handleOutOfRange=ignore is provided, a range check shall be implemented in the receiving signal. If the value is out of bounds, the reception of the ISignal shall not be propagated. In this case the RTE shall behave as if no reception occurred. ⌋*(SRS_Rte_00180)*

[SWS_Rte_08035] ⌈ If for an ISignal to be received an ISignalProps with handleOutOfRange=saturate is provided, a range check shall be implemented in the receiving signal. If the value is out of bounds, the value actually received shall be set to the lower respectively the upper limit. ⌋*(SRS_Rte_00180)*

[SWS_Rte_08036] ⌈ If for an ISignal to be received an ISignalProps with handleOutOfRange=default is provided, a range check shall be implemented in the receiving signal. If the value is out of bounds and the initValue is not equal to the invalidValue, the value actually received shall be set to the initValue. ⌋*(SRS_Rte_00180)*

[SWS_Rte_08037] ⌈ If for an ISignal to be received an ISignalProps with handleOutOfRange=invalid is provided, a range check shall be implemented in the receiving signal. If the value is out of bounds, the value actually received shall be set to the invalidValue. ⌋*(SRS_Rte_00180)*

[SWS_Rte_08038] ⌈ If for an ISignal to be received an ISignalProps with handleOutOfRange=externalReplacement is provided, a range check shall be implemented in the receiving signal. If the value is out of bounds, the value actually received shall be replaced by the value sourced from the ReceiverComSpec.externalReplacement (e.g. constant, NVRAM parameter). ⌋*(SRS_Rte_00180)*



Configuration handleInvalid	RTE status		DE propagation
	init != invalid	init == invalid	
keep	RTE_E_NEVER_RECEIVED	RTE_E_NEVER_RECEIVED	init value
replace	RTE_E_NEVER_RECEIVED	REJECT	init value
dontInvalidate	RTE_E_NEVER_RECEIVED	RTE_E_NEVER_RECEIVED	init value

Configuration handleInvalid	RTE status		DE propagation
	init != invalid	init == invalid	
keep	RTE_E_OK	RTE_E_INVALID	init value
replace	RTE_E_OK	REJECT	init value
dontInvalidate	RTE_E_OK	RTE_E_OK	init value

Configuration handleInvalid	RTE status		DE propagation
	init != invalid	init == invalid	
keep	RTE_E_INVALID	RTE_E_INVALID	last valid value[1]
replace	RTE_E_OK	REJECT	init value
dontInvalidate	RTE_E_OK	RTE_E_OK	value

Configuration handleOutOfRange	RTE status		DE propagation
	handleOutOfRange Status == silent[5]	handleOutOfRange Status == indicate[4,5]	
none	RTE_E_OK	RTE_E_OK	value
ignore	RTE_E_OK	RTE_E_OUT_OF_RANGE	last valid value[2]
saturate	RTE_E_OK	RTE_E_OUT_OF_RANGE	lower/upper limit
default[4]	RTE_E_OK	RTE_E_OUT_OF_RANGE	init value[3]
invalid[4]	RTE_E_INVALID	RTE_E_INVALID	invalid value
external Replacement[5]	RTE_E_OK	RTE_E_OUT_OF_RANGE	external replacement value

RTE status	DE propagation
RTE_E_OK	value

1. If no valid value was received previously then the init value shall be propagated
2. In case of queued communication the RTE behaves as if no value was enqueued
3. Init value shall not be equal to invalid value
4. Applicable only in combination with a non-queued COMSPEC
5. Applicable only in combination with a receiver COMSPEC

Figure 4.44: Overview for data invalidation and range checks

4.4 Modes

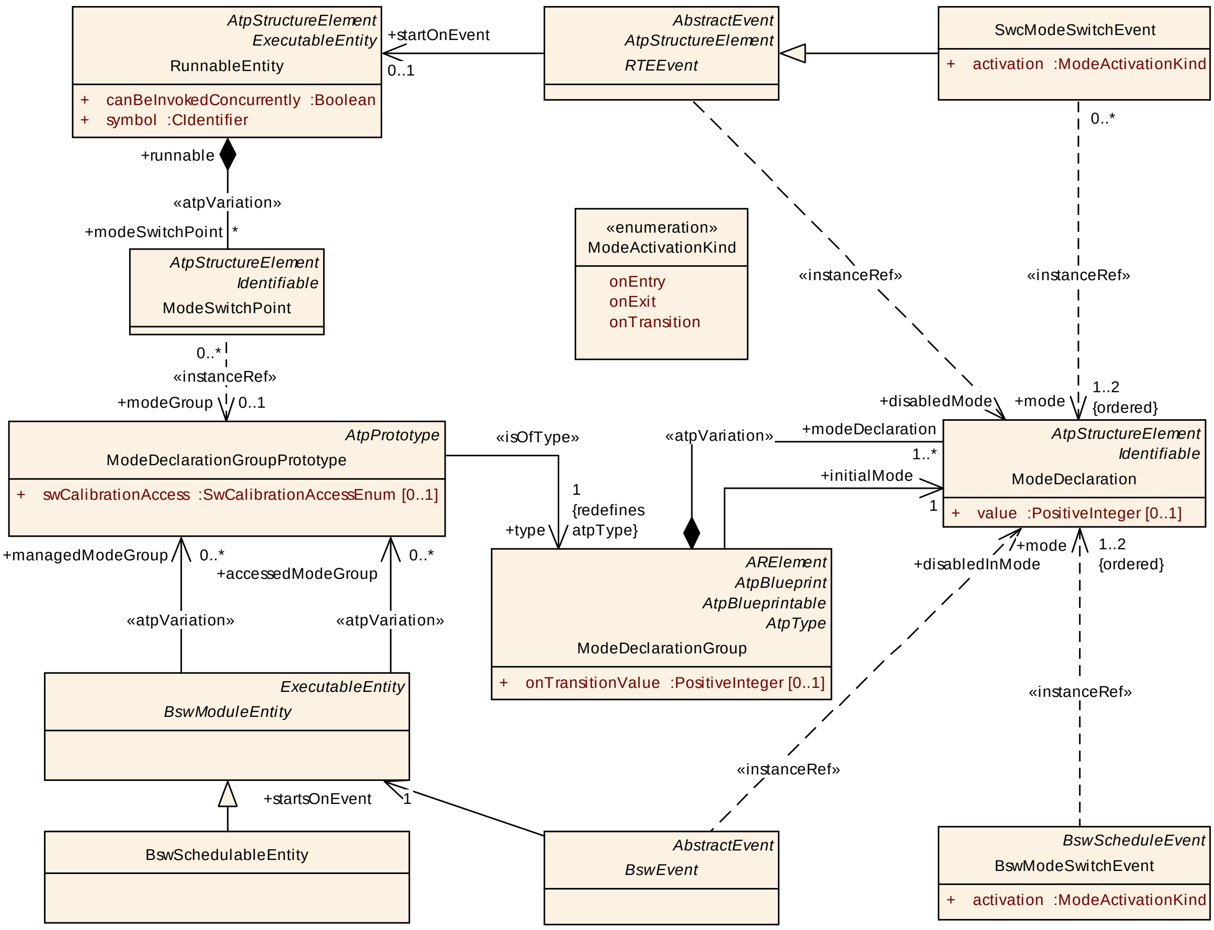


Figure 4.45: Summary of the use of ModeDeclarations by an AUTOSAR software-components and Basic Software Modules as defined in the *Software Component Template Specification* [2] and *Specification of BSW Module Description Template* [9].

The purpose of modes is to start `RunnableEntity`s and *Basic Software Schedulable Entities* on the transition between modes and to disable (/enable) specified triggers of `RunnableEntity`s and *Basic Software Schedulable Entities* in certain modes. Here, we use the specification of modes from the *Software Component Template Specification* [2]. Further on the document *Specification of BSW Module Description Template* [9] describes how modes are described for *Basic Software Modules*.

The first subsection 4.4.1 describes how modes can be used by an AUTOSAR software-component or *Basic Software Module* `mode user`. The role of the `mode manager` who initiates mode switches is described in section 4.4.2. How `ModeDeclaration`s are connected to a state machine is described in subsection 4.4.3. The behaviour of the RTE and *Basic Software Scheduler* regarding mode switches is detailed in subsection 4.4.4.

One usecase of modes is described in section 4.6.2 for the initialization and finalization of AUTOSAR software-components. Modes can be used for handling of communication states as well as for specific application purposes. The specific definition of modes and their use is not in the scope of this document.

The status of the modes will be notified to the AUTOSAR software-component mode user by mode communication - mode switch notifications - as described in the subsection 4.4.7. The port for receiving (or sending) a mode switch notification is called mode switch port.

A *Basic Software Module* mode users and the *Basic Software Module* mode manager are not necessarily using ports. *Basic Software Modules* without *AUTOSAR Interfaces* are connected via the configuration of the *Basic Software Scheduler*.

4.4.1 Mode User

To use modes, an AUTOSAR software-component (mode user) has to reference a ModeDeclarationGroup by a ModeDeclarationGroupPrototype of a require mode switch port, see section 4.4.7. The ModeDeclarationGroup contains the required modes. Alternatively the mode manager can also contain a ModeAccessPoint for a provided mode switch port and can combine the roles of mode user and mode manager for the same ModeDeclarationGroupPrototype.

An *Basic Software Module* (mode user) has to define a requiredModeGroup ModeDeclarationGroupPrototype.The ModeDeclarationGroup referred by these ModeDeclarationGroupPrototype contains the required modes. Similar to a software-component mode user, the Basic Software Module mode manager can also contain a accessedModeGroup for a providedModeGroup ModeDeclarationGroupPrototype. By this it combines the roles of mode user and mode manager for the same ModeDeclarationGroupPrototype.

The ModeDeclarations can be used in two ways by the mode user (see also figure 4.45):

1. Modes can be used to trigger runnables: The SwcInternalBehavior of the AUTOSAR SW-C or the BswInternalBehavior of the BSW module can define a SwcModeSwitchEvent respectively a BswModeSwitchEvent referencing the required ModeDeclaration. This SwcModeSwitchEvent or BswModeSwitchEvent can then be used as trigger for a RunnableEntity / *Basic Software Schedulable Entity*. Both SwcModeSwitchEvent and BswModeSwitchEvent carry an attribute ModeActivationKind which can be 'exit', 'entry', or 'transition'.

   A RunnableEntity or *Basic Software Schedulable Entity* that is triggered by a SwcModeSwitchEvent or a BswModeSwitchEvent with ModeActivationKind 'exit' is triggered on exiting the mode. For simplicity it will be called OnExit ExecutableEntity. Correspondingly, an OnTransition ExecutableEntity is triggered by a SwcModeSwitchEvent or a BswModeSwitchEvent with ModeActivationKind 'transition' and will be executed during the transition between two modes, and an OnEntry ExecutableEntity is triggered by a SwcModeSwitchEvent or a BswModeSwitchEvent with ModeActivationKind 'entry' and will be executed when the mode is entered.

Since a RunnableEntity as well as a *Basic Software Schedulable Entity* can be triggered by multiple RTEEvents respectively BswEvents, both can be an *OnExit-*, *OnTransition* and *OnEntry ExecutableEntity* at the same time.

RTE does not support a WaitPoint for a SwcModeSwitchEvent (see [SWS_Rte_01358]).

2. An RTEEvent or BswEvent that starts a ExecutableEntity can contain a ModeDisablingDependency.

[SWS_Rte_02503] ⌈ If a RunnableEntity *r* is referenced with startOnEvent by an RTEEvent *e* that has a ModeDisablingDependency on a mode *m*, then RTE shall not activate runnable *r* on any occurrence of *e* while the mode *m* is active. ⌋*(SRS_Rte_00143, SRS_Rte_00052)*

[SWS_Rte_07530] ⌈ If a *Basic Software Schedulable Entity* *r* is referenced with startsOnEvent by an BswEvent *e* that has a ModeDisablingDependency on a mode *m*, then *Basic Software Scheduler* shall not activate Basic Software Schedulable Entitys *r* on any occurrence of *e* while the mode *m* is active. ⌋*(SRS_Rte_00213)*

Note: As a consequence of [SWS_Rte_02503] and [SWS_Rte_07530] in combination with [SWS_Rte_02661], RTE or *Basic Software Scheduler* will not start runnable or BswSchedulableEntity *r* on any occurrence of *e* while the mode *m* is active.

The mode disabling is active during the transition to a mode, during the mode itself and during the transition for exiting the mode. For a precise definition see section 4.4.4.

The existence of a ModeDisablingDependency prevents the RTE to start the mode disabling dependent ExecutableEntity by the disabled RTEEvent / BswEvent during the mode, referenced by the ModeDisablingDependency, and during the transitions from and to that mode. ModeDisablingDependencys override any activation of a RunnableEntity and *Basic Software Schedulable Entity* by the disabled RTEEvents / BswEvents. This is also true for the SwcModeSwitchEvent and BswModeSwitchEvent.

A RunnableEntity as well as a *Basic Software Schedulable Entity* can not be 'enabled' explicitly. RunnableEntitys are *Basic Software Schedulable Entities* are only 'enabled' by the absence of any active ModeDisablingDependencys.

Note that ModeDisablingDependencys do not prevent the wake up from a WaitPoint by the 'disabled' RTEEvent. This allows the wake-uped RunnableEntity to run until completion if a transition occurred during the RunnableEntitys execution.

[SWS_Rte_02504] ⌈ The existence of a ModeDisablingDependency shall not instruct the RTE to kill a running runnable at a mode switch. ⌋*(SRS_Rte_00143)*

[SWS_Rte_07531] ⌈ The existence of a ModeDisablingDependency shall not instruct the *Basic Software Scheduler* to kill a running *Basic Software Schedulable Entity* at a mode switch. ⌋*(SRS_Rte_00213)*

The RTE and the *Basic Software Scheduler* can be configured to switch schedule tables to implement mode disabling dependencies for cyclic triggers of RunnableEntitys or *Basic Software Schedulable Entities*. Sets of mutual exclusive modes can be mapped to different schedule tables. The RTE shall implement the switch between schedule tables according to the mapping of modes to schedule tables in RteModeScheduleTableRef, see [SWS_Rte_05146].

The mode user can specify in the ModeSwitchReceiverComSpec (software components) or BswModeReceiverPolicy (BSW modules) that it is able to deal with asynchronous mode switch behavior (supportsAsynchronousModeSwitch == TRUE). If all mode users connected to the same ModeDeclarationGroupPrototype of the mode manager support the asynchronous mode switch behavior, the related mode machine instance can be implemented with the asynchronous mode switching procedure. Otherwise, the synchronous mode switching procedure has to be applied (see [SWS_Rte_07150]).

4.4.2 Mode Manager

Entering and leaving modes is initiated by a mode manager. A mode manager might be a basic software module, for example the Basic Software Mode Manager (BswM), the communication manager (ComM), or the ECU state manager (EcuM). The mode manager may also be an AUTOSAR SW-C. In this case, it is called an application mode manager.

The mode manager contains the master state machine to represent the modes.

To provide modes, an AUTOSAR software-component (mode manager) has to reference a ModeDeclarationGroup by a ModeDeclarationGroupPrototype of a provide mode switch port, see section 4.4.7. The ModeDeclarationGroup contains the provided modes.

An *Basic Software Module* (mode manager) has to define a providedModeGroup ModeDeclarationGroupPrototype. The ModeDeclarationGroup referred by these ModeDeclarationGroupPrototype contains the provided modes.

The RTE / *Basic Software Scheduler* will take the actions necessary to switch between the modes. This includes the termination and execution of several *ExecutableEntities* from all mode users that are connected to the same *ModeDeclarationGroupPrototype* of the mode manager. To do so, the RTE / *Basic Software Scheduler* needs a state machine to keep track of the currently active modes and transitions initiated by the mode manager. The RTE's / *Basic Software Scheduler*'s mode machine is called mode machine instance. There is exactly one mode machine instance for each *ModeDeclarationGroupPrototype* of a mode manager's provide mode switch port respectively providedModeGroup ModeDeclarationGroupPrototype.

It is the responsibility of the mode manager to advance the RTE's / *Basic Software Scheduler*'s mode machine instance by sending mode switch notifications to the mode users. The mode switch notifications are implemented by a non blocking API (see 5.6.6 / 6.5.7). So, the mode switch notifications alone provide only a loose coupling between the state machine of the mode manager and the mode machine instance of the RTE / *Basic Software Scheduler*. To prevent, that the mode machine instance lags behind and the states of the mode manager and the RTE / *Basic Software Scheduler* get out of phase, the mode manager can use acknowledgment feedback for the mode switch notification. RTE / *Basic Software Scheduler* can be configured to send an acknowledgment of the mode switch notification to the mode manager when the requested transition is completed.

At the mode manager, the acknowledgment results in an ModeSwitchedAckEvent. As with DataSendCompletedEvents, this event can be picked up with the polling or blocking Rte_SwitchAck API. And the event can be used to trigger a mode switch acknowledge ExecutableEntity to pick up the status. Note: The *Basic Software Scheduler* do not support WaitPoints. Therefore the SchM_SwitchAck never blocks.

Some possible usage patterns for the acknowledgement are:

- The most straight forward method is to use a sequence of Rte_Switch and a blocking Rte_SwitchAck to send the mode switch notification and wait for the completion. This requires the use of an extended task.
- Another possibility is to have a cyclic RunnableEntity / *Basic Software Schedulable Entity* (maybe the same that switches the modes via Rte_Switch / SchM_Switch) to poll for the acknowledgement using Rte_SwitchAck / SchM_SwitchAck.
- The acknowledgement can also be polled from a RunnableEntity or *Basic Software Schedulable Entity* that is started by the *ModeSwitchedAckEvent*.

The mode manager can also use the Rte_Mode / SchM_Mode API to read the currently active mode from the RTE's / *Basic Software Scheduler*'s perspective.

4.4.3 Refinement of the semantics of ModeDeclarations and ModeDeclarationGroups

To implement the logic of mode switches, the RTE / *Basic Software Scheduler* needs some basic information about the available modes. For this reason, RTE / *Basic Software Scheduler* will make the following additional assumptions about the modes of one ModeDeclarationGroup:

1. **[constr_9013] Exactly one mode or one mode transition shall be active** ⌈ Whenever any RunnableEntity or *Basic Software Schedulable Entity* is running, there shall always be exactly one mode or one mode transition active of each ModeDeclarationGroupPrototype. ⌋

2. Immediately after initialization of a mode machine instance, RTE / *Basic Software Scheduler* will execute a transition to the initial mode of each ModeDeclarationGroupPrototype (see [SWS_Rte_02544]).

   RTE / *Basic Software Scheduler* will enforce the mode disablings of the initial modes and trigger the OnEntry ExecutableEntitys (if any defined) of the initial modes of every ModeDeclarationGroupPrototype immediately after initialization of the RTE / *Basic Software Scheduler*.

In other words, RTE / *Basic Software Scheduler* assumes, that the modes of one ModeDeclarationGroupPrototype belong to exactly one state machine without nested states. The state machines cover the whole lifetime of the atomic AUTOSAR SW-Cs[10] and mode dependent AUTOSAR Basic Software Modules [11].

4.4.4 Order of actions taken by the RTE / *Basic Software Scheduler* upon interception of a mode switch notification

This section describes what the 'communication' of a mode switch to a mode user actually does. What does the RTE *Basic Software Scheduler* do to switch a mode and especially in which order.

Mode switch procedures

Depending on the needs of mode users for synchronicity, the mode machine instance can be implemented with two different realizations.

- synchronous mode switching procedure
- asynchronous mode switching procedure

The differences between these two realizations are the omitted waiting conditions in case of asynchronous mode switching procedure. For instance with asynchronous behavior a software component can not rely that all mode disabling dependent ExecutableEntitys of the previous mode are terminated before OnEntry ExecutableEntitys and OnExit ExecutableEntitys are started. On one hand this might put some effort to the software component designer to enable the components implementation to support this kind of scheduling but on the other hand it enables fast and lean mode switching.

[10] The lifetime of an atomic AUTOSAR SW-C is considered to be the time span in which the SW-C's runnables are being executed.

[11] The lifetime of an mode dependent AUTOSAR Basic Software Module is considered to be the time span in which the *Basic Software Schedulable Entities* are being executed.

[SWS_Rte_07150] ⌈ The RTE generator shall use the synchronous mode switching procedure if at least one mode user of the mode machine instance does not support the asynchronous mode switch behavior. ⌋*(SRS_Rte_00143, SRS_Rte_00213)*

[SWS_Rte_07151] ⌈ The RTE generator shall apply the asynchronous mode switch behavior, if all mode users support the asynchronous mode switch behavior and if it is configured for the related mode machine instance. ⌋*(SRS_Rte_00143, SRS_Rte_00213)*

Typical usage of modes to protect resources

RTE / *Basic Software Scheduler* can start and prevent the execution of RunnableEntitys and BswSchedulableEntity. In the context of mode switches,

- RTE / *Basic Software Scheduler* starts OnExit ExecutableEntitys for leaving the previous mode. This is typically used by 'clean up ExecutableEntitys' to free resources that were used during the previous mode.
- RTE / *Basic Software Scheduler* starts OnEntry ExecutableEntitys for entering the next mode. This is typically used by 'initialization ExecutableEntitys' to allocate resources that are used in the next mode.
- And RTE / *Basic Software Scheduler* can prevent the execution of mode disabling dependent ExecutableEntitys within a mode. This is typically used with time triggered 'work ExecutableEntity' that use a resource which is not available in a certain mode.

According to this use case, during the execution of 'clean up ExecutableEntitys' and 'initialization ExecutableEntitys' the 'work ExecutableEntitys' should be disabled to protect the resource. Also, if the same resource is used (by different SW-C's) in two successive modes, the 'clean up ExecutableEntitys' should be safely terminated before the 'initialization ExecutableEntitys' of the next mode are executed (synchronous mode switching procedure). In summary, this would lead to the following sequence of actions by the RTE / *Basic Software Scheduler* upon reception of the mode switch notification:

1. activate mode disablings for the next mode
2. wait for the newly disabled ExecutableEntitys to terminate in case of synchronous mode switching procedure
3. execute 'clean up ExecutableEntitys'
4. wait for the 'clean up ExecutableEntitys' to terminate in case of synchronous mode switching procedure
5. execute 'initialization ExecutableEntitys'
6. wait for the 'initialization ExecutableEntitys' to terminate in case of synchronous mode switching procedure

7. deactivate `mode disablings` for the previous modes and enable `ExecutableEntity`s that have been disabled in the previous mode.

RTE / *Basic Software Scheduler* can also start `OnTransition ExecutableEntitys` on a transition between two modes which is not shown in this use case example.

Often, only a fraction of the SW-Cs, Runnable Entities, Basic Software modules and Basic Software Schedulable Entities of one ECU depends on the modes that are switched. Consequently, it should be possible to design the system in a way, that the mode switch does not influence the performance of the remaining software.

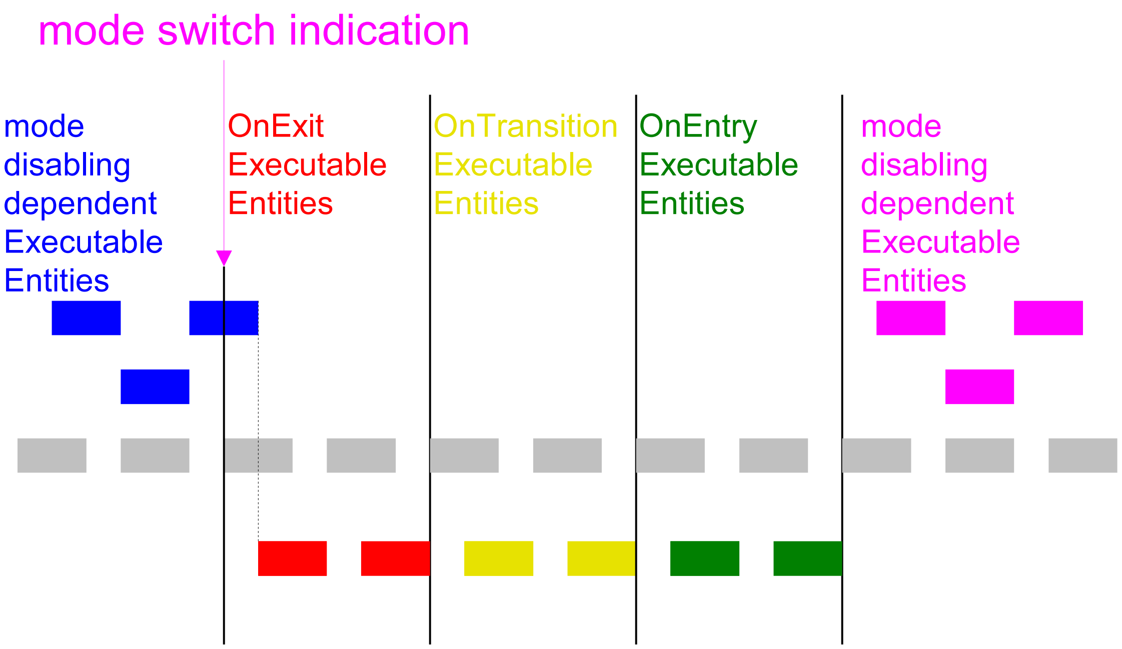


Figure 4.46: This figure shall illustrate what kind of ExecutableEntities will run in what order during a synchronous mode transition. The boxes indicate activated ExecutableEntities. Mode disabling dependant ExecutableEntities are printed in blue (old mode) and pink (new mode). OnExit, OnTransition, and OnEntry ExecutableEntity are printed in red, yellow, and green.

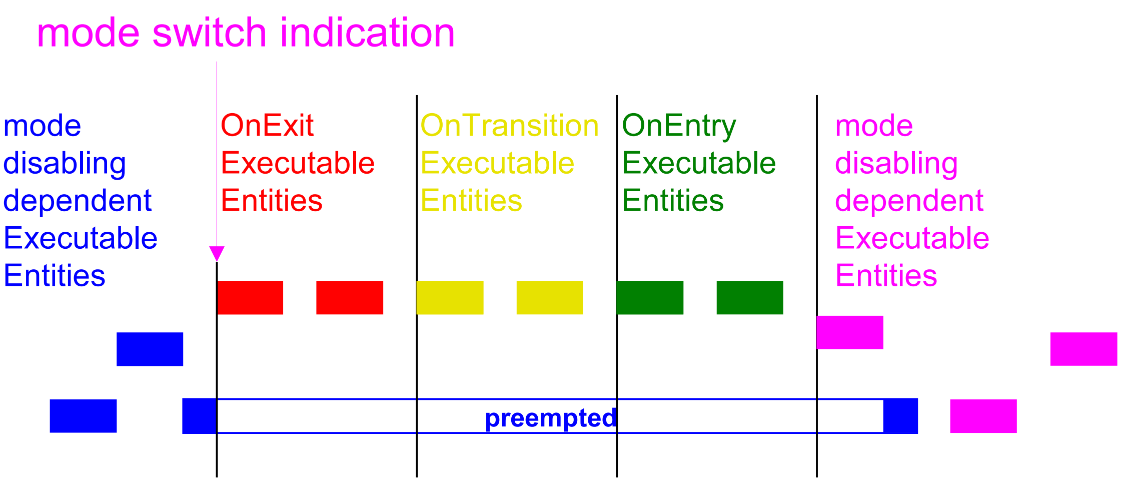


Figure 4.47: This figure shall illustrate what kind of ExecutableEntity will run in what order during an asynchronous mode transition where the ExecutableEntities are triggered on a mode change are mapped to a higher priority task than the Mode Dependent ExecutableEntity. The boxes indicate activated ExecutableEntity. Mode disabling dependant ExecutableEntity are printed in blue (old mode) and pink (new mode). OnExit, OnTransition, and OnEntry ExecutableEntity are printed in red, yellow, and green.

The remainder of this section lists the requirements that guarantee the behavior described above.

All runnables with dependencies on modes have to be executed or terminated during mode transitions. Restriction [SWS_Rte_02500] requires these runnables to be of category 1 to guarantee finite execution time.

For simplicity of the implementation to guarantee the order of runnable executions, the following restriction is made:

All `OnEntry ExecutableEntitys`, `OnTransition ExecutableEntitys`, and `OnExit ExecutableEntitys` of the same `mode machine instance` should be mapped to the same task in the execution order following OnExit, OnTransition, OnEntry (see [SWS_Rte_02662]).

A `mode machine instance` implementing an asynchronous mode switch procedure might be fully implemented inside the `Rte_Switch` or `SchM_Switch` API. In this case the `OnEntry ExecutableEntitys`, `OnTransition ExecutableEntitys`, `OnExit ExecutableEntitys` and `mode switch acknowledge ExecutableEntitys` are not mapped to tasks as described in chapter 7.6.1.

[SWS_Rte_07173] ⌈ The RTE generator shall support invocation of `OnEntry ExecutableEntitys`, `OnTransition ExecutableEntitys`, `OnExit ExecutableEntitys` and `mode switch acknowledge ExecutableEntitys` via direct function call, if all following conditions are fulfilled:

- if the asynchronous mode switch behavior is configured (see [SWS_Rte_07151])
- the `OnEntry ExecutableEntitys`, `OnTransition ExecutableEntitys`, `OnExit ExecutableEntitys` and `mode switch acknowledge ExecutableEntitys` do not define a 'minimum start distance'
- the `mode manager` and `mode user` are in the same Partition
- if the preconditions of table 4.5 are fulfilled

⌋ *(SRS_Rte_00143, SRS_Rte_00213)*

Further on the requirements [SWS_Rte_05083], [SWS_Rte_07155] and [SWS_Rte_07157] has to be considered.

[SWS_Rte_02667] ⌈ Within the mode manager's `Rte_Switch` / `SchM_Switch` API call to indicate a mode switch, one of the following shall be done:

1. If the corresponding mode machine instance is in a transition, and the queue for mode switch notifications is full, Rte_Switch / SchM_Switch shall return an error immediately.
2. If the corresponding mode machine instance is in a transition, and the queue for mode switch notifications is not full, the mode switch notification shall be queued.
3. If the mode machine instance is not in a transition, Rte_Switch / SchM_Switch shall initiate the transition as described by the sequence in [SWS_Rte_02665] which in turn activates the mode disablings (see [SWS_Rte_02661]) of the next mode.

⌋*(SRS_Rte_00143, SRS_Rte_00213)*

The following list holds the requirements for the steps of a mode transition.

- **[SWS_Rte_02661]** ⌈ At the beginning of a transition of a mode machine instance, the RTE / *Basic Software Scheduler* shall activate the mode disablings of the next mode (see also [SWS_Rte_02503]), if any ModeDisablingDependencys for that mode are defined. ⌋*(SRS_Rte_00143, SRS_Rte_00213)*
- **[SWS_Rte_07152]** ⌈ If any ModeDisablingDependencys for the next mode are defined (as specified by [SWS_Rte_02661]), the RTE / *Basic Software Scheduler* shall wait until the newly disabled RunnableEntitys and *Basic Software Schedulable Entities* are terminated, in case of synchronous mode switching procedure. ⌋*(SRS_Rte_00143, SRS_Rte_00213)*

  Note: To guarantee in case of synchronous mode switching all activated mode disabling dependent ExecutableEntitys of this mode machine instance have terminated before the start of the OnExit ExecutableEntitys of the transition, RTE generator can exploit the restriction [SWS_Rte_02663] that mode disabling dependent ExecutableEntitys run with higher or equal priority than the OnExit ExecutableEntitys and the OnEntry ExecutableEntitys.
- **[SWS_Rte_02562]** ⌈ RTE / *Basic Software Scheduler* shall execute the OnExit ExecutableEntitys of the previous mode. ⌋*(SRS_Rte_00143, SRS_Rte_00052, SRS_Rte_00213)*
- **[SWS_Rte_07153]** ⌈ If any OnExit ExecutableEntity is configured the RTE / *Basic Software Scheduler* shall wait after its execution ([SWS_Rte_02562]) until all OnExit ExecutableEntitys are terminated in case of synchronous mode switching procedure. ⌋*(SRS_Rte_00143, SRS_Rte_00213)*
- **[SWS_Rte_02707]** ⌈ RTE / *Basic Software Scheduler* shall execute the OnTransition ExecutableEntitys configured for the transition from previous mode to next mode. ⌋*(SRS_Rte_00143, SRS_Rte_00052, SRS_Rte_00213)*

- **[SWS_Rte_02708]** ⌈ If any OnTransition ExecutableEntity is configured, the RTE / *Basic Software Scheduler* shall wait after its execution ([SWS_Rte_02707]) until all OnTransition ExecutableEntitys are terminated in case of synchronous mode switching procedure. ⌋*(SRS_Rte_00143, SRS_Rte_00213)*

- **[SWS_Rte_02564]** ⌈ RTE / *Basic Software Scheduler* shall execute the OnEntry ExecutableEntitys of the next mode. ⌋*(SRS_Rte_00143, SRS_Rte_00052, SRS_Rte_00213)*

- **[SWS_Rte_07154]** ⌈ If any OnEntry ExecutableEntity is configured the RTE shall wait after its execution ([SWS_Rte_02564]) until all OnEntry ExecutableEntitys are terminated in case of synchronous mode switching procedure. ⌋*(SRS_Rte_00143, SRS_Rte_00213)*

- **[SWS_Rte_02563]** ⌈ The RTE / *Basic Software Scheduler* shall deactivate the previous mode disablings and only keep the mode disablings of the next mode. ⌋*(SRS_Rte_00143, SRS_Rte_00213)*

  With this, the transition is completed.

- **[SWS_Rte_02587]** ⌈ At the end of the transition, RTE / *Basic Software Scheduler* shall trigger the ModeSwitchedAckEvents connected to the mode manager's *ModeDeclarationGroupPrototype*. ⌋*(SRS_Rte_00143, SRS_Rte_00213)*

  This will result in an acknowledgment on the mode manager's side which allows the mode manager to wait for the completion of the mode switch.

  The dequeuing of the mode switch notification shall also be done at the end of the transition, see [SWS_Rte_02721].

[SWS_Rte_02665] ⌈ During a transition of a mode machine instance each applicable of the steps

1. [SWS_Rte_02661] (The transition is entered in parallel with this step),
2. [SWS_Rte_07152],
3. [SWS_Rte_02562],
4. [SWS_Rte_07153],
5. [SWS_Rte_02707],
6. [SWS_Rte_02708],
7. [SWS_Rte_02564],
8. [SWS_Rte_07154],
9. [SWS_Rte_02563] (The transition is completed with this step), and
10. immediately followed by [SWS_Rte_02587]

shall be executed in the order as listed. If a step is not applicable, the order of the remaining steps shall be unchanged. ⌋(SRS_Rte_00143, SRS_Rte_00213)

[SWS_Rte_02668] ⌈ Immediately after the execution of a transition as described in [SWS_Rte_02665], RTE / *Basic Software Scheduler* shall check the queue for pending mode switch notifications of this mode machine instance. If a mode switch notification can be dequeued, the mode machine instance shall enter the corresponding transition directly as described by the sequence in [SWS_Rte_02665]. ⌋(SRS_Rte_00143, SRS_Rte_00213)

In the case of a fast sequence of two mode switches, the Rte_Mode or SchM_Mode API will not indicate an intermediate mode, if a mode switch notification to the next mode is indicated before the transition to the intermediate mode is completed.

[SWS_Rte_02630] ⌈ In case of synchronous mode switch procedure, the RTE shall execute all steps of a mode switch (see [SWS_Rte_02665]) synchronously for the whole mode machine instance. ⌋(SRS_Rte_00143, SRS_Rte_00213)

I.e., the mode transitions will be executed synchronously for all mode users that are connected to the same mode manager's ModeDeclarationGroupPrototype.

[SWS_Rte_02669] ⌈ If the next mode and the previous mode of a transition are the same, the transition shall still be executed. ⌋(SRS_Rte_00143, SRS_Rte_00213)

Mapping of ModeDeclarations

There exist several use cases (especially if software is reused), where mode users are connected to mode managers providing ModeDeclarationGroups with different ModeDeclarations than the user.

Examples:

- A mode manager can be able to differentiate more fin grained sub states as it is required by the generic mode user. But due to the definition of the mode communication it is not possible to use two p-ports at the mode manager because this would lead to two independent and unsynchronized mode machine instances in the RTE.
- A generic mode user can support additionally modes which are not used by all mode managers.

This would normally lead to an error as incompatible ports are connected. To overcome this limitation the Software Component Template [2] provides a mapping between different ModeDeclarations so that the RTE can translated on mode to the other.

[SWS_Rte_08511] ⌈ If a ModeDeclaration of a mode user is mapped to a single ModeDeclaration of a mode manager the related mode of the mode user is entered or exit when the mapped mode of the mode manager is entered or exit. ⌋(SRS_Rte_00236)

[SWS_Rte_08512] ⌈ If one ModeDeclaration of a mode user is mapped to several ModeDeclarations of a mode manager the related mode of the mode user is entered when any of the mapped modes of the mode manager mapped by one modeDeclarationMapping is entered. The related mode of the mode user is exit when any of the mapped modes of the mode manager mapped by one modeDeclarationMapping is exit and if the new mode is not mapped by the same modeDeclarationMapping to related mode of the mode user. ⌋*(SRS_Rte_00236)*

Note: If one ModeDeclaration of a mode user is mapped to several ModeDeclarations of a mode manager by the means of several modeDeclarationMappings the semantics is defined in a way that the individual mode transitions of the mode manager are getting visible as "exit" and "enter" events for the mode user. Further on the transition phase gets visible by the RTE_TRANSITION return value in the case that Rte_Mode-API is called during such a transition phase.

If one ModeDeclaration of a mode user is mapped to several ModeDeclarations of a mode manager by the means of a single modeDeclarationMapping the semantics is defined in a way that the individual mode transitions of the mode manager are **not** visible for the mode user.

Example:
The mode manager and the mode user have different ModeDeclarationGroups which are mapped by several modeDeclarationMappings. The ModeDeclarationGroup of the mode manager is more fine grained, so more than one of its ModeDeclarations has to be mapped onto the same ModeDeclaration of the mode user. The modeDeclarationMappings can be seen in table 4.12. The complete example is listed as ARXML in Appendix F.1.

modeDeclarationMapping	**ModeDeclarations of the mode manager**	**Mapped ModeDeclarations of the mode user**
StartUp_2_STARTUP	StartUp	STARTUP
Run_2_RUN	Run	RUN
PostRunX_2_POST_RUN	PostRun1 PostRun2	POST_RUN
ShutDown_2_SHUTDOWN	ShutDown	SHUTDOWN
Sleep_Hibernate_2_SHUTDOWN	Sleep Hibernate	SHUTDOWN

Table 4.12: Example of a modeDeclarationMapping which maps ModeDeclarations from mode manager to ModeDeclarations of the mode user

Table 4.13 shows a possible scenario how mode transitions of a mode manager will be seen from the point of view of a mode user when the modeDeclarationMapping maps more than one ModeDeclaration of the mode manager's ModeDeclarationGroup onto the same ModeDeclaration of the mode user's ModeDeclarationGroup.

Mode transitions of the mode manager	**Mode transitions of the mode user resulting out of the mapping**
Undefined → StartUp	Undefined → STARTUP
StartUp → Run	STARTUP → RUN

Run → PostRun1	RUN → POST_RUN
PostRun1 → PostRun2	— (no transition)
PostRun2 → ShutDown	POST_RUN → SHUTDOWN
ShutDown → Sleep	SHUTDOWN → SHUTDOWN
Sleep → Hibernate	— (no transition)

Table 4.13: Possible scenario of mode transitions by the mode manager and the resulting transitions from the point of view of the mode user

A configuration that maps several ModeDeclarations of a mode user to a single ModeDeclaration representing a mode of a mode manager shall be rejected (see also [constr_1209]). This is not valid as it violates the principle that modes are mutually exclusive.

[SWS_Rte_08513] ⌈ The RTE-Generator shall reject configurations violating [constr_1209]. ⌋*(SRS_Rte_00236)*

If a modeDeclarationMapping exists that references a ModeDeclaration representing a mode of the mode manager then ModeDeclarationMappings shall exist that map all ModeDeclarations of the mode manager to ModeDeclarations of the mode user (see also [constr_1210]).

[SWS_Rte_08514] ⌈ The RTE-Generator shall reject configurations violating [constr_1210]. ⌋*(SRS_Rte_00236)*

Note: It is only supported that modes of the mode user might not be mapped.

4.4.5 Assignment of mode machine instances to RTE and Basic Software Scheduler

[SWS_Rte_07533] ⌈ A mode machine instance shall be assigned to the RTE if the correlating ModeDeclarationGroupPrototype is instantiated in a port of a software-component and if the ModeDeclarationGroupPrototype is not synchronized (*synchronizedModeGroup* of a *SwcBswMapping*) with a providedModeGroup ModeDeclarationGroupPrototype of a Basic Software Module instance. ⌋*(SRS_Rte_00143)*

[SWS_Rte_07534] ⌈ A mode machine instance shall be assigned to the *Basic Software Scheduler* if the correlating ModeDeclarationGroupPrototype is a providedModeGroup ModeDeclarationGroupPrototype of a Basic Software Module instance. ⌋*(SRS_Rte_00213)*

[SWS_Rte_07535] ⌈ The RTE Generator shall create only one mode machine instance if a *ModeDeclarationGroupPrototype* instantiated in a port of a software-component is synchronized (*synchronizedModeGroup* of a SwcBswMapping) with a providedModeGroup ModeDeclarationGroupPrototype of a Basic Software Module instance. The related common mode machine instance shall be assigned to the *Basic Software Scheduler*. ⌋*(SRS_Rte_00143, SRS_Rte_00213, SRS_Rte_00214)*

In case of synchronized ModeDeclarationGroupPrototypes the correlating common mode machine instance is initialized during the execution of the SchM_Init. At this point of time the scheduling of RunnableEntitys is not enabled due to the uninitialized RTE. Therefore situation occurs, that the RunnableEntitys being OnEntry ExecutableEntitys are not called if the mode machine instance is initialized. Further on the current mode of such mode machine instance might be still switched until the RTE gets initialized. Nevertheless the *OnEntry Runnables* of the current active mode are executed.

[SWS_Rte_07582] ⌈ For common mode machine instances the *OnEntry Runnable Entities* of the current active mode are executed during the initialization of the RTE if the common mode machine instance is not in transition. ⌋*(SRS_Rte_00214)*

[SWS_Rte_07583] ⌈ A common mode machine instances is not allowed to enter transition phase during the RTE initialization if the common mode machine instances has *OnEntry Runnable Entities*, *OnTransition Runnable Entities* or *OnExit Runnable Entities* ⌋*(SRS_Rte_00214)*

Note: [SWS_Rte_07582] and [SWS_Rte_07583] shall ensure a deterministic behavior that the software components receiving a Mode Switch Request from a common mode machine instances are receiving the current active mode during RTE initialization.

[SWS_Rte_07564] ⌈ The RTE generator shall reject configurations where ModeSwitchPoint(s) referencing a ModeDeclarationGroupPrototype in a mode switch port and a managedModeGroup association(s) to a providedModeGroup ModeDeclarationGroupPrototype are not defined mutual exclusively to one of two synchronized ModeDeclarationGroupPrototypes. ⌋*(SRS_Rte_00143, SRS_Rte_00213, SRS_Rte_00214, SRS_Rte_00018)*

[constr_9014] *ModeSwitchPoint*(s) and *managedModeGroup*(s) are mutually exclusive for synchronized *ModeDeclarationGroupPrototypes* ⌈ Only one of two synchronized *ModeDeclarationGroupPrototype*s shall mutual exclusively be referenced by *ModeSwitchPoint*(s) or *managedModeGroup* association(s). ⌋

Note: [constr_9014] shall ensure in the combination with the existence conditions of the Rte_Switch, Rte_Mode, Rte_SwitchAck, SchM_Switch, SchM_Mode and SchM_SwitchAck that either the port based RTE API or the *Basic Software Scheduler* API ([SWS_Rte_07201] and [SWS_Rte_07264]) offered to the implementation of the mode manager.

4.4.6 Initialization of mode machine instances

A mode machine instance can either be initialized during Rte_Start or during Rte_Init. The initialization during Rte_Init enables a defined order when which mode machine instance gets initialized and the belonging *OnEntry Runnable Entities* are scheduled.

[SWS_Rte_06766] ⌈ RTE shall initiate the transition to the initial modes of each mode machine instance belonging to the RTE during Rte_Start if the *OnEntry Runnable Entities* for the initialMode are not mapped to any RteInitialization-RunnableBatch container. ⌋*(SRS_Rte_00143, SRS_Rte_00144, SRS_Rte_00116)*

[SWS_Rte_06767] ⌈ RTE shall initiate the transition to the initial modes of each mode machine instance belonging to the RTE during Rte_Init if the *OnEntry Runnable Entities* for the initialMode are mapped to one or several RteInitialization-RunnableBatch container. ⌋*(SRS_Rte_00143, SRS_Rte_00144, SRS_Rte_00116, SRS_Rte_00240)*

Please note the restrictions on the mapping to RteInitializationRunnable-Batch containers [constr_9062], [constr_9063] and [constr_9064].

[SWS_Rte_02544] ⌈ During the transition to the initial modes of mode machine instances belonging to the RTE, the steps defined in the following requirements have to be omitted as no previous mode is defined:

- [SWS_Rte_02562],
- [SWS_Rte_07153],
- [SWS_Rte_02707],
- [SWS_Rte_02708],
- [SWS_Rte_02563],
- [SWS_Rte_02587]

If applicable, the steps described by the following requirements still have to be executed for entering the initial mode:

- [SWS_Rte_02661],
- [SWS_Rte_02564]

⌋*(SRS_Rte_00143, SRS_Rte_00144, SRS_Rte_00116)*

[SWS_Rte_07532] ⌈ *Basic Software Scheduler* shall initiate the transition to the initial modes of each mode machine instance belonging to the *Basic Software Scheduler* during SchM_Init. During the transition to the initial modes, the steps defined in the following requirements have to be omitted as no previous mode is defined:

- [SWS_Rte_02562],
- [SWS_Rte_07153],
- [SWS_Rte_02707],
- [SWS_Rte_02708],

- [SWS_Rte_02563],
- [SWS_Rte_02587]

If applicable, the steps described by the following requirements still have to be executed for entering the initial mode:

- [SWS_Rte_02661],
- [SWS_Rte_02564]

⌋*(SRS_Rte_00213)*

4.4.7 Notification of mode switches

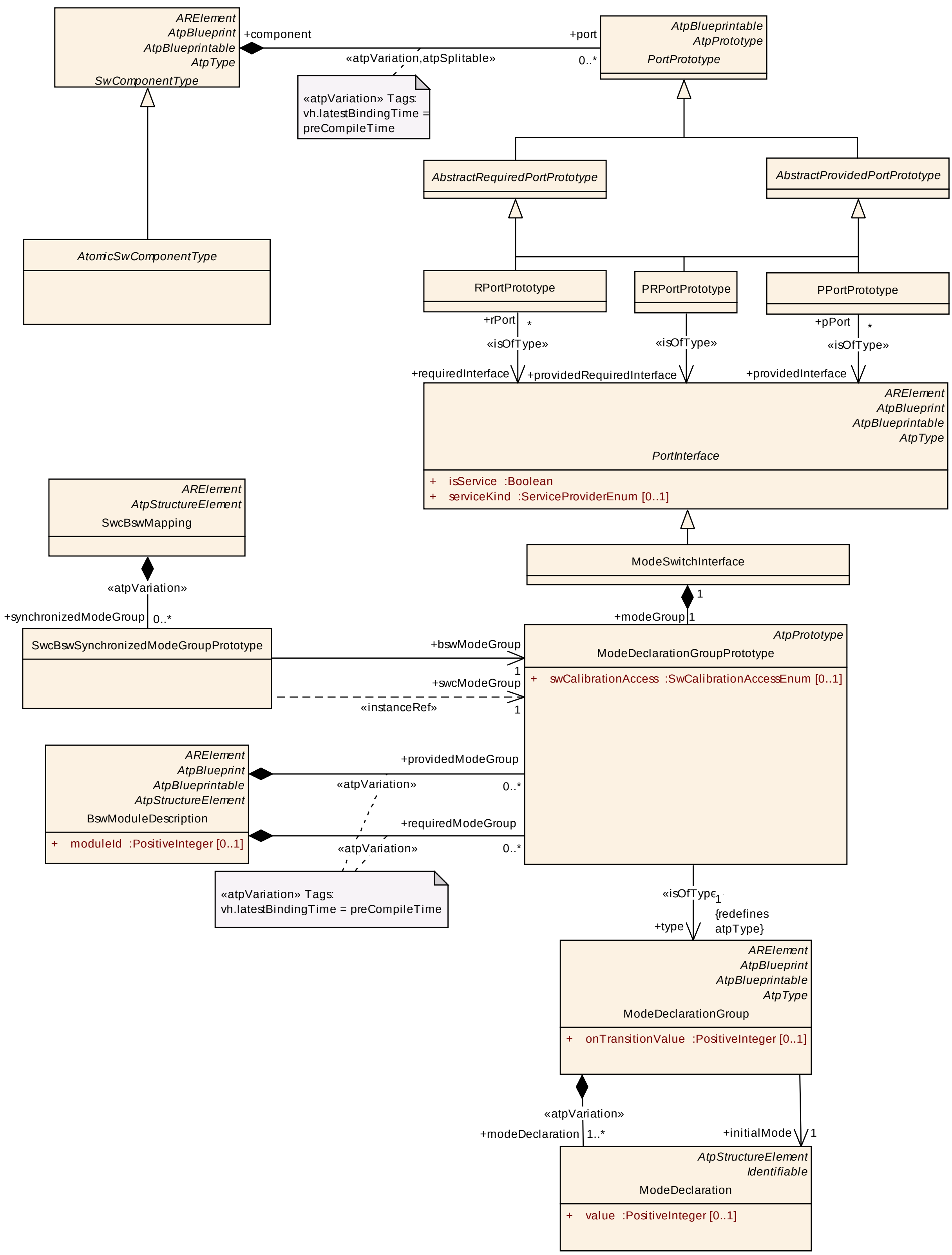


Figure 4.48: Definition of a ModeSwitchInterface.

- **[SWS_Rte_02549]** ⌈ Mode switches shall be communicated via RTE by ModeDeclarationGroupPrototypes of a ModeSwitchInterface as defined in [2], see Fig. 4.48. ⌋*(SRS_Rte_00144)*

  The mode switch ports of the mode manager and the mode user are of the type of a ModeSwitchInterface.

- **[SWS_Rte_07538]** ⌈ Mode switches shall be communicated via *Basic Software Scheduler* via providedModeGroup and requiredModeGroup *ModeDeclarationGroupPrototype*s as defined in [9], see Fig. 4.48. Which *ModeDeclarationGroupPrototype*s are connected to each other is defined by the configuration of the *Basic Software Scheduler*. ⌋*(SRS_Rte_00213)*

- RTE / *Basic Software Scheduler* only requires the notification of switches between modes.

- AUTOSAR does not support inter ECU communication of mode switch notifications.

  RTE does not support a configuration in which the mode users of one mode machine instance are distributed over several partitions, see [SWS_Rte_02724].

  Rationale: Mode switch communication requires high synchronization effort. Such a high coupling should be avoided between ECUs and between partitions. This does not prevent distributed mode management.

  For the distributed mode management mode requests can be distributed via ServiceProxySwComponentTypes and the BswM of each target ECU to the mode users of the BswMs.

- **[SWS_Rte_02508]** ⌈ A mode switch shall be notified asynchronously as indicated by the use of a ModeSwitchInterface. ⌋*(SRS_Rte_00144)*

  Rationale: This simplifies the communication. Due to [SWS_Rte_02724] the communication is local and no handshake is required to guarantee reliable transmission.

  RTE offers the Rte_Switch API to the mode manager for this notification, see 5.6.6.

  Basic Software Scheduler offers the SchM_Switch API to the mode manager for this notification, see 6.5.7.

- The mode manager might still require a feedback to keep it's internal state machine synchronized with the RTE / *Basic Software Scheduler* view of active modes.

  The RTE generator shall support an AcknowledgementRequest from the mode switch port / providedModeGroup ModeDeclarationGroupPrototype of a mode manager, see [SWS_Rte_02587], to notify the mode manager of the completion of a mode switch.

- **[SWS_Rte_02566]** ⌈ A ModeSwitchInterface shall support 1:n communication. ⌋*(SRS_Rte_00144)*

  Rationale: This simplifies the configuration and the communication. One mode switch can be notified to all receivers simultaneously.

  A ModeSwitchInterface does not support n:1 communication, see [SWS_Rte_02670].

- **[SWS_Rte_07539]** ⌈ The connection of providedModeGroup and requiredModeGroup ModeDeclarationGroupPrototype shall support 1:n communication. ⌋*(SRS_Rte_00213)*

- **[SWS_Rte_02624]** ⌈ A mode switch shall be notified with event semantics, i.e., the mode switch notifications shall be buffered by RTE or *Basic Software Scheduler* to which the mode machine instance is assigned. ⌋*(SRS_Rte_00144)*

  The queueing of mode switches (and SwcModeSwitchEvents) depends like that of DataReceivedEvents on the settings for the receiving port, see section 4.3.1.10.2.

- **[SWS_Rte_02567]** ⌈ A ModeSwitchInterface shall only indicate the next mode of the transition. ⌋*(SRS_Rte_00144)*

- **[SWS_Rte_07541]** ⌈ A providedModeGroup ModeDeclarationGroupPrototype shall only indicate the next mode of the transition. ⌋*(SRS_Rte_00213)*

  The API takes a single parameter (plus, optionally, the instance handle) that indicates the requested 'next mode'. For this purpose, RTE and *Basic Software Scheduler* will use identifiers of the modes as defined in [SWS_Rte_02568] and [SWS_Rte_07294].

- **[SWS_Rte_02546]** ⌈ The RTE shall keep track of the active modes of a mode manager's ModeDeclarationGroupPrototypes (mode machine instances) which is assigned to the RTE. ⌋*(SRS_Rte_00143, SRS_Rte_00144)*

- **[SWS_Rte_07540]** ⌈ The *Basic Software Scheduler* shall keep track of the active modes of a mode manager's ModeDeclarationGroupPrototypes (mode machine instances) which is assigned to the *Basic Software Scheduler*. ⌋*(SRS_Rte_00213, SRS_Rte_00144)*

  Rationale: This allows the RTE / *Basic Software Scheduler* to guarantee consistency between the timing for firing of SwcModeSwitchEvents / BswModeSwitchEvents and disabling the start of ExecutableEntities by ModeDisablingDependency without adding additional interfaces to a mode manager with fine grained substates on the transitions.

- The RTE offers an Rte_Mode API to the SW-C to get information about the active mode, see section 5.6.29.

- The *Basic Software Scheduler* offers an SchM_Mode API to the Basic Software Module to get information about the active mode, see section 6.5.8.
- In addition to the mode switch ports, the mode manager may offer an AUTOSAR interface for requesting and releasing modes as a means to keep modes alive like for ComM and EcuM.

4.4.8 Mode switch acknowledgment

In case of mode switch communication, the mode manager may specify a ModeSwitchedAckEvent or BswModeSwitchedAckEvent to receive a notification from the RTE that the mode transition has been completed, see [SWS_Rte_02679] and [SWS_Rte_07559].

The ModeSwitchedAckEvent is triggered by the RTE regardless which runnable entity has requested the mode switch notification, even if the meta model implies a link to a specific ModeSwitchPoint.

[SWS_Rte_02679] ⌈ If acknowledgment is enabled for a provided ModeDeclarationGroupPrototype and a ModeSwitchedAckEvent references a RunnableEntity as well as the ModeDeclarationGroupPrototype, the RunnableEntity shall be activated when the mode switch acknowledgment occurs or when the RTE detects that the partition to which the mode users are mapped was stopped or restarted. ⌋*(SRS_Rte_00051, SRS_Rte_00143)*

Note the constraint that all mode users are in the same partition ([SWS_Rte_02724]).

The related *Entry Point Prototype* is defined in [SWS_Rte_02512].

[SWS_Rte_07559] ⌈ If acknowledgment is enabled for a provided (providedModeGroup) ModeDeclarationGroupPrototype and a BswModeSwitchedAckEvent references a BswSchedulableEntity as well as the ModeDeclarationGroupPrototype, the BswSchedulableEntity shall be activated when the mode switch acknowledgment occurs or when a timeout was detected by the *Basic Software Scheduler*. [SWS_Rte_02587]. ⌋*(SRS_Rte_00213, SRS_Rte_00143)*

The related *Entry Point Prototype* is defined in [SWS_Rte_07283].

Requirement [SWS_Rte_02679] and [SWS_Rte_07559] merely affects when the runnable is activated. The Rte_SwitchAck and SchM_SwitchAck shall still be created, according to requirement [SWS_Rte_02678] and [SWS_Rte_07558] to actually read the acknowledgment.

[SWS_Rte_02730] ⌈ A ModeSwitchedAckEvent that references a RunnableEntity and is referenced by a WaitPoint shall be an invalid configuration which is rejected by the RTE generator. ⌋*(SRS_Rte_00051, SRS_Rte_00018, SRS_Rte_00143)*

The attributes ModeSwitchedAckRequest and BswModeSwitchAckRequest allow to specify a timeout.

[SWS_Rte_07056] ⌈ If ModeSwitchedAckRequest or BswModeSwitchAckRequest with a timeout greater than zero is specified, the RTE shall ensure that timeout monitoring is performed, regardless of the receive mode of the acknowledgment. ⌋*(SRS_Rte_00069, SRS_Rte_00143)*

[SWS_Rte_07060] ⌈ Regardless of an occurred timeout during a mode transition the RTE shall complete the transition of a mode machine instance as defined in [SWS_Rte_02665]. ⌋*(SRS_Rte_00069, SRS_Rte_00143)*

If a WaitPoint is specified to collect the acknowledgment, two timeout values have to be specified, one for the ModeSwitchedAckRequest and one for the WaitPoint.

[SWS_Rte_07057] ⌈ The RTE generator shall reject configuration violating [constr_4012] in software component template [2]. ⌋*(SRS_Rte_00018, SRS_Rte_00143)*

[SWS_Rte_07058] ⌈ The status information about the success or failure of the mode transition shall be buffered with last-is-best semantics. When a new mode switch notification is sent or when the mode switch notification was completed after a timeout, the status information is overwritten. ⌋*(SRS_Rte_00143)*

[SWS_Rte_07058] implies that once the ModeSwitchedAckEvent or BswModeSwitchedAckEvent has occurred, repeated API calls (Rte_SwitchAck or SchM_SwitchAck to retrieve the acknowledgment can return different values.

[SWS_Rte_07059] ⌈ If the timeout value of the ModeSwitchedAckRequest or BswModeSwitchAckRequest is 0, no timeout monitoring shall be performed. ⌋*(SRS_Rte_00069, SRS_Rte_00143)*

4.4.9 Mode switch error handling

Since the mode switch communication may cross partitions basically two error scenarios are possible:

- The partition of the mode users gets terminated.
- The partition of the mode manager gets terminated.

In both cases additionally the terminated partition may be restarted. For both error scenarios the RTE offers functionality to handle the errors.

4.4.9.1 Mode User gets terminated

When a mode manager is getting out of sync with the mode user(s) (because the partition of the mode user has been terminated) a sequence of error reactions is defined.

This shall support on the one hand to inform the mode manager about the fact that the mode users are absent. This might be used by the mode manager to set internal states. This supports an active error handling by the mode manager as well as a synchronization of the mode manager to the mode user's partition restart.

Furthermore the RTE offers the ability to switch into a default mode automatically. This feature can be used to ensure that either the mode users are re-initialized as during ECU start (default mode is initial mode) or that the mode users are re-initialized by a dedicated mode (default mode is different from initial mode) which in turn may be used to ensure a secure behavior of the mode user's, for instance suppressing the actuator self tests in the running system.

[SWS_Rte_06794] ⌈ The RTE Generator shall take the modeManagerErrorBehavior from the ModeDeclarationGroup typing the ModeDeclarationGroupPrototype in the ModeSwitchInterface of the PPortPrototype/PRPortPrototype. ⌋*(SRS_Rte_00143, SRS_Rte_00144)*

[SWS_Rte_06772] ⌈ The RTE shall clear all mode switch notifications in the queue when the partition of the mode users gets terminated. ⌋*(SRS_Rte_00143, SRS_Rte_00144)*

[SWS_Rte_06773] ⌈ The RTE shall activate RunnableEntitys triggered by a SwcModeManagerErrorEvent when the partition of the mode users gets terminated. ⌋*(SRS_Rte_00143, SRS_Rte_00144)*

[SWS_Rte_06774] ⌈ If ModeSwitchedAckRequest or BswModeSwitchAckRequest is specified, the RTE shall detect a timeout when the partition of the mode users gets terminated during an ongoing transition. ⌋*(SRS_Rte_00143, SRS_Rte_00144)*

Also see [SWS_Rte_02679], [SWS_Rte_07559], and [SWS_Rte_03853].

The further behavior of the mode machine instance depends on the attribute ModeDeclarationGroup.modeUserErrorBehavior.

[SWS_Rte_06775] ⌈ If the attribute modeManagerErrorBehavior.errorReactionPolicy is set to lastMode the mode machine instance stays in the last mode before the termination of the mode users. If the partition of the mode users gets terminated during an ongoing transition the last mode is the next mode of the transition. ⌋*(SRS_Rte_00143, SRS_Rte_00144)*

Please note: In case the partition of the mode users gets terminated during an ongoing transition logically the transition is still completed even if the mode users didn't "survive" the transition.

[SWS_Rte_06776] ⌈ If the attribute modeManagerErrorBehavior.errorReactionPolicy is set to defaultMode the RTE shall enqueue the mode defined by modeManagerErrorBehavior.defaultMode to the mode switch notification queue. ⌋*(SRS_Rte_00143, SRS_Rte_00144)*

If the ModeSwitchInterface does not define a specific modeManagerErrorBehavior the RTE uses the initialMode as a default mode.

[SWS_Rte_06777] ⌈ If the attribute modeManagerErrorBehavior is not defined the RTE shall enqueue the mode defined by initialMode to the mode switch notification queue. ⌋*(SRS_Rte_00143, SRS_Rte_00144)*

[SWS_Rte_06778] ⌈ The RTE shall execute the error reactions in case the partition of the mode users gets terminated in following order:

1. [SWS_Rte_06772]
2. [SWS_Rte_06773]
3. [SWS_Rte_06774]
4. [SWS_Rte_06775] or [SWS_Rte_06776] or [SWS_Rte_06777]

⌋*(SRS_Rte_00143, SRS_Rte_00144)*

If the partition of the mode users is capable to restart (PartitionCanBeRestarted == true) the mode manager shall be able to enqueue new mode switch requests during the restart of the partition. This shall support a dedicated error handling by the mode manager depending on other environmental conditions. In this case the mode manager may decide which transitions are appropriate to get the mode users either back in an operational mode or in a secure default mode. Therefore the errorReactionPolicy equals lastMode avoids any automatically forced mode transitions by the error handling of the RTE.

[SWS_Rte_06779] ⌈ RTE shall support the enqueueing of new mode switch requests during the restart of the mode user's partition by the mode manager after the call of Rte_PartitionRestarting. ⌋*(SRS_Rte_00143, SRS_Rte_00144)*

[SWS_Rte_06780] ⌈ When the partition with the mode users is restarted (after call of Rte_PartitionRestart), RTE shall dequeue queued mode switch notifications. ⌋*(SRS_Rte_00143, SRS_Rte_00144)*

When the first mode switch notification after a partition restart is dequeued the previous mode is defined as "last mode" or "on transition" depending on the modeManagerErrorBehavior.errorReactionPolicy. See [SWS_Rte_06783] and [SWS_Rte_06784].

Initialization of mode machine instance during mode user's partition restart

Depending on the modeManagerErrorBehavior the RTE has to re-initialize the mode machine instance during the restart of the mode user's partition. In case modeManagerErrorBehavior.errorReactionPolicy is set to defaultMode the behavior is similar as during the transition to the initial mode (see [SWS_Rte_02544]). During the initialization of the RTE resources for a restarting mode user partition only a subset of the single steps of a mode transition is applicable.

[SWS_Rte_06796] ⌈ During the transition to the default mode (next mode is default mode) of mode machine instances when the mode user's partition restarts, the steps defined in the following requirements have to be omitted as no previous mode is applicable:

- [SWS_Rte_02562],
- [SWS_Rte_07153],
- [SWS_Rte_02707],
- [SWS_Rte_02708],
- [SWS_Rte_02563],
- [SWS_Rte_02587]

If applicable, the steps described by the following requirements still have to be executed for entering the default mode:

- [SWS_Rte_02661],
- [SWS_Rte_02564]

⌋*(SRS_Rte_00143, SRS_Rte_00144)*

In case modeManagerErrorBehavior.errorReactionPolicy is set to lastMode the behavior indicates a stable mode during the re-initialization in order to provide the means to the mode manager to explicitly decide on the appropriate mode to handle the fault.

[SWS_Rte_06797] ⌈ If the attribute modeManagerErrorBehavior.errorReactionPolicy is set to lastMode the RTE / Basic Software Scheduler shall activate the mode disablings of the last mode during the partition restart, if any ModeDisablingDependencys for that mode are defined. ⌋*(SRS_Rte_00143, SRS_Rte_00144)*

4.4.9.2 Mode Manager gets terminated

When a mode user gets out of sync with the mode manager (because the partition of the mode manager has been terminated) a sequence of error reactions is defined.

Hereby the RTE offers the ability to automatically switch into a default mode. This feature can be used to ensure that the mode users are automatically switched into a defined mode which in turn may be used to ensure a secure behavior of the mode users, for instance switching off some actuators.

As an alternative the mode machine instance can stay in the last mode which can be used to keep the "status quo" until the mode manager is restarted.

[SWS_Rte_06795] ⌈ The RTE Generator shall take the modeUserErrorBehavior from the ModeDeclarationGroup typing the ModeDeclarationGroupPrototype in the ModeSwitchInterface of the PPortPrototype/PRPortPrototype. ⌋*(SRS_Rte_00143, SRS_Rte_00144)*

[SWS_Rte_06785] ⌈ If the partition of the mode manager gets terminated during an ongoing transition, the RTE shall complete the transition. ⌋*(SRS_Rte_00143, SRS_Rte_00144)*

[SWS_Rte_06786] ⌈ If the partition of the mode manager gets terminated during an ongoing transition, the RTE shall skip the mode switch acknowledgment. ⌋*(SRS_Rte_00143, SRS_Rte_00144)* For mode switch acknowledgment see [SWS_Rte_02587] and section 4.4.8

[SWS_Rte_06787] ⌈ The RTE shall clear all mode switch notifications in the queue when the partition of the mode manager gets terminated and after an ongoing transition is completed. ⌋*(SRS_Rte_00143, SRS_Rte_00144)*

[SWS_Rte_06788] ⌈ If the attribute modeUserErrorBehavior.errorReactionPolicy is set to lastMode the mode machine instance stays in the last mode before the termination of the mode manager. ⌋*(SRS_Rte_00143, SRS_Rte_00144)*

[SWS_Rte_06789] ⌈ If the attribute modeUserErrorBehavior.errorReactionPolicy is set to defaultMode the RTE shall enqueue the mode defined by modeUserErrorBehavior.defaultMode to the mode switch notification queue. ⌋*(SRS_Rte_00143, SRS_Rte_00144)*

[SWS_Rte_06790] ⌈ If the attribute modeUserErrorBehavior is not defined the RTE shall enqueue the mode defined by initialMode to the mode switch notification queue. ⌋*(SRS_Rte_00143, SRS_Rte_00144)*

[SWS_Rte_06791] ⌈ The RTE shall execute the error reactions in case the partition of the mode manager gets terminated in the following order:

1. [SWS_Rte_06785], [SWS_Rte_06786]
2. [SWS_Rte_06787]
3. [SWS_Rte_06788] or [SWS_Rte_06789] or [SWS_Rte_06790]

⌋*(SRS_Rte_00143, SRS_Rte_00144)*

[SWS_Rte_06792] ⌈ The RTE shall dequeue queued mode switch notifications and execute them regardless whether the partition with the mode manager is terminated, restarting or restarted. Thereby the restart of the mode manager's partition shall not abort the ongoing transition of a mode machine instance. ⌋*(SRS_Rte_00143, SRS_Rte_00144)*

This ensures that the defaultMode in the mode switch notification queue gets effective.

[SWS_Rte_06793] ⌈ The RTE shall activate RunnableEntitys triggered by a SwcModeManagerErrorEvent when the partition of the mode manager is restarted ⌋*(SRS_Rte_00143, SRS_Rte_00144)*

4.5 External and Internal Trigger

4.5.1 External Trigger Event Communication

4.5.1.1 Introduction

With the mechanism of the trigger event communication a software component or a *Basic Software Module* acting as a `Trigger Source` is able to request the activation of *Runnable Entities* respectively *Basic Software Schedulable Entities* of connected `Trigger Sinks`. Typically but not necessarily these *Runnable Entities* and *Basic Software Schedulable Entities* are executed in a sequential order.

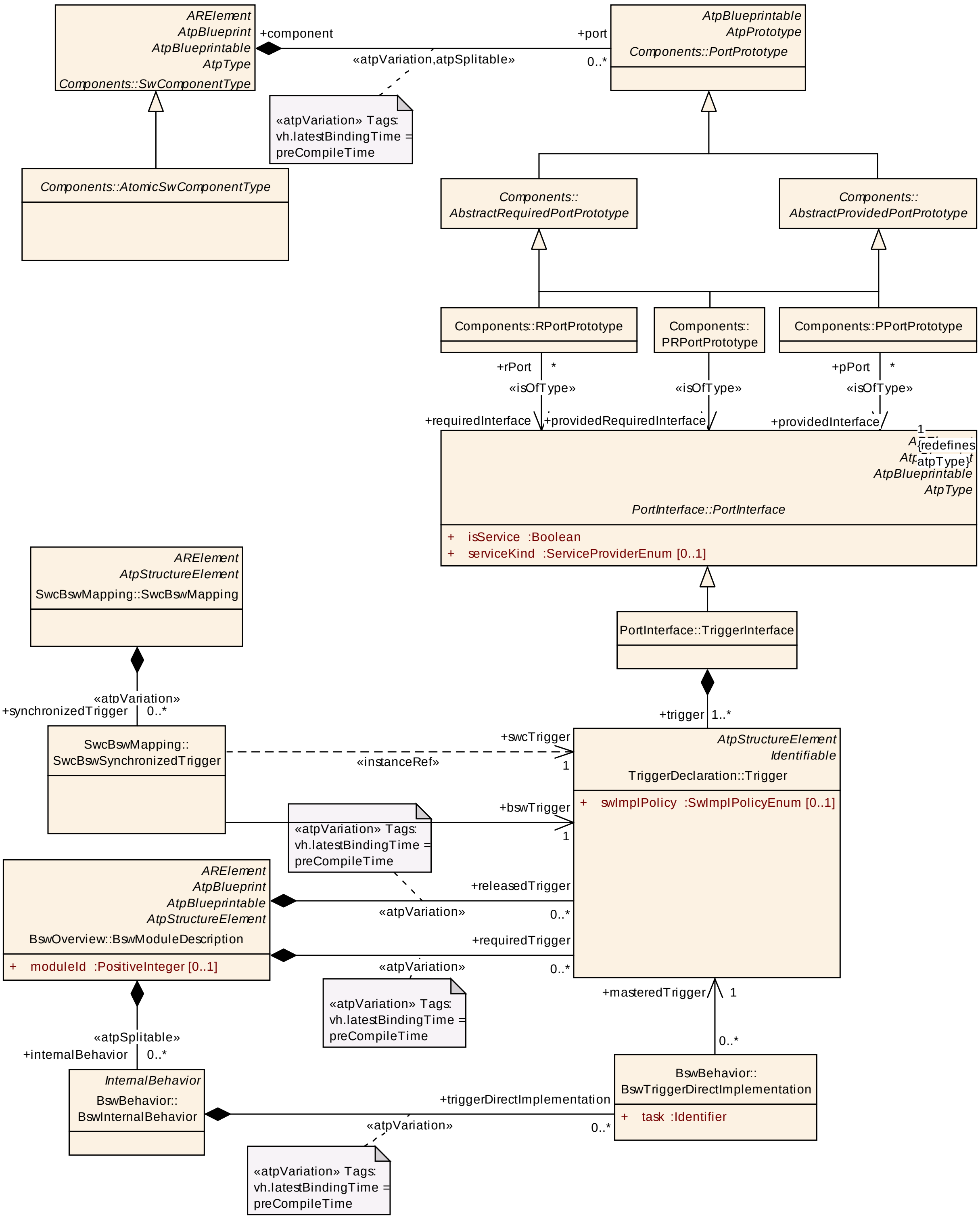


Figure 4.49: Summary of the use of Trigger by an AUTOSAR software-components and Basic Software Modules as defined in the *Software Component Template Specification* [2] and *Specification of BSW Module Description Template* [9].

[SWS_Rte_07212] ⌈ The RTE shall support *External Trigger Event Communication*. ⌋ *(SRS_Rte_00162)*

[SWS_Rte_07542] ⌈ The *Basic Software Scheduler* shall support the activation of *Basic Software Schedulable Entities* occurrence of External Trigger Events. ⌋*(SRS_Rte_00216)*

4.5.1.2 Trigger Sink

A AUTOSAR software-component `Trigger Sink` has a dedicated require `trigger port`. The `trigger port` is typed by an *TriggerInterface* declaring one or more *Trigger*. See figure 4.49. The *Runnable Entities* of the of the software component are activated at the occurrence of the external event by the means of a `ExternalTriggerOccurredEvent`.

An *Basic Software Module* `Trigger Sink` has to define a *requiredTrigger Trigger*. The *Basic Software Schedulable Entities* of the of the *Basic Software Module* are activated at the occurrence of the external event by the means of a `BswExternalTriggerOccurredEvent`. See figure 4.49.

Basically there are two approaches to implement the activation of `triggered ExecutableEntitys`s. In one case the `triggered ExecutableEntitys`s of the `Trigger Sinks`s triggered by one `Trigger` of the `Trigger Source` are mapped in one or more tasks. In this case the event communication can be implemented by the means of activating an Operating System Task. Please note that the tasks may belong to different partitions.

[SWS_Rte_07213] ⌈ The RTE generator shall support invocation of `triggered ExecutableEntitys` via OS Task. ⌋*(SRS_Rte_00162, SRS_Rte_00216)*

In the other case the Event Communication is mapped to a function call which means that the `triggered ExecutableEntitys` of the `Trigger Sinks` are executed in the `Rte_Trigger` API respectively `SchM_Trigger` API used to raise the trigger event in the `Trigger Sinks`.

[SWS_Rte_07214] ⌈ The RTE generator shall support invocation of `triggered ExecutableEntitys` via direct function call, if all of the follwing conditions are fulfilled:

- the `triggered ExecutableEntitys` do not define a 'minimum start distance'
- the `Trigger Sink` and `Trigger Source` are in the same Partition
- if no *BswTriggerDirectImplementation* is defined.
- if the preconditions of table 4.5 are fulfilled
- no queuing for the `Trigger Source` is configured

⌋*(SRS_Rte_00162, SRS_Rte_00216)*

4.5.1.3 Trigger Source

An AUTOSAR software-component `Trigger Source` has a dedicated provide `trigger port`. The `trigger port` is typed by an *TriggerInterface* declaring one or more *Trigger*. See figure 4.49. To be able to connect a provide `trigger port` and a require `trigger port`, both ports must be categorized by the same or by compatible *TriggerInterface*(s).

An *Basic Software Module* `Trigger Source` has to define a *releasedTrigger Trigger*. See figure 4.49. The connection of *releasedTrigger* and *requiredTrigger Trigger* is defined by the ECU configuration of the *Basic Software Scheduler*.

To inform the RTE about an occurrence of the external trigger event the RTE provides the `Rte_Trigger` to an AUTOSAR software-component `Trigger Source`.

[SWS_Rte_07543] ⌈ The call of the `Rte_Trigger` API shall activate all *Runnable Entities* that are activated by *ExternalTriggerOccurredEvent*s associated to a connected *Trigger* of the `Trigger Source` if either no queuing for the `Trigger` is configured or if queuing for the `Trigger` is configured and the trigger queue is empty. ⌋*(SRS_Rte_00162)*

For Basic Software Module `Trigger Source` are two options defined to interfaces with *Basic Software Scheduler*.

The first option is that the *Basic Software Module* `Trigger Source` inform the *Basic Software Scheduler* about an occurrence of the external trigger event by the call of the `SchM_Trigger` API.

[SWS_Rte_07544] ⌈ The call of the `SchM_Trigger` API shall activate all `ExecutableEntity`s that are activated by *ExternalTriggerOccurredEvent*s associated to a connected *Trigger* of the `Trigger Source` if either no queuing for the `Trigger` is configured or if queuing for the `Trigger` is configured and the trigger queue is empty. ⌋*(SRS_Rte_00216)*

The second option is that the *Basic Software Module* `Trigger Source` directly takes care about the activation of the particular OS task to which the *ExternalTriggerOccurredEvent*s of the `triggered ExecutableEntitys` are mapped. In this case the `Trigger Source` has to define a *BswTriggerDirectImplementation*. The name of the used OS tasks is annotated by the *task* attribute. If an *BswTriggerDirectImplementation* is defined no `SchM_Trigger` API is generated by the RTE generator. see [SWS_Rte_07548] and [SWS_Rte_07264].

[SWS_Rte_07545] ⌈ The RTE generator shall reject configurations where a *BswTriggerDirectImplementation* is specified and an `ExecutableEntity` that is activated by an *ExternalTriggerOccurredEvent* associated to a connected *Trigger* of the `Trigger Source` is mapped to an OS task different from the one defined by the task attribute of the *BswTriggerDirectImplementation*. ⌋*(SRS_Rte_00216, SRS_Rte_00018)*

[SWS_Rte_07548] ⌈ The RTE generator shall reject configurations where a *issuedTrigger* association and a *BswTriggerDirectImplementation* is defined for the same *releasedTrigger Trigger*. ⌋*(SRS_Rte_00216, SRS_Rte_00018)*

[constr_9007] *issuedTrigger* and *BswTriggerDirectImplementation* are mutually exclusive ⌈ A *releasedTrigger Trigger* shall not be referenced by both a *issuedTrigger* and a *BswTriggerDirectImplementation*. ⌋

Note: This shall ensure in the combination with the existence conditions ([SWS_Rte_07264]) of the `SchM_Trigger` that either the Trigger API or the direct task activation is offered to the implementation of the `Trigger Source`.

Note also that several OS tasks might be used to implement a `Trigger` (several `BswTriggerDirectImplementation` can be defined for a *releasedTrigger*).

If the `BswTriggerDirectImplementation` is defined for a *releasedTrigger* which `swImplPolicy` attribute is set to `queued` it is part of the `Trigger Source` to implement the queue or to use the means of the OS (OsTaskActivation > 1) to queue the number of raised triggers. (OsTaskActivation > 1). Further details about queuing of triggers is described in 4.5.5.

4.5.1.4 Multiplicity

4.5.1.4.1 Multiple Trigger

A trigger interface contains one or more `Trigger`. A port of an AUTOSAR software-component that provides an AUTOSAR trigger interface to the component can independently raise events related to each `Trigger` defined in the interface .

[SWS_Rte_07215] ⌈ The RTE API shall support independent event raising for each `Trigger` in a trigger interface. ⌋*(SRS_Rte_00162)*

Further on a *Basic Software Module* `Trigger Source` can define several *releasedTrigger Trigger* which can be independently raised.

[SWS_Rte_07546] ⌈ The *Basic Software Scheduler* API shall support independent event raising for each *releasedTrigger* `Trigger`. ⌋*(SRS_Rte_00216)*

4.5.1.4.2 Multiple Trigger Sinks Single Trigger Source

The concept of external event communication supports, that a `Trigger Source` activates one or more `triggered ExecutableEntitys` in one or more `Trigger Sinks`.

[SWS_Rte_07216] ⌈ The RTE generator shall support `triggered ExecutableEntitys` triggered by the same `Trigger` of a `Trigger Source` ('$1:n$' communication where $n \geq 1$). ⌋*(SRS_Rte_00162, SRS_Rte_00216)*

The execution order of the triggered ExecutableEntitys in the trigger sinks depends from the RteEventToTaskMapping described in chapter 7.6.1 and the configured priorities of the operating system.

4.5.1.4.3 Multiple Trigger Sources Single Trigger Sink

The RTE generator does not support multiple Trigger Sources communicating events to the same Trigger in a Trigger Sink ('$n:1$' communication where $n>1$).

[SWS_Rte_07039] ⌈ The RTE generator shall reject configurations where multiple Trigger Sources communicating events to the same Trigger in a Trigger Sink ('$n:1$' communication where $n>1$). ⌋*(SRS_Rte_00018)*

[constr_9008] The same Trigger in a *Trigger Sink* must not be connected to multiple *Trigger Sources* ⌈ The same Trigger in a *Trigger Sink* must not be connected to multiple *Trigger Source*s. ⌋

4.5.1.5 Synchronized Trigger

If two Triggers are synchronized by the definition of a SwcBswSynchronizedTrigger then the Trigger in the referenced provide trigger port and the referenced releasedTrigger Trigger are treated as one common Trigger. This means that all ExecutableEntitys activated by an ExternalTriggerOccurredEvent associated to one of the connected *Trigger*s are activated together.

[SWS_Rte_07218] ⌈ The RTE and *Basic Software Scheduler* shall activate together all ExecutableEntitys that are activated by ExternalTriggerOccurredEvents associated to a synchronized connected *Trigger*. ⌋*(SRS_Rte_00162, SRS_Rte_00216, SRS_Rte_00217)*

[SWS_Rte_07549] ⌈ The RTE generator shall reject configurations where a synchronized Trigger is referenced by more than one type of access method, where the type is one of the following:

1. ExternalTriggeringPoint
2. issuedTrigger
3. BswTriggerDirectImplementation

⌋*(SRS_Rte_00216, SRS_Rte_00217, SRS_Rte_00018)*

[constr_9009] Synchronized Trigger shall not be referenced by more than one type of access method ⌈ A synchronized Trigger shall only be referenced by either ExternalTriggeringPoints, issuedTriggers or BswTriggerDirectImplementations. ⌋

Note: This shall ensure in the combination with the existence conditions of the Rte_Trigger and SchM_Trigger that only one kind of Trigger API ([SWS_Rte_07201] and [SWS_Rte_07264]) or the direct task activation is offered to the implementation of the Trigger Source.

4.5.2 Inter Runnable Triggering

With the mechanism of *Inter Runnable Triggering* one *Runnable Entity* is able to request the activation of *Runnable Entities* of the same software-component instance.

[SWS_Rte_07220] ⌈ The RTE shall support Inter Runnable Triggering. ⌋*(SRS_Rte_00163)*

Similar to External Trigger Event Communication (described in chapter 4.5.1) the activation of triggered runnables can be implemented by means of activating an Operating System Task or by direct function call.

[SWS_Rte_07555] ⌈ The call of the Rte_IrTrigger API shall activate all triggered runnables which *InternalTriggerOccurredEvent*s are associated with the related *InternalTriggeringPoint* of the same software-component instance if either no queuing for the InternalTriggeringPoint is configured or if queuing for the InternalTriggeringPoint is configured and the trigger queue is empty. ⌋*(SRS_Rte_00163)*

[SWS_Rte_07221] ⌈ The RTE shall support for Inter Runnable Triggering that triggered runnables entities are invoked via OS Task activation. ⌋*(SRS_Rte_00163)*

[SWS_Rte_07224] ⌈ The RTE shall support for *Inter Runnable Triggering* that triggered runnables are invoked via direct function call if all of the following conditions are fulfilled:

- none of the triggered Basic Software Schedulable Entitys activated by this InternalTriggeringPoint define a 'minimum start distance'
- no queuing for the InternalTriggeringPointis configured

⌋*(SRS_Rte_00163)*

4.5.2.1 Multiplicity

A InternalTriggeringPoint might be referenced by more than one InternalTriggerOccurredEvent. Therefore one RunnableEntity is able to request the activation of several RunnableEntity's with the mechanism of Inter Runnable Triggering contemporaneously.

[SWS_Rte_07223] ⌈ The RTE shall support multiple RunnableEntity's triggered by the same InternalTriggeringPoint ('$1 : n$' Inter Runnable Triggering where $n \geq 1$). ⌋*(SRS_Rte_00163)*

The execution order of the runnable entities in the trigger sinks depends from the Runnable Entity to task mapping described in chapter 7.6.1 and the configured priorities of the operating system.

4.5.3 Inter Basic Software Module Entity Triggering

The *Inter Basic Software Module Entity Triggering* is similar to the mechanism of *Inter Runnable Triggering* (see chapter 4.5.2) with the exception that it is used inside a Basic Software Module. It can be used to request the activation of a *Basic Software Schedulable Entity* by a *Basic Software Entity* of the same a *Basic Software Module* instance.

[SWS_Rte_07551] ⌈ The *Basic Software Scheduler* shall support *Inter Basic Software Module Entity Triggering*. ⌋*(SRS_Rte_00230)*

Similar to External Trigger Event Communication (described in chapter 4.5.1) the activation of triggered *Basic Software Schedulable Entity* can be implemented by means of activating an Operating System Task or by direct function call.

[SWS_Rte_07552] ⌈ The call of the `SchM_ActMainFunction` API shall activate all `triggered Basic Software Schedulable Entitys` which *BswInternalTriggerOccurredEvents* are associated by the related *activationPoint* of the same a *Basic Software Module* instance if either no queuing for the `BswInternalTriggeringPoint` is configured or if queuing for the `BswInternalTriggeringPoint` is configured and the trigger queue is empty.. ⌋*(SRS_Rte_00230)*

[SWS_Rte_07553] ⌈ The *Basic Software Scheduler* shall support for *Inter Basic Software Module Entity Triggering* that `triggered Basic Software Schedulable Entitys` are invoked via OS Task activation. ⌋*(SRS_Rte_00230)*

[SWS_Rte_07554] ⌈ The *Basic Software Scheduler* shall support for *Inter Basic Software Module Entity Triggering* that `triggered Basic Software Schedulable Entitys` are invoked via direct function call if

- the `triggered Basic Software Schedulable Entitys` do not define a 'minimum start distance'
- if the preconditions of table 4.5 are fulfilled
- no queuing for the `BswInternalTriggeringPoint`is configured

⌋*(SRS_Rte_00230)*

Note: Typically the feature of *Inter Basic Software Module Entity Triggering* is used to decouple the execution context of *Basic Software Entities*. But if this decoupling is really required depends from the particular scheduling concept and microcontroller performance.

4.5.4 Intra ECU Trigger Communication

The trigger communication is also possible in case of intra-ECU communication. In this case, a software component on an ECU can act as a Trigger Source for a software component on another ECU, so requesting the activation of software components on the other ECU.

[SWS_Rte_08409] ⌈ The RTE shall support intra-ECU Trigger Communication. ⌋

[SWS_Rte_08410] ⌈ The RTE shall support the activation of RunnableEntitys occurrence of Trigger Events coming from another ECU. ⌋

[SWS_Rte_08411] ⌈ In case of an issued Trigger the RTE shall send the ISignal to COM. The API call argument of the signal has no meaning. ⌋

[SWS_Rte_08412] ⌈ In case of a received Trigger the RTE shall only care about the COM Notification which indicates a reception of the zero size signal. The value of such signal shall not be read (Com_ReceiveSignal shall not be called). ⌋

[SWS_Rte_08072] ⌈ The RTE generator shall reject configurations violating the [constr_3065]. ⌋*(SRS_Rte_00018)*

4.5.5 Queuing of Triggers

The queuing of triggers ensures that the number of executions of triggered ExecutableEntitys is equal to the number of released triggers. Further on it ensures that the number of activations of triggered ExecutableEntitys is equal for all associated triggered ExecutableEntitys of a Trigger Emitter if the associated triggered ExecutableEntitys are not activated by other RTEEvents. Therefore the trigger queue is rather a counter than a real queue.

[SWS_Rte_07087] ⌈ The RTE shall support the queuing of triggers for

- *External Trigger Event Communication*
- *Inter Runnable Triggering*
- *Inter Basic Software Module Entity Triggering*

if the RteTriggerSourceQueueLength / RteBswTriggerSourceQueueLength is configured > 0. ⌋*(SRS_Rte_00235)*

The attribute swImplPolicy specifies a queued or non queued processing of the Trigger Emitter. Since the setup of a queue might have other side effects on the dynamic behavior of the ECU its still an design decision of the ECU integrator to configure a trigger queue.
Therefore it is possible to configure a trigger queue regardless on the value of the attribute swImplPolicy of the Trigger Emitter.

[SWS_Rte_07088] ⌈ The RTE shall enqueue a trigger when the RTE gets informed about the occurrence of a trigger by the call of the related API (Rte_IrTrigger, Rte_Trigger, SchM_Trigger, SchM_ActMainFunction) if queuing for this Trigger Emitter is configured and if the maximum queue length (RteTrigger-SourceQueueLength / RteBswTriggerSourceQueueLength) is not exceeded. ⌋*(SRS_Rte_00235)*

[SWS_Rte_07089] ⌈ The RTE shall dequeue a trigger when the Trigger Emitter is informed about the end of execution of all triggered ExecutableEntitys which are triggered by this Trigger Emitter. ⌋*(SRS_Rte_00235)*

[SWS_Rte_07090] ⌈ The RTE shall activate all triggered ExecutableEntitys associated to a Trigger Emitter when it has successfully dequeued a trigger from the trigger queue of the Trigger Emitter except for the last dequeued trigger. ⌋*(SRS_Rte_00235)*

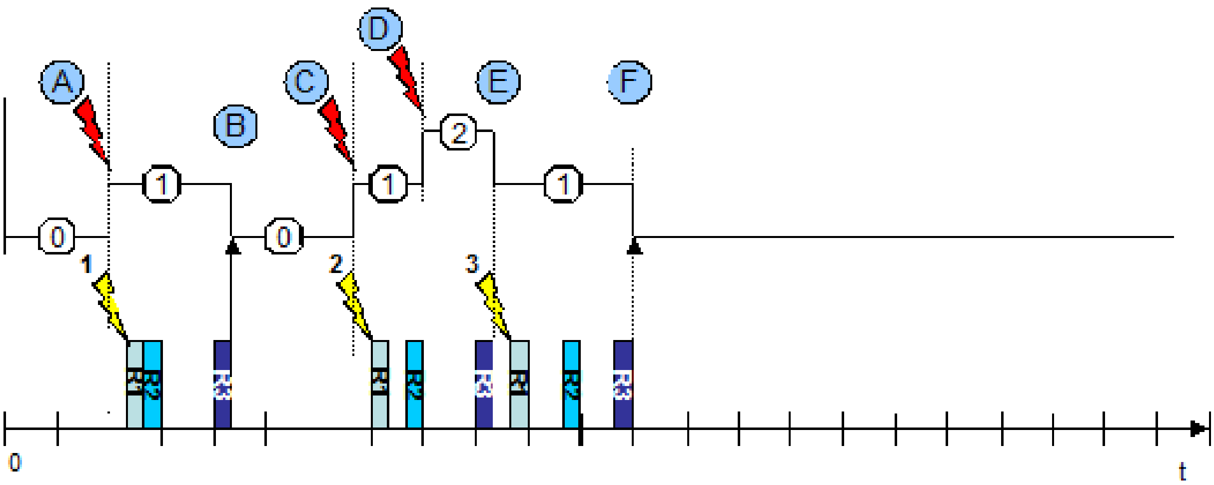


Figure 4.50: Queued activation of ExecutableEntitys

The figure 4.50 illustrates the basic behavior of a trigger queue.

- At "'A"' the RTE gets informed by the call of the API about the occurrence of a Trigger. Since no trigger is in the queue all associated triggered ExecutableEntitys are activated ([SWS_Rte_07544], [SWS_Rte_07555], [SWS_Rte_07552]) and the trigger is enqueued ([SWS_Rte_07088]).
- At "'B"' all triggered ExecutableEntitys which are triggered by this Trigger Emitter have terminated. The RTE dequeues the trigger but since it is the last dequeued trigger the associated triggered ExecutableEntitys are not activated again.
- At "'C"' the RTE gets informed by the call of the API about the occurrence of a Trigger. Enqueuing of triggers and activating of triggered ExecutableEntitys is done as in "'A"'
- At "'D"' the RTE gets informed again by occurrence of a trigger. Since a trigger is already in the queue the associated triggered ExecutableEntitys are

not activated ([SWS_Rte_07544], [SWS_Rte_07555], [SWS_Rte_07552]). Nevertheless the trigger is enqueued ([SWS_Rte_07088]).

- At "'E"' all triggered ExecutableEntitys which are triggered by this Trigger Emitter have terminated. The RTE dequeues the trigger ([SWS_Rte_07089]) and activates all associated triggered ExecutableEntitys ([SWS_Rte_07090]).
- At "'E"' all triggered ExecutableEntitys which are triggered by this Trigger Emitter have terminated. Dequeuing of triggers is done as in "'B"'

Implementation hint:
One possible solution to implement the queue for the number of released triggers is to use the means of the operation systems which already can queue the activation requests for a OS task (OsTaskActivation > 1). This for sure is only possible if all ExternalTriggerOccurredEvents, InternalTriggerOccurredEvents, BswExternalTriggerOccurredEvent and BswInternalTriggerOccurredEvent connected to the same Trigger Emitter with configured queuing are mapped exclusively to one OS task.

4.5.6 Activation of triggered ExecutableEntities

The activation of triggered ExecutableEntitys is done like described in chapter 4.2.3. See also Fig. 4.17.

If the triggered ExecutableEntitys are activated synchronous or asynchronous depends how the *RTEEvent*s and *BswEvent*s are mapped to OS tasks.

If all *ExternalTriggerOccurredEvent*s of the Trigger Sinks which are associated to connected *Trigger* of the Trigger Source

- either are mapped to OS task(s) with higher priority as the OS task where the *Executable Entity* calling the Rte_Trigger respectively the SchM_Trigger API is mapped
- or are activated by direct function call

the triggering behaves synchronous. This means that all "triggered" *Executable Entities* of the Trigger Sinks are executed before the Rte_Trigger or SchM_Trigger API returns.

If any *ExternalTriggerOccurredEvent* of the Trigger Sinks which are associated to connected *Trigger* of the Trigger Source

are mapped to an OS task with lower priority as the OS task where the *Executable Entity* calling the Rte_Trigger respectively the SchM_Trigger API is mapped the triggering behaves asynchronous. This means that **not** all triggered ExecutableEntitys of the Trigger Sinks are executed before the Rte_Trigger or SchM_Trigger API returns.

4.6 Initialization and Finalization

4.6.1 Initialization and Finalization of the RTE

RTE and *Basic Software Scheduler* have a nested life cycle. It is only permitted to initialize the RTE if the *Basic Software Scheduler* is initialized ([constr_9036]). Further on it is only supported to finalize the *Basic Software Scheduler* after the RTE is finalized ([constr_9056]).

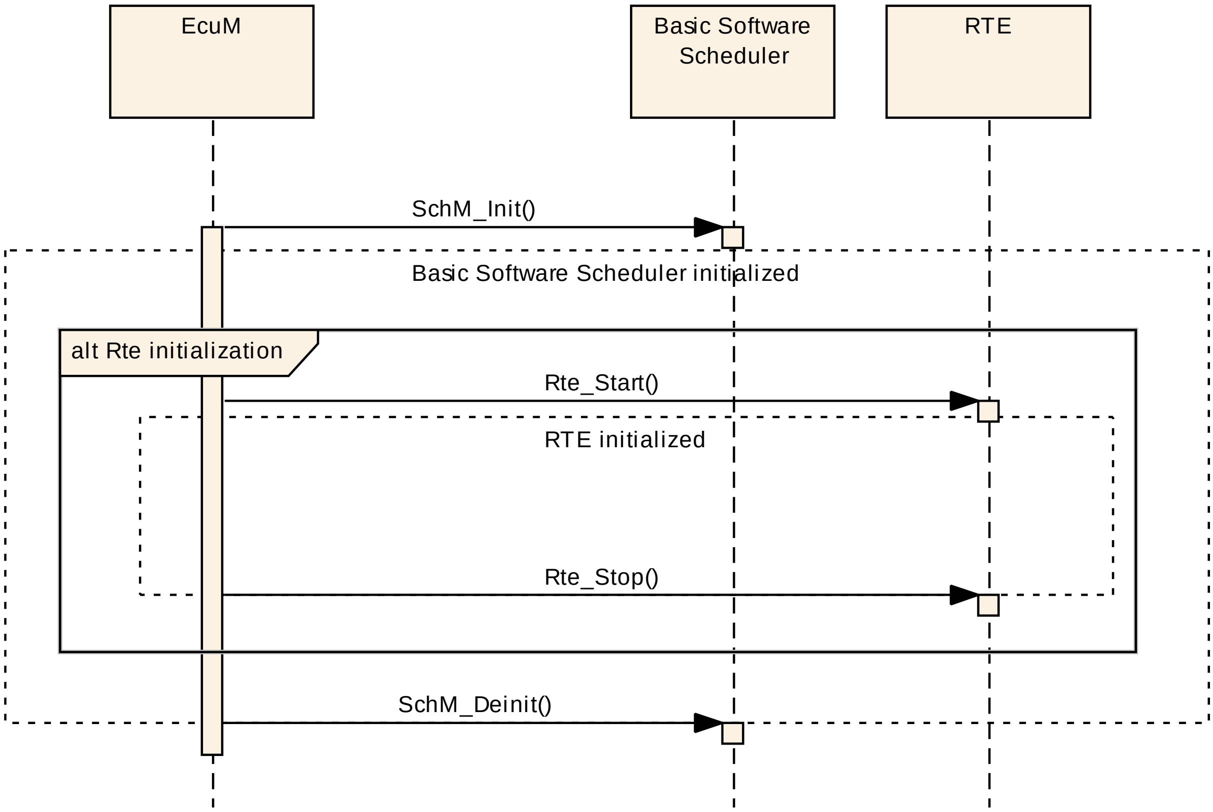


Figure 4.51: Nested life cycle of RTE and *Basic Software Scheduler*

4.6.1.1 Initialization of the Basic Software Scheduler

Before the *Basic Software Scheduler* is initialized only the API calls `SchM_Enter` and `SchM_Exit` are available ([SWS_Rte_07578]).

The ECU state manager calls the startup routine `SchM_Init` of the *Basic Software Scheduler* before any *Basic Software Module* needs to be scheduled.

The initialization routine of the *Basic Software Scheduler* will return within finite execution time (see [SWS_Rte_07273]).

The *Basic Software Scheduler* will initialize the `mode machine instances` ([SWS_Rte_02544])assigned to the *Basic Software Scheduler*. This will activate the `mode disablings` of all initial modes during `SchM_Init` and trigger the execution of the OnEntry ExecutableEntitys of the initial modes. After initialization of the *Basic Software Scheduler* internal data structure and mode machine instances the activation of *Basic Software Schedulable Entities* triggered by BswTimingEvents starts.

[SWS_Rte_07574] ⌈The call of SchM_Init shall start the activation of BswSchedulableEntitys triggered by BswTimingEvents. ⌋*(SRS_Rte_00211)*

[SWS_Rte_07584] ⌈The call of SchM_Init shall start the activation of BswSchedulableEntitys triggered by BswBackgroundEvents. ⌋*(SRS_Rte_00211)*

Note: In case of OS task where BswEvents and RTEEvents are mapped to the RTE Generator has to ensure, that RunnableEntitys are not activated before the RTE is initialized or after the RTE is finalized. See [SWS_Rte_07580] and [SWS_Rte_02538].

[SWS_Rte_07580] ⌈The *Basic Software Scheduler* has to prevent the activation of RunnableEntitys before the RTE is initialized. ⌋*(SRS_Rte_00220)*

4.6.1.2 Initialization of the RTE

The ECU state manager calls the startup routine Rte_Start of the RTE at the end of startup phase II when the OS is available and all basic software modules are initialized.

The initialization routine of the RTE will return within finite execution time (see [SWS_Rte_02585]).

Before the RTE is initialized completely, there is only a limited capability of RTE to handle incoming data from COM:

The RTE will initialize the mode machine instances ([SWS_Rte_02544]) assigned to the RTE. This will activate the mode disablings of all initial modes during Rte_Start and trigger the execution of the OnEntry ExecutableEntitys of the initial modes. Further on for common mode machine instances the *OnEntry Runnable Entities* of the current active mode are executed during the initialization of the RTE ([SWS_Rte_07582]). common mode machine instances can not enter the transition phase during RTE initialization ([SWS_Rte_07583]).

[SWS_Rte_07575] ⌈The call of Rte_Start shall start the activation of RunnableEntitys triggered by TimingEvents if the Rte_StartTiming API does not exist. ⌋*(SRS_Rte_00072)*

[SWS_Rte_07178] ⌈The call of Rte_Start shall start the activation of RunnableEntitys triggered by BackgroundEvents if the Rte_StartTiming API does not exist. ⌋*(SRS_Rte_00072)*

[SWS_Rte_06759] ⌈The call of Rte_StartTiming shall start the activation of RunnableEntitys triggered by TimingEvents if the Rte_StartTiming API does exist. ⌋*(SRS_Rte_00072, SRS_Rte_00240)*

[SWS_Rte_06760] ⌈ The call of Rte_StartTiming shall start the activation of RunnableEntitys triggered by BackgroundEvents if the Rte_StartTiming API does exist. ⌋*(SRS_Rte_00072, SRS_Rte_00240)*

[SWS_Rte_07615] ⌈ The call of Rte_Start shall be executed on every core independently. ⌋

[SWS_Rte_07616] ⌈ The Rte_Start includes the partition specific startup activities of RTE for all partitions that are mapped to the core, from which the Rte_Start is called. ⌋

4.6.1.3 Stop and restart of the RTE

Partitions of the ECU can be stopped and restarted. In a stopped or restarting partition, the OS has killed all running tasks. RTE has to react to stopping and restarting partitions.

The RTE does not execute ExecutableEntitys of a terminated or restarting partition.

[SWS_Rte_07604] ⌈ The RTE shall not activate, start or release ExecutableEntity execution-instances of a terminated or restarting partition. ⌋*(SRS_Rte_00195)*

The RTE is notified of the termination (respectively, the beginning of restart) of a partition by the Rte_PartitionTerminated (respectively, Rte_PartitionRestarting) API. At this point in time, the tasks containing the runnables of this partition are already killed by the OS. In case of restart, RTE is notified by the Rte_RestartPartition API when the communication can be re-initialized and re-enabled.

[SWS_Rte_07604] also applies to ExecutableEntitys whose execution started before the notification to the RTE. RTE can rely on the OS functionality to stop or restart an OS application and all related OS objects.

When a partition is restarted, the RTE will restore an initial environment for its SW-Cs.

[SWS_Rte_02735] ⌈ When the Rte_RestartPartition API for a partition is called, the RTE shall restore an initial environment for its SW-Cs on this partition. ⌋

The SW-Cs themselves are responsible to restore their internal initial environment and should not rely on any initialization performed by the compiler. This should be done in initialization runnables.

[SWS_Rte_07610] ⌈ The RTE Generator shall reject configurations where the handleTerminationAndRestart attribute of a SW-C is not set to canBeTerminatedAndRestarted and this SW-C is mapped on a Partition with the PartitionCanBeRestarted parameter set to TRUE. ⌋*(SRS_Rte_00018, SRS_Rte_00196)*

When a partition is terminated or is being restarted, it is important that the runnable entities of this partition are not activated before the partition returns to the ACTIVE state.

In case of partition restart or termination, event sent to this partition or activation of tasks of this partition are discarded. The RTE can use these mechanism to ensure that ExecutableEntitys are not activated.

4.6.1.4 Finalization of the RTE

The finalization routine Rte_Stop of the RTE is called by the ECU state manager at the beginning of shutdown phase I when the OS is still available. (For details of the ECU state manager, see [7]. For details of Rte_Start and Rte_Stop see section 5.8.)

[SWS_Rte_02538] ⌈ The RTE shall not activate, start or release RunnableEntitys on a core after Rte_Stop has been called on this core. ⌋*(SRS_Rte_00116, SRS_Rte_00220)*

Note: RTE does not kill the tasks during the 'running' state of the runnables.

[SWS_Rte_02535] ⌈ RTE shall ignore incoming client server communication requests, before RTE is initialized completely and when it is stopped. ⌋*(SRS_Rte_00116)*

[SWS_Rte_02536] ⌈ Incoming data and events from sender receiver communication shall be ignored, before RTE is initialized completely and when it is stopped. ⌋*(SRS_Rte_00116)*

4.6.1.5 Finalization of the *Basic Software Scheduler*

The ECU state manager calls the finalization routine SchM_Deinit of the *Basic Software Scheduler* if the scheduling of *Basic Software Modules* has to be stopped.

[SWS_Rte_07586] ⌈ The BSW Scheduler shall neither activate nor start BswSchedulableEntitys on a core after SchM_Deinit has been called on this core. ⌋*(SRS_Rte_00116)*

Note: The BSW Scheduler does not kill the tasks during the 'running' state of the BswSchedulableEntitys.

4.6.2 Initialization and Finalization of AUTOSAR Software-Components

For the initialization and finalization of AUTOSAR software components, RTE provides the mechanism of mode switches. A SwcModeSwitchEvent of an appropriate ModeDeclaration can be used to trigger a corresponding initialization or finalization runnable (see [SWS_Rte_02562]). Runnables that shall not run during initialization or finalization can be disabled in the corresponding modes with a ModeDisablingDependency (see [SWS_Rte_02503]).

Since category 2 runnables have no predictable execution time and can not be terminated using ModeDisablingDependencies, it is the responsibility of the implementer to set meaningful termination criteria for the cat 2 runnables. These criteria could include mode information. At latest, all runnables will be terminated by RTE during the shutdown of RTE, see [SWS_Rte_02538].

It is appropriate to use user defined modes that will be handled in a proprietary application mode manager.

All runnables that are triggered by entering an initial mode, are activated immediately after the initialization of RTE. They can be used for initialization. In many cases it might be preferable to have a multi step initialization supported by a sequence of different initialization modes.

In addition to the mode-based approach RunnableEntitys to be used for initialization purposes can be activated by InitEvents as well. More information is provided in section 4.2.2.11.

4.7 Variant Handling Support

4.7.1 Overview

The *AUTOSAR Template*s support the creation of *Variant*s in a subset of its model elements. The *Variant Handling* support in the in *AUTOSAR Template*s is driven by the purpose to describe variability in a *AUTOSAR System* on several aspects, e.g.

- Virtual Functional Bus
- Component `SwcInternalBehavior` and `SwcImplementation`
- Deployment of the software components to ECUs
- Communication Matrix
- Basic Software Modules

This approach requires that the RTE Generator is able to process the described Variability in input configurations and partially to implement described variability in the generated RTE and Basic Software Scheduler code.

In the meta-model all locations that may exhibit variability are marked with the stereotype ≪`atpVariation`≫. This allows the definition of possible variation points. Tagged Values are used to specify additional information.

There are four types of locations in the meta-model which may exhibit variability:

- Aggregations
- Associations
- Attribute Values
- Classes providing property sets

More details about the AUTOSAR Variant Handling Concept can be found in the AUTOSAR Generic Structure Template [10].

[SWS_Rte_06543] ⌈The RTE generator shall support the `VariationPoint`s defined in the *AUTOSAR Meta Model* ⌋*(SRS_Rte_00201, SRS_Rte_00202, SRS_Rte_00229, SRS_Rte_00191)*

The list of `VariationPoint`s shall provide an overview about the most prominent ones which impacting the generated RTE code. Further on tables will show which implementation of variability is standardized due to the relevance for contract phase. (see tables 4.15, 4.17, 4.18, 4.19, 4.20, 4.21, 4.25, 4.26, 4.28 and 4.29. But please note that these tables are not listing all possible variation of the input configuration. For that the related Template Specifications are relevant.

4.7.2 Choosing a Variant and Binding Variability

To understand the later definition it is required to clarify the difference between *Choosing a Variant* and *Resolving Variability*.

A particular *PreBuild Variant* in a variant rich input configuration is chosen by assigning particular values to the `SwSystemconst`s with the means of `PredefinedVariant`s and associated `SwSystemconstantValueSet`s. With this information `SwSystemconstDependentFormula`s can be evaluated which determines PreBuild conditions of `VariationPoint`s and attribute values. Nevertheless the input configuration contains still the information of all potential variants.

A particular *PostBuild Variant* in a variant rich input configuration is chosen by assigning particular values to the `PostBuildVariantCriterion` with the means of `PredefinedVariant`s and associated `PostBuildVariantCriterionValueSet`s. With this information `PostBuildVariantCondition`s can be evaluated for instance to check the consistency of chosen *PostBuild Variant*. Nevertheless the input configuration contains still the information of all potential variants.

From an RTE perspective this information is mainly used to generate the *RTE Post Build Variant Sets* which are used to bind the `PostBuild Variability` during initialization of the RTE (call of `SchM_Init`).

The variability of an input configuration is bound if information related to other variants is removed and only the information of the bound variant is kept. Binding respectively resolving variability in the scope of this specification means that the generated code only implements the particular variant which results out of the chosen variant of the input configuration.

If the variability can not be resolved in a particular phase of the *RTE Generation Process* (see chapter 3) the generated RTE files have to be able to support the potential variants by implementing all potential variants.

If the variability is relevant for the software components contract the RTE Generator uses standardized *Condition Value Macros* to implement the `PreBuild Variability`. These *Condition Value Macros* are set in the *RTE PreBuild Data Set Contract Phase* and *RTE PreBuild Data Set Generation Phase* to the resulting value of the evaluated `ConditionByFormula` of the related `VariationPoint`.

For further definition see sections 4.7.2.3, 4.7.2.4, 4.7.2.5, 4.7.2.6 and 4.7.2.7.

4.7.2.1 General impact of Binding Times on RTE generation

In the AUTOSAR meta-model, each `VariationPoint` is associated with a tag named `vh.latestBindingTime`. The value of the tag yields the applicable latest binding time for the given `VariationPoint`.

Each `VariationPoint` with a `swSyscond` has an attribute `bindingTime` in its `ConditionByFormula`, which defines when the pre-build condition may be evaluated earliest for this VariationPoint. This controls the capability of the software implementation to bind the variant earliest at a certain point of time.

Even if the variability is chosen earlier (for instance by assigning SwSystemconstValues to the SwSystemconsts used by the VariationPoint's condition) the RTE generator has to respect potential later binding of the VariationPoints.

Please note that variability with the bindingTime PreCompileTime and postBuildVariantConditions has a particular semantic for the RTE generation and impacts the generated output.

For instance a conditional existence RTE API which is bound at PreCompileTime requires that the RTE generator inserts specific pre processor statements.

RTE Phase	System Designe Time	Code Generation Time	Pre Compile Time	Link Time	Post Build
RTE Contract Phase	R	R	I	n/a	n/a
Basic Software Scheduler Contract Phase	R	R	I	n/a	n/a
RTE PreBuild Data Set Contract Phase	n/a	n/a	RV	n/a	n/a
Basic Software Scheduler Generation Phase	R	R	I	n/a	I
RTE Generation Phase	R	R	I	n/a	I
RTE PreBuild Data Set Generation Phase	n/a	n/a	RV	n/a	n/a
RTE PostBuild Data Set Generation Phase	n/a	n/a	n/a	n/a	RV

Table 4.14: Overview impact of Binding Times on RTE generation

R	resolve variability, a particular variant is the output
I	implement variability, all possible variants in the output
RV	provide values to resolve implemented variability *PreBuild* or *PostBuild*
n/a	not applicable

4.7.2.2 Choosing a particular variant

A particular variant of the variant rich input configuration is chosen via the ECU configuration For that purpose a set of PredefinedVariants is configured to chosen a variant in the input configuration and to later on bind the variability in subsequent phases of the *RTE Generation Process* 3. For further information see document [10].

[SWS_Rte_06500] ⌈ For each PreBuild Variability in the input configuration the RTE Generator shall choose a particular variant according to the PredefinedVariants selected by the parameter EcucVariationResolver. ⌋*(SRS_Rte_00201, SRS_Rte_00202, SRS_Rte_00229, SRS_Rte_00191)*

[SWS_Rte_06546] ⌈ For each PostBuild Variability in the input configuration the RTE Generator shall choose a particular variant according to the PredefinedVariants selected by the parameter RtePostBuildVariantConfiguration. ⌋*(SRS_Rte_00201, SRS_Rte_00202, SRS_Rte_00229, SRS_Rte_00191)*

Having variants chosen the RTE generator can apply further consistency checks on the particular variants.

4.7.2.3 SystemDesignTime

Variability with latest binding time SystemDesignTime (called SystemDesignTime Variability) has to be bound before the *RTE Contract Phase* respectively *Basic Software Scheduler Contract Phase*. Such variability is resolved by RTE generator in all generation phases. Due to that such kind of variability results always in a particular variant and needs no special code generation rules for RTE generator.

[SWS_Rte_06501] ⌈ The RTE generator shall bind SystemDesignTime Variability in the *RTE Contract Phase*, *Basic Software Scheduler Contract Phase*, *RTE Generation Phase* and *Basic Software Scheduler Generation Phase* (3). ⌋*(SRS_Rte_00191)*

[SWS_Rte_06502] ⌈ The RTE Generator shall reject input configurations during the *RTE Contract Phase* where not a particular variant is chosen for each SystemDesignTime Variability affecting the software components contract. ⌋*(SRS_Rte_00201, SRS_Rte_00018)*

[SWS_Rte_06503] ⌈ The RTE Generator shall reject input configurations during the *Basic Software Scheduler Contract Phase* where not a particular variant is chosen for each SystemDesignTime Variability affecting the *Basic Software Scheduler* contract. ⌋*(SRS_Rte_00229, SRS_Rte_00018)*

[SWS_Rte_06504] ⌈ The RTE Generator shall reject input configurations during the *Basic Software Scheduler Generation Phase* where not a particular variant is chosen for each SystemDesignTime Variability affecting the *Basic Software Scheduler* generation. ⌋*(SRS_Rte_00229, SRS_Rte_00018)*

[SWS_Rte_06505] ⌈ The RTE Generator shall reject input configurations during the *RTE Generation Phase* where not a particular variant is chosen for each SystemDesignTime Variability affecting the *RTE* generation. ⌋*(SRS_Rte_00201, SRS_Rte_00202, SRS_Rte_00018)*

4.7.2.4 CodeGenerationTime

During *RTE Contract Phase*, *RTE Generation Phase* and *Basic Software Scheduler Generation Phase* the variability with latest binding time *CodeGenerationTime* (called CodeGenerationTime Variability) has to be bound and the RTE generator resolves the variability. This denotes that the code is generated for a particular variant. To do this it is required that a particular variant for each CodeGenerationTime Variability has to be chosen.

[SWS_Rte_06507] ⌈ The RTE generator shall bind CodeGenerationTime Variability in the *RTE Contract Phase*, *Basic Software Scheduler Contract Phase*, *RTE Generation Phase* and *Basic Software Scheduler Generation Phase* (see sections 3.1.1, 3.1.2, 3.4.1 and 3.4.2). ⌋*(SRS_Rte_00229, SRS_Rte_00191)*

[SWS_Rte_06547] ⌈ The RTE Generator shall reject input configurations during the *RTE Contract Phase* where not a particular variant is chosen for each CodeGenerationTime Variability affecting the software components contract. ⌋*(SRS_Rte_00191, SRS_Rte_00018)*

[SWS_Rte_06548] ⌈ The RTE Generator shall reject input configurations during the *Basic Software Scheduler Contract Phase* where not a particular variant is chosen for each CodeGenerationTime Variability affecting the *Basic Software Scheduler* contract. ⌋*(SRS_Rte_00229, SRS_Rte_00018)*

[SWS_Rte_06508] ⌈ The RTE Generator shall reject input configurations during the *Basic Software Scheduler Generation Phase* where not a particular variant is chosen for each CodeGenerationTime Variability affecting the *Basic Software Scheduler* generation. ⌋*(SRS_Rte_00229, SRS_Rte_00018)*

[SWS_Rte_06509] ⌈ The RTE Generator shall reject input configurations during the *RTE Generation Phase* where not a particular variant is chosen for each CodeGenerationTime Variability affecting the *RTE* generation. ⌋*(SRS_Rte_00191, SRS_Rte_00018)*

4.7.2.5 PreCompileTime

Variability with latest binding time *PreCompileTime* (called PreCompileTime Variability) is relevant for the *RTE Contract Phase* and *Basic Software Scheduler Contract Phase* as well as for the *RTE Generation Phase* and *Basic Software Scheduler Generation Phase*. The *Application Header File*, *Application Types Header File*, *Module Interlink Header* and *Module Interlink Types Header* and the generated RTE / *Basic Software Scheduler* has to support the potential variability of the software components and *Basic Software Modules*. The variability is resolved during the execution of the pre processor of the C-Complier.

[SWS_Rte_06510] ⌈ The RTE generator shall implement PreCompileTime Variability in the *RTE Contract Phase*, *Basic Software Scheduler Contract Phase*, *RTE Generation Phase*, *Basic Software Scheduler Generation Phase* via pre processor

statements in the generated RTE code (see sections 3.1.1, 3.1.2, 3.4.1 and 3.4.2).
⌋(*SRS_Rte_00191*)

[SWS_Rte_06553] ⌈ The RTE Generator shall use the defined *Attribute Value Macro* instead of immediate values if the value depends on an AttributeValueVariationPoint where the bindingTime is set to preCompileTime.
⌋(*SRS_Rte_00191*)

4.7.2.6 LinkTime

The latest Binding Time *LinkTime* will not be supported for *VariationPoint*s relevant for the RTE Generator.

[SWS_Rte_06511] ⌈ The RTE generator shall reject configuration which defines RTE or *Basic Software Scheduler* relevant LinkTime Variability. ⌋(*SRS_Rte_00018*)

4.7.2.7 PostBuild

Variability with latest binding time *PostBuild* (called PostBuild Variability) might be bound / rebound after the generated RTE is compiled and has been linked to the executable. The generated RTE binary code has to contain all variants. Which variant is executed during ECU runtime is decided by variant selectors.

[SWS_Rte_06512] ⌈ The RTE generator shall implement PostBuild Variability in the *RTE Generation Phase* and *Basic Software Scheduler Generation Phase* via C statements in the generated RTE code (see 3.4.1 and 3.4.2). ⌋(*SRS_Rte_00191*)

Combining PreBuild and PostBuild Variability

According document [10] it is supported that a VariationPoint defines a PreBuild Variability in conjunction with PostBuild Variability. If the *PreBuild condition* is false, it is not expected that the element which is subject to variability including the code evaluating the *PostBuild condition* gets implemented at all.

[SWS_Rte_06549] ⌈ In cases where a VariationPoint defines a SystemDesignTime Variability or CodeGenerationTime Variability in conjunction with PostBuild Variability the PostBuild Variability shall only be implemented by the RTE Generator in the generated RTE code if the condition of the PreBuild Variability evaluates to true. ⌋(*SRS_Rte_00191*)

[SWS_Rte_06550] ⌈ In cases where a VariationPoint defines a PreCompileTime Variability in conjunction with PostBuild Variability the PostBuild Variability shall only be effective in the RTE executable if the condition of the PreCompileTime Variability evaluates to true. ⌋(*SRS_Rte_00191*)

In this case the PostBuild Variability implemented according [SWS_Rte_06512] depends from the PreCompileTime Variability implemented according [SWS_Rte_06510].

4.7.3 Variability affecting the RTE generation

4.7.3.1 Software Composition

This section describes the affects of the existence of variation points with regards to compositions. Though the application software compositions have been flattened and effectively eliminated after allocation to an ECU there is still one composition to consider for the RTE (i.e. the RootSwCompositionPrototype). The RootSwCompositionPrototype contains the atomic software components allocated to the respective ECU, its assembly connections,its delegation connections and the connections of the delegation ports to system signals. Once the variability is resolved for a variation point it must adhere to the constraints and limitations that apply to a model that does not have any variations. For example dangling connectors are not allowed and as such their existence will lead to undefined behavior if such configurations still exist after resolving post-build variation points.

Also within this specification section the wording "'a variant is enabled or disabled"' refers to the variation point's *SwSystemconstDependentFormula* and/or *PostBuildVariantCondition* evaluating to "'true or false"' respectively.

4.7.3.1.1 Variant existence of SwComponentPrototypes

[SWS_Rte_06601] ⌈ If a variant is disabled for the aggregation of a SwComponentPrototype in a CompositionSwComponentType then all RTEEvents destined for Runnables in the respective SwComponentPrototype shall be blocked; No RTEEvent is allowed to reach any Runnable that is contained in a "'disabled"' SwComponentPrototype. ⌋*(SRS_Rte_00206, SRS_Rte_00207, SRS_Rte_00204)*

Potential misconfigurations of connectors connecting to ports of "'disabled"' SWC's will result in undefined behavior; It is the responsibility of the person considering the variability of the SwComponentPrototype to make the connections also variable and valid when a variant selection results in the elimination of a SwComponentPrototype from a composition. It is recommended to use predefined variants to ensure proper configurations are established.

4.7.3.1.2 Variant existence of SwConnectors

[SWS_Rte_06602] ⌈ If a variant is disabled for a SwConnector (i.e. AssemblySwConnector or DelegationSwConnector) aggregated in a CompositionSwComponentType then the PortPrototypes at each end of the connector shall behave as an unconnected port (see section 5.2.7 for the defined RTE behavior) **if** no other variant enables a SwConnector between these ports. ⌋*(SRS_Rte_00206, SRS_Rte_00207)*

4.7.3.1.3 COM related Variant existence

This section describes the impact on the RTE interaction with the COM layer as a result of variability of DataMappings (i.e. SenderReceiverToSignalMapping and SenderReceiverToSignalGroupMapping in the SystemMapping) as well as the existence of variants for ISignals The Meta Model allows for mapping the same data to different SystemSignals as well as associating a SystemSignal with 1 or more ISignals.

[SWS_Rte_06603] ⌈ If a variant is enabled for a SystemMapping aggregating a DataMapping then the RTE shall call the appropriate API's for the applicable mapping type. ⌋*(SRS_Rte_00206, SRS_Rte_00207)*

[SWS_Rte_06604] ⌈ The appropriate API shall be determined based on the existence of variants of ISignals to which a SystemSignal is associated to. For each enabled ISignal the RTE shall call the proper COM API to send and receive data SystemSignals ⌋*(SRS_Rte_00206, SRS_Rte_00207)*

For example for an instance mapping from a VariableDataPrototype to a SystemSignal the RTE shall call the corresponding Com_SendSignal with the proper *SignalId* and *SignalDataPtr* based on the selected variant DataMapping.

The existence of variants of ISignals is determined by the System element (see also [constr_3028]).

[SWS_Rte_06605] ⌈ Delegation ports on a RootSwCompositionPrototype for which no DataMapping exists (i.e. no variant DataMapping is enabled) shall be considered unconnected because no path exists to a designated SystemSignal. Since this is a delegation port all enabled delegation connectors linking SWC R-ports to the respective delegation port must be considered unconnected (see section 5.2.7). P-Ports shall behave as documented in section 4.7.3.1.2. ⌋*(SRS_Rte_00206, SRS_Rte_00207)*

4.7.3.1.4 Variant existence of *PortPrototypes*

[SWS_Rte_06606] ⌈ If no variant is enabled for a delegation port on a RootSwCompositionPrototype then all connected R-Ports using a DelegationSwConnector to this delegation port shall be considered unconnected (see section 5.2.7). The behavior of the P-ports shall be as defined in section 4.7.3.1.2. ⌋*(SRS_Rte_00206, SRS_Rte_00207)*

Note on variant disabling criteria: In a proper variant configuration the following should be followed: when a PortPrototype is eliminated from any SwComponentType then any associated SwConnector should also have a variation point removing the connection since the connection is illegal.

4.7.3.2 Atomic Software Component and its Internal Behavior

4.7.3.2.1 RTE API which is subject to variability

Following VariationPoints in the Meta Model do control the variant existence of RTE API for a software component. If a RTE API is variant existent, the API mapping and the related entries in the component data structure are 'variant' as well. This means, if a RTE API does not exist the API mapping does not exist as well. A part of the component data structure entries are related to the existences of the port. In these cases the *component data structure entry* depends from the existence of the PortPrototype.

Variation Point Condition Value Macro	RTE API which is subject to variability	form	kind infix
ExclusiveArea [SWS_Rte_06518]	Rte_Enter, Rte_Exit	component internal	ExAr
VariableDataPrototype in the role arTypedPerInstanceMemory [SWS_Rte_06518]	Rte_Pim	component internal	PIM
PerInstanceMemory [SWS_Rte_06518]	Rte_Pim	component internal	PIM
ParameterDataPrototype in the role perInstanceParameter [SWS_Rte_06518]	Rte_CData	component internal	Prm
ParameterDataPrototype in the role sharedParameter [SWS_Rte_06518]	Rte_CData	component internal	Prm
ServerCallPoint [SWS_Rte_06515]	Rte_Call	component port	
AsynchronousServerCallResultPoint [SWS_Rte_06515]	Rte_Result	component port	
InternalTriggeringPoint [SWS_Rte_06519]	Rte_IrTrigger	entity internal	IRT
ExternalTriggeringPoint [SWS_Rte_06515]	Rte_Trigger	component port	
ModeSwitchPoint [SWS_Rte_06515]	Rte_Switch, Rte_SwitchAck	component port	
ModeAccessPoint	Rte_Mode	component port	

[SWS_Rte_06515]			
VariableAccess in the role dataReadAccess [SWS_Rte_06515]	Rte_IRead , Rte_IStatus, Rte_IsUpdated	entity port	
VariableAccess in the role dataWriteAccess [SWS_Rte_06515]	Rte_IWrite, Rte_IWriteRef, Rte_IInvalidate, Rte_IFeedback	entity port	
VariableAccess in the role dataSendPoint [SWS_Rte_06515]	Rte_Write, Rte_Invalidate, Rte_Feedback	component port	
VariableAccess in the role dataReceive-PointByArgument [SWS_Rte_06515]	Rte_Read	component port	
VariableAccess in the role dataReceive-PointByValue [SWS_Rte_06515]	Rte_DRead	component port	
VariableAccess in the role readLocalVari-able referring an explicitInterRunnable-Variable [SWS_Rte_06518]	Rte_IrvRead	component internal	IRV
VariableAccess in the role writtenLo-calVariable referring an explicitInter-RunnableVariable [SWS_Rte_06518]	Rte_IrvWrite	component internal	IRV
VariableAccess in the role readLocalVari-able referring an implicitInterRunnable-Variable [SWS_Rte_06519]	Rte_IrvIRead	entity internal	IRV
VariableAccess in the role writtenLo-calVariable referring an implicitInter-RunnableVariable [SWS_Rte_06519]	Rte_IrvIWrite	entity internal	IRV
PortPrototype referring a ParameterInter-face [SWS_Rte_06515]	Rte_Prm	component port	
PortAPIOption with attribute indirectAPI [SWS_Rte_06520]	Rte_Port		

Table 4.15: variant existence of RTE API

column	**description**
kind infix	The column kind infix defines infix strings to differentiate condition value macros belonging to variation points of different API sets
form	The column form specifies which names for the macro of the condition value are concatenated to ensure a unique name space of the macro.

form	description
component port	The related API is provide for the whole software component and belongs to a software components port
entity port	The related API is provide for a particular RunnableEntity and belongs to a software components port
component internal	The related API is provide for the whole software component and belongs to a software component internal functionality
entity internal	The related API is provide per RunnableEntity and belongs to a software component internal functionality

Table 4.16: Key to table 4.15

[SWS_Rte_06517] ⌈ The RTE generator shall treat RTE API as variant RTE API only if all elements (e.g. VariableAccess) in the input configuration controlling the existence of the same RTE API are subject to variability. ⌋*(SRS_Rte_00203)*

4.7.3.2.2 Conditional API options

Following variation points in the Meta Model do control the variant properties of RTE API or allocated Memory.

Variation Point Condition Value Macro	Subject to variability
PortAPIOption with attribute portArgValue not standardized	PortDefinedArgumentValue is passed to a RunnableEntity
PortAPIOption with attribute indirectAPI not standardized	Number of Ports which are supporting indirect API, see Rte_NPorts and Rte_Ports

Table 4.17: Conditional API options

4.7.3.2.3 Runnable Entity's and RTEEvents

Following variation points in the Meta Model do control the variant existence and activation of RunnableEntitys.

Variation Point Condition Value Macro	Subject to variability
RunnableEntity [SWS_Rte_06530]	Existence of the RunnableEntity prototype
RTEEvent	Activation of the RunnableEntity

not standardized	

Table 4.18: variation on Runnable Entity's and RTEEvents

4.7.3.2.4 Conditional Memory Allocation

Following variation points in the Meta Model do control the variant existence of RTE memory allocation for the software component instance.

Variation Point Condition Value Macro	Subject to variability
implicitInterRunnableVariable not standardized	variable definition implementing the implicitInterRunnableVariable
explicitInterRunnableVariable not standardized	variable definition implementing the explicitInterRunnableVariable
arTypedPerInstanceMemory not standardized	variable definition implementing the arTypedPerInstanceMemory
PerInstanceMemory not standardized	variable definition implementing the PerInstanceMemory
perInstanceParameter not standardized	constant definition implementing the perInstanceParameter
sharedParameter not standardized	variable definition implementing the sharedParameter
InstantiationDataDefProps, SwDataDefProps not standardized	Allocation of the memory objects described via swAddrMethod, accessibility for MCD systems described via swCalibrationAccess, displayFormat, mcFunction

Table 4.19: Conditional Memory Allocation

4.7.3.3 NvBlockComponent and its Internal Behavior

Variation Point Condition Value Macro	Subject to variability

PortPrototype of a NvBlockSwComponentType typed by NvDataInterface not standardized	Existence of the ability to access the memory objects of the ramBlock
NvBlockDataMapping of a NvBlockDescriptor not standardized	Existence of the ability to access the memory objects of the ramBlock
provide PortPrototype of a NvBlockSwComponentType typed by ClientServerInterface, RunnableEntity and referring OperationInvokedEvent not standardized	Existence of the *Block Management* port and the ability to access the *Block Management* API of the *NvRAM Manager*
require PortPrototype of a NvBlockSwComponentType typed by ClientServerInterface, RoleBasedPortAssignment and referring the PortPrototype not standardized	Existence of the *callback notification* port
NumericalValueSpecification or TextValueSpecification of the ramBlock or romBlocks initValue ValueSpecification (aggregated or referred one) not standardized	initialization values of the memory objects implementing the ramBlock or romBlock
InstantiationDataDefProps not standardized	Allocation of the memory objects implementing the ramBlock or romBlock described via swAddrMethod, accessibility for MCD systems described via swCalibrationAccess, displayFormat, mcFunction

Table 4.20: variation in NvBlockSwComponentTypes

4.7.3.4 Parameter Component

Variation Point Condition Value Macro	Subject to variability
PortPrototype of a ParameterSwComponentType not standardized	Existence of the memory objects / definitions related to the ParameterDataPrototypes in the PortInterface referred by the PortPrototype
NumericalValueSpecification or TextValueSpecification of the ParameterProvideComSpecs initValue ValueSpecification (aggregated or referred one) not standardized	initialization values of the memory objects / definitions related to the ParameterDataPrototypes

Table 4.21: variation in ParameterSwComponentTypes

4.7.3.5 Data Type

Following variation points in the Meta Model do control the variant generation of data types.

Variation Point Condition Value Macro	Subject to variability
ImplementationDataTypeElement [SWS_Rte_06542]	Existence of the structure or union element
arraySize [SWS_Rte_06541]	Number of elements in the array
CompuMethod upperLimit	Upper limit of the ImplementationDataType
CompuMethod lowerLimit	Lower limit of the ImplementationDataType
CompuMethod v attributes	Coefficients of nominator and denominator

Table 4.22: variation in ImplementationDataTypes

Variation Point Condition Value Macro	Subject to variability
DataConstr upperLimit [SWS_Rte_06551]	Upper limit of the ApplicationPrimitiveDataType
DataConstr lowerLimit [SWS_Rte_06552]	Lower limit of the ApplicationPrimitiveDataType
CompuMethod upperLimit	Upper limit of the ApplicationPrimitiveDataType
CompuMethod lowerLimit	Lower limit of the ApplicationPrimitiveDataType
CompuMethod v attributes	Coefficients of nominator and denominator

Table 4.23: variation in ApplicationDataTypes and related meta classes

4.7.3.6 Constants

Variation Point Condition Value Macro	Subject to variability

NumericalValueSpecification value	numerical value
ApplicationValueSpecification v (swArraysize)	size of compound primitives
ApplicationValueSpecification v (value) attributes	physical value

Table 4.24: variation in ValueSpecifications

4.7.3.7 Basic Software Modules and its Internal Behavior

4.7.3.7.1 Basic Software Interfaces

Variation Point Condition Value Macro	Subject to variability
providedEntry not standardized	Existence of the provided BswModuleEntry
outgoingCallback not standardized	Existence of the expected BswModuleEntry
ModeDeclarationGroupPrototype in role providedMode-Group not standardized	Existence of the provided ModeDeclarationGroup-Prototype
ModeDeclarationGroupPrototype in role requiredMode-Group not standardized	Existence of the required ModeDeclarationGroup-Prototype
Trigger in role releasedTrigger not standardized	Existence of the released Trigger
Trigger in role requiredTrigger not standardized	Existence of the required Trig-ger

Table 4.25: variability affecting *Basic Software Interfaces*

4.7.4 Variability affecting the Basic Software Scheduler generation

4.7.4.1 Basic Software Scheduler API which is subject to variability

The VariationPoints listed in table 4.26 in the input configuration are controlling the variant existence of *Basic Software Scheduler* API.

Variation Point Condition Value Macro	Subject to variability	form	kind infix
ExclusiveArea [SWS_Rte_06535]	SchM_Enter, SchM_Exit	module internal	ExAr

managedModeGroup association to providedModeGroup ModeDeclarationGroupPrototype [SWS_Rte_06536]	SchM_Switch, SchM_SwitchAck	module external	MMod
accessedModeGroup association to providedModeGroup or requiredModeGroup ModeDeclarationGroupPrototype [SWS_Rte_06536]	SchM_Mode	module external	AMod
issuedTrigger association to releasedTrigger Trigger [SWS_Rte_06536]	SchM_Trigger	module external	Tr
BswModuleCallPoint [SWS_Rte_06536]	SchM_Call	module external	SrvCall
BswAsynchronousServerCallResultPoint [SWS_Rte_06536]	SchM_Result	module external	SrvRes
dataSendPoint association to providedData [SWS_Rte_06536]	SchM_Send	module external	DSP
dataReceivePoint association to requiredData [SWS_Rte_06536]	SchM_Receive	module external	DRP
BswInternalTriggeringPoint [SWS_Rte_06536]	SchM_ActMainFunction	entity internal	ITr
perInstanceParameter ParameterDataPrototype [SWS_Rte_06535]	SchM_CData	module internal	PIP

Table 4.26: variant existence of Basic Software Scheduler API

column	description
kind infix	The column kind infix defines infix strings to differentiate condition value macros belonging to variation points of different API sets
form	The column form specifies which names for the macro of the condition value are concatenated to ensure a unique name space of the macro.

form	description
module external	The related API is provide for the whole module and belongs to a module interface
module internal	The related API is provide for the whole module and belongs to a module internal functionality
entity internal	The related API is provide per ExecutableEntity and belongs to a module internal functionality

Table 4.27: Key to table 4.26

[SWS_Rte_06537] ⌈ The RTE generator shall treat the existence of *Basic Software Scheduler* API as subject to variability only if all elements (e.g. managedModeGroup association) in the input configuration controlling the existence of the same *Basic Software Scheduler* API are subject to variability. ⌋*(SRS_Rte_00229)*

4.7.4.2 Basic Software Entities

The VariationPoints listed in table 4.28 in the input configuration are controlling the variant existence of BswModuleEntitys and the variant activation of BswSchedulableEntitys.

Variation Point Condition Value Macro	Subject to variability
BswSchedulableEntity [SWS_Rte_06532]	Existence of the BswSchedulableEntity prototype
BswEvent not standardized	Activation of the BswSchedulableEntity

Table 4.28: variability affecting BswSchedulableEntitys

4.7.4.3 API behavior

The VariationPoints listed in table 4.29 in the input configuration are controlling the variant behavior of *Basic Software Scheduler* API.

Variation Point Condition Value Macro	Subject to variability
BswModeSenderPolicy not standardized	Queue length in the mode machine instance dependent from the attribute
BswModeReceiverPolicy not standardized	attribute supportsAsynchronousModeSwitch has to be considered according the bound variant

Table 4.29: variant existence of BswSchedulableEntity

4.7.5 Variability affecting SWC implementation

In this section some examples will be given in order to describe the affects of variability with regard to SWC implementation. The implemented variability in SWCs is described through VariationPointProxys and can be resolved by pre-build evaluation, by post-build evaluation or by the combination of them. Furthermore for each VariationPointProxy AUTOSAR defines the categorys VALUE and CONDITION (see Software Component Template [2]). In the following code examples one scenario for each category will be described. The first scenario addresses the post-build case and the second one the case of combination of pre-build and post-build.

Scenario for category VALUE

VariationPointProxy FRIDA
postBuildValueAccess Rte_PBCon_FRIDA = 3
might result for example in something like:

```
/* Generated RTE-Code */

const Rte_PBCon_FRIDA 3
```

```
/* SWC-Code */

if (Rte_PBCon_FRIDA == 3) {
     /* code depending on proxy FRIDA */
     }
else {
     /* functional alternative, if FRIDA is not selected */
     }
```

Scenario for category CONDITION

SystemConstant FRANZ = 10
VariationPointProxy HUGO
conditionAccess Rte_SysCon_HUGO = (FRANZ == 10)
postBuildVariantCondition A = 3, postBuildVariantCondition B = 5
might result for example in something like:

```
/* Generated RTE-Code */

#define Rte_SysCon_HUGO 1

#define Rte_PBCon_HUGO (
     Rte_SysCon_HUGO  &&
     RteInternal_EvalPostBuildVariantCondition_HUGO_A &&
     RteInternal_EvalPostBuildVariantCondition_HUGO_B
     )
```

```
/*SWC-Code*/

/* ensure that no code for HUGO remains in
   the binary, if HUGO is not selected */
#if Rte_SysCon_HUGO

/* check during run time, if HUGO is
   active due to post-build conditions */
if (Rte_PBCon_HUGO) {
     /* code depending on proxy HUGO */
     }
else {
```

```
        /* functional alternative, if HUGO is not selected */
        }

#else
/* functional alternative is always
   active since HUGO is not selected */
#endif
```

Since the post-build data structure is not standardized the algorithm for the evaluation of the expressions `RteInternal_EvalPostBuildVariantCondition_HUGO_A` and `RteInternal_EvalPostBuildVariantCondition_HUGO_B` is up to the implementer.

In contrast to `Rte_SysCon` the `Rte_PBCon` API has no guarantee, that it can be resolved in the pre-processor. It is subject to the optimization of the compiler to reduce code size. If one wants to be absolutely sure, that no superfluous code exists even with non optimizing compilers, he needs to implement a pre-processor directive in addition (see example).

4.8 Development errors

Errors which can occur at runtime in the RTE are classified as development errors. The RTE uses a BSW module report these types of errors to the DET [23] (Development Error Tracer).

4.8.1 DET Report Identifiers

[SWS_Rte_06630] ⌈ The RTE shall report development errors to the DET and use its assigned module identifier (i.e. 2) to identify itself to the DET. ⌋*(SRS_BSW_00337)*

[SWS_Rte_07676] ⌈ Development errors shall be reported to the DET if and only if `RteDevErrorDetect` is enabled. ⌋*(SRS_BSW_00337)*

[SWS_Rte_06631] ⌈ The RTE shall use the OS Application Identifier as the Instance Id to enable the developer to identify in which runtime section of the RTE the error occurs. This Instance ID is even unique across multi cores and so implicitly allows the development error to be traced to a specific core. ⌋*(SRS_BSW_00337)*

[SWS_Rte_06632] ⌈ The RTE shall use the Service Id as identified in the table 4.31. Each RTE API template, RTE callback template and RTE API will have an Identifier. This ID Service ID must be used when running code in the context of the respective RTE call. ⌋*(SRS_BSW_00337)*

4.8.2 DET Error Identifiers

Only a limited set of development identifiers are currently recognized. Each of these need to be detected either at runtime or during initialization of the RTE. To report these errors extra development code must be generated by the RTE generator.

[SWS_Rte_06633] ⌈ An RTE_E_DET_ILLEGAL_SIGNAL_ID (0x01) shall be reported at runtime by the RTE when it receives a COM callback for a signal name (e.g. `Rte_COMCbk_<sn>`, `Rte_COMCbkTAck_<sn>`) which was not expected within the context of the currently-selected postBuild variant. See section 5.9.2 for the list of possible COM callback template API. ⌋*(SRS_BSW_00337)*

[SWS_Rte_06634] ⌈ An RTE_E_DET_ILLEGAL_VARIANT_CRITERION_VALUE (0x02) shall be reported by the RTE when it determines that a value is assigned to a variant criterion which is not in the list of possible values for that criterion. This error shall be detected during the RTE initialization phase. ⌋*(SRS_BSW_00337)*

[SWS_Rte_07684] ⌈ An RTE_E_DET_ILLEGAL_VARIANT_CRITERION_VALUE (0x02) shall be reported by the *Basic Software Scheduler* when the `SchM_Init` API is called with a `NULL` parameter. ⌋*(SRS_BSW_00337)*

[SWS_Rte_06635] ⌈ An RTE_E_DET_ILLEGAL_INVOCATION (0x03) shall be reported by the RTE when it determines that an RTE API is called by a Runnable which should not call that RTE API. The RTE can identify the active Runnable when it dispatches the RTE Event and if it subsequently receives a call from that Runnable to an API that is not part of its contract then this particular error ID must me logged. ⌋*(SRS_BSW_00337)*

[SWS_Rte_06637] ⌈ An RTE_E_DET_WAIT_IN_EXCLUSIVE_AREA (0x04) shall be reported by the RTE when an application has called an Rte_Enter API and subsequently asks the RTE to enter a wait state. This is illegal because it would lock the ECU. ⌋*(SRS_BSW_00337)*

[SWS_Rte_07675] ⌈ An RTE_E_DET_ILLEGAL_NESTED_EXCLUSIVE_AREA (0x05) shall be reported by the RTE when an application violates [constr_9029]. ⌋*(SRS_BSW_00337)*

[SWS_Rte_07685] ⌈ An RTE_E_DET_SEG_FAULT (0x06) shall be reported by the RTE when the parameters of an RTE API call contain a direct or indirect reference to memory that is not accessible from the callers partition as defined in [SWS_Rte_02752] and [SWS_Rte_02753]. ⌋*(SRS_BSW_00337)*

[SWS_Rte_07682] ⌈ If `RteDevErrorDetectUninit` is enabled, an RTE_E_DET_UNINIT (0x07) shall be reported by the RTE when one of the APIs :

- Specified in 5.6.
- `Rte_NvMNotifyInitBlock`.
- `Rte_PartitionTerminated`.
- `Rte_PartitionRestarting`.

- Rte_RestartPartition.

is called before Rte_Start, after Rte_Stop or After the partition to witch the API belongs is terminated. ⌋*(SRS_BSW_00337)*

Note:

- In production mode, No checks are performed.
- In development mode, if an error is detected the API behaviour is undefined and it is left to the Rte implementer.

Rational: The introduction of this developpement check should not introduce big changes to production mode configuration.

[SWS_Rte_07683] ⌈ If RteDevErrorDetectUninit is enabled, an RTE_E_DET_UNINIT (0x07) shall be reported by the *Basic Software Scheduler* / RTE when one of the APIs SchM_Switch, SchM_Mode, SchM_SwitchAck, SchM_Trigger, SchM_ActMainFunction, or Rte_Start is called before SchM_Init. ⌋*(SRS_BSW_00337)*

4.8.3 DET Error Classification

The following abbreviations are used to identify the DET error in table 4.31.

Abbreviation	RTE DET Error
ISI	RTE_E_DET_ILLEGAL_SIGNAL_ID
IVCV	RTE_E_DET_ILLEGAL_VARIANT_CRITERION_VALUE
II	RTE_E_DET_ILLEGAL_INVOCATION
INEA	RTE_E_DET_ILLEGAL_NESTED_EXCLUSIVE_AREA
WIEA	RTE_E_DET_WAIT_IN_EXCLUSIVE_AREA
UNINIT	RTE_E_DET_UNINIT

Table 4.30: Abbreviations of RTE DET Errors to APIs

The following table 4.31 indicates which DET errors are relevant for the various RTE APIs, and the service ID associated with the RTE APIs (see [SWS_Rte_06632]):

API name	Service ID	ISI	IVCV	II	INEA	WIEA	UNINIT
Rte_Ports APIs	0x10						X
Rte_NPorts APIs	0x11						X
Rte_Port APIs	0x12						X
Rte_Send APIs	0x13						X
Rte_Write APIs	0x14						X
Rte_Switch APIs	0x15						X

Rte_Invalidate APIs	0x16						X
Rte_Feedback APIs	0x17					X	X
Rte_SwitchAck APIs	0x18					X	X
Rte_Read APIs	0x19						X
Rte_DRead APIs	0x1A						X
Rte_Receive APIs	0x1B					X	X
Rte_Call APIs	0x1C					X	X
Rte_Result APIs	0x1D					X	X
Rte_Pim APIs	0x1E						X
Rte_CData APIs	0x1F						X
Rte_Prm APIs	0x20						X
Rte_IRead APIs	0x21						X
Rte_IWrite APIs	0x22						X
Rte_IWriteRef APIs	0x23						X
Rte_IInvalidate APIs	0x24						X
Rte_IStatus APIs	0x25						X
Rte_IrvIRead APIs	0x26						X
Rte_IrvIWrite APIs	0x27						X
Rte_IrvRead APIs	0x28						X
Rte_IrvWrite APIs	0x29						X
Rte_Enter APIs	0x2A						X
Rte_Exit APIs	0x2B				X		X
Rte_Mode APIs	0x2C						
Rte_Trigger APIs	0x2D						X
Rte_IrTrigger APIs	0x2E						X
Rte_IFeedback APIs	0x2F						X
Rte_IsUpdated APIs	0x30						X
trigger by TimingEvent	0x50			X			
trigger by BackgroundEvent	0x51			X			
trigger by SwcModeSwitchEvent	0x52			X			
trigger by AsynchronousServerCallReturnsEvent	0x53			X			
trigger by DataReceiveErrorEvent	0x54			X			
trigger by OperationInvokedEvent	0x55			X			
trigger by DataReceivedEvent	0x56			X			
trigger by DataSendCompletedEvent	0x57			X			
trigger by ExternalTriggerOccurredEvent	0x58			X			
trigger by InternalTriggerOccurredEvent	0x59			X			
trigger by DataWriteCompletedEvent	0x5A			X			
Rte_Start API	0x70						X
Rte_Stop API	0x71						

Rte_PartitionTerminated APIs	0x72						
Rte_PartitionRestarting APIs	0x73						
Rte_RestartPartition APIs	0x74						
Rte_Init API	0x75						
Rte_StartTiming API	0x76						
Rte_COMCbkTAck_<sn> callbacks	0x90	X					
Rte_COMCbkTErr_<sn> callbacks	0x91	X					
Rte_COMCbkInv_<sn> callbacks	0x92	X					
Rte_COMCbkRxTOut_<sn> callbacks	0x93	X					
Rte_COMCbkTxTOut_<sn> callbacks	0x94	X					
Rte_COMCbk_<sg> callbacks	0x95	X					
Rte_COMCbkTAck_<sg> callbacks	0x96	X					
Rte_COMCbkTErr_<sg> callbacks	0x97	X					
Rte_COMCbkInv_<sg> callbacks	0x98	X					
Rte_COMCbkRxTOut_<sg> callbacks	0x99	X					
Rte_COMCbkTxTOut_<sg> callbacks	0x9A	X					
Rte_COMCbk_<sn> callbacks	0x9F	X					
Rte_SetMirror callbacks	0x9B						
Rte_GetMirror callbacks	0x9C						
Rte_NvMNotifyJobFinished callbacks	0x9D						
Rte_NvMNotifyInitBlock callbacks	0x9E						X
SchM_Init API	0x00		X				
SchM_Deinit API	0x01						
SchM_GetVersionInfo API	0x02						
SchM_Enter APIs	0x03						X
SchM_Exit APIs	0x04				X		X
SchM_ActMainFunction APIs	0x05						X
SchM_Switch APIs	0x06						X
SchM_Mode APIs	0x07						X
SchM_SwitchAck APIs	0x08						X
SchM_Trigger APIs	0x09						X

Table 4.31: Applicability of RTE DET Errors to APIs

4.9 Bypass Support

Rapid prototyping can be used during electronic control unit development to evaluate and test new software control algorithms for various functions.

With Fullpass technology the original ECU is totally replaced by a Rapid Prototyping Unit (RPU).

With Bypass technology the original ECU and software stays in the control loop to supports the majority of the control algorithms and interface with sensors, actuators and communication buses: only the specific control algorithm that shall be prototyped is deported into the RPU (external bypass) or even directly executed in the original ECU (internal bypass). Bypass mainly consists in replacing at run time inputs and/or outputs of the original software algorithms by value computed by the prototype algorithm under test.

The RTE does not directly implement bypass but the RTE provides supports for the integration of such implementation by CDD and/or integration code.

4.9.1 Bypass description

In order to describe a rapid prototyping system as an Autosar Software Component a System Description with the category `RPT_SYSTEM` is used. This System Description is not relevant for the RTE itself but is only a support for the ECU integrator to setup the rapid prototyping solution.

[SWS_Rte_07833] ⌈ RTE shall ignore definitions in System Description of category `RPT_SYSTEM`. ⌋*(SRS_Rte_00244)*

4.9.2 Component wrapper method

The component wrapper method consists in wrapping the original software component implementation with a CDD that implements the bypass. With this method the CDD is able to take the control of the AUTOSAR interfaces of the software component because there is no more direct call between RTE and the SWC but everything go through the CDD.

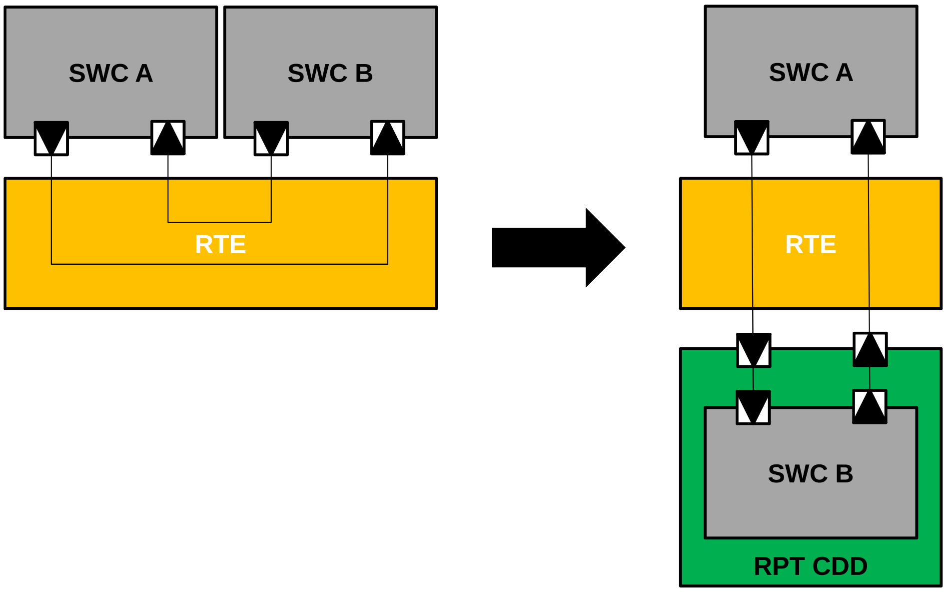


Figure 4.52: Component wrapper method

The RTE supports the component wrapper method by generating the SWC interfaces with a c-namespace including an additional [Byps_] infix for the bypassed SWC (i.e. `SWC B` in Figure 4.52). This includes:

- naming of Application Header File
- naming of the Application Type Header File
- naming of the RTE APIs (excepted life cycle APIs)
- naming of the runnables
- naming of the instance handle
- naming of the Component Data Structure type
- naming of the memory sections

The component wrapper method for bypass support is enabled per software component type.

[SWS_Rte_07840] ⌈The component wrapper method for bypass support is enabled for a software component type if the general switch `RteBypassSupport` is set to `COMPONENT_WRAPPER` and the individual switch for this software component type `RteBypassSupportEnabled` is set to true. ⌋*(SRS_Rte_00244)*

[SWS_Rte_07841] ⌈ The component wrapper method for bypass support is disabled for a software component type if the general switch `RteBypassSupport` is set to value different from `COMPONENT_WRAPPER` or if the individual switch for this software component type `RteBypassSupportEnabled` is not configured or is set to false. ⌋*(SRS_Rte_00244)*

[SWS_Rte_07834] ⌈ If the component wrapper method for bypass support is enabled for a software component type, the RTE generator shall include the optional infix `[Byps\_]` to the name of all the elements generated for this software component type that are defined in this specification with the optional infix `[Byps\_]`. ⌋*(SRS_Rte_00244)*

[SWS_Rte_07835] ⌈ If the component wrapper method for bypass support is disabled for a software component type, the RTE generator shall remove the optional infix `[Byps\_]` to the name of all the elements generated for this software component type that are defined in this specification with the optional infix `[Byps\_]`. ⌋*(SRS_Rte_00244)*

4.9.3 Direct buffer access method

The direct buffer access method provides runtime direct read and write access to the RTE buffers that implement the ECU communication infrastructure.

The RTE supports the direct buffer access method by generating the `McSupportData` for these buffers. This is already supported by the RTE measurement and calibration support but for the rapid prototyping purpose additional elements shall be generated.

The component wrapper method for bypass support is enabled per software component type.

The component wrapper method for bypass support is enabled for a software component type if the individual switch for this software component type `RteBypassSupportEnabled` is set to true.

[SWS_Rte_07836] ⌈ If the direct buffer access method for bypass support is enabled for a software component type, the RTE generator shall generate `McSupportData` with `mcDataAccessDetails` for each preemption area specific buffer that implements the implicit communication for this software component type. ⌋*(SRS_Rte_00244)*

5 RTE Reference

"Everything should be as simple as possible, but no simpler."
– *Albert Einstein*

5.1 Scope

This chapter presents the RTE API from the perspective of AUTOSAR applications and basic software – the same API applies to all software whether they are AUTOSAR software-components or basic software.

Section 5.2 presents basic principles of the API including naming conventions and supported programming languages. Section 5.3 describes the header files used by the RTE and the files created by an RTE generator. The data types used by the API are described in Section 5.5 and Sections 5.6 and 5.7 provide a reference to the RTE API itself including the definition of runnable entities. Section 5.11 defines the events that can be monitored during VFB tracing.

5.1.1 Programming Languages

The RTE is required to support components written using the C and C++ programming languages [SRS_Rte_00126] as well as legacy software modules. The ability for multiple languages to use the same generated RTE is an important step in reducing the complexity of RTE generation and therefore the scope for errors.

[SWS_Rte_01167] ⌈ The RTE shall be generated in C. ⌋*(SRS_Rte_00126)*

[SWS_Rte_01168] ⌈ All RTE code, whether generated or not, shall conform to the HIS subset of the MISRA C standard [24]. In technically reasonable, exceptional cases MISRA violations are permissible. Except for MISRA rule #11, such violations shall be clearly identified and documented. ⌋*(SRS_BSW_00007)*

Specified MISRA violations are defined in Appendix C.

In realistic use cases, the RTE will generate C identifiers (functions, types, variables, etc) whose name will be longer than the maximum size supported by the MISRA C standard (rule #11). Users should configure the RTE to indicate the maximum C identifiers' size supported by their tool chain to make sure that no issues will be caused by these MISRA violation.

[SWS_Rte_07300] ⌈ If a RteToolChainSignificantCharacters limit has been configured, the RTE generator shall provide the list of C RTE identifiers whose name is not unique when only the first RteToolChainSignificantCharacters characters are considered. ⌋*(SRS_BSW_00007)*

The RTE API presented in Section 5.6 is described using C. The API is also directly accessible from an AUTOSAR software-component written using C++ provided all API functions and instances of data structures are imported with C linkage.

[SWS_Rte_01011] ⌈ The RTE generator shall ensure that, for a component written in C++, all imported RTE symbols are declared using C linkage. ⌋*(SRS_Rte_00138)*

For the RTE API for C and C++ components the import of symbols occurs within the application header file (Section 5.3.3).

5.1.2 Generator Principles

5.1.2.1 Operating Modes

An object-code component is compiled against an application header file that is created during the first "RTE Contract" phase of RTE generation. The object code is then linked against an RTE created during the second "RTE Generation" phase. To ensure that the object-code component and the RTE code are compatible the RTE generator supports *compatibility mode* that uses well-defined data structures and types for the component data structure. In addition, an RTE generator may support a *vendor* operating mode that removes compatibility between RTE generators from different vendors but permits implementation specific, and hence potentially more efficient, data structures and types.

[SWS_Rte_01195] ⌈ All RTE operating modes shall be source-code compatible at the SW-C level. ⌋*(SRS_Rte_00024, SRS_Rte_00140)*

Requirement [SWS_Rte_01195] ensures that a SW-C can be used in any operating mode as long as the source is available. The converse is not true – for example, an object-code SW-C compiled after the "RTE Contract" phase must be linked against an RTE created by an RTE generator operating in the same operating mode. If the vendor mode is used in the "RTE Contract" phase, an RTE generator from the same vendor (or one compatible to the vendor-mode features of the RTE generator used in the "RTE Contract" phase) has to be used for the "RTE Generation" phase.

5.1.2.1.1 Compatibility Mode

Compatibility mode is the default operating mode for an RTE generator and guarantees compatibility even between RTE generators from different vendors through the use of well-defined, "standardized", data structures. The data structures that are used by the generated RTE in the compatibility mode are defined in Section 5.4.

Support for compatibility mode is required and therefore is guaranteed to be implemented by all RTE generators.

[SWS_Rte_01151] ⌈ The *compatibility mode* shall be the default operating mode and shall be supported by all RTE generators, whether they are for the “RTE Contract” or “RTE Generation” phases. ⌋ *(SRS_Rte_00145)*

The compatibility mode uses custom (generated) functions with standardized names and data structures that are defined during the “RTE Contract” phase and used when compiling object-code components.

[SWS_Rte_01216] ⌈ SW-Cs that are compiled against an “RTE Contract” phase application header file (i.e. object-code SW-Cs) generated in compatibility mode shall be compatible with an RTE that was generated in compatibility mode. ⌋ *(SRS_Rte_00145)*

The use of well-defined data structures imposes tight constraints on the RTE implementation and therefore restricts the freedom of RTE vendors to optimize the solution of object-code components but have the advantage that RTE generators from different vendors can be used to compile a binary-component and to generate the RTE.

Note that even when an RTE generator is operating in compatibility mode the data structures used for *source-code* components are not defined thus permitting vendor-specific optimizations to be applied.

5.1.2.1.2 Vendor Mode

Vendor mode is an optional operating mode where the data structures defined in the “RTE Contract” phase and used in the “RTE Generation” phase are implementation specific rather than “standardized”.

[SWS_Rte_01152] ⌈ An RTE generator may optionally support *vendor mode*. ⌋ *(SRS_Rte_00083)*

The data structures defined and declared when an RTE generator operates in vendor mode are implementation specific and therefore *not* described in this document. This omission is deliberate and permits vendor-specific optimizations to be implemented for object-code components. It also means that RTE generators from different vendors are unlikely to be compatible when run in the vendor mode.

[SWS_Rte_01234] ⌈ An AUTOSAR software-component shall be assumed to be operating in “compatibility” mode unless “vendor mode” is explicitly requested. ⌋ *(SRS_Rte_00145, SRS_Rte_00146)*

The potential for more efficient implementations of object-code components offered by the vendor mode comes at the expense of requiring high cohesion between object-code components (compiled after the “RTE Contract” phase) and the generated RTE. However, this is not as restrictive as it may seem at first sight since the tight coupling is also reflected in many other aspects or the AUTOSAR methodology, not least of which is the requirement that the same compiler (and compatible options) is used when compiling both the object-code component and the RTE.

5.1.2.2 Optimization Modes

The actual RTE code is generated – based on the input information – for each ECU individually. To allow optimization during the RTE generation one of the two general optimization directions can be specified: MEMORY consumption or execution RUNTIME.

[SWS_Rte_05053] ⌈ The RTE Generator shall optimize the generated RTE code either for memory consumption or execution runtime depending on the provided input information RteOptimizationMode. ⌋*(SRS_Rte_00023)*

5.1.2.3 Build support

The generated RTE code has to respect several rules in order to be integrated with other AUTOSAR software in the build process.

[SWS_Rte_05088] ⌈ All memory[1] allocated by the RTE shall be wrapped in the segment declarations defined in the *Specification of Memory Mapping* [25] using RTE as the <MSN> (Module Short Name). ⌋*(SRS_Rte_00148, SRS_Rte_00169)*

Due to the structure of the AUTOSAR Meta Model the input configuration might contain several DataPrototypes which are resulting only in one memory object. In this case it is required to define rules which SwAddrMethod is used to allocate the memory and to decide about its initialization. Therefore precedence rules for SwAddrMethods are defined by [SWS_Rte_07590] and [SWS_Rte_07591].

[SWS_Rte_07589] ⌈ For AutosarDataPrototype implementations the <SEGMENT> infix for the Memory Allocation Keyword shall be set to the shortName of the preceding SwAddrMethod if there is one defined and if [SWS_Rte_07592] is not applicable. ⌋*(SRS_Rte_00148, SRS_Rte_00169)*

[SWS_Rte_07047] ⌈ If the memoryAllocationKeywordPolicy of the preceding SwAddrMethod is set to addrMethodShortName the <ALIGNMENT> suffix with leading underscore of the Memory Allocation Keyword used by the AutosarDataPrototype implementations and PerInstanceMemory implementations shall be omitted. ⌋*(SRS_Rte_00148, SRS_Rte_00169)*

[SWS_Rte_07048] ⌈ If the memoryAllocationKeywordPolicy of the preceding SwAddrMethod is set to addrMethodShortNameAndAlignment the <ALIGNMENT> suffix with leading underscore of the Memory Allocation Keyword used by the AutosarDataPrototype implementations and PerInstanceMemory implementations shall be set to the resulting alignment as defined in [SWS_Rte_07049], [SWS_Rte_07050], [SWS_Rte_07051], [SWS_Rte_07052] and [SWS_Rte_07053]. ⌋*(SRS_Rte_00148, SRS_Rte_00169)*

[SWS_Rte_08303] ⌈ The alignment of a PerInstanceMemory shall be set to UNSPECIFIED. ⌋*(SRS_Rte_00013, SRS_Rte_00077)*

[1] *memory* refers to all elements in the generated RTE which will later occupy space in the ECU's memory and is directly associated with the RTE. This includes code, static data, parameters, etc.

[SWS_Rte_07049] ⌈The alignment defined by the preceding (see [SWS_Rte_07196]) swAlignment attribute of a AutosarDataPrototype precedes the alignment defined by the ImplementationDataType related to the AutosarDataPrototype as defined in [SWS_Rte_07050], [SWS_Rte_07051], [SWS_Rte_07052] and [SWS_Rte_07053].⌋*(SRS_Rte_00148, SRS_Rte_00169)*

[SWS_Rte_07050] ⌈The alignment of a AutosarDataPrototype related to a Primitive Implementation Data Type or Array Implementation Data Type shall be set to the baseTypeSize of the referred SwBaseType.⌋*(SRS_Rte_00148, SRS_Rte_00169)*

[SWS_Rte_07051] ⌈The alignment of a AutosarDataPrototype related to a Structure Implementation Data Type or Union Implementation Data Type shall be set to to biggest baseTypeSize of the SwBaseTypes used by the elements.⌋*(SRS_Rte_00148, SRS_Rte_00169)*

Note: According [SWS_Rte_07051] structures and unions are aligned according the size of the biggest primitive element in the structure.

[SWS_Rte_07052] ⌈The alignment of a AutosarDataPrototype related to a Redefinition Implementation Data Type shall be determined from the redefined ImplementationDataType.⌋*(SRS_Rte_00148, SRS_Rte_00169)*

[SWS_Rte_07053] ⌈The alignment of a AutosarDataPrototype related to a Pointer Implementation Data Type shall be set to UNSPECIFIED.⌋*(SRS_Rte_00148, SRS_Rte_00169)*

Note: If the RTE generator does not implement the memory objects related to VariableDataPrototypes and ParameterDataPrototypes for instance due to communication via IOC the assigned SwAddrMethods might have no effect on the generated RTE code.

[SWS_Rte_07592] ⌈If the RTE Generator requires several non automatic memory objects per AutosarDataPrototypes (e.g. due to partitioning) the RTE Generator is permitted to select the <SEGMENT> infix for the auxiliary memory objects.⌋*(SRS_Rte_00148, SRS_Rte_00169)*

Note: For definitions and declarations for memory objects allocated by the RTE and implementing AutosarDataPrototypes without an assigned SwAddrMethod the RTE Generator is permitted to select the <SEGMENT> infix but still has to follow [SWS_Rte_05088].

[SWS_Rte_07590] ⌈The SwAddrMethod of a AutosarDataPrototype in the PPortPrototype precedes the assigned SwAddrMethod(s) of the AutosarDataPrototype in the RPortPrototype and PRPortPrototype.⌋*(SRS_Rte_00148, SRS_Rte_00169)*

[SWS_Rte_06741] ⌈The SwAddrMethod of a AutosarDataPrototype in the PRPortPrototype precedes the assigned SwAddrMethod(s) of the AutosarDataPrototype in the RPortPrototype.⌋*(SRS_Rte_00148, SRS_Rte_00169)*

[SWS_Rte_07591] ⌈The SwAddrMethod of the ramBlocks has always higher precedence as the assigned SwAddrMethods of the VariableDataPrototypes in the PortPrototypes. ⌋*(SRS_Rte_00148, SRS_Rte_00169)*

[SWS_Rte_05089] ⌈The RTE Generator shall provide information on the used memory segments and their attributes from [SWS_Rte_05088] in the generated *Basic Software Module Description*(see [SWS_Rte_05086]). The information shall be provided in the MemorySection elements of the *Basic Software Module Description* [9]. ⌋*(SRS_Rte_00148, SRS_Rte_00169, SRS_Rte_00170)*

[SWS_Rte_05090] ⌈The RTE Generator shall provide information about the generated artifacts which are produced during the RTE generation, using the generated *Basic Software Module Description*(see [SWS_Rte_05086]). The information shall be provided in the BswImplementation::generatedArtifact elements of the *Basic Software Module Description* [9]. ⌋

5.1.2.4 Debugging support

For the support of the AUTOSAR Debugging (see [26]) several requirements have to be respected.

[SWS_Rte_05094] ⌈Each variable that shall be accessible by AUTOSAR Debugging, shall be defined as global variable. ⌋

[SWS_Rte_05095] ⌈All type definitions of variables which shall be debugged, shall be accessible by the Rte types header file *Rte_Type.h*. ⌋

[SWS_Rte_05096] ⌈The declaration of variables in the header file shall be such, that it is possible to calculate the size of the variables by C-'sizeof()'. ⌋

[SWS_Rte_05097] ⌈Variables available for debugging shall be described in the respective *Basic Software Module Description* (see [SWS_Rte_05086], [9]) using the elements BswDebugInfo. ⌋

[SWS_Rte_05098] ⌈If the state of a Runnable Entity is kept in a variable in the generated RTE, it shall be possible to debug the state of this Runnable Entity ([SWS_Rte_02697]). ⌋

[SWS_Rte_05105] ⌈If the Mode Machine Instance is kept in a variable in the generated RTE, it shall be possible to debug the state of this Mode Machine Instance, ⌋

5.1.2.5 Software Component Namespace

The concept of RTE requires that objects and definitions which are related to one software component are generated in a global name space. Nevertheless in this global name space labels have to be unique for instance to support a correct linkage by C Linker Locater. To ensure unique labels such objects and definitions related to a

specific software component are typically prefixed or infixed with the component type symbol.
When AtomicSwComponentTypes of several vendors are integrated in the same ECU name clashes might occur if the identical component type name is accidentally used twice. To ease the dissolving of name clashes the RTE supports the superseding of the AtomicSwComponentType.shortName with the SymbolProps.symbol attribute.

The resulting name related to an AtomicSwComponentType is called component type symbol in this document.

[SWS_Rte_06714] ⌈ The component type symbol shall be the value of the SymbolProps.symbol attribute of the AtomicSwComponentType if the symbol attribute is defined. ⌋

[SWS_Rte_06715] ⌈ The component type symbol shall be the shortName of the AtomicSwComponentType if no symbol attribute for this AtomicSwComponentTypeis defined. ⌋

Please note that the component type symbol is not applied for file names, e.g *Application Header File* or includes of Memory Mapping Header files. Its expected that a build environment can handle two equally named files.

5.1.3 Generator external configuration switches

There are use-cases where there is need to influence the behavior of the RTE Generator without changing the RTE Configuration description. In order to support such use-cases this section collects the *external configuration switches*.

Note: it is not specified how these switches shall be implemented in the actual RTE Generator implementation.

Unconnected R-Port check

[SWS_Rte_05099] ⌈ The RTE Generator shall support the *external configuration switch* strictUnconnectedRPortCheck which, when enabled, forces the RTE Generator to consider unconnected R-Ports as an error. ⌋*(SRS_Rte_00139)*

Missing input configuration check

[SWS_Rte_05148] ⌈ The RTE Generator shall support the *external configuration switch* strictConfigurationCheck which, when enabled, forces the RTE Generator to consider missing input configuration information as an error. If the *external configuration switch* strictConfigurationCheck is not provided the value shall be considered as *true*. ⌋

For Details on the use-cases please refer to section 3.7.

Missing initialization values

[SWS_Rte_07680] ⌈ The RTE Generator shall support the *external configuration switch* `strictInitialValuesCheck`. This switch, when enabled, forces the RTE Generator to check initial values against constraints defined in [TPS_SYST_02011], [SWS_Rte_07642] and [SWS_Rte_07681]. Not fulfilled constraints shall be considered as errors by the RTE Generator. ⌋*(SRS_Rte_00108)*

5.2 API Principles

[SWS_Rte_01316] ⌈ The RTE shall be configured and/or generated for each ECU. ⌋*(SRS_Rte_00021)*

Part of the process is the customization (i.e. configuration or generation) of the RTE API for each AUTOSAR software-component on the ECU. The customization of the API implementation for each AUTOSAR software-component, whether by generation anew or configuration of library code, permits improved run-time efficiency and reduces memory overheads.

The design of the RTE API has been guided by the following core principles:

- The API should be orthogonal – there should be only one way of performing a task.
- **[SWS_Rte_01314]** ⌈ The API shall be compiler independent. ⌋*(SRS_Rte_00100)*
- **[SWS_Rte_03787]** ⌈ The RTE implementation shall use the compiler abstraction. ⌋*(SRS_Rte_00149)*

  The consequence of [SWS_Rte_03787] is that no additional memory modifiers (e.g. volatile) are permitted in the signatures of the RTE APIs.
- **[SWS_Rte_01315]** ⌈ The API shall support components where the source-code is available [SRS_Rte_00024] and where only object-code is available [SRS_Rte_00140]. ⌋*(SRS_Rte_00024, SRS_Rte_00140)*
- The API shall support the multiple instantiation of AUTOSAR software-components [SRS_Rte_00011] that share code [SRS_Rte_00012].

Two forms of the RTE API are available to software-components; direct and indirect. The direct API has been designed with regard to efficient invocation and includes an API mapping that can be used by an RTE generator to optimize a component's API, for example, to permit the direct invocation of the generated API functions or even eliding the generated RTE completely. The indirect API cannot be optimized using the API mapping but has the advantage that the handle used to access the API can be stored in memory and accessed, via an iterator, to apply the same API to multiple ports.

5.2.1 RTE Namespace

All RTE symbols (e.g. function names, global variables, etc.) visible within the global namespace are required to use the “Rte” prefix.

[SWS_Rte_01171] ⌈ All externally visible symbols created by the RTE generator shall use the prefix `Rte_`.

This rule shall not be applied for the following symbols:

- type names representing AUTOSAR Data Types (specified in [SWS_Rte_07104], [SWS_Rte_07109], [SWS_Rte_07110], [SWS_Rte_07111], [SWS_Rte_07114], [SWS_Rte_07144], [SWS_Rte_07148])
- enumeration literals of implementation data types (specified in [SWS_Rte_03810])
- range limits of `ApplicationDataType`s (specified in [SWS_Rte_05052])

This rule shall be applied for RTE internal types to avoid name clashes with other modules and SWCs. ⌋*(SRS_BSW_00307, SRS_BSW_00300, SRS_Rte_00055)*

In order to maintain control over the RTE namespace the creation of symbols in the global namespace using the prefix `Rte_` is reserved for the RTE generator.

The generated RTE is required to work with components written in several source languages and therefore should not use language specific features, such as C++ namespaces, to ensure symbol name uniqueness.

5.2.2 Direct API

The direct invocation form is the form used to present the RTE API in Section 5.6. The RTE direct API mapping is designed to be optimizable so that the instance handle is elided (and therefore imposes zero run-time overhead) when the RTE generator can determine that exactly one instance of a component is mapped to an ECU.

All runnable entities for a AUTOSAR software-component type are passed the same instance handle type (as the first formal parameter) and can therefore use the same type definition from the component’s application header file.

The direct API can also be further optimized for source code components via the API mapping.

The direct API is typically implemented as macros that are modified by the RTE generator depending on configuration. This technique places certain restrictions on how the API can be used within a program, for example, it is not possible in C to take the address of a macro and therefore direct API functions cannot be placed within a function table or array. If it is required by the implementation of a software-component to derive a pointer to an object for the port API the `PortAPIOption enableTakeAddress` can be used. For instance in an implementation of an `AUTOSAR Service` this feature might be used to setup a constant function pointer table storing the configuration of callback functions per ID. Additionally the indirect API provides support for API addresses and iteration over ports.

[SWS_Rte_07100] ⌈ If a PortPrototype is referenced by PortAPIOption with enableTakeAddress = TRUE the RTE generator has to provide "C" functions and non function like macro for the API related to this port. ⌋

The PortAPIOption enableTakeAddress = TRUE is not supported for software-components supporting multiple instantiation.

5.2.3 Indirect API

The indirect API is an optional form of API invocation that uses indirection through a port handle to invoke RTE API functions rather than direct invocation. This form is less efficient (the indirection cannot be optimized away) but supports a different programming style that may be more convenient. For example, when using the indirect API, an array of port handles of the same interface and provide/require direction is provided by RTE and the same RTE API can be invoked for multiple ports by iterating over the array.

Both direct and indirect forms of API call are equivalent and result in the same generated RTE function being invoked.

Whether the indirect API is generated or not can be specified for each software component and for each port prototype of the software component separately with the indirectAPI attribute.

The semantics of the port handle must be the same in both the “RTE Contract” and “RTE Generation” phases since the port handle accesses the standardized data structures of the RTE.

It is possible to mix the indirect and direct APIs within the same SW-C, if the indirect API is present for the SW-C.

The indirect API uses port handles during the invocation of RTE API calls. The type of the port handle is determined by the port interface that types the port which means that if a component declares multiple ports typed by the same port interface the port handle points to an array of port data structures and the same API invoked for each element.

The port handle type is defined in Section 5.4.2.5.

5.2.3.1 Accessing Port Handles

An AUTOSAR SW-C needs to obtain port handles using the instance handle before the indirect API can be used. The definition of the instance handle in Section 5.4.2 defines the “Port API” section of the component data structure and these entries can be used to access the port handles in either object-code or source-code components.

The API Rte_Ports and Rte_NPorts provides port data handles of a given interface. Example 5.1 shows how the indirect API can be used to apply the same operation to multiple ports in a component within a loop.

Example 5.1

The port handle points to an array that can be used within a loop to apply the same operation to each port. The following example sends the same data to each receiver:

```
void TT1(Rte_Instance self)
{
  Rte_PortHandle_interface1_P my_array;
  my_array=Rte_Ports_interface1_P(self);
  int s;
  for(s = 0; s < Rte_NPorts_interface1_P(self); s++) {
    my_array[s].Send_a(23);
  }
}
```

Note that if csInterface1 is a client/server interface with an operation op, the mechanism sketched in Example5.1 only works if op is invoked either by all clients synchronously or by all clients asynchronously, since the signature of Rte_Call and the existence of Rte_Result depend on the kind of invocation (see restriction [SWS_Rte_03605].

5.2.4 VariableAccess in the dataReadAccess and dataWriteAccess roles

The RTE is required to support access to data with implicit semantics. The required semantics are subject to two constraints:

- For VariableAccess in the dataReadAccess role, the data accessed by a runnable entity must not change during the lifetime of the runnable entity.
- For VariableAccess in the dataWriteAccess role, the data written by a runnable entity is only visible to other runnable entities after the accessing runnable entity has terminated.

The generated RTE satisfies both requirements through data copies that are created when the RTE is generated based on the known task and runnable mapping.

Example 5.2

Consider a data element, a, of port p which is accessed using a VariableAccess in the dataReadAccess role by runnable re1 and a VariableAccess in the dataWriteAccess role by runnable re2. Furthermore, consider that re1 and re2 are mapped to different tasks and that execution of re1 can pre-empt re2.

In this example, the RTE will create two different copies to contain a to prevent updates from re2 'corrupting' the value access by re1 since the latter must remain unchanged during the lifetime of re1.

The RTE API includes three API calls to support VariableAccesses in the dataReadAccess and dataWriteAccess roles for a software-component; Rte_IRead (see Section 5.6.18), Rte_IWrite, and Rte_IWriteRef (see Section 5.6.19 and 5.6.20). The API calls Rte_IRead and Rte_IWrite access the data copies (for read and write access respectively). The API call Rte_IWriteRef returns a reference to the data copy, thus enabling the runnable to write the data directly. This is especially useful for Structure Implementation Data Type and Array Implementation Data Type. The use of an API call for reading and writing enables the definition to be changed based on the task and runnable mapping without affecting the software-component code.

Example 5.3

Consider a data element, a, of port p which is declared as being accessed using VariableAccesses in the dataWriteAccess role by runnables re1 and re2 within component c. The RTE API for component c will then contain four API functions to write the data element;

```
void Rte_IWrite_re1_p_a(Rte_Instance self, <type> val);
void Rte_IWrite_re2_p_a(Rte_Instance self, <type> val);
<type> Rte_IWriteRef_re1_p_a(Rte_Instance self);
<type> Rte_IWriteRef_re2_p_a(Rte_Instance self);
```

The API calls are used by re1 and re2 as required. The definitions of the API depend on where the data copies are defined. If both re1 and re2 are mapped to the same task then each can access the same copy. However, if re1 and re2 are mapped to different (pre-emptable) tasks then the RTE will ensure that each API access a different copy.

The Rte_IRead and Rte_IWrite use the "data handles" defined in the component data structure (see Section 5.4.2).

5.2.5 Per Instance Memory

The RTE is required to support Per Instance Memory [SRS_Rte_00013].

The component's instance handle defines a particular instance of a component and is therefore used when accessing the *Per Instance Memory* using the Rte_Pim API.

The Rte_Pim API does not impose the RTE to apply a data consistency mechanism for the access to *Per Instance Memory*. An application is responsible for consistency of accessed data by itself. This design decision permits efficient (zero overhead) access when required. If a component possesses multiple runnable entities that require concurrent access to the same *Per Instance Memory*, an exclusive area can be used to ensure data consistency, either through explicit Rte_Enter and Rte_Exit API calls or by declaring that, implicitly, the runnable entities run inside an exclusive area.

Thus, the *Per Instance Memory* is exclusively used by a particular software-component instance and needs to be declared and allocated (statically).

In general there are two different kinds of *Per Instance Memory* available which are varying in the typing mechanisms. 'C' typed PerInstanceMemory is typed by the description of a 'C' typedef whereas arTypedPerInstanceMemory (*AUTOSAR Typed Per Instance Memory*) is typed by the means of an AutosarDataType. Nevertheless both kinds of *Per Instance Memory* are accessed via the Rte_Pim API.

[SWS_Rte_07161] ⌈ The generated RTE shall declare arTypedPerInstanceMemory in accordance to the associated ImplementationDataType of a particular arTypedPerInstanceMemory. ⌋*(SRS_Rte_00013, SRS_Rte_00077)*

Note: The related *AUTOSAR data type* will generated in the RTE Types Header File (see chapter 5.3.6).

[SWS_Rte_02303] ⌈ The generated RTE shall declare 'C' typed PerInstanceMemory in accordance to the attribute type of a particular PerInstanceMemory. ⌋*(SRS_Rte_00013, SRS_Rte_00077)*

In addition, the attribute type needs to be defined in the corresponding software-component header. Therefore, the attribute typeDefinition of the PerInstanceMemory contains its definition as plain text string. It is assumed that this text is valid 'C' syntax, because it will be included verbatim in the application header file.

[SWS_Rte_02304] ⌈ The generated RTE shall define the type of a 'C' typed PerInstanceMemory by interpreting the text string of the attribute typeDefinition of a particular PerInstanceMemory as the 'C' definition. This type shall be named according to the attribute type of the PerInstanceMemory. ⌋*(SRS_Rte_00013, SRS_Rte_00077)*

[SWS_Rte_07133] ⌈ The type of a 'C' typed PerInstanceMemory shall be defined in the *RTE Types Header File* as

```
typedef <typedefinition> Rte_PimType_<cts>_<type>;
```

where <typedefinition> is the content of the typeDefinition attribute of the PerInstanceMemory,
<type> is the type name defined in the type attribute of the the PerInstanceMemory and
<cts> the component type symbol of the AtomicSwComponentType to which the PerInstanceMemory belongs.. ⌋*(SRS_Rte_00013, SRS_Rte_00077)*

[SWS_Rte_03782] ⌈ The type of a 'C' typed PerInstanceMemory shall be defined in the *Application Header File* as

```
typedef Rte_PimType_<cts>_<type> <type>;
```

where <cts> is the component type symbol of the AtomicSwComponentType to which the PerInstanceMemory belongs and
<type> is the type name defined in the type attribute of the PerInstanceMemory.
⌋*(SRS_Rte_00013, SRS_Rte_00077)*

[SWS_Rte_07134] ⌈ The RTE generator shall generate type definitions for 'C' typed PerInstanceMemory (see [SWS_Rte_07133] and [SWS_Rte_03782]) only once for all 'C' typed PerInstanceMemorys of same Software Component Type defining identical couples of type and typeDefinition attributes. ⌋*(SRS_Rte_00013, SRS_Rte_00165)*

Note: This shall support, that a Software Component Type can define several PerInstanceMemory's using the identical 'C' type.

[SWS_Rte_07135] ⌈ The RTE generator shall reject configurations, violating [constr_2007], where 'C' typed PerInstanceMemorys with identical type attributes but different typeDefinition attributes in the same Software Component Type are defined. ⌋*(SRS_Rte_00013, SRS_Rte_00018)*

Note: This would lead to an compiler error due to incompatible redefinition of a 'C' type.

[SWS_Rte_02305] ⌈ The generated RTE shall instantiate (or allocate) declared PerInstanceMemory. ⌋*(SRS_Rte_00013, SRS_Rte_00077)*

[SWS_Rte_07182] ⌈ The generated RTE shall initialize declared PerInstanceMemory according the initValue attribute if

- an initValue is defined

  AND

- no SwAddrMethod is defined for PerInstanceMemory.

⌋*(SRS_Rte_00013, SRS_Rte_00077)*

[SWS_Rte_08304] ⌈ Variables implementing PerInstanceMemory shall be initialized by RTE if

- an initValue is defined

  AND

- a SwAddrMethod is defined for PerInstanceMemory

  AND

- the RteInitializationStrategy for the sectionInitializationPolicy of the related SwAddrMethod is NOT configured to RTE_INITIALIZATION_STRATEGY_NONE.

⌋*(SRS_Rte_00013, SRS_Rte_00077)*

[SWS_Rte_07183] ⌈ The generated RTE shall instantiate (or allocate) declared arTypedPerInstanceMemory. ⌋*(SRS_Rte_00013, SRS_Rte_00077)*

[SWS_Rte_07184] ⌈ The generated RTE shall initialize declared arTypedPerInstanceMemory according the ValueSpecification of the VariableDataPrototype defining the arTypedPerInstanceMemory if the general initialization conditions in [SWS_Rte_07046] are fulfilled. ⌋*(SRS_Rte_00013, SRS_Rte_00077)*

[SWS_Rte_05062] ⌈ In case the PerInstanceMemory or arTypedPerInstanceMemory is used as a permanent ram mirror for the *NvRam manager* the name for the instantiated PerInstanceMemory or arTypedPerInstanceMemory shall be taken from the input information RteNvmRamBlockLocationSymbol. Otherwise the RTE generator is free to choose an arbitrary name. ⌋*(SRS_Rte_00013, SRS_Rte_00077)*

Note that, in cases where a PerInstanceMemory is not initialized due to [SWS_Rte_07182] or [SWS_Rte_07184], the memory allocated for a PerInstanceMemory is not initialized by the generated RTE, but by the corresponding software-component instances.

[SWS_Rte_07693] ⌈ In case a ParameterDataPrototype in the role perInstanceParameter is used as a romBlock for the NVRam Manager, then the name for the instantiated ParameterDataPrototype shall be taken from the input information RteNvmRomBlockLocationSymbol. Otherwise the RTE generator is free to choose an arbitrary name. ⌋*(SRS_Rte_00154)*

Example 5.4

This description of a software component

```
<AR-PACKAGE>
  <SHORT-NAME>SWC</SHORT-NAME>
  <ELEMENTS>
    <APPLICATION-SW-COMPONENT-TYPE>
      <SHORT-NAME>TheSwc</SHORT-NAME>
      <INTERNAL-BEHAVIORS>
        <SWC-INTERNAL-BEHAVIOR>
          <SHORT-NAME>TheSwcInternalBehavior</SHORT-NAME>
          <PER-INSTANCE-MEMORYS>
            <PER-INSTANCE-MEMORY>
              <SHORT-NAME>MyPIM</SHORT-NAME>
              <TYPE>MyMemType</TYPE>
              <TYPE-DEFINITION>struct {uint16 val1; uint8 * val2;}</
                 TYPE-DEFINITION>
            </PER-INSTANCE-MEMORY>
          </PER-INSTANCE-MEMORYS>
        </SWC-INTERNAL-BEHAVIOR>
      </INTERNAL-BEHAVIORS>
    </APPLICATION-SW-COMPONENT-TYPE>
  </ELEMENTS>
</AR-PACKAGE>
```

will e. g. result in the following code:

In the *RTE Types Header File*:

```
/* typedef to ensure unique typename */
/* according to the attributes */
```

```
/* 'type' and 'typeDefinition' */
typedef struct{
    uint16 val1;
    uint8 * val2;
} Rte_PimType_TheSwc_MyMemType;
```

In the respective *Application Header File*:

```
/* typedef visible within the scope */
/* of the component according to the attributes */
/* 'type' and 'typeDefinition' */
typedef Rte_PimType_TheSwc_MyMemType MyMemType;
```

In Rte.c:

```
/* declare and instantiate mem1 */
/* "mem1" name may be taken from RteNvmRamBlockLocationSymbol */
Rte_PimType_TheSwc_MyMemType mem1;
```

Note that the name used for the definition of the `PerInstanceMemory` may be used outside of the RTE. One use-case is to support the definition of the link between the NvRam Manager's permanent blocks and the software-components. The name in `RteNvmRamBlockLocationSymbol` is used to configure the location at which the NvRam Manager shall store and retrieve the permanent block content. For a detailed description please refer to the AUTOSAR Software Component Template [2].

5.2.6 API Mapping

The RTE API is implemented by macros and generated API functions that are created (or configured, depending on the implementation) by the RTE generator during the "RTE Generation" phase. Typically one customized macro or function is created for each "end" of a communication though the RTE generator may elide or combine custom functions to improve run-time efficiency or memory overheads.

[SWS_Rte_01274] ⌈ The API mapping shall be implemented in the application header file. ⌋ *(SRS_BSW_00330, SRS_Rte_00027, SRS_Rte_00051, SRS_Rte_00083, SRS_Rte_00087)*

The RTE generator is required to provide a mapping from the RTE API name to the generated function [SRS_Rte_00051]. The API mapping provides a level of indirection necessary to support binary components and multiple component instances. The indirection is necessary for two reasons. Firstly, some information may not be known when the component is created, for example, the component's instance name, but are necessary to ensure that the names of the generated functions are unique. Secondly, the names of the generated API functions should be unique (so that the ECU image can link correctly) and the steps taken to ensure this may make the names not "user-friendly". Therefore, the primary rationale for the API mapping is to provide the required abstraction that means that a component does not need to concern itself with the preceding problems.

The requirements on the API mapping depend on the phase in which an RTE generator is operating. The requirements on the API mapping are only binding for RTE generators operating in compatibility mode.

5.2.6.1 "RTE Contract" Phase

Within the "RTE Contract" phase the API mapping is required to convert from the source API call (as defined in Section 5.6) to the runnable entity provided by a software-component or the implementation of the API function created by the RTE generator.

When compiled against a "RTE Contract" phase header file a software-component that can be multiple instantiated is required to use a general API mapping that uses the instance handle to access the function table defined in the component data structure.

[SWS_Rte_03706] ⌈ If a software-component `supportsMultipleInstantiation`, the "RTE Contract" phase API mapping shall access the generated RTE functions using the instance handle to indirect through the generated function table in the component data structure. ⌋*(SRS_Rte_00051)*

Example 5.5

For a require client-server port 'p1' with operation 'a' with a single argument, the general form of the API mapping would be:

```
#define Rte_Call_p1_a(s,v)  ((s)->p1.Call_a(v))
```

Where `s` is the instance handle.

[SWS_Rte_06516] ⌈ The RTE Generator shall wrap each API mapping and API function definition of a variant existent API according table 4.15 if the variability shall be implemented.

```
#if (<condition> [||<condition>])

<API Mapping>

#endif
```

where `condition` are the condition value macro(s) of the `VariationPoint`s relevant for the conditional existence of the RTE API (see table 4.15), `API Mapping` is the code according an invariant API Mapping (see also [SWS_Rte_01274], [SWS_Rte_03707], [SWS_Rte_03837], [SWS_Rte_01156]) ⌋*(SRS_Rte_00201)*

Note: In case of explicit communication any existent access points in the meta model might result in the related API which results in a or condition for the pre processor.

Example 5.6

For a require client-server port ‘p1’ with operation ‘a’ with a single argument of the component ‘c1’ defining a ServerCallPoint which is subject of variability in runnable ‘run1’, the general form of the conditional API mapping would be:

```
#if (Rte_VPCon_c1_run1_p1_a==TRUE)

#define Rte_Call_p1_a(s,v) ((s)->p1.Call_a(v))

#endif
```

[SWS_Rte_03707] ⌈ If a software-component does not supportsMultipleInstantiation, the “RTE Contract” phase API mapping shall access the generated RTE functions directly. ⌋*(SRS_Rte_00051)*

[SWS_Rte_08073] ⌈ In compatibility mode or “RTE Contract” phase, the API mapping for Rte_PBCon shall access the generated RTE functions directly. ⌋*(SRS_Rte_00051)*

When accessed directly, the names of the generated functions are formed according to the following rule:

[SWS_Rte_03837] ⌈ The function generated for API calls Rte_<name>_<api_extension> that are intended to be called by the software component shall be

Rte_<name>_<cts>_<api_extension>,

where <name> is the API root (e.g. Receive),
<cts> the component type symbol of the AtomicSwComponentType,
and <api_extension> is the extension of the API dependent on <name> (e.g. <re>_<p>_<o>). ⌋*(SRS_Rte_00051)*

[SWS_Rte_01156] ⌈ In compatibility mode, the following API calls shall be implemented as macros:

- Rte_Pim
- Rte_IRead
- Rte_IWrite
- Rte_IWriteRef
- Rte_IStatus
- Rte_IrvIRead
- Rte_IrvIWrite

The generated macros for these API calls shall map to the relevant fields of the component data structure. ⌋*(SRS_Rte_00051)*

Note that the rule described in [SWS_Rte_03837] does not apply for the life cycle APIs, nor for the callback APIs, nor for the APIs that are implemented as macros (see [SWS_Rte_01156]).

The functions generated that are the destination of the API mapping, which is created during the “RTE Contract” phase, are created by the RTE generator during the second “RTE Generation” phase.

[SWS_Rte_01153] ⌈ The generated function (or runnable) shall take the same parameters, in the same order, as the API mapping. ⌋*(SRS_Rte_00051)*

Example 5.7

For a require client-server port ‘p1’ with operation ‘a’ with a single argument for component type ‘c1’ for which multiple instantiation is forbidden, the following mapping would be generated:

```
#define Rte_Call_p1_a Rte_Call_c1_p1_a
```

5.2.6.2 “RTE Generation” Phase

There are no requirements on the *form* that the API mapping created during the “RTE Generation” phase should take. This is because the application header files defined during this phase are used by source-code components and therefore compatibility between the generated RTE and source-code components is automatic.

The RTE generator is required to produce the component data structure instances required by object-code components and multiple instantiated source-code components.

If multiple instantiation of a software-component is forbidden, then the API mapping specified for the “RTE Contract” phase (Section 5.2.6.1) defines the names of the generated functions. If multiple instantiation is possible, there are no corresponding requirements that define the name of the generated function since all accesses to the generated functions are performed via the component data structure which contains well-defined entries (Sections 5.4.2.5 and 5.4.2.5).

5.2.6.3 Function Elision

Using the “RTE Generation” phase API mapping, it is possible for the RTE generator to elide the use of generated RTE functions.

[SWS_Rte_01146] ⌈ If the API mapping elides an RTE function the “RTE Generation” phase API mapping mechanism shall ensure that the invoking component still receives a “return value” so that no changes to the AUTOSAR software-component are necessary. ⌋*(SRS_Rte_00051)*

In C, the elision of API calls can be achieved using a comma expression[2]

Example 5.8

As an example, consider the following component code:

```
Std_ReturnType s;
s = Rte_Send_p1_a(self,23);
```

Furthermore, assume that the communication attributes are specified such that the sender-receiver communication can be performed as a direct assignment and therefore no RTE API call needs to be generated. However, the component source cannot be modified and expects to receive an `Std_ReturnType` as the return. The "RTE Generation" phase API mapping could then be rewritten as:

```
#define Rte_Send_p1_a(s,a) (<var> = (a), RTE_E_OK)
```

Where `<var>` is the implementation dependent name for an RTE created cache between sender and receiver.

5.2.6.4 API Naming Conventions

An AUTOSAR software-component communicates with other components (including basic software) through ports and therefore the names that constitute the RTE API are formed from the combination of the API call's functionality (e.g. Call, Send) that defines the API root name and the access point through which the API operates.

For any API that operates through a port, the API's access point includes the port name.

A `SenderReceiverInterface` can support multiple data items and a `ClientServerInterface` can support multiple operations, any of which can be invoked through the requiring port by a client. The RTE API therefore needs a mechanism to indicate which data item/operation on the port to access and this is implemented by including the data item/operation name in the API's access point.

As described above, the RTE API mapping is responsible for mapping the RTE API name to the correct generated RTE function. The API mapping permits an RTE generator to include targeted optimization as well as removing the need to implement functions that act as routing functions from generic API calls to particular functions within the generated RTE.

For C and C++ the RTE API names introduce symbols into global scope and therefore the names are required to be prefixed with `Rte_` [SWS_Rte_01171].

[2] This is contrary to MISRA Rule 42 "*comma expression shall not be used except in the control expression of a for loop*". However, a comma expression is valid, legal, C and the elision cannot be achieved without a comma expression and therefore the rule must be relaxed.

5.2.6.5 API Parameters

All API parameters fall into one of two classes; parameters that are strictly read-only ("In" parameters) and parameters whose value may be modified by the API function ("In/Out" and "Out" parameters).

The type of these parameters is taken from the data element prototype or operation prototype in the interface that characterizes the port for which the API is being generated.

In the following, requirement [SWS_Rte_06806] reflects the standard defined by [27]. The remaining requirements are include to ensure the consistency between different RTE implementations.

[SWS_Rte_06804] ⌈ All input parameters using the `P2CONST` macro shall use `memclass AUTOMATIC` and `ptrclass RTE_APPL_DATA`. ⌋*(SRS_Rte_00060, SRS_BSW_00007)*

[SWS_Rte_06805] ⌈ All parameters using the `VAR` macro shall use `memclass AUTOMATIC`. ⌋*(SRS_Rte_00059, SRS_BSW_00007)*

[SWS_Rte_06806] ⌈ All output and bi-directional parameters (i.e. both input and output) parameters shall use the `P2VAR` macro. ⌋*(SRS_Rte_00061, SRS_BSW_00007)*

[SWS_Rte_06807] ⌈ All parameters using the `P2VAR` macro shall use `memclass AUTOMATIC` and `ptrclass RTE_APPL_DATA`. ⌋*(SRS_Rte_00059, SRS_Rte_00060, SRS_BSW_00007)*

- **"In" Parameters**

  [SWS_Rte_01017] ⌈ All input parameters that are a `Primitive Implementation Data Type` shall be passed by value. ⌋*(SRS_Rte_00059, SRS_Rte_00061)*

  [SWS_Rte_01018] ⌈ All input parameters that are of type `Structure Implementation Data Type` or `Union Implementation Data Type` shall be passed by reference. ⌋*(SRS_Rte_00060, SRS_Rte_00061)*

  [SWS_Rte_05107] ⌈ All input parameters that are an `Array Implementation Data Type` shall be passed as an array expression (that is a pointer to the array base type). ⌋*(SRS_Rte_00060, SRS_Rte_00061)*

  [SWS_Rte_07661] ⌈ All input parameters that are a data type of category `DATA_REFERENCE` shall be passed as a pointer to the data type specified by the `SwPointerTargetProps`. ⌋*(SRS_Rte_00059, SRS_Rte_00061)*

  [SWS_Rte_07086] ⌈ All input parameters that are passed by reference ([SWS_Rte_01018]) or passed as an array expression ([SWS_Rte_05107]) shall be declared as pointer to const with the means of the `P2CONST` macro. ⌋*(SRS_Rte_00060, SRS_BSW_00007)*

  Please note that the description of the `P2CONST` macro can be found in [28].

- **“Out” Parameters**

  [SWS_Rte_01019] ⌈ All output parameters that are of type Primitive Implementation Data Type shall be passed by reference. ⌋ *(SRS_Rte_00061)*

  [SWS_Rte_07082] ⌈ All output parameters that are of type Structure Implementation Data Type or Union Implementation Data Type shall be passed by reference. ⌋ *(SRS_Rte_00060, SRS_Rte_00061)*

  [SWS_Rte_05108] ⌈ All output parameters that are an Array Implementation Data Type shall be passed as an array expression (that is a pointer to the array base type). ⌋ *(SRS_Rte_00060, SRS_Rte_00061)*

  [SWS_Rte_07083] ⌈ All output parameters that are of type Pointer Implementation Data Type shall be passed as a pointer to the Pointer Implementation Data Type. ⌋ *(SRS_Rte_00059, SRS_Rte_00061)*

- **“In/Out” Parameters**

  [SWS_Rte_01020] ⌈ All bi-directional parameters (i.e. both input and output) that are of type Primitive Implementation Data Type or Structure Implementation Data Type or Union Implementation Data Type shall be passed by reference. ⌋ *(SRS_Rte_00059, SRS_Rte_00061)*

  [SWS_Rte_05109] ⌈ All bi-directional parameters (i.e. both input and output) that are an Array Implementation Data Type shall be passed as an array expression (that is a pointer to the array base type). ⌋ *(SRS_Rte_00061)*

  [SWS_Rte_07084] ⌈ All input, output and bi-directional parameters which related DataPrototype is typed or mapped to an Redefinition Implementation Data Type shall be treated according the kind of data type redefined by the Redefinition Implementation Data Type. The possible kinds of data types supported by RTE are listed in 5.3.4.2. ⌋ *(SRS_Rte_00059, SRS_Rte_00060, SRS_Rte_00061)*

In order to indicate the direction of the individual API parameters, the descriptions of the API signatures in this API reference chapter use the direction qualifiers ”IN”, ”OUT”, and ”INOUT”. These direction qualifiers are not part of the actual API prototypes. Especially, the user cannot expect that these direction qualifiers are available for the application.

Example 5.9

This would be the Rte_Write API generated for the example 5.6 (example of a two dimension array typed by an ImplementationDataType):

```
FUNC(Std_ReturnType, RTE_CODE) Rte_Write_<p>_<o>(P2CONST(uint8,
    AUTOMATIC, AUTOMATIC) data)
```

Which can be used in the SWC code:

```
status = Rte_Write_<p>_<o> (&array[0][0]);
```

5.2.6.6 Return Values

A subset of the RTE API's returning the values instead of using OUT Parameters. In the API section these API signatures defining a `<return>` value. In addition to the following rules some of the APIs might specify additionally const qualifiers.

[SWS_Rte_07069] ⌈ The RTE Generator shall determine the `<return>` type according the applicable `ImplementationDataType` of the `DataPrototype` for which the API provides access. ⌋*(SRS_Rte_00059)*

[SWS_Rte_08300] ⌈ A pointer return value of an RTE API shall be declared as pointer to const with the means of the `FUNC_P2CONST` macro or `P2CONST` if the pointer is not used to modify the addressed object. ⌋*(SRS_Rte_00059)*

Please note that the `FUNC_P2CONST` macro is applicable if the RTE API is implemented as an real function and the `P2CONST` might be used if the RTE API is implemented as a macro.

Requirement [SWS_Rte_08300] applies for instance for the RTE APIs `Rte_Prm`, `Rte_CData`, `Rte_IrvRead`, `Rte_IrvIRead` in the cases where the API grants access to composite data (arrays, structures, unions).

Please note, that the the implementation of the C data types are specified in section 5.3.4 "RTE Types Header File".

[SWS_Rte_07070] ⌈ If the `DataPrototype` is associated to a `Primitive Implementation Data Type` the RTE API shall return the value of the `DataPrototype` for which the API provides access. The type name shall be equal to the `shortName` of these `ImplementationDataType`. ⌋*(SRS_Rte_00059)*

Example 5.10

Consider an RTE API call return a primitive as defined in the example 5.3 for a singly instantiated SW-C. The signature of the API will be:

```
MyUint8 Rte_IRead_<re>_<p>_<o>(void);
```

Please note that the usage of Compiler Abstraction is not shown in the example.

[SWS_Rte_07071] ⌈ If the `DataPrototype` is associated to a `Structure Implementation Data Type` or `Union Implementation Data Type`, the RTE API shall return a pointer to a variable holding the `DataPrototype` value provided by the API. The type name shall be equal to the `shortName` of these `ImplementationDataType`. ⌋*(SRS_Rte_00059)*

Example 5.11

Consider an RTE API call return a structure as defined in the example 5.7 for a singly instantiated SW-C. The signature of the API will be:

```

FUNC_P2CONST(RecA, RTE_VAR_FAST_INIT, RTE_CODE)
```

```
Rte_IRead_<re>_<p>_<o>(void);
```

Please note that the usage of Compiler Abstraction assumes that the SwAddrMethod of the accessed VariableDataPrototype is named "'VAR_FAST_INIT"'. Further on the example does not respect the principles of API mapping.

[SWS_Rte_07072] ⌈ If the DataPrototype is associated to an Array Implementation Data Type the RTE API shall return an array expression (that is a pointer to the array base type) pointing to variable holding the value of the DataPrototype for which the API provides access. If the leaf ImplementationDataTypeElement is typed by a SwBaseType the array type name shall be equal to the nativeDeclaration attribute of the SwBaseType. If the leaf ImplementationDataTypeElement is typed by an ImplementationDataType the type name shall be equal to the shortName of these ImplementationDataType. ⌋*(SRS_Rte_00059)*

Example 5.12

Consider an RTE API call return an array as defined in the example 5.5 for a singly instantiated SW-C. The signature of the API will be:

```
FUNC_P2CONST(unsigned char, RTE_VAR_POWER_ON_INIT, RTE_CODE)
            Rte_IRead_<re>_<p>_<o>(void);
```

Please note that the usage of Compiler Abstraction assumes that the SwAddrMethod of the accessed VariableDataPrototype is named "'VAR_POWER_ON_INIT"'. Further on the example does not respect the principles of API mapping.

Example 5.13

Consider an RTE API call return an array as defined in the example 5.6 for a singly instantiated SW-C. The signature of the API will be:

```
FUNC_P2CONST(uint8, RTE_VAR_NO_INIT, RTE_CODE)
            Rte_IRead_<re>_<p>_<o>(void);
```

Please note that the usage of Compiler Abstraction assumes that the SwAddrMethod of the accessed VariableDataPrototype is named "'VAR_NO_INIT"'. Further on the example does not respect the principles of API mapping.

[SWS_Rte_07073] ⌈ If the DataPrototype is associated to a Pointer Implementation Data Type the RTE API shall return the value of the DataPrototype for which the API provides access. The type name shall be equal to the shortName of these ImplementationDataType. ⌋*(SRS_Rte_00059)* Please not that in this case the value is a pointer.

[SWS_Rte_07074] ⌈ If the DataPrototype is associated to a Redefinition Implementation Data Type the RTE Generator shall determine the API return value behaviour as described in [SWS_Rte_07070], [SWS_Rte_07071], [SWS_Rte_07072], [SWS_Rte_07073], [SWS_Rte_07074] according the referenced ImplementationDataType. Nevertheless except for Array Implementation Data Type the type name shall be equal to the shortName of these ImplementationDataType. ⌋*(SRS_Rte_00059)*

Please note that Redefinition Implementation Data Type might redefine an other Redefinition Implementation Data Type again.

5.2.6.7 Return References

A subset of the RTE API's returning a reference to the memory location where the data can be accessed instead of using IN/OUT Parameters. In the API section these API signatures defining a `<return reference>` value.

[SWS_Rte_06808] ⌈ A `<return reference>` shall use the `FUNC_P2VAR` or `P2VAR` macro. ⌋*(SRS_BSW_00007)*

[SWS_Rte_06809] ⌈ A `<return reference>` which uses either the `P2VAR` or the `FUNC_P2VAR` macro shall use `memclass AUTOMATIC` and `ptrclass RTE_DATA`. ⌋*(SRS_BSW_00007)*

[SWS_Rte_07076] ⌈ The RTE Generator shall determine the `<return reference>` type according the applicable ImplementationDataType of the DataPrototype for which the API provides access. ⌋*(SRS_Rte_00059)*

Please note, that the the implementation of the C data types are specified in section 5.3.4 "RTE Types Header File".

[SWS_Rte_07077] ⌈ If the DataPrototype is associated to a Primitive Implementation Data Type the RTE API shall return a pointer to variable holding the data of the value of the DataPrototype for which the API provides access. The type name shall be equal to the shortName of these ImplementationDataType. ⌋*(SRS_Rte_00059)*

Example 5.14

Consider an RTE API call return a reference to a primitive as defined in the example 5.3 for a singly instantiated SW-C. The signature of the API will be:

```
MyUint8 * Rte_IWriteRef_<re>_<p>_<o>(void);
```

Please note that the usage of Compiler Abstraction is not shown in the example.

[SWS_Rte_07078] ⌈ If the DataPrototype is associated to a Structure Implementation Data Type or Union Implementation Data Type the RTE API shall return a pointer to variable holding the value of the DataPrototype for which the API provides access. The type name shall be equal to the shortName of these ImplementationDataType. ⌋*(SRS_Rte_00059)*

Example 5.15

Consider an RTE API call return a reference to a structure as defined in the example 5.7 for a singly instantiated SW-C. The signature of the API will be:

```
RecA * Rte_IWriteRef_<re>_<p>_<o>(void);
```

Please note that the usage of Compiler Abstraction is not shown in the example.

[SWS_Rte_07079] ⌈ If the DataPrototype is associated to an Array Implementation Data Type the RTE API shall return an array expression (that is a pointer to the array base type) pointing to variable holding the value of the DataPrototype for which the API provides access. If the leaf ImplementationDataTypeElement is typed by a SwBaseType the array type name shall be equal to the nativeDeclaration attribute of the SwBaseType. If the leaf ImplementationDataTypeElement is typed by an ImplementationDataType the type name shall be equal to the shortName of these ImplementationDataType. ⌋*(SRS_Rte_00059)*

Example 5.16

Consider an RTE API call return a reference to an array as defined in the example 5.5 for a singly instantiated SW-C. The signature of the API will be:

```
unsigned char * Rte_IWriteRef_<re>_<p>_<o>(void);
```

Example 5.17

Consider an RTE API call return a reference to an array as defined in the example 5.6 for a singly instantiated SW-C. The signature of the API will be:

```
uint8 * Rte_IWriteRef_<re>_<p>_<o>(void);
```

Please note that the usage of Compiler Abstraction is not shown in the examples.

[SWS_Rte_07080] ⌈ If the DataPrototype is associated to a Pointer Implementation Data Type the RTE API shall return a pointer pointing to variable holding the value of the DataPrototype for which the API provides access. The type name shall be equal to the shortName of these ImplementationDataType. ⌋*(SRS_Rte_00059)* Please not that in this case the value is a pointer again.

[SWS_Rte_07081] ⌈ If the DataPrototype is associated to a Redefinition Implementation Data Type the RTE Generator shall determine the API return value behaviour as described in [SWS_Rte_07077], [SWS_Rte_07078], [SWS_Rte_07079], [SWS_Rte_07080], [SWS_Rte_07081] according the referenced ImplementationDataType. Nevertheless except for Array Implementation Data Type the

type name shall be equal to the shortName of these ImplementationDataType.
⌋(SRS_Rte_00059)

Please note that Redefinition Implementation Data Type might redefine an other Redefinition Implementation Data Type again.

5.2.6.8 Error Handling

In RTE, error and status information is defined with the data type Std_ReturnType, see Section 5.5.1.

It is possible to distinguish between infrastructure errors and application errors. Infrastructure errors are caused by a resource failure or an invalid input parameter. Infrastructure errors usually occur in the basic software or hardware along the communication path of a data element. Application errors are reported by a SW-C or by AUTOSAR services. RTE has the capability to treat application errors that are forwarded

- by return value in client server communication or
- by signal invalidation in sender receiver communication with data semantics.

Errors that are detected during an RTE API call are notified to the caller using the API's return value.

[SWS_Rte_01034] ⌈ Error states (including 'no error') shall only be passed as return value of the RTE API to the AUTOSAR SW-C. ⌋*(SRS_Rte_00094)*

Requirement [SWS_Rte_01034] ensures that, irrespective of whether the API is blocking or non-blocking, the error is collected at the same time the data is made available to the caller thus ensuring that both items are accessed consistently.

Certain RTE API calls operate asynchronously from the underlying communication mechanism. In this case, the return value from the API indicates only errors detected during that API call. Errors detected after the API has terminated are returned using a different mechanism [SWS_Rte_01111]. RTE also provides an 'implicit' API for direct access to virtually shared memory. This API does not return any errors. The underlying communication is decoupled. Instead, an API is provided to pick up the current status of the corresponding data element.

5.2.6.9 Success Feedback

The RTE supports the notification of results of transmission attempts to an AUTOSAR software-component.

The Rte_Feedback API [SWS_Rte_01083] or the Rte_IFeedback API [SWS_Rte_07367] can be configured to return the transmission result as either a blocking or non-blocking API or via activation of a runnable entity.

5.2.7 Unconnected Ports

[SWS_Rte_01329] ⌈ The RTE shall handle both require and provide ports that are not connected. ⌋*(SRS_Rte_00139)*

The handling of require ports as an error is described in requirement [SWS_Rte_05099].

[SWS_Rte_06030] ⌈ The RTE shall consider a `PRPortPrototype` as always connected. ⌋*(SRS_Rte_00139)*

Note: [SWS_Rte_06030] is the consequence of [TPS_SWCT_01573]. This is because a `PRPortPrototype` is logically an overlay of require and provide semantics hence the `PRPortPrototype` needs no further explicitly defined connection in the form of an `SwConnector` or signal mapping.

The API calls for unconnected ports are specified to behave as if the port was connected but the remote communication point took no action.

Unconnected require ports are regarded by the RTE generator as an invalid configuration (see [SWS_Rte_03019]) if the strict handling has been enabled (see [SWS_Rte_05099]).

5.2.7.1 Data Elements

5.2.7.1.1 Explicit Communication

[SWS_Rte_01330] ⌈ A `Rte_Read` API for an unconnected require port typed by a `SenderReceiverInterface` or `NvDataInterface` shall return the `RTE_E_UNCONNECTED` code and provide the `initValue` as if a sender was connected but did not transmit anything. ⌋*(SRS_Rte_00094, SRS_Rte_00139, SRS_Rte_00200)*

[SWS_Rte_07663] ⌈ A `Rte_DRead` API for an unconnected require port typed by a `SenderReceiverInterface` or `NvDataInterface` shall return the `initValue` as if a sender was connected but did not transmit anything. ⌋*(SRS_Rte_00139, SRS_Rte_00200)*

Requirements [SWS_Rte_01330] and [SWS_Rte_07663] apply to elements with "‘data"’ semantics and therefore "last is best"’ semantics. This means that the initial value will be returned.

[SWS_Rte_01331] ⌈ A blocking or non-blocking `Rte_Receive` API for an unconnected require port typed by a `SenderReceiverInterface` shall return `RTE_E_UNCONNECTED` immediately. ⌋*(SRS_Rte_00094, SRS_Rte_00107, SRS_Rte_00110, SRS_Rte_00139, SRS_Rte_00200)*

The existence of blocking and non-blocking `Rte_Read`, `Rte_DRead` and `Rte_Receive` API calls is controlled by the presence of `VariableAccess`es in the dataReceivePointByValue or dataReceivePointByArgument role, DataReceivedEvents and WaitPoints within the SW-C description [SWS_Rte_01288], [SWS_Rte_01289] and [SWS_Rte_01290].

[SWS_Rte_01344] ⌈ A blocking or non-blocking Rte_Feedback API for a VariableDataPrototype of an unconnected provide port shall return RTE_E_UNCONNECTED immediately. ⌋ *(SRS_Rte_00094, SRS_Rte_00122, SRS_Rte_00139)*

The existence of blocking and non-blocking Rte_Feedback API is controlled by the presence of VariableAccesses in the dataSendPoint role, DataSendCompletedEvents and WaitPoints within the SW-C description for a VariableDataPrototype with acknowledgement enabled, see [SWS_Rte_01283], [SWS_Rte_01284], [SWS_Rte_01285] and [SWS_Rte_01286].

[SWS_Rte_01332] ⌈ The Rte_Send or Rte_Write API for an unconnected provide port typed by a SenderReceiverInterface or NvDataInterface shall discard the input parameters and return RTE_E_OK. ⌋ *(SRS_Rte_00139)*

The existence of Rte_Send or Rte_Write is controlled by the presence of VariableAccesses in the dataSendPoint role within the SW/C description [SWS_Rte_01280] and [SWS_Rte_01281].

[SWS_Rte_03783] ⌈ The Rte_Invalidate API for an unconnected provide port typed by a SenderReceiverInterface shall return RTE_E_OK. ⌋ *(SRS_Rte_00139)*

The existence of Rte_Invalidate is controlled by the presence of VariableAccesses in the dataSendPoint role within the SW/C description for a VariableDataPrototype which is marked as invalidatable by an associated InvalidationPolicy. The handleInvalid attribute of the InvalidationPolicy has to be set to keep or replace to enable the invalidation support for this dataElement ([SWS_Rte_01282]).

5.2.7.1.2 Implicit Communication

[SWS_Rte_07378] ⌈ An Rte_IFeedback API for a VariableDataPrototype of an unconnected provide port shall return RTE_E_UNCONNECTED immediately. ⌋ *(SRS_Rte_00139, SRS_Rte_00185)*

The existence of an Rte_IFeedback API is controlled by the presence of VariableAccesses in the dataWriteAccess role, and DataWriteCompletedEvents within the SWC description for a VariableDataPrototype with acknowledgement enabled, see [SWS_Rte_07646], [SWS_Rte_07647].

[SWS_Rte_01346] ⌈ An Rte_IRead API for an unconnected require port typed by a SenderReceiverInterface or NvDataInterface shall return the initial value. ⌋ *(SRS_Rte_00139)*

The existence of Rte_IRead is controlled by the presence of a VariableAccess in the dataReadAccess role in the SW-C description [SWS_Rte_01301].

[SWS_Rte_01347] ⌈ An Rte_IWrite API for an unconnected provide port typed by a SenderReceiverInterface or NvDataInterface shall discard the written data. ⌋*(SRS_Rte_00139)*

The existence of Rte_IWrite is controlled by the presence of a VariableAccess in the dataWriteAccess role in the SW-C description [SWS_Rte_01302].

[SWS_Rte_03784] ⌈ An Rte_IInvalidate API for an unconnected provide port typed by a SenderReceiverInterface shall perform no action. ⌋*(SRS_Rte_00139)*

The existence of Rte_IInvalidate is controlled by the presence of a VariableAccess in the dataWriteAccess role in the SW-C description for a VariableDataPrototype which is marked as invalidatable by an associated InvalidationPolicy. The handleInvalid attribute of the InvalidationPolicy has to be set to keep or replace to enable the invalidation support for this dataElement ([SWS_Rte_03801]).

[SWS_Rte_03785] ⌈ An Rte_IStatus API for an unconnected require port typed by a SenderReceiverInterface shall return RTE_E_UNCONNECTED. ⌋*(SRS_Rte_00094, SRS_Rte_00139, SRS_Rte_00200)*

The existence of Rte_IStatus is controlled by the presence of a VariableAccess in the dataReadAccess role in the SW-C description for a VariableDataPrototype with data element outdated notification or data element invalidation [SWS_Rte_02600].

5.2.7.2 Mode Switch Ports

For the mode user an unconnected mode switch port behaves as if it was connected to a mode manager that never sends a mode switch notification.

[SWS_Rte_02638] ⌈ A Rte_Mode API for an unconnected mode switch port of a mode user shall return the initial state. ⌋*(SRS_Rte_00139)*

[SWS_Rte_02639] ⌈ Regarding the modes of an unconnected mode switch port of a mode user, the mode disabling dependencies on the initial mode shall be permanently active and the mode disabling dependencies on all other modes shall be inactive. ⌋*(SRS_Rte_00139)*

[SWS_Rte_02640] ⌈ Regarding the modes of an unconnected mode switch port of a mode user, RTE will only generate a SwcModeSwitchEvent for entering the initial mode which occurs directly after startup. ⌋*(SRS_Rte_00139)*

[SWS_Rte_02641] ⌈ The Rte_Switch API for an unconnected mode switch port of the mode manager shall discard the input parameters and return RTE_E_OK. ⌋*(SRS_Rte_00139)*

[SWS_Rte_02642] ⌈ A blocking or non blocking `Rte_SwitchAck` API for an unconnected mode switch port of the mode manager shall return `RTE_E_UNCONNECTED` immediately. ⌋*(SRS_Rte_00139)*

[SWS_Rte_01375] ⌈ A provided `mode switch port` of a `mode manager` shall be considered unconnected only if there are no connections at the composition level and no `ModeAccessPoint` exists for the provided `mode switch port` and no *synchronizedModeGroup* refers to the provided `mode switch port`. ⌋*(SRS_Rte_00139)*

5.2.7.3 Client-Server

[SWS_Rte_01333] ⌈ The `Rte_Result` API for an unconnected asynchronous require port typed by a `ClientServerInterface` shall return `RTE_E_UNCONNECTED` immediately. ⌋*(SRS_Rte_00094, SRS_Rte_00139, SRS_Rte_00200)*

[SWS_Rte_01334] ⌈ The `Rte_Call` API for an unconnected require port typed by a `ClientServerInterface` shall return `RTE_E_UNCONNECTED` immediately. ⌋*(SRS_Rte_00094, SRS_Rte_00139, SRS_Rte_00200)*

5.2.8 Non-identical port interfaces

Two ports are permitted to be connected provided that they are characterized by compatible, but not necessarily identical, interfaces. For the full definition of whether two interfaces are compatible, see the Software Component Template [2].

[SWS_Rte_01368] ⌈ The RTE generator shall report an error if the [constr_1036] and the [constr_1069] are violated so if two connected ports are connected by incompatible interfaces. ⌋*(SRS_Rte_00137)*

A significant issue in determining whether two interfaces are compatible is that the interface characterizing the require port may be a strict subset of the interface characterizing the provide port. This means that there may be provided data elements or operations for which there is no corresponding element in the require port. This can be imagined as a multi-strand wire between the two ports (the assembly connector) where each strand represents the connection between two data elements or operations, and where some of the strands from the ‘provide’ end are not connected to anything at the ‘require’ end.

Define, for the purposes of this section, an “unconnected element” as a data element or operation that occurs in the provide interface, but for which no corresponding data element or operation occurs in a particular R-Port’s interface.

[SWS_Rte_01369] ⌈ For each data element or operation within the provide interface, every connected requirer with an “unconnected element” must be treated as if it were not connected. ⌋*(SRS_Rte_00137)*

Note that requirement [SWS_Rte_01369] means that in the case of a 1:n Sender-Receiver the `Rte_Write` call may transmit to some but not all receivers.

The extreme is if all connected requirers have an “unconnected element”:

[SWS_Rte_01370] ⌈ For a data element or operation in a provide interface which is an unconnected element in every connected R-Port, the generated `Rte_Send`, `Rte_Write`, `Rte_IWrite`, or `Rte_IWriteRef` APIs must act as if the port were unconnected. ⌋*(SRS_Rte_00137)*

See Section 5.2.7 for the required behavior in this case.

5.3 RTE Modules

Figure 5.1 defines the relationship between header files and how those files are included by modules implementing AUTOSAR software-components and by general, non-component, code.

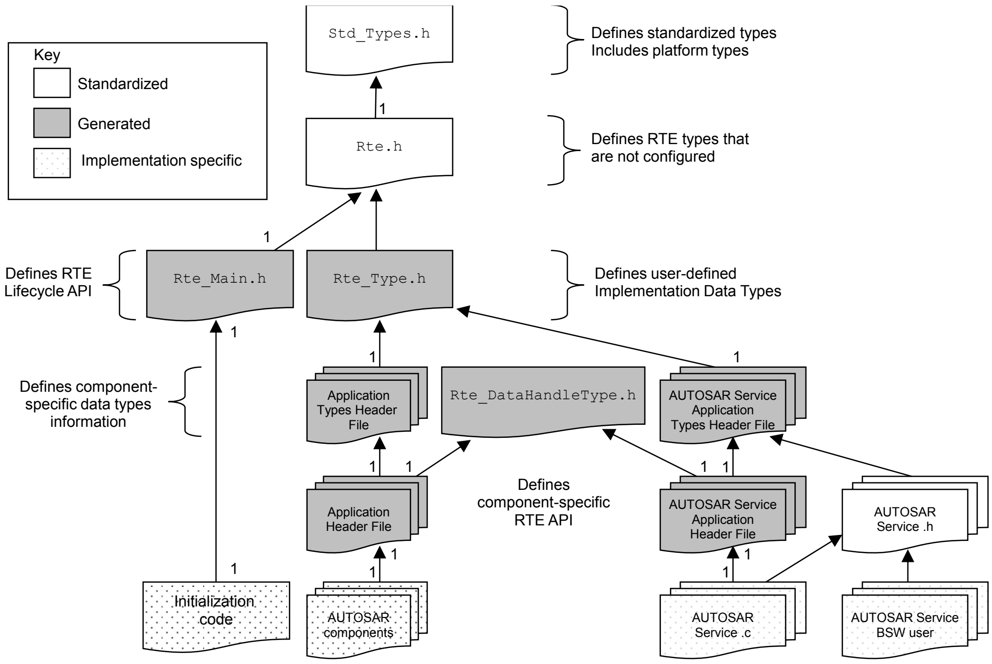


Figure 5.1: Relationships between RTE Header Files

The output of an RTE generator can consist of both generated code and configuration for “library” code that may be supplied as either object code or source code. Both configured and generated code reference standard definitions that are defined in the *RTE Header File*.

The relationship between the *RTE header file*, *Application Header File*s, the *Lifecycle Header File* and AUTOSAR software-components is illustrated in Figure 5.1.

In general a RTE can be partitioned in several files. The partitioning depends from the RTE vendors software design and generation strategy. Nevertheless it shall be possible to clearly identify code and header files which are part of the RTE module.

[SWS_Rte_07139] ⌈ Every file of the RTE beside Rte.h and Rte.c shall be named with the prefix `Rte_`. ⌋*(SRS_BSW_00300)*

5.3.1 RTE Header File

The RTE header file defines fixed elements of the RTE that do not need to be generated or configured for each ECU.

[SWS_Rte_01157] ⌈ For C/C++ AUTOSAR software-components, the name of the RTE header file shall be `Rte.h`. ⌋*(SRS_BSW_00300)*

Typically the contents of the RTE header file are fixed for any particular implementation and therefore it is not created by the RTE generator. However, customization for each generated RTE is not forbidden.

[SWS_Rte_01164] ⌈ The RTE header file shall include the file `Std_Types.h`. ⌋*(SRS_Rte_00149, SRS_Rte_00150, SRS_BSW_00353)*

The file `Std_Types.h` is the standard AUTOSAR file [29] that defines basic data types including platform specific definitions of unsigned and signed integers and provides access to the compiler abstraction.

The contents of the RTE header file are not restricted to standardized elements that are defined within this document – it can also contain definitions specific to a particular implementation.

5.3.2 Lifecycle Header File

[SWS_Rte_08309] ⌈ The RTE generator shall provide declarations for RTE and SchM Lifecycle APIs (see Section 5.8 and 6.7) through the Lifecycle header file. ⌋*(SRS_Rte_00051)*

[SWS_Rte_01158] ⌈ For C/C++ AUTOSAR software-components, the name of the lifecycle header file shall be `Rte_Main.h`. ⌋*(SRS_BSW_00300)*

[SWS_Rte_01159] ⌈ The lifecycle header file shall include the *RTE header file*. ⌋*(SRS_Rte_00051)*

5.3.3 Application Header File

The application header file [SRS_Rte_00087] is central to the definition of the RTE API. An application header file defines the RTE API and any associated data structures that are required by the SW-C to use the RTE implementation. But the application header file is not allowed to create objects in memory.

[SWS_Rte_01000] ⌈ The RTE generator shall create an application header file for each software-component type (excluding `ParameterSwComponentType`s and `NvBlockSwComponentType`s) defined in the input. ⌋*(SRS_Rte_00087, SRS_Rte_00024, SRS_Rte_00140)*

[SWS_Rte_03786] ⌈ The application header file shall not contain code that creates objects in memory. ⌋*(SRS_Rte_00087, SRS_BSW_00308)*

RTE generation consists of two phases; an initial “RTE Contract” phase and a second “RTE Generation” phase (see Section 2.3). Object-code components are compiled after the first phase of RTE generation and therefore the application header file should conform to the form of definitions defined in Sections 5.4.1 and 5.5.2. In contrast, source-code components are compiled after the second phase of RTE generation and therefore the RTE generator produces an optimized application header file based on knowledge of component instantiation and deployment.

5.3.3.1 File Name

[SWS_Rte_01003] ⌈ The name of the *Application Header File* of an AUTOSAR software component shall be `Rte_[Byps_]<name>.h`. `<name>` is the AUTOSAR software component type name. `[Byps_]` is an optional infix used when component wrapper method for bypass support is enabled for the related software component type (See chapter 4.9.2). ⌋*(SRS_BSW_00300)*

Example 5.18

The following declaration in the input XML:

```
<APPLICATION-SW-COMPONENT-TYPE>
    <SHORT-NAME>Source</SHORT-NAME>
</APPLICATION-SW-COMPONENT-TYPE>
```

should result in the application header file `Rte_Source.h` being generated when the component wrapper method for bypass support is disabled.

The component type name is used rather than the component instance name for two reasons; firstly the same component code is used for all component instances and, secondly, the component instance name is an internal identifier, and should not appear outside of generated code.

5.3.3.2 Scope

RTE supports two approaches for the scope of the application header file, a SW-C based, and a runnable based approach.

1. Always, the application header file provides only the API that is specific for one atomic SW-C, see [SWS_Rte_01004].
2. The scope of the application header file can be further reduced to one runnable by using the mechanism described in [SWS_Rte_02751].

Many of the RTE APIs are specific to runnables. The restrictions for the usage of the generated APIs are defined in the ‘Existence’ parts of each API subsection in 5.6. To prevent run time errors by the misuse of APIs that are not supported for a runnable, it is recommended to use the runnable based approach of the application header file.

[SWS_Rte_01004] ⌈ The application header file for a component shall contain only information relevant to that component. ⌋*(SRS_Rte_00087, SRS_Rte_00017, SRS_Rte_00167)*

[SWS_Rte_02751] ⌈ If the pre-compiler Symbol RTE_RUNNABLEAPI_<rn> is defined for a runnable with short name <rn> when the application header file is included, the application header file shall not declare APIs that are not valid to be used by the runnable *rn*. ⌋*(SRS_Rte_00017)*

For example, to restrict the application header file of the SW-C mySwc to the API of the runnable myRunnable, the following sequence can be used:

```
#define RTE_RUNNABLEAPI_myRunnable
#include <Rte_mySwc.h>

// runnable source code
```

Note that this mechanism does not support to restrict the application header file to the super set of two or more runnable APIs. In other words, runnables should be kept in separate source files, if the runnable based approach is used.

Requirements [SWS_Rte_01004] and [SWS_Rte_02751] mean that compile time checks ensure that a component (or runnable) that uses the application header file only accesses the generated data structures and functions to which it has been configured. Any other access, e.g. to fields not defined in the customized data structures or RTE API, will fail with a compiler error [SRS_Rte_00017].

The definitions of the RTE API contained in the application header file can be optimized during the “RTE Generation” phase when the mapping of software-components to ECUs and the communication matrix is known. Consequently multiple application header files must not be included in the same source module to avoid conflicting definitions of the RTE API definitions that the files contains.

Figure 5.2 illustrates the code structure for the declaration of the entry point of a runnable entity that provides the implementation for a ServerPort in component c1. The RTE generator is responsible for creating the API and tasks used to execute the server and the symbol name of the entry point is extracted from the attribute symbol of the runnable entity. The example shows that the first parameter of the entry point function is the software-component’s instance handle [SWS_Rte_01016].

```
#include <Rte_c1.h>

void
runnable_entry(Rte_Instance self)
{
  /* ... server code ... */
}
```

Figure 5.2: Skeleton server runnable entity

Figure 5.2 includes the component-specific application header file `Rte_c1.h` created by the RTE generator. The RTE generator will also create the supporting data structures and the task body to which the runnable is mapped.

The RTE is also responsible for preventing conflicting concurrent accesses when the runnable entity implementing the server operation is triggered as a result of a request from a client received via the communication service or directly via inter-task communication.

5.3.3.3 File Contents

Multiple application header file must not be included in the same module ([SWS_Rte_01004]) and therefore the file contents should contain a mechanism to enforce this requirement.

[SWS_Rte_01006] ⌈ An application header file shall include the following mechanism before any other definitions.

```
#ifdef RTE_APPLICATION_HEADER_FILE
#error Multiple application header files included.
#endif /* RTE_APPLICATION_HEADER_FILE */
#define RTE_APPLICATION_HEADER_FILE
```

⌋*(SRS_Rte_00087)*

[SWS_Rte_07131] ⌈ The application header file shall include the *Application Types Header File*. ⌋*(SRS_Rte_00087)*

The name of the *Application Types Header File* is defined in Section 5.3.6.

[SWS_Rte_07924] ⌈ The application header file shall include the *RTE Data Handle Types Header File* (see Section 5.3.5). ⌋*(SRS_Rte_00087)*

[SWS_Rte_01005] ⌈ The application header file shall be valid for both C and C++ source. ⌋*(SRS_Rte_00126, SRS_Rte_00138)*

Requirement [SWS_Rte_01005] is met by ensuring that all definitions within the application header file are defined using C linkage if a C++ compiler is used.

[SWS_Rte_03709] ⌈ All definitions within in the application header file shall be preceded by the following fragment;

```
#ifdef __cplusplus
extern "C" {
#endif /* __cplusplus */
```

⌋*(SRS_Rte_00126, SRS_Rte_00138)*

[SWS_Rte_03710] ⌈ All definitions within the application header file shall be suffixed by the following fragment;

```
#ifdef __cplusplus
} /* extern "C" */
#endif /* __cplusplus */
```

⌋*(SRS_Rte_00126, SRS_Rte_00138)*

5.3.3.3.1 Instance Handle

The RTE uses an instance handle to identify different instances of the same component type. The definition of the instance handle type [SWS_Rte_01148] is unique to each component type and therefore should be included in the application header file.

[SWS_Rte_01007] ⌈ The application header file shall define the type of the instance handle for the component. ⌋*(SRS_Rte_00012)*

All runnable entities for a component are passed the same instance handle type (as the first formal parameter [SWS_Rte_01016]) and can therefore use the same type definition from the component's application header file.

5.3.3.3.2 Runnable Entity Prototype

The application header file also includes a prototype for each runnable entity entry point ([SWS_Rte_01132]) and the API mapping ([SWS_Rte_01274]).

5.3.3.3.3 Initial Values

[SWS_Rte_05078] ⌈ The *Application Header File* shall define the init value of non-queued `VariableDataPrototype`s of sender receiver or non volatile data ports and typed by an `ImplementationDataType` or `ApplicationDataType` of category `VALUE`.

```
#define Rte_InitValue_<Port>_<DEPType> <initValue><suffix>
```

where `<Port>` is the `PortPrototype` `shortName`, `<DEPType>` is the `shortName` of the `VariableDataPrototype`, and `<initValue>` is the `initValue` specified in the `NonqueuedReceiverComSpec` respectively `NonqueuedSenderComSpec`. `<suffix>` shall be "U" for unsigned data types and empty for signed data types. ⌋*(SRS_Rte_00068, SRS_Rte_00087, SRS_Rte_00108)*

Note that the `initValue` defined may be subject to change due to the fact that for COM configuration it may be possible to change this value during ECU Configuration or even post-build time.

5.3.3.3.4 PerInstanceMemory

The *Application Header File* shall type definitions for PerInstanceMemory's as defined in Chapter 5.2.5, [SWS_Rte_07133].

5.3.3.3.5 RTE-Component Interface

The application header file defines the "interface" between a component and the RTE. The interface consists of the RTE API for the component and the prototypes for runnable entities. The definition of the RTE API requires that both relevant data structures and API calls are defined.

The data structures required to support the API are defined in the Application Header file (CDS) (see chapter 5.3.3), in the Application Types Header file (see chapter 5.3.6), in the RTE Types Header file (see chapter 5.3.1) and in the RTE Data Handle Types Header file (see chapter 5.3.5).

The data structure types are declared in the header files whereas the instances are defined in the generated RTE. The necessary data structures for object-code software-components are defined in chapter 5.5.2 and chapter 5.4.2.

The RTE generator is required [SWS_Rte_01004] to limit the contents of the application header file to only that information that is relevant to that component type. This requirement includes the definition of the API mapping. The API mapping is described in chapter 5.2.6.

[SWS_Rte_01276] ⌈ Only RTE API calls that are valid for the particular software-component type shall be defined within the component's application header file. ⌋*(SRS_Rte_00051, SRS_Rte_00017, SRS_Rte_00167)*

Requirement [SWS_Rte_01276] ensures that attempts to invoke invalid API calls will be rejected as a compile-time error [SRS_Rte_00017].

5.3.3.3.6 Application Errors

The concept of client server supports application specific error codes. Symbolic names for Application Errors are defined in the application header file to avoid conflicting definitions between several `AtomicSwComponentType`s mapped one ECU. See [SWS_Rte_02575] and [SWS_Rte_02576].

5.3.4 RTE Types Header File

The *RTE Types Header File* includes the RTE specific type declarations derived from the ImplementationDataTypes created from the definitions of AUTOSAR meta-model classes within the RTE generator's input. The available meta-model classes are defined by the AUTOSAR software-component template and include classes for defining primitive values, structures, arrays and pointers.

The types declared in the *RTE Types Header File* intend to be used for the implementation of RTE internal data buffers as well as for RTE API.

[SWS_Rte_01160] ⌈ The RTE generator shall create the *RTE Types Header File* including the type declarations corresponding to the ImplementationDataTypes defined in the input configuration as well as the RTE implementation types. ⌋

The RTE Data Types header file should be output for "RTE Contract" and "RTE Generation" phases.

5.3.4.1 File Contents

[SWS_Rte_02648] ⌈ The *RTE Types Header File* shall include the type declarations for all the AUTOSAR Data Types according to [SWS_Rte_07104], [SWS_Rte_07110], [SWS_Rte_06706], [SWS_Rte_06707], [SWS_Rte_06708] [SWS_Rte_07111], [SWS_Rte_07114], [SWS_Rte_07144], [SWS_Rte_07109] and [SWS_Rte_07148] depending on the values of attributes typeEmitter and nativeDeclaration but irrespective of their use by the generated RTE. ⌋

The attribute typeEmitter controls which part of the AUTOSAR toolchain is supposed to provide data type definitions. For legacy reasons the RTE generator is supposed to generate the corresponding data type if the ImplementationDataType defines no typeEmitter.

[SWS_Rte_06709] ⌈ The RTE generator shall generate the corresponding data type definition if the value of attribute typeEmitter is NOT defined. ⌋

[SWS_Rte_06710] ⌈ The RTE generator shall generate the corresponding data type definition if the value of attribute typeEmitter is set to "RTE". ⌋

[SWS_Rte_06711] ⌈ The RTE generator shall reject configurations where the value of the attribute typeEmitter is set to "RTE" and the ImplementationDataType references a SwBaseType without defined nativeDeclaration. ⌋

[SWS_Rte_06712] ⌈ The RTE generator shall silently not generate the corresponding data type definition if the value of attribute typeEmitter is set to anything else but "RTE". ⌋

This requirement ensures the availability of ImplementationDataTypes for the internal use in AUTOSAR software components.

Nevertheless the *RTE Types Header File* does not contain any data type belonging to an ImplementationDataType where typeEmitter is set to anything else but "RTE" regardless if the ImplementationDataType references SwBaseTypes and if this SwBaseTypes define nativeDeclarations.

[SWS_Rte_08732] ⌈ The RTE generator shall generate the type Rte_Cs_TransactionHandleType of the transaction handle for inter-ECU Client-Server communication as a structure:

```
typedef struct {
    [<IntegerTypeClientId> clientId;]
    <IntegerTypeSequenceCounter> sequenceCounter;
} Rte_Cs_TransactionHandleType;
```

where clientId and sequenceCounter contains the client identifier and sequence counter as speficied in [SWS_Rte_02649].

The IntegerTypeClientId and IntegerTypeSequenceCounter are integer types as specified in [SWS_Rte_08711] and [SWS_Rte_08712].

The existence of the clientId fields is specified in [SWS_Rte_08713]. ⌋

The types header file may need types in terms of BSW types (from the file Std_Types.h) or from the implementation specific RTE header file to declare types. However, since the RTE header file includes the file Std_Types.h already so only the RTE header file needs direct inclusion within the types header file.

[SWS_Rte_01163] ⌈ The *RTE Types Header File* shall include the *RTE Header File*. ⌋*(SRS_BSW_00353)*

5.3.4.2 Classification of Implementation Data Types

The type model ImplementationDataTypes is able to express following kinds of data types:

- Primitive Implementation Data Type
- Array Implementation Data Type
- Structure Implementation Data Type
- Union Implementation Data Type
- Redefinition Implementation Data Type
- Pointer Implementation Data Type

A *Primitive Implementation Data Type* is classified that it directly refers by its SwDataDefProps to a SwBaseType in the role baseType. The category attribute is set to VALUE.

An *Array Implementation Data Type* is classified that it defines ImplementationDataTypeElements for each dimension of the array. The swArraySize specifies the number of array elements of the dimension. The category attribute *Array Implementation Data Type* is set to ARRAY.

A *Structure Implementation Data Type* is categorized that it has ImplementationDataTypeElement's. The category attribute of the ImplementationDataType is set to STRUCTURE. Each ImplementationDataTypeElement it self can be one of the listed kinds again.

A *Union Implementation Data Type* is categorized that it has ImplementationDataTypeElement's. The category attribute of the ImplementationDataType is set to UNION. Each ImplementationDataTypeElement it self can be one of the listed kinds again.

A *Redefinition Implementation Data Type* is classified that it refers to other ImplementationDataTypes. The category attribute of the referring ImplementationDataType has to be set to TYPE_REFERENCE.

A *Pointer Implementation Data Type* is classified that its SwDataDefProps has a swPointerTargetProps attribute. The swDataDefProps in the role swPointerTargetProps is specifying the target to which the pointer refers. The category attribute of the ImplementationDataType has to be set to DATA_REFERENCE.

5.3.4.3 Primitive Implementation Data Type

The *RTE Types Header File* declares C types for all Primitive Implementation Data Types where the referred BaseType has a nativeDeclaration attribute.

[SWS_Rte_07104] ⌈ For each Primitive Implementation Data Type with a nativeDeclaration attribute, the *RTE Types Header File* shall include the corresponding type declaration as:

```
typedef <nativeDeclaration> <name>;
```

where <nativeDeclaration> is the nativeDeclaration attribute of the referred BaseType and <name> is the Implementation Data Type symbol of the Primitive Implementation Data Type. ⌋*(SRS_Rte_00055, SRS_Rte_00166, SRS_Rte_00168, SRS_BSW_00353)*

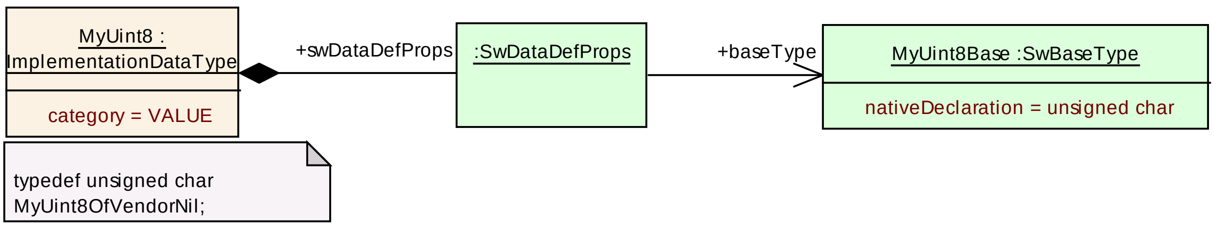


Figure 5.3: Primitive Implementation Data Type

Note: All Primitive Implementation Data Types where the referred Base-Type has **no** nativeDeclaration attribute resulting not in a type declaration. This is intended to prevent the redeclaration of the predefined Standard Types and Platform Types.

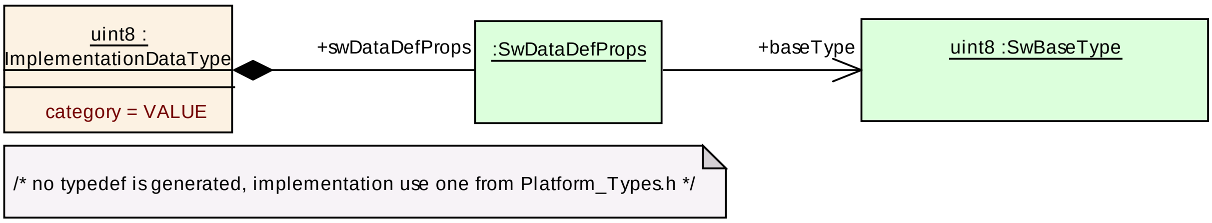


Figure 5.4: Primitive Implementation Data Type included from Platform_Types.h

[SWS_Rte_07105] ⌈ If more than one Primitive Implementation Data Type with equal shortName and equal nativeDeclaration attribute of the referred BaseType are defined, the *RTE Types Header File* shall include only once the corresponding type declaration according to [SWS_Rte_07104]. ⌋*(SRS_Rte_00165)*

Note: This avoids the redeclaration of C types due to the multiple descriptions of equivalent Primitive Implementation Data Types in the ECU extract.

5.3.4.4 Array Implementation Data Type

In addition to the primitive data-types defined in the previous section, it is also necessary for the RTE generator to declare composite data-types: arrays and records.

An array definition following information:

- the array type
- the number of dimensions
- the number of elements for each dimension.

[SWS_Rte_07110] ⌈ For each Array Implementation Data Type which leaf ImplementationDataTypeElement is typed by a BaseType, the *RTE Types Header File* shall include the corresponding type declaration as:

```
typedef <nativeDeclaration> <name>[<size 1>]{[<size 2>]...[<size n>]};
```

where <nativeDeclaration> is the nativeDeclaration attribute of the referred BaseType,

<name> is the Implementation Data Type symbol of the Array Implementation Data Type,

[<size x>] is the arraySize of the Array's ImplementationDataTypeElement.

For each array dimension defined by one Array's ImplementationDataTypeElement one array dimension definition [<size x>] is defined. The array dimension definitions [<size 1>], [<size 2>] ... [<size n>] ordered from the root to the leaf ImplementationDataTypeElement. ⌋*(SRS_Rte_00055, SRS_Rte_00164)*

[SWS_Rte_07111] ⌈ For each Array Implementation Data Type which leaf ImplementationDataTypeElement is typed by an ImplementationDataType, the *RTE Types Header File* shall include the corresponding type declaration as:

```
typedef <type> <name>[<size 1>]{[<size 2>]...[<size n>]};
```

where <type> is the shortName of the referred ImplementationDataType,

<name> is the Implementation Data Type symbol of the Array Implementation Data Type,

[<size x>] is the arraySize of the Array's ImplementationDataTypeElement. For each array dimension defined by one Array's ImplementationDataTypeElement one array dimension definition [<size x>] is defined.

The array dimension definitions [<size 1>], [<size 2>] ... [<size n>] ordered from the root to the leaf ImplementationDataTypeElement. ⌋*(SRS_Rte_00055, SRS_Rte_00164)*

[SWS_Rte_06706] ⌈ For each Array Implementation Data Type which last ImplementationDataTypeElement is of category STRUCTURE, the RTE *Types Header File* shall include the corresponding type declaration as:

```
typedef struct { <elements> } <name>[<size 1>]{[<size 2>]...[<size n>]};
```

where <elements> is the record element specification and

<name> is the Implementation Data Type symbol of the Array Implementation Data Type.

For each record element defined by one ImplementationDataTypeElement one record element specification <elements> is defined. The record element specifications are ordered according the order of the related ImplementationDataTypeElements in the input configuration.
Sequent record elements are separated with a semicolon.

[<size x>] is the arraySize of the Array's ImplementationDataTypeElement. For each array dimension defined by one Array's ImplementationDataTypeElement one array dimension definition [<size x>] is defined.

The array dimension definitions [<size 1>], [<size 2>] ... [<size n>] ordered from the root to the last ImplementationDataTypeElement belonging to the array definition. ⌋*(SRS_Rte_00055, SRS_Rte_00164)*

The definition of the record element specification is defined in section 5.3.4.6.

[SWS_Rte_06707] ⌈ For each Array Implementation Data Type which last ImplementationDataTypeElement is of category UNION, the RTE *Types Header File* shall include the corresponding type declaration as:

`typedef union { <elements> } <name>[<size 1>]{[<size 2>]...[<size n>]};`

where `<elements>` is the record element specification and

`<name>` is the `Implementation Data Type symbol` of the `Array Implementation Data Type`.

For each record element defined by one `ImplementationDataTypeElement` one record element specification `<elements>` is defined. The record element specifications are ordered according the order of the related `ImplementationDataTypeElement`s in the input configuration.
Sequent record elements are separated with a semicolon.

`[<size x>]` is the `arraySize` of the Array's `ImplementationDataTypeElement`. For each array dimension defined by one Array's `ImplementationDataTypeElement` one array dimension definition `[<size x>]` is defined.

The array dimension definitions `[<size 1>], [<size 2>] ... [<size n>]` ordered from the root to the last `ImplementationDataTypeElement` belonging to the array definition. ⌋*(SRS_Rte_00055, SRS_Rte_00164)*

The definition of the record element specification is defined in section 5.3.4.6.

[SWS_Rte_06708] ⌈For each `Array Implementation Data Type` which last `ImplementationDataTypeElement` is of category `DATA_REFERENCE`, the RTE *Types Header File* shall include the corresponding type declaration as:

`typedef <tqlA> <addtqlA> <type> * <tqlB> <addtqlB> <name> [<size 1>]{[<size 2>]...[<size n>]};`

where `<name>` is the `Implementation Data Type symbol` of the `Array Implementation Data Type` and

`[<size x>]` is the `arraySize` of the Array's `ImplementationDataTypeElement`. For each array dimension defined by one Array's `ImplementationDataTypeElement` one array dimension definition `[<size x>]` is defined. The array dimension definitions `[<size 1>], [<size 2>] ... [<size n>]` ordered from the root to the last `ImplementationDataTypeElement` belonging to the array definition. ⌋*(SRS_Rte_00055, SRS_Rte_00164)*

For the definition of `<tqlA>` and `<tqlB>` see [SWS_Rte_07149] and [SWS_Rte_07166].

For the definition of `<addtqlA>` and `<addtqlB>` see [SWS_Rte_07036] and [SWS_Rte_07037].

[SWS_Rte_07112] ⌈ If more than one `Array Implementation Data Type` with equal `shortName` of the `ImplementationDataType` and equal `nativeDeclaration` attribute of the referred `BaseType` are defined, the *RTE Types Header File* shall include only once the corresponding type declaration according to [SWS_Rte_07110]. ⌋*(SRS_Rte_00165)*

[SWS_Rte_07113] ⌈ If more than one Array Implementation Data Type with equal shortName of the ImplementationDataType and equal shortName of the referred ImplementationDataType are defined, the *RTE Types Header File* shall include only once the corresponding type declaration according to [SWS_Rte_07111]. ⌋*(SRS_Rte_00165)*

Note: This avoids the redeclaration of C types due to the multiple descriptions of equivalent Array Implementation Data Types in the ECU extract.

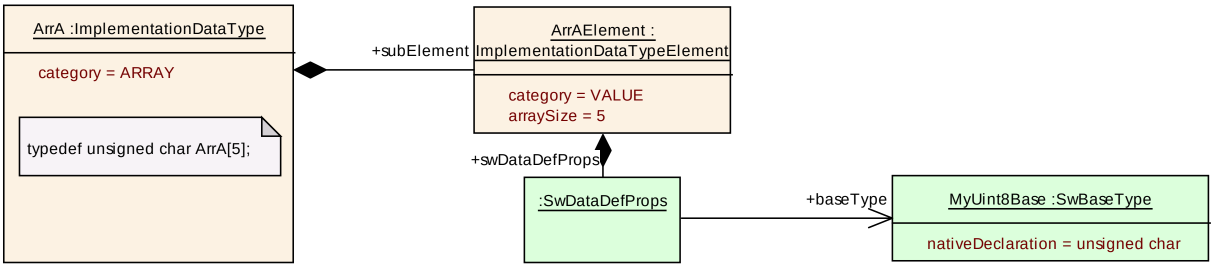


Figure 5.5: Example of a single dimension array typed by an BaseType

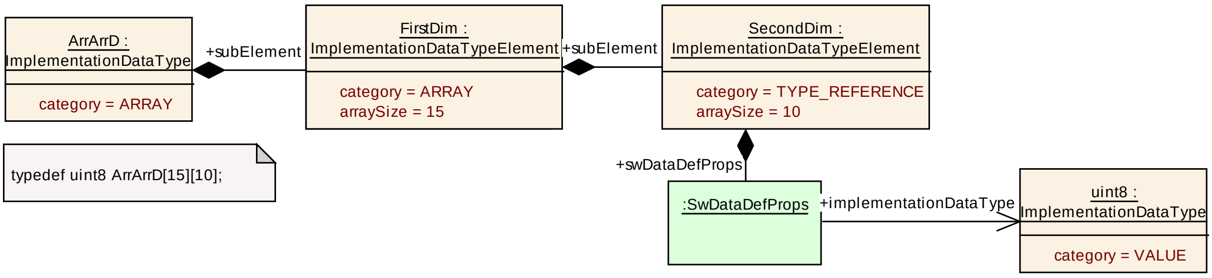


Figure 5.6: Example of a two dimension array typed by an ImplementationDataType

ANSI C does not allow a type declaration to have zero elements and therefore we require that the “number of elements” to be a positive integer.

[constr_9042] Array Implementation Data Types needs at least one element ⌈ The arraySize defining number of elements in one dimension of an *Array Implementation Data Type* shall be an integer that is ≥ 1 for each dimension. ⌋

5.3.4.5 Structure Implementation Data Type and Union Implementation Data Type

[SWS_Rte_07114] ⌈ For each Structure Implementation Data Type, the *RTE Types Header File* shall include the corresponding type declaration as:

```
typedef struct { <elements> } <name>;
```

where <elements> is the record element specification and <name> is the Implementation Data Type symbol of the Structure Implementation Data Type. For each record element defined by one ImplementationDataTypeElement one record element specification <elements> is defined. The record element specifications are ordered according the order of the related `ImplementationDataTypeElement`s in the input configuration. Sequent record elements are separated with a semicolon. ⌋*(SRS_Rte_00055, SRS_Rte_00164)*

5.3.4.6 Union Implementation Data Type

[SWS_Rte_07144] ⌈ For each `Union Implementation Data Type`, the *RTE Types Header File* shall include the corresponding type declaration as:

`typedef union { <elements> } <name>;`

where `<elements>` is the union element specification and `<name>` is the `Implementation Data Type symbol` of the `Union Implementation Data Type`. For each union element defined by one `ImplementationDataTypeElement` one union element specification `<elements>` is defined. The union element specifications are ordered according the order of the related `ImplementationDataTypeElement`s in the input configuration. Sequent union elements are separated with a semicolon. ⌋*(SRS_Rte_00055, SRS_Rte_00164)*

[SWS_Rte_07115] ⌈ Record and Union element specifications `<elements>` shall be generated as

`<nativeDeclaration> <name>;`

if the `ImplementationDataTypeElement` has the `category` attribute set to `VALUE` and if it refers to an `BaseType`. The meaning of the fields is identical to [SWS_Rte_07104] ⌋*(SRS_Rte_00055, SRS_Rte_00164)*

[SWS_Rte_07116] ⌈ Record and Union element specifications `<elements>` shall be generated as

`<type> <name>;`

if the `ImplementationDataTypeElement` has the `category` attribute set to `TYPE_REFERENCE` and if it refers to an `ImplementationDataType`. `<type>` is the `Implementation Data Type symbol` of the referred `ImplementationDataType` and `<name>` is the `shortName` of the `ImplementationDataTypeElement`. ⌋*(SRS_Rte_00055, SRS_Rte_00164)*

[SWS_Rte_07117] ⌈ Record and Union element specifications `<elements>` shall be generated as

`<nativeDeclaration> <name>[<size 1>]{[<size 2>]...[<size n>]};`

if the `ImplementationDataTypeElement` has the `category` attribute set to `ARRAY` and which leaf `ImplementationDataTypeElement` has the `category` attribute set to `VALUE` and is typed by an `BaseType`. The meaning and order of the fields is identical to [SWS_Rte_07110] ⌋*(SRS_Rte_00055, SRS_Rte_00164)*

[SWS_Rte_07118] ⌈ Record and Union element specifications <elements> shall be generated as

```
<type> <name>[<size 1>]{[<size 2>]...[<size n>]};
```

if the ImplementationDataTypeElement has the category attribute set to ARRAY and which leaf ImplementationDataTypeElement has the category attribute set to TYPE_REFERENCE and is typed by an ImplementationDataType. The meaning and order of the fields is identical to [SWS_Rte_07111] ⌋*(SRS_Rte_00055, SRS_Rte_00164)*

[SWS_Rte_07119] ⌈ Record and Union element specifications <elements> shall be generated as

```
struct { <elements> } <name>;
```

if the ImplementationDataTypeElement has the category attribute set to STRUCTURE. The meaning and order of the fields is identical to [SWS_Rte_07114] Sequent elements are separated with a semicolon. ⌋*(SRS_Rte_00055, SRS_Rte_00164)*

[SWS_Rte_07145] ⌈ Record and Union element specifications <elements> shall be generated as

```
union { <elements> } <name>;
```

if the ImplementationDataTypeElement has the category attribute set to UNION. The meaning and order of the fields is identical to [SWS_Rte_07144]. Sequent elements are separated with a semicolon. ⌋*(SRS_Rte_00055, SRS_Rte_00164)*

[SWS_Rte_07146] ⌈ Pointer element specifications <elements> shall be generated as

```
<tqlA> <addtqlA> <type> * <tqlB> <addtqlB> <name>;
```

if the ImplementationDataTypeElement has the category attribute set to DATA_REFERENCE where <name> is the shortName of the ImplementationDataTypeElement. ⌋*(SRS_Rte_00055, SRS_Rte_00164)*

For the definition of <tqlA> and <tqlB> see [SWS_Rte_07149] and [SWS_Rte_07166].

For the definition of <addtqlA> and <addtqlB> see [SWS_Rte_07036] and [SWS_Rte_07037].

For the definition of <type> see [SWS_Rte_07162], [SWS_Rte_07163].

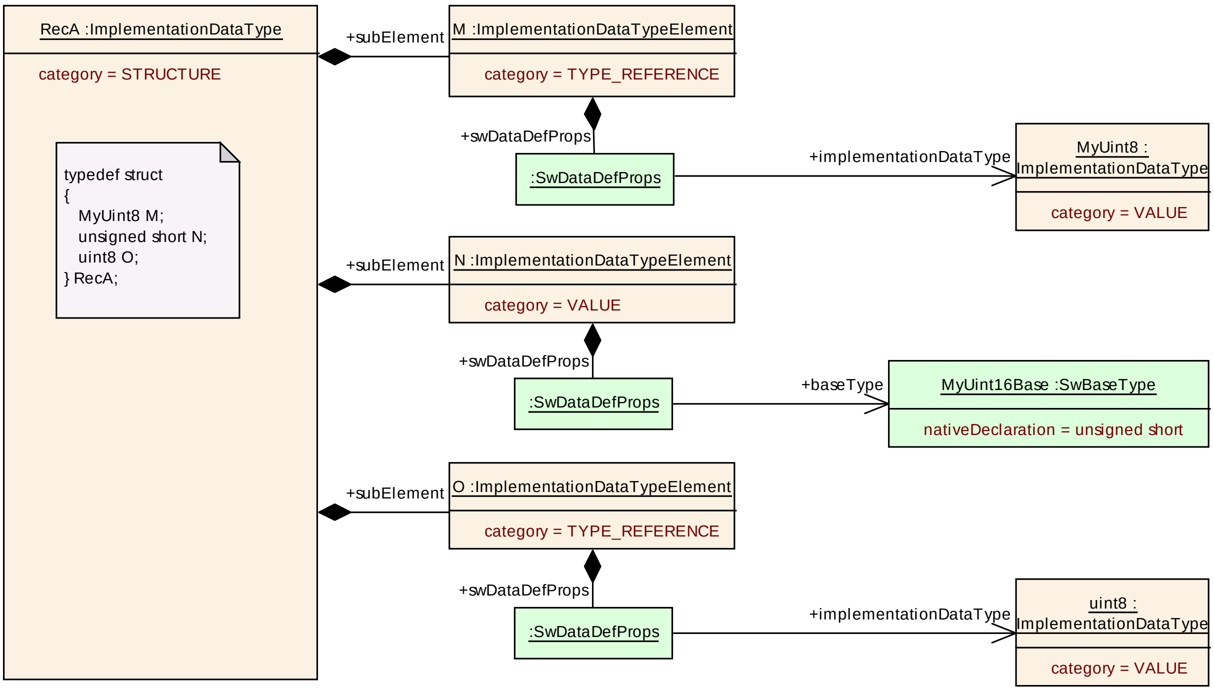


Figure 5.7: Example of a structure type

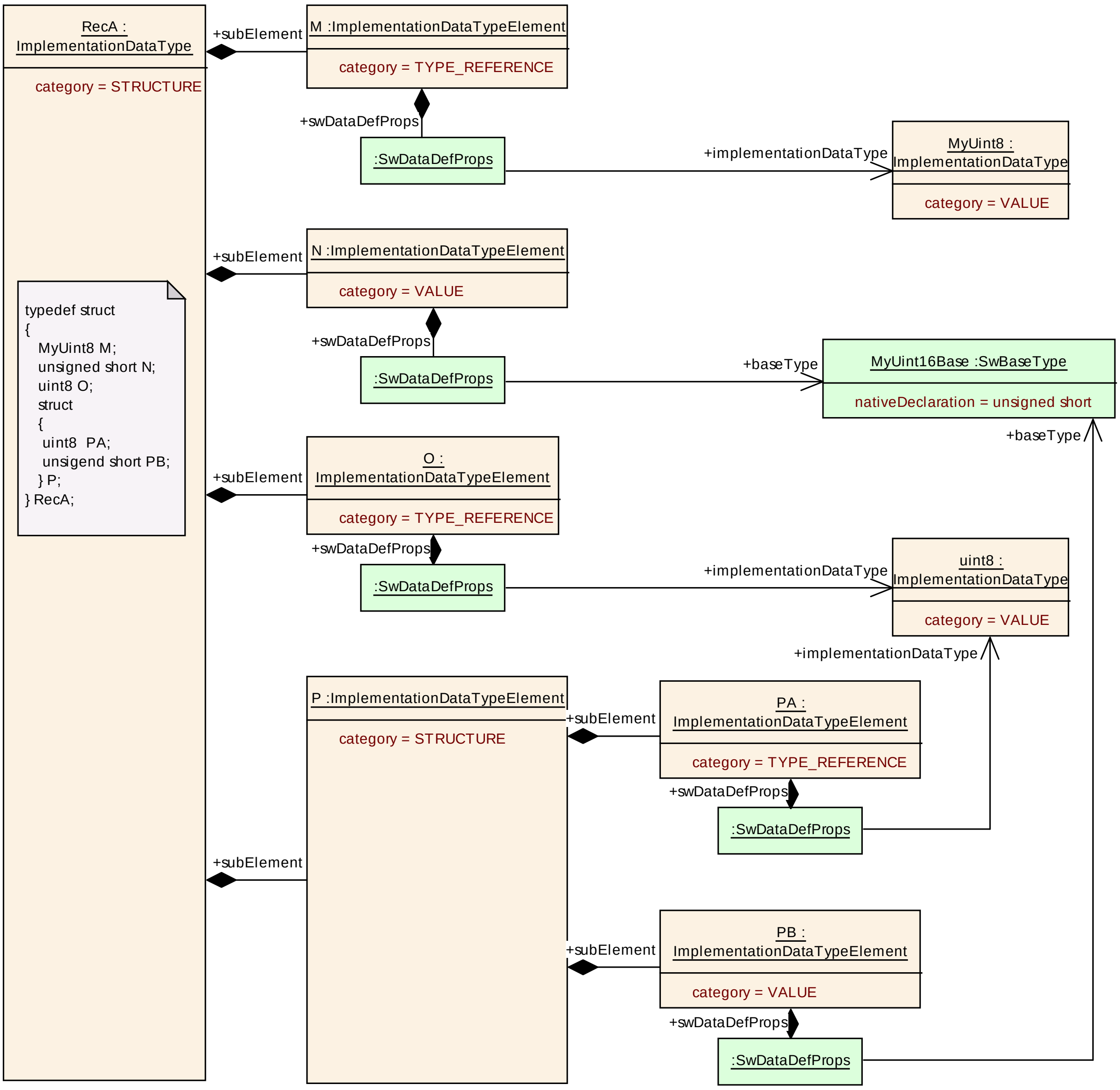


Figure 5.8: Example of a nested structure type

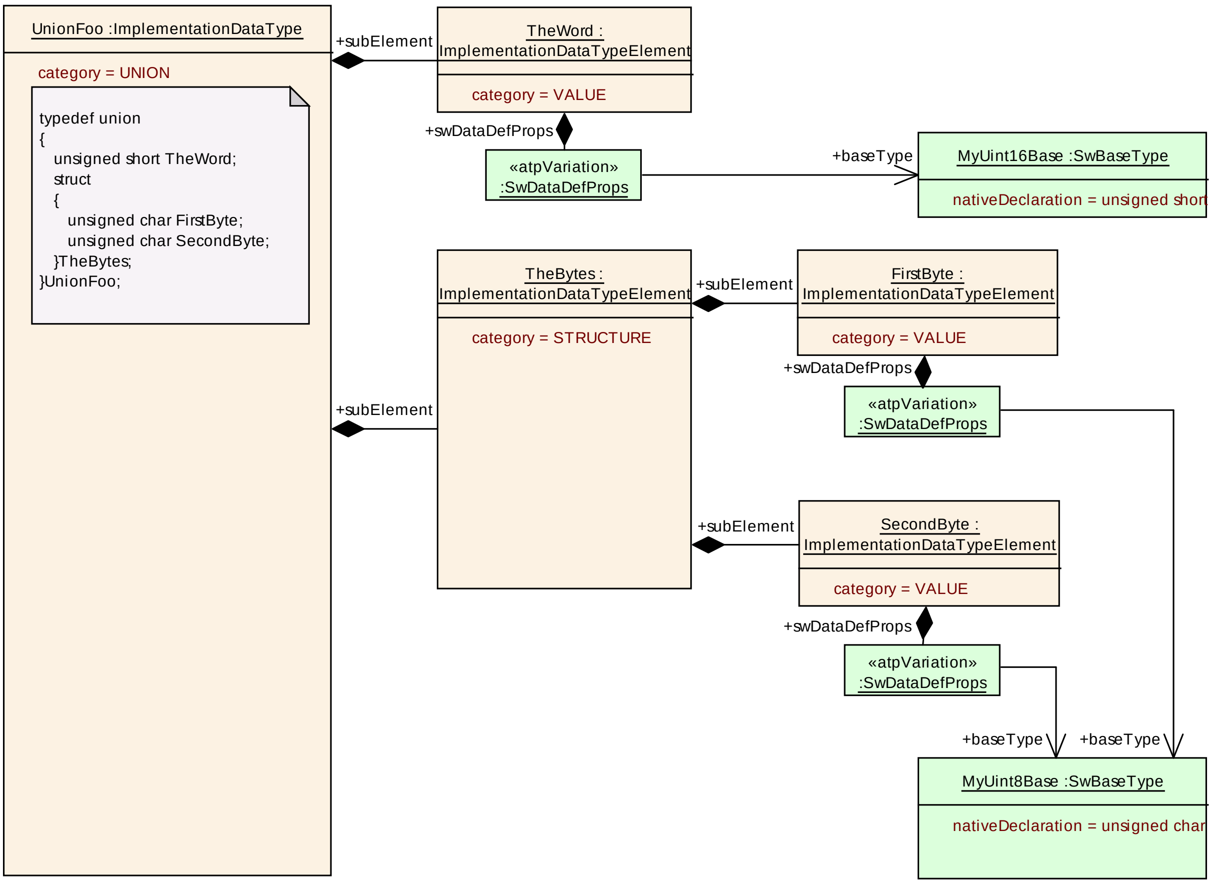


Figure 5.9: Example of a union type

[SWS_Rte_07107] ⌈ If more than one `Structure Implementation Data Type` or `Union Implementation Data Type` with equal `shortName` of the `ImplementationDataType` are defined, the *RTE Types Header File* shall include only once the corresponding type declaration according to [SWS_Rte_07114] or [SWS_Rte_07144]. ⌋*(SRS_Rte_00165)*

Note: This avoids the redeclaration of C types due to the multiple descriptions of equivalent `Structure Implementation Data Types` and `Union Implementation Data Types` in the ECU extract.

ANSI C does not allow a `struct` to have zero elements and therefore we require that a record include at least one element.

[constr_9043] `Structure Implementation Data Types` needs at least one element ⌈ A structure shall include at least one element defined by a `ImplementationDataTypeElement`. ⌋

A union data type describes a kind of structural overlay. Defining only one sub element of a `union` ist therefore not reasonable and indicates an error.

[constr_9044] *Union Implementation Data Type* shall include at least two elements ⌈ A *Union Implementation Data Type* shall include at least two elements defined by `ImplementationDataTypeElements`. ⌋

5.3.4.7 Implementation Data Type redefinition

[SWS_Rte_07109] ⌈ For each Redefinition Implementation Data Type which is typed by an ImplementationDataType, the *RTE Types Header File* shall include the corresponding type declaration as:

```
typedef <type> <name>;
```

where `<type>` is the Implementation Data Type symbol of the referred ImplementationDataType and `<name>` is the Implementation Data Type symbol of the Primitive Implementation Data Type. ⌋*(SRS_Rte_00055, SRS_Rte_00166)*

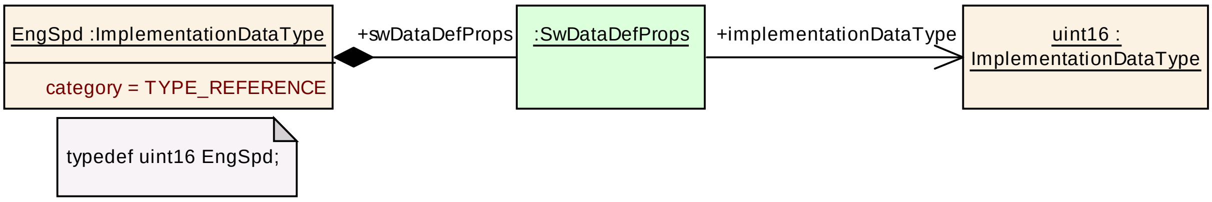


Figure 5.10: Example of an Implementation Data Type redefinition

[SWS_Rte_07167] ⌈ If more than one Redefinition Implementation Data Types with equal shortNames which are referring to compatible Implementation-DataTypes with identical shortNames are defined, the *RTE Types Header File* shall include only once the corresponding type declaration according to [SWS_Rte_07109]. ⌋*(SRS_Rte_00165)*

Note: This avoids the redeclaration of C types due to the multiple descriptions of equivalent Redefinition Implementation Data Type in the ECU extract.

5.3.4.8 Pointer Implementation Data Type

[SWS_Rte_07148] ⌈ For each Pointer Implementation Data Type, the *RTE Types Header File* shall include the corresponding type declaration as:

```
typedef <tqlA> <addtqlA> <type> * <tqlB> <addtqlB> <name>;
```

where `<name>` is the Implementation Data Type symbol of the Pointer Implementation Data Type. ⌋*(SRS_Rte_00055, SRS_Rte_00166)*

[SWS_Rte_07149] ⌈ `<tqlA>` (type qualifier A) of a Pointer Implementation Data Type ([SWS_Rte_07148]) or *Pointer element specifications* ([SWS_Rte_07146]) shall be set to `const` if the `swImplPolicy` of the sw-PointerTargetProps is set to const and shall be omitted for all other values of `swImplPolicy`. ⌋*(SRS_Rte_00055, SRS_Rte_00166)*

[SWS_Rte_07166] ⌈ `<tqlB>` (type qualifier B) of a Pointer Implementation Data Type ([SWS_Rte_07148]) or *Pointer element specifications* ([SWS_Rte_07146]) shall be set to `const` if the `swImplPolicy` of the Sw-

DataDefProps of the ImplementationDataType respectively ImplementationDataTypeElement is set to const and shall be omitted for all other values of swImplPolicy. ⌋*(SRS_Rte_00055, SRS_Rte_00166)*

[SWS_Rte_07036] ⌈ <addtqlA> (additional type qualifier A) of a Pointer Implementation Data Type ([SWS_Rte_07148]) or *Pointer element specification*s ([SWS_Rte_07146]) shall be set to the content of the additionalNativeTypeQualifier attribute of the swPointerTargetProps if the attribute exists and shall be omitted if such additionalNativeTypeQualifier attribute dose not exist. ⌋*(SRS_Rte_00055, SRS_Rte_00166)*

[SWS_Rte_07037] ⌈ <addtqlB> (additional type qualifier B) of a Pointer Implementation Data Type ([SWS_Rte_07148]) or *Pointer element specification*s ([SWS_Rte_07146]) shall be set to the content of the additionalNativeTypeQualifier attribute of the SwDataDefProps of the ImplementationDataType respectively ImplementationDataTypeElement and shall be omitted if such additionalNativeTypeQualifier attribute dose not exist. ⌋*(SRS_Rte_00055, SRS_Rte_00166)*

[SWS_Rte_07162] ⌈ <type> shall be set to the nativeDeclaration attribute of the referred BaseType if the targetCategory of a Pointer Implementation Data Type ([SWS_Rte_07148]) or *Pointer element specification*s ([SWS_Rte_07146]) is set to VALUE ⌋*(SRS_Rte_00055, SRS_Rte_00166)*

[SWS_Rte_07163] ⌈ <type> shall be the shortName of the referred ImplementationDataType if the targetCategory of a Pointer Implementation Data Type ([SWS_Rte_07148]) or *Pointer element specification*s ([SWS_Rte_07146]) is set to TYPE_REFERENCE ⌋*(SRS_Rte_00055, SRS_Rte_00166)*

[SWS_Rte_07169] ⌈ If more than one Pointer Implementation Data Types with equal shortNames which are resulting in the same C pointer type declaration are defined, the *RTE Types Header File* shall include only once the corresponding type declaration according to [SWS_Rte_07148]. ⌋*(SRS_Rte_00165)*

Note: This avoids the redeclaration of C types due to the multiple descriptions of equivalent Pointer Implementation Data Type in the ECU extract.

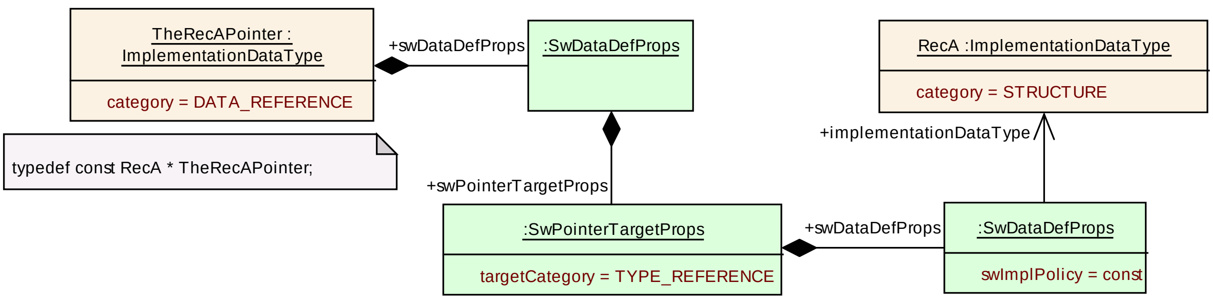


Figure 5.11: Example of a Pointer Implementation Data Type

5.3.4.9 ImplementationDataTypes with VariationPoints

[SWS_Rte_06539] ⌈

The RTE Generator shall wrap each code related to ImplementationDataTypeElements which are subject to variability in Structure Implementation Data Type and Union Implementation Data Type (see 4.22 if the variability shall be implemented.

```
#if (<condition>)

<elements>

#endif
```

where <condition> are the *condition value macro*(s) of the VariationPoints according table 4.22 and

<elements> is the code according invariant ImplementationDataTypeElements (see also [SWS_Rte_07115], [SWS_Rte_07116], [SWS_Rte_07117], [SWS_Rte_07118], [SWS_Rte_07119], [SWS_Rte_07145], [SWS_Rte_07146])

⌋*(SRS_Rte_00201)*

[SWS_Rte_06540] ⌈ The RTE Generator shall implement the <size x> of an Array Implementation Data Type for each arraySize which is subject to variability with the corresponding *attribute value macro* according table 4.22 if the variability shall be implemented. ⌋*(SRS_Rte_00201)*

5.3.4.10 Naming of data types

The Implementation Data Type symbol is defined as follows:

[SWS_Rte_06716] ⌈ The Implementation Data Type symbol shall be the shortName of the ImplementationDataType if no symbol attribute for this ImplementationDataTypeis defined. ⌋*(SRS_Rte_00167)*

Example 5.19

The Primitive Implementation Data Type in example 5.3 results in the type definition:

```
/* RTE Types Header File */
typedef unsigned char MyUint8;
```

[SWS_Rte_06717] ⌈ The Implementation Data Type symbol shall be the value of the SymbolProps.symbol attribute of the ImplementationDataType if the symbol attribute is defined. ⌋*(SRS_Rte_00167)*

[SWS_Rte_06718] ⌈ If the *RTE Types Header File* contains a generated C data type whose Implementation Data Type symbol differs from the ImplementationDataType shortName, the *Application Type Header Files* of each software component using the type shall contain a definition which redefines the Implementation Data Type symbol to the shortName of the ImplementationDataType. ⌋*(SRS_Rte_00167)*

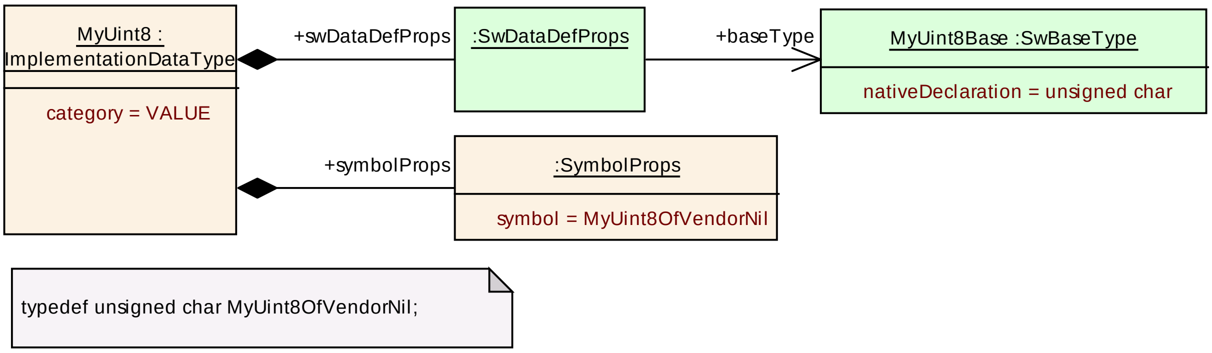


Figure 5.12: Primitive Implementation Data Type with SymbolProps

Example 5.20

If the input configuration contains a two ImplementationDataTypes with same name but different definition the SymbolProps can be used to avoid the name clash. The Primitive Implementation Data Type in example 5.12 results in following definition:

```
/* RTE Types Header File */
typedef unsigned char MyUint8OfVendorNil;
```

The *Application Types Header File* an using component contain the remapping to the original name:

```
/* Application Types Header File */
define MyUint8 MyUint8OfVendorNil;
```

[SWS_Rte_06719] ⌈ The RTE generator shall reject configurations where ImplementationDataTypes result in the same Implementation Data Type symbol but whose definition would not resulting in the same type declaration. ⌋*(SRS_Rte_00018)*

Note: This would result in compiler errors due to incompatible redefinition of C types.

[SWS_Rte_06724] ⌈ The RTE generator shall reject configurations where the same software component uses ImplementationDataTypes with equal shortNames which would result in the mapping to different Implementation Data Type symbols. ⌋*(SRS_Rte_00018)*

Note: This would result in compiler errors due to incompatible redefinition of the mapping from ImplementationDataType.shortName to Implementation Data Type symbol

5.3.4.11 C/C++

The following requirements apply to RTEs generated for C and C++.

[SWS_Rte_01161] ⌈ The name of the *RTE Types Header File* shall be `Rte_Type.h`. ⌋*(SRS_BSW_00300)*

[SWS_Rte_01162] ⌈ Within the *RTE Types Header File*, each data type shall be declared using `typedef`. ⌋*(SRS_Rte_00126)*

A `typedef` is used when declaring a new data type instead of a `#define` even though C only provides weak type checking since other static analysis tools can then be used to overlay strong type checking onto the C before it is compiled and thus detect type errors before the module is even compiled.

5.3.5 RTE Data Handle Types Header File

The *RTE Data Handle Types Header File* contains the Data Handle type declarations necessary for the component data structures (see Section 5.4.2). The *RTE Data Handle Types Header File* code is not allowed to create objects in memory.

[SWS_Rte_07920] ⌈ The RTE generator shall create the *RTE Data Handle Types Header File* including the type declarations of `Data Element without Status` ([SWS_Rte_01363], [SWS_Rte_01364], [SWS_Rte_02607]) and `Data Element with Status` ([SWS_Rte_01365], [SWS_Rte_01366], [SWS_Rte_03734], [SWS_Rte_02666], [SWS_Rte_02589], [SWS_Rte_02590]). ⌋

[SWS_Rte_07921] ⌈ The *RTE Data Handle Types Header File* shall not contain code that creates object in memory. ⌋*(SRS_BSW_00308)*

The *RTE Data Handle Types Header File* should be an output of the “RTE Contract” and “RTE Generation” phases.

5.3.5.1 File Name

[SWS_Rte_07922] ⌈ The name of the *RTE Data Handle Types Header File* shall be `Rte_DataHandleType.h`. ⌋*(SRS_BSW_00300)*

5.3.5.2 File Contents

The *RTE Data Handle Types Header File* contains the type declarations of `Data Element without Status` and `Data Element with Status` (see Section 5.4.2).

[SWS_Rte_07923] ⌈ The *RTE Data Handle Types Header File* shall include the following mechanism to prevent multiple inclusions.

```
#ifndef RTE_DATA_HANDLE_TYPE_H
```

```
#define RTE_DATA_HANDLE_TYPE_H

/* File contents */

#endif /* RTE_DATA_HANDLE_TYPE_H */
```

⌋*(SRS_Rte_00126)*

5.3.6 Application Types Header File

The *Application Types Header File* provides a component local name space for enumeration literals and range values. The *Application Types Header File* is not allowed to create objects in memory.

The *Application Types Header File* file should be identical output for "RTE Contract" and "RTE Generation" phases.

[SWS_Rte_07120] ⌈ The RTE generator shall create an *Application Types Header File* for each software-component type (excluding `ParameterSwComponentType`s and `NvBlockSwComponentType`s) defined in the input. ⌋*(SRS_Rte_00024, SRS_Rte_00140, SRS_BSW_00447)*

[SWS_Rte_07121] ⌈ The *Application Types Header File* shall not contain code that creates objects in memory. ⌋*(SRS_BSW_00308)*

5.3.6.1 File Name

[SWS_Rte_07122] ⌈ The name of the *Application Types Header File* shall be formed by prefixing the AUTOSAR software-component type name with `Rte_[Byps_]` and appending the result with `_Type.h`. `[Byps_]` is an optionnal infix used when component wrapper method for bypass support is enabled for the related software component type (See chapter 4.9.2). ⌋*(SRS_BSW_00300, SRS_Rte_00167)*

Example 5.21

The following declaration in the input XML:

```
<APPLICATION-SW-COMPONENT-TYPE>
    <SHORT-NAME>Source</SHORT-NAME>
</APPLICATION-SW-COMPONENT-TYPE>
```

should result in the *Application Types Header File* `Rte_Source_Type.h` being generated when the component wrapper method for bypass support is disabled.

5.3.6.2 Scope

[SWS_Rte_07123] ⌈ The *Application Types Header File* for a component shall contain only information relevant for that component. ⌋*(SRS_Rte_00167, SRS_Rte_00017)*

[SWS_Rte_07124] ⌈ The *Application Types Header File* shall be valid for both C and C++ source. ⌋*(SRS_Rte_00126, SRS_Rte_00138)*

Requirement [SWS_Rte_07124] is met by ensuring that all definitions within the *Application Types Header File* are defined using C linkage if a C++ compiler is used.

[SWS_Rte_07125] ⌈ All definitions within in the *Application Types Header File* shall be preceded by the following fragment;

```
#ifdef __cplusplus
extern "C" {
#endif /* __cplusplus */
```

⌋*(SRS_Rte_00126, SRS_Rte_00138)*

[SWS_Rte_07126] ⌈ All definitions within the application types header file shall be suffixed by the following fragment;

```
#ifdef __cplusplus
} /* extern "C" */
#endif /* __cplusplus */
```

⌋*(SRS_Rte_00126, SRS_Rte_00138)*

[SWS_Rte_07678] ⌈ The *Application Types Header File* shall be protected against multiple inclusions:

```
#ifndef RTE_<SWC>_TYPE_H
#define RTE_<SWC>_TYPE_H
...
/*
 * Contents of file
 */
...
#endif /* !RTE_<SWC>_TYPE_H */
```

Where `<SWC>` is the AUTOSAR software-component type name.[3] ⌋*(SRS_Rte_00126)*

5.3.6.3 File Contents

In contrast to the *Application Header File* the *Application Types Header File* supports that multiple *Application Types Header File*'s are included in the same module. This is necessary if for instance a BSW module uses several AUTOSAR Services.

[SWS_Rte_07127] ⌈ The *Application Types Header File* shall include the *RTE Types Header File*. ⌋*(SRS_Rte_00087)*

[3] No additional capitalization is applied to the names.

The name of the *RTE Types Header File* is defined in Section 5.3.4.

5.3.6.4 RTE Modes

The *Application Types Header File* shall contain identifiers for the `ModeDeclaration`s and type definitions for `ModeDeclarationGroup`'s as defined in Chapter 5.5.3

5.3.6.5 Enumeration Data Types

The *Application Types Header File* shall contain the enumeration constants as defined in Chapter 5.5.4

5.3.6.6 Range Data Types

The *Application Types Header File* shall contain definitions of `Range` constants as defined in Chapter 5.5.5

5.3.6.7 Implementation Data Type symbols

The *Application Type Header File* may contain definitions to redefine the `Implementation Data Type symbol` to the `shortName` of the `ImplementationDataType` in order to provide the expected type name to the software component implementation. See section 5.3.4.10.

5.3.7 VFB Tracing Header File

The VFB Tracing Header File defines the configured VFB Trace events.

[SWS_Rte_01319] ⌈ The VFB Tracing Header File shall be created by the RTE Generator during *RTE Generation Phase* only. ⌋*(SRS_Rte_00045)*

The VFB Tracing Header file is included by the generated RTE and by the user in the module(s) that define the configured hook functions. The header file includes prototypes for the configured functions to ensure consistency between the invocation by the RTE and the definition by the user.

5.3.7.1 C/C++

The following requirements apply to RTEs generated for C and C++.

[SWS_Rte_01250] ⌈ The name of the VFB Tracing Header File shall be `Rte_Hook.h`. ⌋*(SRS_Rte_00045)*

5.3.7.2 File Contents

[SWS_Rte_01251] ⌈ The VFB Tracing header file shall include the *RTE Configuration Header File* (Section 5.3.8). ⌋*(SRS_Rte_00045)*

[SWS_Rte_01357] ⌈ The VFB Tracing header file shall include the *RTE Types Header file* (Section 5.3.4). ⌋*(SRS_Rte_00003, SRS_Rte_00004)*

[SWS_Rte_03607] ⌈ The VFB Tracing header file shall include `Os.h`. ⌋*(SRS_Rte_00005, SRS_Rte_00008)*

[SWS_Rte_01320] ⌈ The VFB Tracing header file shall contain the following code immediately after the include of the *RTE Configuration Header File*.

```
#ifndef RTE_VFB_TRACE
#define RTE_VFB_TRACE (FALSE)
#endif /* RTE_VFB_TRACE */
```

⌋*(SRS_Rte_00008, SRS_Rte_00005)*

Requirement [SWS_Rte_01320] enables VFB tracing to be globally enabled/disabled within the RTE Configuration Header File and ensures that it defaults to 'disabled'.

[SWS_Rte_01236] ⌈ For each trace event hook function defined in Section 5.11.5, the RTE generator shall define the following code sequence in the VFB Tracing header file:

```
#if defined(<trace event>) && (RTE_VFB_TRACE == FALSE)
#undef <trace event>
#endif
#if defined(<trace event>)
#undef <trace event>
extern void <trace event>(<params>);
#else
#define <trace event>(<params>) ((void)(0))
#endif /* <trace event> */
```

where `<trace event>` is the name of trace event hook function and `<params>` is the list of parameter names of the trace event hook function prototype as defined in Section 5.11.5. ⌋*(SRS_Rte_00008)*

The code fragment within [SWS_Rte_01236] benefits from a brief analysis of its structure. The first `#if` block ensures that an individually configured trace event in the RTE Configuration Header File [SWS_Rte_01324] is disabled if tracing is globally disabled [SWS_Rte_01323]. The second `#if` block emits the prototype for the hook function only if enabled in the RTE Configuration file and thus ensures that only configured trace events are prototyped. The `#undef` is required to ensure that the trace event function is invoked as a function by the generated RTE. The `#else` block comes into effect if the trace event is disabled, either individually [SWS_Rte_01325] or globally, and ensures that it has no run-time effect. Within the #else block the definition to ((void)(0)) enables the hook function to be used within the API Mapping in a comma-expression.

An individual trace event defined in Section 5.11.5 actually defines a class of hook functions. A member of the class is created for each RTE object created (e.g. for each API function, for each task) and therefore an individual trace event may give rise to many hook function definitions in the VFB Tracing header file.

Example 5.22

Consider an API call Rte_Write_p1_a for an instance of SW-C c. This will result in two trace event hook functions being created by the RTE generator:

```
Rte_WriteHook_c_p1_a_Start
```

and

```
Rte_WriteHook_c_p1_a_Return
```

5.3.8 RTE Configuration Header File

The *RTE Configuration Header File* contains user definitions that affect the behavior of the generated RTE.

The directory containing the required *RTE Configuration Header File* should be included in the compiler's include path when using the VFB tracing header file. The *RTE Configuration Header File* is generated by the RTE generator.

5.3.8.1 C/C++

The following requirements apply to RTEs generated for C and C++.

[SWS_Rte_01321] ⌈ The name of the *RTE Configuration Header File* shall be Rte_Cfg.h. ⌋*(SRS_Rte_00008, SRS_Rte_00045)*

5.3.8.2 File Contents

[SWS_Rte_07641] ⌈ The *RTE Configuration Header File* shall include the file Std_Types.h. ⌋*(SRS_Rte_00149, SRS_Rte_00150, SRS_BSW_00353)*

5.3.8.2.1 VFB tracing configuration

[SWS_Rte_01322] ⌈ The RTE generator shall globally enable VFB tracing when `RTE_VFB_TRACE` is defined in the *RTE Configuration Header File* as a vale which does not evaluate as `FALSE`. ⌋*(SRS_Rte_00008, SRS_Rte_00005)*

Note that, as observed in Section 5.11, VFB tracing enables debugging of software components, not the RTE itself.

[SWS_Rte_01323] ⌈ The RTE generator shall globally disable VFB tracing when `RTE_VFB_TRACE` is defined in the RTE configuration header file as `FALSE`. ⌋*(SRS_Rte_00008, SRS_Rte_00005)*

As well as globally enabling or disabling VFB tracing, the RTE Configuration header file also configures those individual VFB tracing events that are *enabled*.

[SWS_Rte_01324] ⌈ The RTE generator shall enable VFB tracing for a given hook function when there is a `#define` in the *RTE Configuration Header File* for the hook function name and tracing is globally enabled. ⌋*(SRS_Rte_00008)*

Note that the particular value assigned by the `#define`, if any, is not significant.

[SWS_Rte_01325] ⌈ The RTE generator shall disable VFB tracing for a given hook function when there is no `#define` in the *RTE Configuration Header File* for the hook function name even if tracing is globally enabled. ⌋*(SRS_Rte_00008)*

Example 5.23

Consider the trace events from Example 5.22. The trace event for API start is enabled by the following definition;

```
#define Rte_WriteHook_i1_p1_a_Start
```

And the trace event for API termination is enabled by the following definition;

```
#define Rte_WriteHook_i1_p1_a_Return
```

5.3.8.2.2 Condition Value Macros

The *Condition Value Macros* are generated in the *PreBuild Data Set Contract Phase* and *PreBuild Data Set Generation Phase*. To do this a particular variant out of the `PreBuild Variability` of the input configuration has to be chosen by the means described in by [SWS_Rte_06500].

[SWS_Rte_06514] ⌈ If evaluated `BooleanValueVariationPoint`s or `ConditionByFormula`s are resulting to true the `<value>` for *Condition Value Macros* shall be coded as `TRUE` and if these are resulting to false the value shall be coded as `FALSE`. ⌋*(SRS_Rte_00201, SRS_Rte_00203)*

[SWS_Rte_06513] ⌈ For each `VariationPointProxy` which `bindingTime` = `PreCompileTime` the *RTE Configuration Header File* shall contain a definition of a *Condition Value Macro* in the *RTE PreBuild Data Set Contract Phase* and *RTE PreBuild Data Set Generation Phase*

```
#define Rte_SysCon_<cts>_<name> <value>
```

Where `<cts>` is the `component type symbol` of the `AtomicSwComponentType`,

`<name>` is the `shortName` of the `VariationPointProxy` and

`<value>` is the evaluated value of the `AttributeValueVariationPoint` or `ConditionByFormula`. ⌋*(SRS_Rte_00203, SRS_Rte_00167)*

This requirements makes the `SwSystemconst` values available to resolve the `PreBuild Variability` in the software components via the Preprocessor. This might be used to

- read the actual value of the value assigned to a `SwSystemconst`
- read the setting of an attribute (e.g. array size) dependent from a `SwSystemconst`
- check the existence of a conditional existent object, e.g. an code fragment implementing a particular functionality

[SWS_Rte_03854] ⌈ For each `VariationPointProxy` which `bindingTime` = `PreCompileTime` the *RTE Application Header File* shall contain a definition

```
#define Rte_SysCon_<name> Rte_SysCon_<cts>_<name>
```

where `<cts>` is the `component type symbol` of the `AtomicSwComponentType` and

`<name>` is the `shortName` of the `VariationPointProxy`. ⌋*(SRS_Rte_00203, SRS_Rte_00167)*

[SWS_Rte_06515] ⌈ For each RTE API which is subject to variability and following the form *component port* or *entity port* in table 4.15 the *RTE Configuration Header File* shall contain one definition of a *Condition Value*

```
#define Rte_VPCon_<cts>_<re>[_<resl>]_<p>_<o>[_<psl>] <value>
```

where `<cts>` is the `component type symbol` of the `AtomicSwComponentType`,

`<re>` is the short name of the `RunnableEntity`,

`<resl>` is the `shortLabel` of the `RunnableEntity`'s `VariationPoint` containing the reference element (e.g. a `VariableAccess`) to the `PortInterface` element,

`<p>` is the name of the `PortPrototype`,

`<o>` is the short name of the `PortInterface` element and

<psl> is the shortLabel of the PortPrototype's VariationPoint which is referred by the VariableAccess

If there is no VariationPoint at the RunnableEntity owning the VariableAccess the <resl> with leading underscore is omitted ([_<resl>]).

If there is no VariationPoint at the PortPrototype referred by the VariableAccess the <psl> with leading underscore is omitted ([_<psl>]).

<value> is the evaluated value of the ConditionByFormula of the VariationPoint vary the existence of the RTE API in table 4.15. ⌋*(SRS_Rte_00201, SRS_Rte_00167)*

[SWS_Rte_06518] ⌈ For each RTE API which is subject to variability and following the form *component internal* in table 4.15 the *RTE Configuration Header File* shall contain one definition of a *Condition Value*

```
#define Rte_VPCon_<cts>_<ki>_<name>_<sl> <value>
```

where <cts> is the component type symbol of the AtomicSwComponentType,

<ki> is the *kind infix* according table 4.15,

<name> is the short name of the element which is subject to variability in table 4.15 and is defining the API name infix,

<sl> is the shortLabel of the elements' VariationPoint defining the API name infix.

<value> is the evaluated value of the ConditionByFormula of the VariationPoint defining the variant existence of the RTE API in table 4.15. ⌋*(SRS_Rte_00201, SRS_Rte_00167)*

[SWS_Rte_06519] ⌈ For each RTE API which is subject to variability and which variability shall be implemented and which is following the form *entity internal* in table 4.15 the *RTE Configuration Header File* shall contain one definition of a *Condition Value*

```
#define Rte_VPCon_<cts>_<re>[_<resl>]_<ki>_<name>_<sl> <value>
```

where <cts> is the component type symbol of the AtomicSwComponentType,

<re> is the short name of the RunnableEntity,

<resl> is the shortLabel of the RunnableEntity's VariationPoint containing the reference element (e.g. a VariableAccess) to the PortInterface element,

<ki> is the *kind infix* according table 4.15 and

<name> is the short name of the element which is subject to variability in table 4.15 and is defining the API name infix.

<sl> is the shortLabel of the elements' VariationPoint defining the API name infix.

If there is no VariationPoint at the RunnableEntity owning the reference element (e.g. a VariableAccess) to the PortInterface element the <resl> with leading underscore is omitted ([_<resl>]).

<value> is the evaluated value of the ConditionByFormula of the VariationPoint defining the variant existence of the RTE API in table 4.15. ⌋*(SRS_Rte_00201, SRS_Rte_00167)*

[SWS_Rte_06520] ⌈ For each PortPrototype which is subject to variability and which variability shall be implemented the *RTE Configuration Header File* shall contain one definition of a *Condition Value*

```
#define Rte_VPCon_<cts>_<p>_<psl> <value>
```

where <cts> is the component type symbol of the AtomicSwComponentType,

<p> is the short name of the PortPrototype and

<psl> is the shortLabel of the PortPrototype's VariationPoint and

<value> is the evaluated value of the ConditionByFormula of the VariationPoint defining the variant existence of the PortPrototype in table 4.15. ⌋*(SRS_Rte_00201, SRS_Rte_00167)*

[SWS_Rte_06530] ⌈ For each RunnableEntity which is subject to variability and which variability shall be implemented the *RTE Configuration Header File* shall contain one definition of a *Condition Value*

```
#define Rte_VPCon_<cts>_<re>_<resl> <value>
```

where <cts> is the component type symbol of the AtomicSwComponentType,

<re> is the short name of the RunnableEntity

<resl> is the shortLabel of the RunnableEntity's VariationPoint containing the reference element (e.g. a VariableAccess) to the PortInterface element,

<value> is the evaluated value of the ConditionByFormula of the VariationPoint defining the variant existence of the RunnableEntity in table 4.18. ⌋*(SRS_Rte_00201, SRS_Rte_00167)*

[SWS_Rte_06541] ⌈ For each arraySize which subject to variability the *RTE Configuration Header File* shall contain one definition of a *Attribute Value*

```
#define Rte_VPVal_<t>_<e 1>[_<e 2> ... _<e n>] <value>
```

where <t> is the shortName of the ImplementationDataType,

[<e x>] are the shortNames of the Array's ImplementationDataTypeElements with a leading underscore ordered from the root to the Array's ImplementationDataTypeElement with the arraySize being subject to variability and

<value> is the evaluated value of the AttributeValueVariationPoint of the arraySize ⌋*(SRS_Rte_00201, SRS_Rte_00167)*

[SWS_Rte_06542] ⌈ For each Array's ImplementationDataTypeElement which subject to variability the *RTE Configuration Header File* shall contain one definition of a *Condition Value*

```
#define Rte_VPCon_<t>_<e 1>[_<e 2> ... _<e n>] <value>
```

where <t> is the shortName of the ImplementationDataType,

[<e x>] are the shortNames of the Array's ImplementationDataTypeElements with a leading underscore ordered from the root to the Array's ImplementationDataTypeElement being subject to variability and

<value> is the evaluated value of the ConditionByFormula of the VariationPoint defining the conditional existence of the Array's ImplementationDataTypeElement ⌋*(SRS_Rte_00201, SRS_Rte_00167)*

[SWS_Rte_06551] ⌈ For each DataConstr referenced by a ApplicationPrimitiveDataType where the upperLimit is subject to preCompileTime variability the *RTE Configuration Header File* shall contain one definition of a *Attribute Value Macro*

```
#define Rte_VPVal_<cts>_<prefix><t>_UpperLimit <upperValue><suffix>
```

where <cts> is the component type symbol of the AtomicSwComponentType,

<t> is the shortName of the ApplicationPrimitiveDataType,

<prefix> is the optional literalPrefix attribute defined by the IncludedDataTypeSet referring the AutosarDataType to which the DataConstr belongs,

<upperValue> are the upperLimit value of the dataConstr referenced by the ApplicationPrimitiveDataType onto which the corresponding CompuMethod has been applied (see [SWS_Rte_07038]). The value in the macro definitions shall always reflect the closed interval, regardless of the interval type specified by the DataConstr.

<suffix> shall be "U" for unsigned data types and empty for signed data types. ⌋*(SRS_Rte_00201, SRS_Rte_00167)*

[SWS_Rte_06552] ⌈ For each DataConstr referenced by a ApplicationPrimitiveDataType where the lowerLimit is subject to preCompileTime variability the *RTE Configuration Header File* shall contain one definition of a *Attribute Value Macro*

```
#define Rte_VPVal_<cts>_<prefix><t>_LowerLimit <lowerValue><suffix>
```

where <cts> is the component type symbol of the AtomicSwComponentType,

<t> is the shortName of the ApplicationPrimitiveDataType,

<prefix> is the optional literalPrefix attribute defined by the IncludedDataTypeSet referring the AutosarDataType to which the DataConstr belongs,

<lowerValue> are the lowerLimit value of the dataConstr referenced by the ApplicationPrimitiveDataType onto which the corresponding CompuMethod has been applied (see [SWS_Rte_07038]). The value in the macro definitions shall always reflect the closed interval, regardless of the interval type specified by the DataConstr.

<suffix> shall be "U" for unsigned data types and empty for signed data types. ⌋*(SRS_Rte_00201, SRS_Rte_00167)*

[SWS_Rte_06535] ⌈ For each *Basic Software Scheduler* API which is subject to variability and following the form *module internal* in table 4.26 the *RTE Configuration Header File* shall contain one definition of a *Condition Value*

```
#define SchM_VPCon_<bsnp>[_<vi>_<ai>]_<ki>_<name>_<sl> <value>
```

where here

<bsnp> is the *BSW Scheduler Name Prefix* according [SWS_Rte_07593] and [SWS_Rte_07594],

<vi> is the vendorId of the BSW module,

<ai> is the vendorApiInfix of the BSW module,

<ki> is the *kind infix* according table 4.26,

<name> is the short name of the element which is subject to variability in table 4.26 defining the *Basic Software Scheduler* API name infix and

<sl> is the shortLabel of the elements' VariationPoint defining the API name infix.

<value> is the evaluated value of the ConditionByFormula of the VariationPoint defining the variant existence of the *Basic Software Scheduler* API in table 4.26.

The sub part in squared brackets [_<vi>_<ai>] is omitted if no vendorApiInfix is defined for the *Basic Software Module*. See [SWS_Rte_07528]. ⌋*(SRS_Rte_00229, SRS_BSW_00347)*

[SWS_Rte_06536] ⌈ For each *Basic Software Scheduler* API which is subject to variability and which variability shall be implemented and which is following the form *module external* and *entity internal* in table 4.26 the *RTE Configuration Header File* shall contain one definition of a *Condition Value*

```
#define SchM_VPCon_<bsnp>[_<vi>_<ai>]_<ki>_
                        <entity>[_<esl>]_<name>[_<sl>] <value>
```

where here

<bsnp> is the *BSW Scheduler Name Prefix* according [SWS_Rte_07593] and [SWS_Rte_07594],

<vi> is the vendorId of the BSW module,

<ai> is the vendorApiInfix of the BSW module,

<ki> is the *kind infix* according table 4.26,

entity is the shortName of the BswModuleEntity

`<esl>` is the `shortLabel` of the `BswModuleEntity`'s `VariationPoint` containing the subject to variability,

`<name>` is the `shortName` of the element/referenced element which is subject to variability in table 4.26 defining the *Basic Software Scheduler* API name infix and

`<sl>` is the `shortLabel` of the elements's `VariationPoint` defining the API name infix.

`<value>` is the evaluated value of the `ConditionByFormula` of the `VariationPoint` defining the variant existence of the *Basic Software Scheduler* API in table 4.26.

The sub part in squared brackets `[_<vi>_<ai>]` is omitted if no `vendorApiInfix` is defined for the *Basic Software Module*. See [SWS_Rte_07528].

If there is no `VariationPoint` at the `BswModuleEntity` referring to the subject to variability in table 4.26 the `<esl>` with leading underscore is omitted (`[_<esl>]`).

If there is no `VariationPoint` at the elements defining the *Basic Software Scheduler* API name infix 4.26 the `<sl>` with leading underscore is omitted (`[_<sl>]`). ⌋*(SRS_Rte_00229, SRS_BSW_00347)*

[SWS_Rte_06532] ⌈ For each `BswSchedulableEntity` which is subject to variability and which variability shall be implemented the *RTE Configuration Header File* shall contain one definition of a *Condition Value*

```
#define SchM_VPCon_<bsnp>[_<vi>_<ai>]_<entry>_<esl> <value>
```

where here

`<bsnp>` is the *BSW Scheduler Name Prefix* according [SWS_Rte_07593] and [SWS_Rte_07594],

`<vi>` is the `vendorId` of the BSW module,

`<ai>` is the `vendorApiInfix` of the BSW module,

`<entry>` is the `shortName` of the implemented (`implementedEntry`) entry point and

`<esl>` is the `shortLabel` of the `BswModuleEntity`'s `VariationPoint`

`<value>` is the evaluated value of the `ConditionByFormula` of the `VariationPoint` defining the variant existence of the `BswSchedulableEntity` in table 4.28.

The sub part in squared brackets `[_<vi>_<ai>]` is omitted if no `vendorApiInfix` is defined for the *Basic Software Module*. See [SWS_Rte_07528]. ⌋*(SRS_Rte_00229, SRS_BSW_00347)*

An example about the usage of condition value macros is shown in 5.6.

5.3.9 Generated RTE

Figure 5.1 defines the relationship between generated and standardized header files. It is **not** necessary to standardize the relationship between the C module, `Rte.c`, and the header files since when the RTE is generated the application header files are created anew along with the RTE. This means that details of which header files are included by `Rte.c` can be left as an implementation detail.

5.3.9.1 Header File Usage

[SWS_Rte_01257] ⌈ In compatibility mode, the Generated RTE module shall include `Os.h`. ⌋*(SRS_Rte_00145)*

[SWS_Rte_03794] ⌈ In compatibility mode, the generated RTE module shall include `Com.h`. ⌋*(SRS_Rte_00145)*

[SWS_Rte_01279] ⌈ In compatibility mode, the Generated RTE module shall include `Rte.h`. ⌋*(SRS_Rte_00145)*

[SWS_Rte_01326] ⌈ In compatibility mode, the Generated RTE module shall include the VFB Tracing header file. ⌋*(SRS_Rte_00045, SRS_Rte_00145)*

[SWS_Rte_03788] ⌈ Except for the declaration of entry points for components (see [SWS_Rte_07194]), the RTE shall map its memory objects with the file `Rte_MemMap.h`, using the AUTOSAR memory mapping mechanism (see [25]). ⌋*(SRS_Rte_00148)*

[SWS_Rte_07692] ⌈ The Generated RTE module shall perform Inter Module Checks to avoid integration of incompatible files. The imported included files shall be checked by preprocessing directives.

The following version numbers shall be verified:

- `<MODULENAME>_AR_RELEASE_MAJOR_VERSION`
- `<MODULENAME>_AR_RELEASE_MINOR_VERSION`

Where `<MODULENAME>` is the module short name of the other (external) modules which provide header files included by the Generated RTE module.

If the values are not identical to the expected values, an error shall be reported. ⌋*(SRS_BSW_00004)*

Figure 5.13 provides an example of how the RTE header and generated header files could be used by a generated RTE.

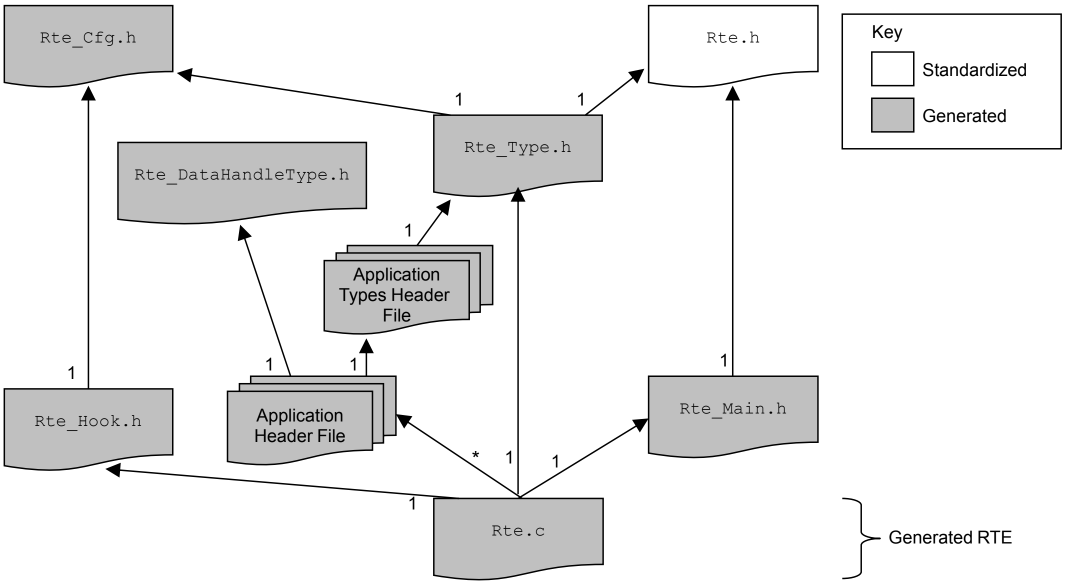


Figure 5.13: Example of header file use by the generated RTE.

In the example in Figure 5.13, the generated RTE C module requires access to the data structures created for each AUTOSAR software-component and therefore includes each application header file[4]. In the example, the generated RTE also includes the RTE header file and the lifecycle header file in order to obtain access to RTE and lifecycle related definitions.

Note: Inclusion of *Application Header File*s of different software components into the RTE C module needs support in the *Application Header File*s in order to avoid that some local definitions of software components are producing name clashes. If the RTE C module does not include any *Application Header File*, some type definitions (e.g. component data structure) might have to be generated twice.

5.3.9.2 C/C++

The following requirements apply to RTEs generated for C and C++.

Note: The `<PartitionName>`s referred to in requirements [SWS_Rte_02712], [SWS_Rte_02713] and [SWS_Rte_02740] are implementation-specific identifiers for the modules. They need not be the same as the `CoreId` identifiers configured for the multi core OS. Refer to section 4.3.4 for a discussion of the allocation of ECU execution logic to partitions and the allocation of partitions to cores.

[SWS_Rte_01169] ⌈ The name of the C module containing the generated RTE code that is shared by all cores of an ECU shall be `Rte.c`. ⌋*(SRS_BSW_00300, SRS_Rte_00126)*

[4]The requirement that a software module include at most one application header file applies only to modules that actually implement a software-component and therefore does not apply to the generated RTE.

[SWS_Rte_02711] ⌈ On a multi core ECU, RTE shall only use global and static variables in the Rte.c module, if it is used in a single image system that supports shared memory. In this case, RTE shall guarantee consistency of this memory, e.g. by using OS mechanisms. ⌋

[SWS_Rte_02712] ⌈ On a multi partition ECU, there shall be additional code and header files named Rte_Partition_<PartitionName> for the core specific code parts of RTE where <PartitionName> is the shortName of the container EcucPartition. ⌋

[SWS_Rte_02713] ⌈ There shall not be symbol redefinitions between different Rte_Partition_<PartitionName> files. ⌋

These requirements makes sure, that all Rte modules can be linked in one image. On a multi core ECU, the RTE may be linked in one image or distributed over separate images, one per core.

An RTE that includes configured code from an object-code or source-code library may use additional modules. Further on due to the encapsulation of a component local name space [SRS_Rte_00167], it might be required to encapsulate part of the generated RTE code in component specific files as well to avoid name clashes in the RTE's implementation.

[SWS_Rte_07140] ⌈ The RTE generator is allowed to partition the generated RTE module in several files additionally to Rte.c and Rte_Partition_<PartitionName>. ⌋*(SRS_Rte_00167)*

5.3.9.3 File Contents

By its very nature the contents of the generated RTE is largely vendor specific. It is therefore only possible to define those common aspects that are visible to the "outside world" such as the names of generated APIs and the definition of component data structures that apply any operating mode.

5.3.9.3.1 Component Data Structures

The *Component Data Structure* (Section 5.4.2) is a per-component data type used to define instance specific information required by the generated RTE.

[SWS_Rte_03711] ⌈ The generated RTE shall contain an instance of the relevant Component Data Structure for each software-component instance on the ECU for which the RTE is generated. ⌋*(SRS_Rte_00011)*

[SWS_Rte_03712] ⌈ The name of a Component Data Structure instantiated by the RTE generator shall be Rte_Instance_<name> where <name> is an automatically generated name, created in some manner such that all instance data structure names are unique. ⌋*(SRS_BSW_00307)*

The software component instance name referred to in [SWS_Rte_03712] is never made visible to the users of the generated RTE. There is therefore no need to specify the precise form that the unique name takes. The `Rte_Instance_` prefix is mandated in order to ensure that no name clashes occur and also to ensure that the structures are readily identifiable in map files, debuggers, etc.

5.3.9.3.2 Generated API

[SWS_Rte_01266] ⌈ The RTE module shall define the generated functions that will be invoked when an AUTOSAR software-component makes an RTE API call. ⌋*(SRS_Rte_00051)*

The semantics of the generated functions are not defined (since these will obviously vary depending on the RTE API call that it is implementing) nor are the implementation details (which are vendor specific). However, the names of the generated functions defined in Section 5.2.6.1.

The signature of a generated function is the same as the signature of the relevant RTE API call (see Section 5.6) with the exception that the instance handle can be omitted since the generated function is applicable to a specific software-component instance.

5.3.9.3.3 Callbacks

In addition to the generated functions for the RTE API, the RTE module includes callbacks invoked by COM when signal events (receptions, transmission acknowledgement, etc.) occur.

[SWS_Rte_01264] ⌈ The RTE module shall define COM callbacks for relevant signals. ⌋*(SRS_Rte_00019)*

The required callbacks are defined in Section 5.9.

[SWS_Rte_03795] ⌈ The RTE generator shall generate a separate header file containing the prototypes of the COM callback functions. ⌋*(SRS_Rte_00019)*

[SWS_Rte_03796] ⌈ The name of the header file containing the callback prototypes shall be `Rte_Cbk.h` in a C/C++ environment. ⌋*(SRS_Rte_00019)*

[SWS_Rte_03796] refers to the callbacks defined in section 5.9.

5.3.9.3.4 Task bodies

The RTE module define task bodies for tasks created by the RTE generator only in compatibility mode.

[SWS_Rte_01277] ⌈ In compatibility mode [SWS_Rte_01257], the RTE module shall define all task bodies created by the RTE generator. ⌋*(SRS_Rte_00145)*

Note that in vendor mode it is assumed that greater knowledge of the OS is available and therefore the above requirement does *not* apply so that specific optimizations, such as creating each task in a separate module, can be applied.

5.3.9.3.5 Lifecycle API

[SWS_Rte_01197] ⌈ The RTE module shall define the RTE lifecycle API. ⌋*(SRS_Rte_00051)*

The RTE lifecycle API is defined in Section 5.8.

5.3.9.4 Reentrancy

All code invoked by generated RTE code that can be subject to concurrent execution must be reentrant. This requirement for reentrancy can be overridden if the generated code is not subject to concurrent execution, for example, if protected by a data consistency mechanism to ensure that access to critical regions is call serialized.

5.3.10 RTE Post Build Variant Sets

[SWS_Rte_06620] ⌈ The RTE generator shall generate in the *Rte_PBCfg.h* file the `SchM_ConfigType` type declaration of the predefined post build variants data structure. This header file must be used by other RTE modules to resolve their runtime variabilities. ⌋*(SRS_Rte_00201)*

[SWS_Rte_06638] ⌈ The RTE generator must generate a *Rte_PBCfg.c* file containing the declarations and initializations of one or more RTE post build variants. Only one of these variants can be active at runtime. ⌋*(SRS_Rte_00201, SRS_BSW_00346)*

Within an RTE with post build variants, one active `RtePostBuildVariantConfiguration` will exist. It is a pointer to this structure that shall be passed to `SchM_Init`. Also note that the container `PredefinedVariant` is only a Meta Model construct to allow the designer to create a validated collection of values assigned to a criterion. It is up to the implementer of the RTE generator to optimize variant configurations either for size and/or performance by using different levels of indirection to the `PostBuildVariantCriterionValue`s. For the least amount of indirection for example one can have the criterion values at the level of the `Sch_ConfigType`. If you use post build loadable then you may want to reduce memory storage by reusing variant sets if they remain unchanged across two or more predefined variants.

The following subsections provide examples for the SchM_ConfigType declaration and instantiation only for demonstration purposes. No requirement what so ever is implied.

RtePostBuildVariantConfiguration is a multipleConfigurationContainer and the RtePostBuildUsedPredefinedVariant reference within the container is PostBuild configurable. This is required to permit that RtePostBuildUsedPredefinedVariant parameters in different RtePostBuildVariantConfiguration containers can refer to different PredefinedVariants. Nevertheless this PostBuild references result in the generation of different *PostBuild Data Sets* whereas the RtePostBuildUsedPredefinedVariant reference itself is not actually post build configurable inside the RTE.

5.3.10.1 Example 1: File Contents Rte_PBCfg.h

An example of a flat data structure to represent the criterion values defined in the *Rte_PBCfg.h* file containing theSchM_ConfigType type which can contain the list of unique PostBuildVariantCriterion members. This approach immediately enforces that only one single criterion assignment can exist. The member names can, for example, follow the template defined below where <sn> is the PostBuildVariantCriterion shortName.

```
struct SchM_ConfigType {
  /* The PostBuildVariantCriterion shortname */
    int VarCri_<sn>;
    .
    .
    .
};
```

5.3.10.2 Example 2: File Contents Rte_PBCfg.h

An example showing an additional level of indirection and as such allows for reuse of variant sets to optimize memory storage acorss for example several predefined variants is shown below. The RTE generator in this case can reuse some PostBuildVariantCriterionValueSets to reduce the memory resource consumption of an ECU. The RTE generator can declare in the *Rte_PBCfg.h* file a structure type for each **distinct unique** collection of PostBuildVariantCriterionValueSets containing the PostBuildVariantCriterions as members. This implies that if two PredefinedVariants are defined each referring to a named PostBuildVariantCriterionValueSet and the list of PostBuildVariantCriterions in each of these PostBuildVariantCriterionValueSets is identical that only one type is defined for these two named PostBuildVariantCriterionValueSets. The name of the type can, for example, follow the pattern below where the <id> is a unique identifier for that type (e.g. a counter).

```
struct Rte_VarSet_<id>_t {
```

```
  /* The PostBuildVariantCriterion shortname */
    int VarCri_<sn>;
    .
    .
    .
};
```

Now the `SchM_ConfigType` type can bedeclared with pointers to these variant sets. The member names of this struct can, for example, follow the template below where `<id>` is a unique identifier.

```
struct SchM_ConfigType {
  /* The PostBuildVariantCriterion shortname */
    Rte_VarSet_<id>_t* VarSet_<id>_Ptr;
    .
    .
    .
};
```

5.3.10.3 Examples: File Contents Rte_PBCfg.c

In correlation with example 1 of the header file the RTE generator can declare and optionaly initialize a default variant configuration named `Rte_VarCfg` in the *Rte_PBCfg.c* file of the `SchM_ConfigType` type.

For example (the initializers are the criterion values):

```
const struct SchM_ConfigType Rte_VarCfg = {1,2,3,4,5};
```

And likewise for the example 2 header file the RTE generator can declare and initialize in the *Rte_PBCfg.c* file all possible `PostBuildVariantCriterionValueSet`s and the `RtePostBuildVariantConfiguration`s using references to these variant sets.

For example:

```
const struct Rte_VarSet_1_t Rte_VarSet_1a = {1,2,3};
const struct Rte_VarSet_1_t Rte_VarSet_1b = {1,4,1};
const struct Rte_VarSet_2_t Rte_VarSet_2 = {2,5,7,3,2};
.
.
.

const struct SchM_ConfigType Rte_VarCfg_1 =
    {&Rte_VarSet_1a,&Rte_VarSet_2};
const struct SchM_ConfigType Rte_VarCfg_2 =
    {&Rte_VarSet_1b,&Rte_VarSet_2};
.
.
.
```

When `SchM_Init` is called, a pointer to the active `SchM_ConfigType` will be passed along which shall be assigned to the named Rte_VarCfgPtr which is of type

SchM_ConfigType*. This pointer shall be used to determine the values for actual used PostBuildVariantCriterions and for variant validation when the DET is enabled.

Example 1 pseudo code evaluating the criterions

```
switch(Rte_VarCfg->VarCri_1)
{
    case 1:
        /* DO SOMETHING */
        break;
    case 2:
        /* DO SOMETHING ELSE */
}
```

Example 2 pseudo code evaluating the criterions

```
switch(Rte_VarCfgPtr->VarSet_1_Ptr->VarCri_1)
{
    case 1:
        /* DO SOMETHING */
        break;
    case 2:
        /* DO SOMETHING ELSE */
}
```

Another type of optimization strategy (besides flattening) that can be applied is double buffering for frequently used variant criterion values. The additional buffer can then be used in the conditions to optimize the performance of the RTE (e.g. `BufferedVarCri_1 = Rte_VarCfgPtr->VarSet_1->VarCri_1`).

5.4 RTE Data Structures

Object-code software components are compiled against an application header file created during the “RTE Contract” phase but are linked against an RTE (and application header file) created during the “RTE Generation” phase. When generated in compatibility mode, an RTE has to work for object-code components compiled against an application header file created in compatibility mode, even if the application header file was created by a different RTE generator. It is thus necessary to define the data structures and naming conventions for the compatibility mode to ensure that the object-code is compatible with the generated RTE. An RTE generated in vendor mode only has to work for those object-code components that were compiled against application header files created in vendor mode by a compatible RTE generator (which in general would mean an RTE generator supplied by the same vendor).

The use of standardized data structures imposes tight constraints on the RTE implementation and therefore restricts the freedom of RTE vendors to optimize the solution of object-code components but has the advantage that RTE generators from different vendors can be used to compile an object-code software-component and to generate the RTE. No such restrictions apply for the vendor mode. If an RTE generator operating in vendor mode is used for an object-code component in both phases, vendor-specific optimizations can be used.

Note that with the exception of data structures required for support object-code software components in compatibility mode, the data structures used for “RTE Generation” phase are not defined. This permits vendor specific API mappings and data structures to be used for a generated RTE without loss of portability.

The following definitions only apply to RTE generators operating in compatibility mode – in this mode the instance handle and the component data structure have to be defined even for those (object-code) software components for which multiple instantiation is forbidden to ensure compatibility.

5.4.1 Instance Handle

The RTE is required to support object-code components as well as multiple instances of the same AUTOSAR software-component mapped to an ECU [SRS_Rte_00011]. To minimise memory overhead all instances of a component on an ECU share code [SRS_Rte_00012] and therefore both the RTE and the component instances require a means to distinguish different instances.

Support for both object-code components and multiple instances requires a level of indirection so that the correct generated RTE custom function is invoked in response to a component action. The indirection is supplied by the instance handle in combination with the API mapping defined in Section 5.2.6.

[SWS_Rte_01012] ⌈ The component instance handle shall identify particular instances of a component. ⌋*(SRS_BSW_00312, SRS_Rte_00011)*

The instance handle is passed to each runnable entity in a component when it is activated by the RTE as the first parameter of the function implementing the runnable entity [SWS_Rte_01016]. The instance handle is then passed back by the runnable entity to the RTE, as the first parameter of each direct RTE API call, so that the RTE can identify the correct component instance making the call. This scheme permits multiple instances of a component on the same ECU to share code.

The instance handle indirection permits the name of the RTE API call that is used within the component to be unique within the scope of a component as well as independent of the component’s instance name. It thus enables object-code AUTOSAR software-components to be compiled before the final “RTE Generation” phase when the instance name is fixed.

[SWS_Rte_01013] ⌈ For the RTE C/C++ API, any call that can operate on different instances of a component that supports multiple instantiation `supportsMultipleInstantiation` shall have an instance handle as the first formal parameter. ⌋*(SRS_Rte_00011)*

[SWS_Rte_03806] ⌈ If a component does not support multiple instantiation, the instance handle parameter shall be omitted in the RTE C/C++ API and in the signature of the RTE Hook functions. ⌋*(SRS_Rte_00011)*

If the component does not support multiple instantiation, the name of the instance handle must be specified, since it is not passed to the API calls and runnable entities as parameters.

[SWS_Rte_03793] ⌈ If a software component does not support multiple instantiation, the name of the instance handle shall be `Rte_Inst_<cts>`, where `<cts>` is the `component type symbol` of the `AtomicSwComponentType`. ⌋*(SRS_Rte_00011)*

The data type of the instance handle is defined in Section 5.5.2.

5.4.2 Component Data Structure

Different component instances share many common features - not least of which is support for shared code. However, each instance is required to invoke different RTE API functions and therefore the instance handle is used to access the component data structure that defines all instance specific data.

It is necessary to define the component data structure to ensure compatibility between the two RTE phases when operating in compatibility mode – for example, a “clever” compiler and linker may encode type information into a pointer type to ensure type-safety. In addition, the structure definition cannot be empty since this is an error in ANSI C.

[SWS_Rte_07132] ⌈ The component data structure type shall be defined in the *Application Header* file. ⌋*(SRS_Rte_00011, SRS_Rte_00167)*

[SWS_Rte_03714] ⌈ The type name of the component data structure shall be `Rte_[Byps]_CDS_<cts>` where `<cts>` is the `component type symbol` of the `AtomicSwComponentType`. `[Byps_]` is an optionnal infix used when component wrapper method for bypass support is enabled for the related software component type (See chapter 4.9.2). ⌋*(SRS_BSW_00305)*

The members of the component data structure include function pointers. It is important that such members are not subject to run-time modification and therefore the component data structure is required to be placed in read-only memory.

[SWS_Rte_03715] ⌈ All instances of the component data structure shall be defined as “const” (i.e. placed in read-only memory). ⌋*(SRS_BSW_00007)*

The elements of the component data structure are sorted into sections, each of which defines a logically related section. The sections defined within the component data structure are:

- **[SWS_Rte_03718]** ⌈ Data Handles section. ⌋*(SRS_Rte_00011, SRS_Rte_00051)*

- **[SWS_Rte_03719]** ⌈ Per-instance Memory Handles section. ⌋*(SRS_Rte_00011, SRS_Rte_00051)*
- **[SWS_Rte_01349]** ⌈ Inter-runnable Variable Handles section. ⌋*(SRS_Rte_00011, SRS_Rte_00051)*
- **[SWS_Rte_03720]** ⌈ Calibration Parameter Handles section. ⌋*(SRS_Rte_00011, SRS_Rte_00051)*
- **[SWS_Rte_03721]** ⌈ Exclusive-area API section. ⌋*(SRS_Rte_00011, SRS_Rte_00051)*
- **[SWS_Rte_03716]** ⌈ Port API section. ⌋*(SRS_Rte_00011, SRS_Rte_00051)*
- **[SWS_Rte_03717]** ⌈ Inter Runnable Variable API section. ⌋*(SRS_Rte_00011, SRS_Rte_00051)*
- **[SWS_Rte_07225]** ⌈ Inter Runnable Triggering API section. ⌋*(SRS_Rte_00011, SRS_Rte_00051)*
- **[SWS_Rte_07837]** ⌈ Instance Id section. ⌋*(SRS_Rte_00011, SRS_Rte_00051, SRS_Rte_00244)*
- **[SWS_Rte_03722]** ⌈ Vendor specific section. ⌋*(SRS_Rte_00011)*

The order of elements within each section of the component data structure is defined as follows;

[SWS_Rte_03723] ⌈ Section entries shall be sorted alphabetically (ASCII / ISO 8859-1 code in ascending order) unless stated otherwise. ⌋*(SRS_Rte_00051)*

The sorting of entries is applied to each section in turn.

Note that there is *no* prefix associated with the name of each entry within a section; the component data structure as a whole has the prefix and therefore there is no need for each member to have the same prefix.

ANSI C does not permit empty structure definitions yet an instance handle is required for the RTE to function. Therefore if there are no API calls then a single dummy entry is defined for the RTE.

[SWS_Rte_03724] ⌈ If all sections of the Component Data Structure are empty the Component Data Structure shall contain a `uint8` with name `_dummy`. ⌋*(SRS_Rte_00126)*

5.4.2.1 Data Handles Section

The data handles section is required to support the `Rte_IRead` and `Rte_IWrite` calls (see Section 5.2.4).

[SWS_Rte_03733] ⌈ Data Handles shall be named `<re>_<p>_<o>` where `<re>` is the runnable entity name that reads (or writes) the data item, `<p>` the port name, `<o>` the data element. ⌋*(SRS_BSW_00305, SRS_Rte_00051)*

A `RunnableEntity` can read and write to the same port/data element in case of a `PRPortPrototype`s where as `PPortPrototype`s and `RPortPrototype`s are inherently uni-directional (a provide port can only be written, a require port can only be read). Please note that for read and write access of a runnable to data in a `PRPortPrototype` only one data handle exist.

[SWS_Rte_02608] ⌈ The Data Handle shall be a pointer to a `Data Element with Status` if and only if either

- the runnable has read access and either
  - `data element outdated` notification or
  - `data element invalidation`

  is activated for this `data element`, or
- the runnable has write access and acknowledgement is enabled for this `data element`.

⌋*(SRS_Rte_00051, SRS_Rte_00185)*

[SWS_Rte_02588] ⌈ Otherwise, the data type for a Data Handle shall be a pointer to a `Data Element without Status`. ⌋*(SRS_Rte_00051)*

See below for the definitions of these terms.

[SWS_Rte_06529] ⌈ The RTE Generator shall wrap each entry of *Data Handles Section* in the component data structure of a variant existent `Rte_IRead` or `Rte_IWrite` API if the variability shall be implemented.

```
#if (<condition>)

<Data Handles Section Entry>

#endif
```

where `condition` is the condition value macro of the `VariationPoint` relevant for the variant existence of the `Rte_IRead` or `Rte_IWrite` API (see [SWS_Rte_06515]), `Data Handles Section Entry` is the code according an invariant *Data Handles Section Entry* (see also [SWS_Rte_03733], [SWS_Rte_02608], [SWS_Rte_02588]) ⌋*(SRS_Rte_00201)*

[SWS_Rte_08777] ⌈ If the software component does not support multiple instantiation, the data handles section shall be empty. ⌋*(SRS_Rte_00051)*

5.4.2.1.1 Data Element without Status

[SWS_Rte_01363] ⌈ The data type for a "Data Element without Status" shall be named `Rte_DE_<dt>` where `<dt>` is the data element's `ImplementationDataType` name. ⌋*(SRS_Rte_00051)*

[SWS_Rte_01364] ⌈ A `Data Element without Status` shall be a structure containing a single member named `value`. ⌋*(SRS_Rte_00051)*

[SWS_Rte_02607] ⌈ The `value` member of a `Data Element without Status` shall have the same data type as the corresponding DataElement. ⌋*(SRS_Rte_00051, SRS_Rte_00147, SRS_Rte_00078)*

Note that requirements [SWS_Rte_01364] and [SWS_Rte_02607] together imply that creating a variable of data type `Rte_DE_<dt>` allocates enough memory to store the data copy.

5.4.2.1.2 Data Element with Status

[SWS_Rte_01365] ⌈ The data type for a "Data Element with Status" shall be named `Rte_DES_<dt>` where `<dt>` is the data element's `ImplementationDataType` name. ⌋*(SRS_Rte_00051)*

[SWS_Rte_01366] ⌈ A `Data Element with Status` shall be a structure containing two members. ⌋*(SRS_Rte_00051)*

[SWS_Rte_03734] ⌈ The first member of each `Data Element with Status` shall be named 'value' ⌋*(SRS_Rte_00051)*

[SWS_Rte_02666] ⌈ The value member of a `Data Element with Status` shall have the type of the corresponding DataElement. ⌋*(SRS_Rte_00051, SRS_Rte_00147, SRS_Rte_00078, SRS_Rte_00185)*

[SWS_Rte_02589] ⌈ The second member of each `Data Element with Status` shall be named 'status'. ⌋*(SRS_Rte_00051, SRS_Rte_00147, SRS_Rte_00078, SRS_Rte_00185)*

[SWS_Rte_02590] ⌈ The status member of a `Data Element with Status` shall be of the `Std_ReturnType` type. ⌋*(SRS_Rte_00147, SRS_Rte_00078, SRS_Rte_00185)*

[SWS_Rte_02609] ⌈ In case of read access, the status member of a `Data Element with Status` shall contain the error status corresponding to the value member. ⌋*(SRS_Rte_00147, SRS_Rte_00078)*

[SWS_Rte_03836] ⌈ In case of write access, the status member of a `Data Element with Status` shall contain the transmission status corresponding to the value member. ⌋*(SRS_Rte_00185)*

5.4.2.1.3 Usage

[SWS_Rte_07136] ⌈ A definition for every required Data Element with Status and every Data Element without Status must be emitted in the *RTE Data Handle Types Header File* (see Section 5.3.5). ⌋*(SRS_Rte_00051)*

Example 5.24

Consider a uint8 data element, a, of port p which is accessed using a VariableAccess in the dataWriteAccess role by runnables re1 and re2 and a VariableAccess in the dataReadAccess role by runnable re2 within component c. data element outdated is defined for this dataElement.

The required data types within the *RTE Data Handle Types Header File* would be:

```
typedef struct {
  uint8 value;
} Rte_DE_uint8;

typedef struct {
  uint8 value;
  Std_ReturnType status;
} Rte_DES_uint8;
```

The component data structure for c would also include:

```
Rte_DE_uint8*  re1_p_a;
Rte_DES_uint8* re2_p_a;
```

A software-component that is supplied as object-code or is multiple instantiated requires "general purpose" definitions of Rte_IRead, Rte_IWrite, and Rte_IStatus that use the data handles to access the data copies created within the generated RTE. For example:

```
#define Rte_IWrite_re1_p_a(s,v)  ((s)->re1_p_a->value = (v))
#define Rte_IWrite_re2_p_a(s,v)  ((s)->re2_p_a->value = (v))
#define Rte_IRead_re2_p_a(s,v)   ((s)->re2_p_a->value)
#define Rte_IStatus_re2_p_a(s)   ((s)->re2_p_a->status)
```

The definitions of Rte_IRead, Rte_IWrite, and Rte_IStatus are type-safe since an attempt to assign an incorrect type will be detected by the compiler.

For source code component that does **not** use multiple instantiation the definitions of Rte_IRead, Rte_IWrite, and Rte_IStatus can remain as above or vendor specific optimizations can be applied without loss of portability.

The values assigned to data handles within *instances* of the component data structure created within the generated RTE depend on the mapping of tasks and runnables – See Section 5.2.4.

5.4.2.2 Per-instance Memory Handles Section

The Per-instance Memory Section Handles section enables to access instance specific memory (sections).

[SWS_Rte_02301] ⌈ The CDS shall contain a handle for each Per-instance Memory. This handle member shall be named `Pim_<name>` where `<name>` is the per-instance memory name. ⌋*(SRS_BSW_00305, SRS_Rte_00051, SRS_Rte_00013)*

The Per-instance Memory Handles are typed; **[SWS_Rte_02302]** ⌈ The data type of each Per-instance Memory Handle shall be a pointer to the type of the per instance memory that is defined in the *Application Header* file. ⌋*(SRS_Rte_00051, SRS_Rte_00013)*

The RTE supports the access to the per-instance memories by the `Rte_Pim` API.

[SWS_Rte_06527] ⌈ The RTE Generator shall wrap each entry of *Per-instance Memory Handles Section* in the component data structure of a variant existent `PerInstanceMemory` or `arTypedPerInstanceMemory` if the variability shall be implemented.

```
#if (<condition>)

<Per-instance Memory Handles Section Entry>

#endif
```

where `condition` is the condition value macro of the `VariationPoint` relevant for the variant existence of the `Rte_Pim` API (see [SWS_Rte_06518]), `Per-instance Memory Handles Section Entry` is the code according an invariant *Per-instance Memory Handles Section Entry* (see also [SWS_Rte_02301], [SWS_Rte_02302]) ⌋*(SRS_Rte_00201)*

Example 5.25

Referring to the specification items [SWS_Rte_02301], [SWS_Rte_02302], and [SWS_Rte_07133] Example 5.4 can be extended –

with respect to the software-component header:

```
struct Rte_CDS_c {
  ...
  /* per-instance memory handle section */
  Rte_PimType_c_MyMemType *Pim_mem;

  ...
};

#define Rte_Pim_mem(s) ((s)->Pim_mem)
```

and in Rte.c:

```
Rte_PimType_c_MyMemType mem1;

```

```
const struct Rte_CDS_c Rte_Instance_c1 = {
  ...
  /* per-instance memory handle section */
  /* Rte_PimType_c_MyMemType Pim_mem */
  &mem1
  ...
};
```

[SWS_Rte_08778] ⌈ If the software component does not support multiple instantiation, the per-instance memory handles section shall be empty. ⌋*(SRS_Rte_00051)*

5.4.2.3 Inter Runnable Variable Handles Section

Each runnable may require separate handling for the inter runnable variables that it accesses. The indirection required for explicit access to inter runnable variables is described in section 5.4.2.7. The inter runnable variable handles section within the component data structure contains pointers to the (shadow) memory of inter runnable variables that can be directly accessed with the implicit API macros. The inter runnable variable handles section does not contain pointers for memory to handle inter runnable variables that are accessed with explicit API only.

[SWS_Rte_02636] ⌈ For each runnable and each inter runnable variable that is accessed implicitly by the runnable, there shall be exactly one inter runnable handle member within the component data structure and this inter runnable variable handle shall point to the (shadow) memory of the inter runnable variable for the runnable. ⌋*(SRS_Rte_00142)*

[SWS_Rte_01350] ⌈ The name of each inter runnable variable handle member within the component data structure shall be `Irv_<re>_<o>` where `<o>` is the Inter-Runnable Variable short name and `<re>` is short name of the runnable name. ⌋*(SRS_Rte_00142)*

[SWS_Rte_01351] ⌈ The data type of each inter runnable variable handle member shall be a pointer to the type of the inter runnable variable. ⌋*(SRS_Rte_00142)*

[SWS_Rte_06528] ⌈ The RTE Generator shall wrap each entry of *Inter Runnable Variable Handles Section* in the component data structure of a variant existent `Rte_IrvRead` or `Rte_IrvWrite` if the variability shall be implemented.

```
#if (<condition> [|| <condition>])

<Inter Runnable Variable Handles Section Entry>

#endif
```

where `condition` are the condition value macro(s) of the `VariationPoint` relevant for the variant existence of the `Rte_IrvRead` or `Rte_IrvWrite` API accessing the same *Inter Runnable Variable* (see [SWS_Rte_06519]), `Inter Runnable Variable Handles Section Entry` is the code according an invariant *Inter Runnable Variable Handles Section Entry* (see also [SWS_Rte_02636], [SWS_Rte_01350], [SWS_Rte_01351]) ⌋*(SRS_Rte_00201)*

[SWS_Rte_08779] ⌈ If the software component does not support multiple instantiation, the inter runnable variable handles section shall be empty. ⌋*(SRS_Rte_00051)*

5.4.2.4 Exclusive-area API Section

The exclusive-area API section includes exclusive areas that are accessed explicitly, using the RTE API, by the SW-C. Each entry in the section is a function pointer to the relevant RTE API function generated for the SW-C instance.

[SWS_Rte_03739] ⌈ The name of each Exclusive-area API section entry shall be <root>_<name> where <root> is either Entry or Exit and <name> is the Exclusive-area name. ⌋*(SRS_Rte_00051, SRS_Rte_00032)*

[SWS_Rte_03740] ⌈ The data type of each Exclusive-area API section entry shall be a function pointer that points to the generated RTE API function. ⌋*(SRS_Rte_00051, SRS_Rte_00032)*

[SWS_Rte_06521] ⌈ The RTE Generator shall wrap each definition of a variant existent Rte_Enter and Rte_Exit in the Exclusive-area API section according table 4.15 if the variability shall be implemented.

```
#if (<condition>)

<Exclusive-area API section entry>

#endif
```

where condition is the condition value macro of the VariationPoint relevant for the variant existence of the Rte_Enter and Rte_Exit API (see [SWS_Rte_06518]), Exclusive-area API section entry is the code according an invariant Exclusive-area section entry (see also [SWS_Rte_03739], [SWS_Rte_03740]) ⌋*(SRS_Rte_00201)*

[SWS_Rte_03812] ⌈ Entries in the Exclusive-area API section shall be sorted alphabetically. ⌋*(SRS_Rte_00051, SRS_Rte_00032)*

Note that two function pointers will be required for each accessed exclusive area; one for the Entry function and one for the Exit function.

[SWS_Rte_08780] ⌈ If the software component does not support multiple instantiation, the exclusive-area API section shall be empty. ⌋*(SRS_Rte_00051)*

5.4.2.5 Port API Section

Port API section comprises zero or more *function references* within the component data structure type that defines all API functions that access a port and can be invoked by the software-component (instance).

[SWS_Rte_02616] ⌈ The function table entries for port access shall be grouped by the port names into port data structures. ⌋*(SRS_Rte_00051)*

Each entry in the port API section of the component data structure is a "port data structure".

[SWS_Rte_02617] ⌈ The name of each *port data structure* in the component data structure shall be `<p>` where `<p>` is the port short-name. ⌋*(SRS_Rte_00051)*

[SWS_Rte_03799] ⌈ The component data structure shall contain a port data structure for port `p` only if the component supports multiple instantiation or if the `indirectAPI` attribute for `p` is set to 'true'. ⌋*(SRS_Rte_00051)*

[SWS_Rte_06522] ⌈ The RTE Generator shall wrap each *port data structure* of a variant existent `PortPrototype` if the variability shall be implemented.

```
#if (<condition>)

<port data structure>

#endif
```

where `condition` is the condition value macro of the `VariationPoint` relevant for the variant existence of the `PortPrototype` (see [SWS_Rte_06520], `port data structure` is the code according an invariant *port data structure*s (see also [SWS_Rte_02617], [SWS_Rte_03799]) ⌋*(SRS_Rte_00201)*

[SWS_Rte_03731] ⌈ The data type name for a port data structure shall be
`struct Rte_PDS_<cts>_<i>_<P/R>`

where `<cts>` is the `component type symbol` of the `AtomicSwComponentType`,

`<i>` is the port interface name and

'P' or 'R' are literals to indicate provide or require ports respectively. ⌋*(SRS_BSW_00305, SRS_Rte_00051)*

[constr_9080] The *shortNames* of *PortInterfaces* shall be unique within a software component if it supports multiple instantiation or `indirectAPI` attribute is set to 'true' ⌈ The *shortNames* of *PortInterfaces* shall be unique within a software component for each set of PPortPrototypes or RPortPrototypes if the software component supports multiple instantiation or if the `indirectAPI` attribute is set to 'true' for at least one require or provide port.

This is required to generate distinguishable Port Data Structure data types. ⌋

[SWS_Rte_08312] ⌈ The RTE generator shall reject a configuration violating the [constr_9080]. ⌋*(SRS_Rte_00051)*

[SWS_Rte_07137] ⌈ The port data structure type(s) shall be defined in the *Application Header* file. ⌋*(SRS_Rte_00051)*

A port data structure type is defined for each port interface that types a port. Thus different ports typed by the same port interface structure share the same port data structure type.

[SWS_Rte_07138] ⌈ The *Application Header* file shall contain a definition of a port data structure type for interface `i` and port type `R` or `P` only if the component supports multiple instantiation or at least one require or provide port exists that has the `indirectAPI` attribute set to 'true'. ⌋*(SRS_Rte_00051)*

[SWS_Rte_06523] ⌈ The RTE Generator shall wrap each *port data structure type* related to variant existent `PortPrototype`s if the variability shall be implemented and if all require `PortPrototype`s or all provide `PortPrototype`s are variant.

```
#if (<condition> [|| <condition>])

<port data structure type>

#endif
```

where `condition` are the condition value macro(s) of the `VariationPoint`s relevant for the variant existence of the `PortPrototype`s requiring the *port data structure type* (see [SWS_Rte_06520]), `port data structure type` is the code according an invariant *port data structure type* (see also [SWS_Rte_03731], [SWS_Rte_07138], [SWS_Rte_03730] [SWS_Rte_02620]) ⌋*(SRS_Rte_00201)*

Note: If any invariant `PortPrototype` requires the *port data structure type* it shall be defined unconditional.

[SWS_Rte_07677] ⌈ The RTE shall support an indirect API for the port access functions listed in table 5.1. ⌋*(SRS_Rte_00051)*

[SWS_Rte_03730] ⌈ A port data structure shall contain a function table entry for each API function associated with the port as referenced in table 5.1. Pure API macros, like `Rte_IRead` and other implicit API functions, do not have a function table entry. ⌋*(SRS_Rte_00051)*

API function	reference
`Rte_Send_<p>_<o>`	5.6.5
`Rte_Write_<p>_<o>`	5.6.5
`Rte_Switch_<p>_<o>`	5.6.6
`Rte_Invalidate_<p>_<o>`	5.6.7
`Rte_Feedback_<p>_<o>`	5.6.8
`Rte_SwitchAck_<p>_<o>`	5.6.9
`Rte_Read_<p>_<o>`	5.6.10

API function	reference
Rte_DRead_<p>_<o>	5.6.10
Rte_Receive_<p>_<o>	5.6.12
Rte_Call_<p>_<o>	5.6.13
Rte_Result_<p>_<o>	5.6.14
Rte_Prm_<p>_<o>	5.6.17
Rte_Mode_<p>_<o>	5.6.29
Rte_Trigger_<p>_<o>	5.6.31
Rte_IsUpdated_<p>_<o>	5.6.34

Table 5.1: Table of API functions that are referenced in the port API section.

[SWS_Rte_02620] ⌈ An API function shall only be included in a port data structure, if it is required at least by one port. ⌋*(SRS_Rte_00051)*

[SWS_Rte_02621] ⌈ If a function table entry is available in a port data structure, the corresponding function shall be implemented for all ports that use this port data structure type. API functions related to ports that are not required by the AUTOSAR configuration shall behave like those for an unconnected port. ⌋*(SRS_Rte_00051)*

APIs may be required only for some ports of a software component instance due to differences in for example the need for transmission acknowledgement. [SWS_Rte_02621] is necessary for the concept of the indirect API. It allows iteration over ports.

[SWS_Rte_01055] ⌈ The name of each function table entry in a port data structure shall be `<name>_<o>` where `<name>` is the API root (e.g. Call, Write) and `<o>` the data element or operation name. ⌋*(SRS_BSW_00305, SRS_Rte_00051)*

Requirement [SWS_Rte_01055] does *not* include the port name in the function table entry name since the port is implicit when using a port handle.

[SWS_Rte_03726] ⌈ The data type of each function table entry in a port data structure shall be a function pointer that points to the generated RTE function. ⌋*(SRS_Rte_00051)*

The signature of a generated function, and hence the definition of the function pointer type, is the same as the signature of the relevant RTE API call (see Section 5.6) with the exception that the instance handle is omitted.

Example 5.26

This example shows a port data structure for the provide ports of the interface type `i2` in an AUTOSAR SW-C `c`.

`i2` is a `SenderReceiverInterface` which contains a data element prototype of type `uint8` with `data semantics`.

If one of the provide ports of c for the interface i2 has a transmission acknowledgement defined and i2 is not used with data element invalidation, the *Application Header* file would include a port data structure type like this:

```
struct Rte_PDS_c_i2_P {
  Std_ReturnType (*Feedback_a)(uint8);
  Std_ReturnType (*Write_a)(uint8);
}
```

If the provide port p1 of the AUTOSAR SW-C c is of interface i2, the generated *Application Header* file would include the following macros to provide the direct API functions Rte_Feedback_p1_a and Rte_Write_p1_a:

```
/*direct API*/
#define Rte_Feedback_p1_a(inst,data)
        ((inst)->p1.Feedback_a)(data)
#define Rte_Write_p1_a(inst,data) ((inst)->p1.Write_a)(data)
```

[SWS_Rte_02618] ⌈ The port data structures within a component data structure shall first be sorted on the port data structure type name and then on the short name of the port. ⌋*(SRS_Rte_00051)*

The requirements [SWS_Rte_03731] and [SWS_Rte_02618] guarantee, that all port data structures within the component data structure are grouped by their interface type and require/provide-direction.

Example 5.27

This example shows the grouping of port data structures within the component data structure.

The *Application Header* file for an AUTOSAR SW-C c with three provide ports p1, p2, and p3 of interface i2 would include a block of port data structures like this:

```
struct Rte_CDS_c {
  ...
  struct Rte_PDS_c_i1_R z;

/* component data structures          *
 * for provide ports of interface i2  */
  struct Rte_PDS_c_i2_P p1;
  struct Rte_PDS_c_i2_P p2;
  struct Rte_PDS_c_i2_P p3;

/*further component data structures*/
  struct Rte_PDS_c_i2_R c;
  ...
}
```

If inst is a pointer to a component data structure, and ph is defined by

```
struct Rte_PDS_c_i2_P *ph = &(inst->p1);
```

ph points to the port data structure p1 of the instance handle inst. Since the three provide port data structures p1, p2, and p3 of interface i2 are ordered sequentially in the component data structure, ph can also be interpreted as an array of port data structures. E.g., ph[2] is equal to inst->p3.

In the following, ph will be called a port handle.

[SWS_Rte_01343] ⌈ RTE shall create *port handle types* for each port data structure using typedef to a pointer to the appropriate port data structure. ⌋*(SRS_Rte_00051)*

[SWS_Rte_01342] ⌈ The *port handle type* name shall be Rte_PortHandle_<i>_<P/R> where <i> is the port interface name and 'P' or 'R' are literals to indicate provide or receive ports respectively. ⌋*(SRS_Rte_00051)*

[SWS_Rte_06524] ⌈ The RTE Generator shall wrap each *port handle type* related to variant existent PortPrototypes if the variability shall be implemented and if all require PortPrototypes or all provide PortPrototypes are variant.

```
#if (<condition> [|| <condition>])

<port handle type>

#endif
```

where condition are the condition value macro(s) of the VariationPoints relevant for the variant existence of the PortPrototypes requiring the *port data structure type* (see [SWS_Rte_06520]), port data structure type is the code according an invariant *port data structure type* (see also [SWS_Rte_01343], [SWS_Rte_01342]) ⌋*(SRS_Rte_00201)*

[SWS_Rte_01053] ⌈ The port handle types shall be written to the application header file. ⌋*(SRS_Rte_00051)*

RTE provides port handles for access to the arrays of port data structures of the same interface type and provide/receive direction by the macro Rte_Ports, see section 5.6.1, and to the number of similar ports by the macro Rte_NPorts, see 5.6.1.

Example 5.28

For the provide port i2 of AUTOSAR SW-C c from example 5.26, the following port handle type will be defined in the *Application Header* file:

```
typedef struct Rte_PDS_c_i2_P *Rte_PortHandle_i2_P;
```

The macros to access the port handles for the indirect API might look like this in the generated *Application Header* file:

```
/*indirect (port oriented) API*/
#define Rte_Ports_i2_P(inst) &((inst)->p1)
#define Rte_NPorts_i2_P(inst) 3
```

So, the port handle ph of the previous example 5.27 could be defined by a user as:

```
Rte_PortHandle_i2_P ph = Rte_Ports_i2_P(inst);
```

To write ‘49’ on all ports p1 to p3, the indirect API can be used within the software component as follows:

```
uint8 p;
Rte_PortHandle_i2_P ph = Rte_Ports_i2_P(inst);
for(p=0;p<Rte_NPorts_i_P(inst);p++) {
  ph[p].Write_a(49);
}
```

Software components may also want to set up their own port handle arrays to iterate over a smaller sub group than all ports with the same interface and direction. Rte_Port can be used to pick the port handle for one specific port, see 5.6.3.

[SWS_Rte_08781] ⌈ If the software component does not support multiple instantiation, the port API section shall be empty. ⌋*(SRS_Rte_00051)*

5.4.2.6 Calibration Parameter Handles Section

The RTE is required to support access to calibration parameters derived by *per-instance* ParameterDataPrototypes (see 4.2.8.3) using the Rte_CData (see section 5.6.16).

[SWS_Rte_03835] ⌈ The name of each Calibration parameter handle shall be CData_<name> where <name> is the ParameterDataPrototype name. ⌋*(SRS_Rte_00051, SRS_Rte_00154, SRS_Rte_00155)*

[SWS_Rte_03949] ⌈ The type of each calibration parameter handle shall be a function pointer that points to the generated RTE function. ⌋*(SRS_Rte_00051, SRS_Rte_00154, SRS_Rte_00155)*

Note that accesses to ParameterDataPrototypes within ParameterSwComponentTypes do not result in any handles within this section since the generated Rte_Prm (see section 5.6.17) API is accessed either directly (single instantiation) or through handles in the port API section (multiple instantiation). Likewise, access to *shared* ParameterDataPrototypes does not result in any handle in the Calibration Parameter Handles Section since, by definition, no per-instance data is present.

[SWS_Rte_08782] ⌈ If the software component does not support multiple instantiation, the calibration parameter handles section shall be empty. ⌋*(SRS_Rte_00051)*

5.4.2.7 Inter Runnable Variable API Section

The Inter Runnable Variable API section comprises zero or more *function table entries* within the component data structure type that defines all explicit API functions to access an inter runnable variable by the software-component (instance). The API for implicit access of inter runnable variables does not have any *function table entries*, since the implicit API uses macro's to access the inter runnable variables or their shadow memory directly, see section 5.4.2.3.

Since the entries of this section are only required to access the explicit InterRunnableVariable API if a software component supports multiple instantiation, it shall be omitted for software components which do not support multiple instantiation.

[SWS_Rte_03725] ⌈ If the component supports multiple instantiation, the member name of each function table entry within the component data structure shall be `<name>_<re>_<o>` where `<name>` is the API root (e.g. IrvRead), `<re>` the runnable name, and `<o>` the inter runnable variable name. ⌋*(SRS_Rte_00051)*

[SWS_Rte_03752] ⌈ The data type of each function table entry shall be a function pointer that points to the generated RTE function. ⌋*(SRS_Rte_00051)*

The signature of a generated function, and hence the definition of the function pointer type, is the same as the signature of the relevant RTE API call (see Section 5.6) with the exception that the instance handle is omitted.

[SWS_Rte_02623] ⌈ If the component supports multiple instantiation, the *Inter Runnable Variable API Section* shall contain pointers to the following API functions:

API function	reference
`Rte_IrvRead_<re>_<o>`	5.6.25
`Rte_IrvWrite_<re>_<o>`	5.6.26

Table 5.2: Table of API functions that are referenced in the inter runnable variable API section

⌋*(SRS_Rte_00051)*

[SWS_Rte_06525] ⌈ The RTE Generator shall wrap each entry of *Inter Runnable Variable API Section* in the component data structure of a variant existent `Rte_IrvRead` or `Rte_IrvWrite` API if the variability shall be implemented.

```
#if (<condition>)

<Inter Runnable Variable API Section Entry>

#endif
```

where `condition` is the condition value macro of the `VariationPoint` relevant for the variant existence of the `Rte_IrvRead` or `Rte_IrvWrite` API (see [SWS_Rte_06519]), `Inter Runnable Variable API Section Entry` is the code according an invariant *Inter Runnable Variable API Section Entry* (see also [SWS_Rte_03725], [SWS_Rte_03752], [SWS_Rte_02623]) ⌋*(SRS_Rte_00201)*

[SWS_Rte_03791] ⌈ If the software component does not support multiple instantiation, the inter runnable variable API section shall be empty. ⌋*(SRS_Rte_00051)*

[SWS_Rte_08783] ⌈ If the software component does not support multiple instantiation, the inter runnable variable API section shall be empty. ⌋*(SRS_Rte_00051)*

5.4.2.8 Inter Runnable Triggering API Section

The Inter Runnable Triggering API Section includes the Inter Runnable Triggering API handles. Each entry in the section is a function pointer to the relevant RTE API function generated for the SW-C instance.

[SWS_Rte_07226] ⌈ The name of each *Inter Runnable Triggering handle* shall be `Rte_IrTrigger_<re>_<name>` where `<re>` is the name of the runnable entity the API might be used and `<name>` is the name of the `InternalTriggeringPoint`. ⌋*(SRS_Rte_00051, SRS_Rte_00163)*

[SWS_Rte_07227] ⌈ The data type of each *Inter Runnable Triggering handle entry* shall be a function pointer that points to the generated RTE API function defined in 5.6.32. ⌋*(SRS_Rte_00051, SRS_Rte_00163)*

[SWS_Rte_06526] ⌈ The RTE Generator shall wrap each entry of *Inter Runnable Triggering handle* in the component data structure of a variant existent `Rte_IrTrigger` API if the variability shall be implemented.

```
#if (<condition>)

<Inter Runnable Variable API Section Entry>

#endif
```

where `condition` is the condition value macro of the `VariationPoint` relevant for the variant existence of the `Rte_IrTrigger` API (see [SWS_Rte_06519], `Inter Runnable Variable API Section Entry` is the code according an invariant *Inter Runnable Variable API Section Entry* (see also [SWS_Rte_03725], [SWS_Rte_03752], [SWS_Rte_02623]) ⌋*(SRS_Rte_00201)*

[SWS_Rte_07228] ⌈ Entries in the Inter Runnable Triggering handles section shall be sorted alphabetically. ⌋*(SRS_Rte_00051, SRS_Rte_00163)*

[SWS_Rte_08784] ⌈ If the software component does not support multiple instantiation, the inter runnable triggering API section shall be empty. ⌋*(SRS_Rte_00051)*

5.4.2.9 Instance Id Section

[SWS_Rte_07838] ⌈ If a software component type supports multiple instantiation, the RTE generator shall add in the Component Data Structure Instance Id Section an element named Instance_Id of type uint8. ⌋*(SRS_Rte_00011, SRS_Rte_00051, SRS_Rte_00244)*

[SWS_Rte_07839] ⌈ For each prototype of a software component type that supports multiple instantiation, the RTE generator shall set the value of the element Instance_Id from 0 to N-1 according to the number (N) of software component prototypes and according to the names of the software component prototypes sorted alphabetically (ASCII / ISO 8859-1 code in ascending order). ⌋*(SRS_Rte_00011, SRS_Rte_00051, SRS_Rte_00244)*

Example: Two prototypes (instances) named A and B of a software component type exist:

- Instance_Id for instance A takes the value 0.
- Instance_Id for instance B takes the value 1.

Note: The Instance_Id should not be used by the runnable implementation. The Instance_Id has been created to support implementation of bypass on software component that supports multiple instantiation.

[SWS_Rte_08785] ⌈ If the software component does not support multiple instantiation, the instance id section shall be empty. ⌋*(SRS_Rte_00051)*

5.4.2.10 Vendor Specific Section

The vendor specific section is used to contain any vendor specific data required to be supported for each instances. By definition the contents of this section are outside the scope of this chapter and only available for use by the RTE generator responsible for the “RTE Generation” phase.

[SWS_Rte_08786] ⌈ If the software component does not support multiple instantiation, the vendor specific section shall be empty. ⌋*(SRS_Rte_00051)*

5.5 API Data Types

Besides the API functions for accessing RTE services, the API also contains RTE-specific data types.

5.5.1 Std_ReturnType

The specification in [29] specifies a standard API return type `Std_ReturnType`. The `Std_ReturnType` defines the "‘status"’ and "‘error values"’ returned by API functions. It is defined as a `uint8` type. The value “0” is reserved for “No error occurred”.

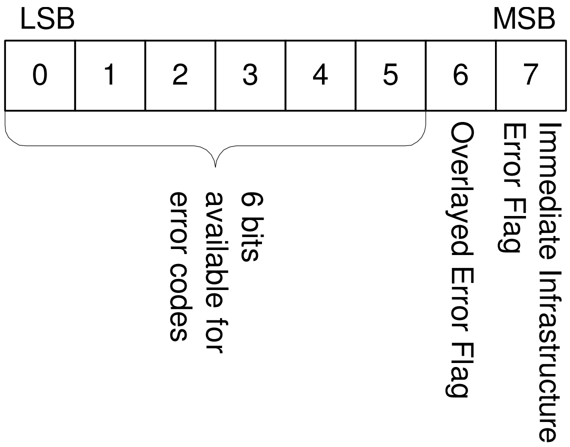


Figure 5.14: Bit-Layout of the Std_ReturnType

Figure 5.14 shows the general layout of `Std_ReturnType`.

The two most significant bits of the `Std_ReturnType` are reserved flags:

- The most significant bit 7 of `Std_ReturnType` is the “Immediate Infrastructure Error Flag” with the following values
  - “1” the error code indicates an immediate infrastructure error.
  - “0” the error code indicates no immediate infrastructure error.
- The second most significant bit 6 of `Std_ReturnType` is the Overlayed Error Flag. The use of this flag depends on the context and will be explained in table 5.4.

In order to avoid explicit access to bit numbers in the code, the RTE provides the three following macros that enables an application to check the return value of an API:

- **[SWS_Rte_07404]** ⌈ For infrastructure errors, this macro is a boolean expression that is true if the corresponding bit is set:

```
#define Rte_IsInfrastructureError(status)  ((status & 128U) !=
    0)
```

⌋

- **[SWS_Rte_07405]** ⌈For overlayed errors, this macro is a boolean expression that is true if the corresponding bit is set:

```
#define Rte_HasOverlayedError(status)  ((status & 64U) != 0)
```

⌋

- **[SWS_Rte_07406]** ⌈For reading only the application error code without the eventual overlayed error, the following macro returns the lower 6 bits of the error code:

```
#define Rte_ApplicationError(status)  (status & 63U)
```

⌋

5.5.1.1 Infrastructure Errors

Infrastructure errors are split into two groups:

- "Immediate Infrastructure Errors" can be associated with the currently available data set. These `Immediate Infrastructure Errors` are mutually exclusive. Only one of these errors can be notified to a SW-C with one API call.

  [SWS_Rte_02593] ⌈ `Immediate Infrastructure Errors` shall override any application level error. ⌋*(SRS_Rte_00084, SRS_Rte_00123)*

  `Immediate Infrastructure Error` codes are used on the receiver side for errors that result in no reception of application data and application errors.

  An `Immediate Infrastructure Error` is indicated in the `Std_ReturnType` by the `Immediate Infrastructure Error Flag` being set.

- "Overlayed Errors" are associated with communication events that happened after the reception of the currently available data set, e.g., `data element outdated` notification, or loss of data elements due to queue overflow.

  [SWS_Rte_01318] ⌈ `Overlayed Error Flags` shall be reported using the unique bit of the `Overlayed Error Flag` within the `Std_ReturnType` type. ⌋*(SRS_Rte_00084, SRS_Rte_00094)*

  An `Overlayed Error` can be combined with any other application or infrastructure error code.

5.5.1.2 Application Errors

[SWS_Rte_02573] ⌈ RTE shall support application errors with the following format definition: Application errors are coded in the least significant 6 bits of `Std_ReturnType` with the `Immediate Infrastructure Error Flag` set to "0". The application error code does not use the `Overlayed Error Flag`. ⌋*(SRS_Rte_00124)*

This results in the following value range for application errors:

range	minimum value	maximum value
application errors	1	63

Table 5.3: application error value range

In client server communication, the server may return any value within the application error range. The client will then receive one of the following:

- An `Immediate Infrastructure Error` to indicate that the communication was not successful, or

- The server return code, or
- The server return code might be overlayed by the Overlayed Error Flag in a future release of RTE. In this release, there is no overlayed error defined for client server communication.

The client can filter the return value, e.g., by using the following code:

```
Std_ReturnType status;
status = Rte_Call_<p>_<o>(<instance>, <parameters>);
if (Rte_HasOverlayedError(status)) {
    /* handle overlayed error flag                            *
     * in this release of the RTE, the flag is reserved *
     * but not used for client server communication     */

}

if(Rte_IsInfrastructureError(status)) {
    /* handle infrastructure error                      */
}
else {
    /* handle application error with error code status  */
    status = Rte_ApplicationError(status);
}
```

5.5.1.3 Predefined Error Codes

For client server communication, application error values are defined per client server interface and shall be passed to the RTE with the interface configuration.

The following standard error and status identifiers are defined:

Symbolic name	Value	Comments
[SWS_Rte_01058] ⌈ RTE_E_OK ⌋(*SRS_BSW_00327*)	0	No error occurred.
Standard Application Error Values:		
[SWS_Rte_02594] ⌈ RTE_E_INVALID ⌋(*SRS_BSW_00327*, *SRS_Rte_00078*)	1	Generic application error indicated by signal invalidation in sender receiver communication with data semantics on the receiver side.
To be defined by the corresponding AUTOSAR Service	1	Returned by AUTOSAR Services to indicate a generic application error.
Immediate Infrastructure Error codes		

Symbolic name	Value	Comments
[SWS_Rte_01060] ⌈ RTE_E_COM_STOPPED ⌋(*SRS_BSW_00327*)	128	An IPDU group was disabled while the application was waiting for the transmission acknowledgment. No value is available. This is not considered a fault, since the IPDU group is switched off on purpose. This semantics are as follows: • The OUT buffers of a client or of explicit read APIs are not modified • no runnable with startOnEvent on a DataReceivedEvent for this VariableDataPrototype is triggered. • the buffers for implicit read access will keep the previous value.
[SWS_Rte_01064] ⌈ RTE_E_TIMEOUT ⌋(*SRS_BSW_00327, SRS_Rte_00069*)	129	A blocking API call returned due to expiry of a local timeout rather than the intended result. OUT buffers are not modified. The interpretation of this being an error depends on the application.
[SWS_Rte_01317] ⌈ RTE_E_LIMIT ⌋(*SRS_BSW_00327*)	130	A internal RTE limit has been exceeded. Request could not be handled. OUT buffers are not modified.
[SWS_Rte_01061] ⌈ RTE_E_NO_DATA ⌋(*SRS_BSW_00327*)	131	An explicit read API call returned no data. (This is no error.)
[SWS_Rte_01065] ⌈ RTE_E_TRANSMIT_ACK ⌋(*SRS_BSW_00327*)	132	Transmission acknowledgement received.
[SWS_Rte_07384] ⌈ RTE_E_NEVER_RECEIVED ⌋(*SRS_BSW_00327, SRS_Rte_00184*)	133	No data received for the corresponding unqueued data element since system start or partition restart.
[SWS_Rte_07655] ⌈ RTE_E_UNCONNECTED ⌋(*SRS_BSW_00327, SRS_Rte_00139, SRS_Rte_00200*)	134	The port used for communication is not connected.

Symbolic name	Value	Comments
[SWS_Rte_02739] ⌈ RTE_E_IN_EXCLUSIVE_AREA ⌋*(SRS_BSW_00327)*	135	The error is returned by a blocking API and indicates that the runnable could not enter a wait state, because one ExecutableEntity of the current task's call stack has entered an ExclusiveArea.
[SWS_Rte_02757] ⌈ RTE_E_SEG_FAULT ⌋*(SRS_BSW_00327)*	136	The error can be returned by an RTE API, if the parameters contain a direct or indirect reference to memory that is not accessible from the callers partition.
[SWS_Rte_08065] ⌈ RTE_E_OUT_OF_RANGE ⌋*(SRS_BSW_00327, SRS_Rte_00180)*	137	The received data is out of range.
[SWS_Rte_08725] ⌈ RTE_E_SERIALIZATION_ERROR ⌋*(SRS_Rte_00091, SRS_BSW_00327)*	138	An error during serialization or deserialization occured.
[SWS_Rte_08726] ⌈ RTE_E_SERIALIZATION_LIMIT ⌋*(SRS_Rte_00091, SRS_BSW_00327)*	139	Buffer for serialization operation could not be created.

Overlayed Errors These errors do not refer to the data returned with the API. They can be overlayed with other Application- or Immediate Infrastructure Errors.		
[SWS_Rte_02571] ⌈ RTE_E_LOST_DATA ⌋*(SRS_BSW_00327, SRS_Rte_00107, SRS_Rte_00110, SRS_Rte_00094)*	64	An API call for reading received data with event semantics indicates that some incoming data has been lost due to an overflow of the receive queue or due to an error of the underlying communication stack.
[SWS_Rte_02702] ⌈ RTE_E_MAX_AGE_EXCEEDED ⌋*(SRS_BSW_00327, SRS_Rte_00078)*	64	An API call for reading received data with data semantics indicates that the available data has exceeded the aliveTimeout limit. A COM signal outdated callback will result in this error.

Table 5.4: RTE Error and Status values

The underlying type for Std_ReturnType is defined as a uint8 for reasons of compatibility – it avoids RTEs from different vendors assuming a different size if an enum was the underlying type. Consequently, #define is used to declare the error values:

```
typedef uint8 Std_ReturnType;

#define RTE_E_OK 0U
```

[SWS_Rte_01269] ⌈ The standard errors as defined in table 5.4 including RTE_E_OK shall be defined in the RTE Header File. ⌋*(SRS_Rte_00051)*

[SWS_Rte_02575] ⌈ Application Error Identifiers with exception of RTE_E_INVALID shall be defined in the Application Header File. ⌋*(SRS_Rte_00124, SRS_Rte_00167)*

[SWS_Rte_02576] ⌈ The application errors shall have a symbolic name defined as follows:

```
#define RTE_E_<interface>_<error> <error value>U
```

where <interface> PortInterface and <error> ApplicationError are the interface and error names from the configuration.[5] ⌋*(SRS_Rte_00123)*

An Std_ReturnType value can be directly compared (for equality) with the above pre-defined error identifiers.

[SWS_Rte_07143] ⌈ The RTE generator shall generate symbolic name for application errors with equal <interface> name, <error> name and <error value> only once. ⌋*(SRS_Rte_00165)*

5.5.2 Rte_Instance

The Rte_Instance data type defines the handle used to access instance specific information from the component data structure.

[SWS_Rte_01148] ⌈ The underlying data type for an instance handle shall be a pointer to a *Component Data Structure*. ⌋*(SRS_Rte_00011, SRS_Rte_00051)*

The component data structure (see Section 5.4.2) is uniquely defined for a component type and therefore the data type for the instance handle is automatically unique for each component type.

The instance handle type is defined in the application header file [SWS_Rte_01007].

To avoid long and complex type names within SW-C code the following requirement imposes a fixed name on the instance handle data type.

[SWS_Rte_01150] ⌈ The name of the instance handle type shall be defined, using typedef as Rte_[Byps_]Instance. [Byps_] is an optionnal infix used when component wrapper method for bypass support is enabled for the related software component type (See chapter 4.9.2). ⌋*(SRS_BSW_00305)*

[5] No additional capitalization is applied to the names.

[SWS_Rte_06810] ⌈ The instance handle typedef shall use the `CONSTP2CONST` macro with `memclass AUTOMATIC` and `ptrclass RTE_CONST`. ⌋*(SRS_BSW_00007)*

Requirement [SWS_Rte_06810] uses `memclass AUTOMATIC` rather than `memclass TYPEDEF` because the instance handle is used as a function parameter and hence automatic. This means the typedef is guaranteed to be compatible when the RTE implementation must use a pointer to the component data structure rather than the instance handle typedef.

5.5.3 RTE Modes

An `Rte_ModeType` is used to hold the identifiers for the `ModeDeclarations` of a `ModeDeclarationGroup`.

[SWS_Rte_02627] ⌈ For each `ModeDeclarationGroupPrototype`, used in the SW-C's ports, the *Application Types Header File* shall contain a type definition

```
#ifndef RTE_MODETYPE_<prefix><ModeDeclarationGroup>
#define RTE_MODETYPE_<prefix><ModeDeclarationGroup>
typedef <type> Rte_ModeType_<prefix><ModeDeclarationGroup>;
#endif
```

where `<ModeDeclarationGroup>` is the `shortName` of the `ModeDeclarationGroup`,

`<prefix>` is the optional `prefix` attribute defined by the `IncludedModeDeclarationGroupSet` referring the `ModeDeclarationGroup` and

`<type>` is the `shortName` of the `ImplementationDataType` which is mapped to the `ModeDeclarationGroup` by a `ModeRequestTypeMap`. ⌋*(SRS_Rte_00144)*

Note: *ImplementationDataTypes* are generated in the *RTE Types Header* file.

Note: The type definition specified in [SWS_Rte_02627] is deprecated to avoid incompatible or duplicate type definitions. It is recommended to not use this type in software components anymore (see [SWS_Rte_02628]).

[SWS_Rte_02738] guarantees that for each `ModeDeclarationGroup`, used in the SW-C's ports, there is a unique mapping to an `ImplementationDataType`.

For a `ModeDeclarationGroup` of category "`ALPHABETIC_ORDER`", the value `<n>U` within the `Rte_ModeType_<ModeDeclarationGroup>` is reserved to express a transition between modes, where `<n>` is the number of modes declared within the group. For `ModeDeclarationGroups` of category "`EXPLICIT_ORDER`", a transition between modes is represented by the explicitly specified `onTransitionValue`.

[SWS_Rte_02659] ⌈ For each `ModeDeclarationGroup` of category "`ALPHABETIC_ORDER`", used in the SW-C's ports, the *Application Types Header File* shall contain a definition

```
#ifndef RTE_TRANSITION_<prefix><ModeDeclarationGroup>
#define RTE_TRANSITION_<prefix><ModeDeclarationGroup> <n>U
```

```
#endif
```

where <ModeDeclarationGroup> is the *shortName* of the ModeDeclarationGroup,

<prefix> is the optional prefix attribute defined by the IncludedModeDeclarationGroupSet referring the ModeDeclarationGroup and

<n> is the number of modes declared within the group.[6] ⌋*(SRS_Rte_00144)*

[SWS_Rte_03858] ⌈ For each ModeDeclarationGroup of category "EXPLICIT_ORDER", used in the SW-C's ports, the *Application Types Header File* shall contain a definition

```
#ifndef RTE_TRANSITION_<prefix><ModeDeclarationGroup>
#define RTE_TRANSITION_<prefix><ModeDeclarationGroup> \
          <onTransitionValue>U
#endif
```

where <ModeDeclarationGroup> is the *shortName* of the ModeDeclarationGroup,

<prefix> is the optional prefix attribute defined by the IncludedModeDeclarationGroupSet referring the ModeDeclarationGroup and

<onTransitionValue> is the onTransitionValue of the *ModeDeclarationGroup*. ⌋*(SRS_Rte_00144)*

[SWS_Rte_07640] ⌈ The RTE Generator shall reject configurations where two ModeDeclarationGroups, used in the SW-C's ports, with the same name but different ModeDeclarations exists. ⌋*(SRS_Rte_00144, SRS_Rte_00018)*

The rational for [SWS_Rte_07640] is to protect against conditions which would lead to [SWS_Rte_02659]and [SWS_Rte_02627] to generate conflicting types or macro definitions.

[SWS_Rte_02568] ⌈ For each mode of a ModeDeclarationGroup of category "ALPHABETIC_ORDER", used in the SW-C's ports, the *Application Types Header File* shall contain a definition

```
#ifndef RTE_MODE_<prefix><ModeDeclarationGroup>_<ModeDeclaration>
#define RTE_MODE_<prefix><ModeDeclarationGroup>_<ModeDeclaration> \
          <index>U
#endif
```

where <ModeDeclarationGroup> is the short name of the ModeDeclarationGroup,

<prefix> is the optional prefix attribute defined by the IncludedModeDeclarationGroupSet referring the ModeDeclarationGroup

<ModeDeclaration> is the *shortName* of a ModeDeclaration, and <index> is the index of the ModeDeclarations in alphabetic ordering (ASCII / ISO 8859-1 code

[6] No additional capitalization is applied to the names.

in ascending order) of the *shortNames* within the ModeDeclarationGroup[7]. The lowest index shall be '0' and therefore the range of assigned values is 0..<n-1> where <n> is the number of modes declared within the group. ⌋*(SRS_Rte_00144)*

[SWS_Rte_03859] ⌈ For each mode of a ModeDeclarationGroup of category "EXPLICIT_ORDER", used in the SW-C's ports, the *Application Types Header File* shall contain a definition

```
#ifndef RTE_MODE_<prefix><ModeDeclarationGroup>_<ModeDeclaration>
#define RTE_MODE_<prefix><ModeDeclarationGroup>_<ModeDeclaration> \
          <value>U
#endif
```

where <ModeDeclarationGroup> is the short name of the ModeDeclarationGroup,

<prefix> is the optional prefix attribute defined by the IncludedModeDeclarationGroupSet referring the ModeDeclarationGroup

<ModeDeclaration> is the *shortName* of a ModeDeclaration, and <value> is the *value* specified at the ModeDeclaration. ⌋*(SRS_Rte_00144)*

5.5.4 Enumeration Data Types

Enumeration is not a plain primitive ImplementationDataType. Rather a range of integers can be used as a structural description. The mapping of integers on "labels" in the enumeration is actually modeled in the SwC-T with the semantics class CompuMethod of a SwDataDefProps [2]. Enumeration data types are modeled as ImplementationDataTypes having a SwDataDefProps referencing a CompuMethod that contains only CompuScales with point ranges (i. e. lower and upper limit of a CompuScale are identical).

[SWS_Rte_03809] ⌈ The *Application Types Header File* shall include the definitions of all constants of ImplementationDataTypes and ApplicationDataTypes for each ImplementationDataType/ApplicationDataTypes used (referenced) by this software component.

This includes constants for CompuMethods referenced by ImplementationDataTypeElements of ImplementationDataTypes directly referenced by the software component and constants for CompuMethods of ImplementationDataTypes which are referenced indirectly via ImplementationDataTypes / ImplementationDataTypeElements of category TYPE_REFERENCE. ⌋*(SRS_Rte_00167)*

[SWS_Rte_03809] is applicable regardless if the AutosarDataType is referenced by an DataPrototypes in PortInterfaces used for SwComponentTypes Ports, DataPrototypes defined in the InternalBehavior of the SwComponentType or AutosarDataTypes which are only referenced by the IncludedDataTypeSet.

[7] No additional capitalization is applied to the names.

This requirement ensures the availability of AutosarDataType constants for the internal use in AUTOSAR software components, for example enumeration constants.

The name of those constants bases on the CompuScale symbolic name as defined in [TPS_SWCT_01569].

[SWS_Rte_03810] ⌈ For each CompuScale which has a point range and is located in the *compuInternalToPhys* container of a CompuMethod referenced by an *ImplementationDataType* or ApplicationPrimitiveDataType according [SWS_Rte_03809] with *category* "TEXTTABLE", "SCALE_LINEAR_AND_TEXTTABLE", "SCALE_RATIONAL_AND_TEXTTABLE", or BITFIELD_TEXTTABLE, the *Application Types Header File* file shall contain a definition

```
#ifndef <prefix><EnumLiteral>
#define <prefix><EnumLiteral> <value><suffix>
#endif /* <prefix><EnumLiteral> */
```

where the name of the enumeration literal <EnumLiteral> is derived according to the following rule:

if (attribute symbol of CompuScale is available and not empty) {
    <EnumLiteral> **:=** C identifier specified in symbol attribute of CompuScale
} **else** {
    if (string specified in the VT element of the CompuConst of the CompuScale
        is a valid C identifier) {
        <EnumLiteral> **:=**
            string specified in the VT element of the CompuConst of the CompuScale
    } **else** {
        if (attribute shortLabel of CompuScale is available and not empty) {
            <EnumLiteral> **:=**
                string specified in shortLabel attribute of CompuScale
        }
    }
}

<prefix> is the optional literalPrefix attribute defined by the IncludedDataTypeSet referring the AutosarDataType using the CompuMethod.
<value> is the value representing the CompuScale's point range.
<suffix> shall be "U" for unsigned data types and empty for signed data types. ⌋*(SRS_Rte_00167)*

Please note that the prefix can either be defined that the IncludedDataTypeSet with a literalPrefix attribute references the ApplicationDataType or it references the ImplementationDataType.

[SWS_Rte_03810] implies that the RTE does add prefix to the names of the enumeration constants on explicit demand only. This is necessary in order to handle enumeration constants supplied by Basic Software modules which all use their own prefix convention. Such Enumeration constant names have to be unique in the whole AUTOSAR system.

[SWS_Rte_08401] ⌈ In the case that the same ImplementationDataType or ApplicationPrimitiveDataType is referenced via different IncludedDataTypeSets with different literalPrefix attributes, the definition according to [SWS_Rte_03810] has to be provided once for each different literalPrefix. ⌋*(SRS_Rte_00167)*

[SWS_Rte_03851] ⌈ If the input of the RTE generator contains a CompuMethod with category "TEXTTABLE", "SCALE_LINEAR_AND_TEXTTABLE", "SCALE_RATIONAL_AND_TEXTTABLE", or BITFIELD_TEXTTABLE that contains a CompuScale with a point range, and

- neither the attribute symbol of the CompuScale is available and not empty,
- nor the string specified in the VT element of the CompuConst of the CompuScale is a valid C identifier,
- nor the attribute shortLabel of CompuScale is available and not empty,

the RTE generator shall reject this input as an invalid configuration. ⌋*(SRS_Rte_00018)*

[SWS_Rte_03813] ⌈ The RTE shall reject configurations where the same software component type uses ImplementationDataTypes and ApplicationPrimitiveDataTypes referencing two or more CompuMethods with category "TEXTTABLE", "SCALE_LINEAR_AND_TEXTTABLE", "SCALE_RATIONAL_AND_TEXTTABLE", or BITFIELD_TEXTTABLE that both contain a CompuScale with a different point range and an identical CompuScale symbolic names as an invalid configuration. The only exception is that the usage of the ImplementationDataTypes and ApplicationPrimitiveDataTypes are defined with non identical <literalPrefix>es. ⌋*(SRS_Rte_00018)*

[SWS_Rte_07175] ⌈ The RTE generator shall reject configurations violating the [constr_1133]. ⌋*(SRS_Rte_00018)*

This rejects configurations where an ImplementationDataType or an ApplicationPrimitiveDataType references a CompuMethod which is of category "TEXTTABLE", "SCALE_LINEAR_AND_TEXTTABLE", "SCALE_RATIONAL_AND_TEXTTABLE", or BITFIELD_TEXTTABLE and has CompuScales with identical CompuScale symbolic names but different CompuScale.lowerLimit or CompuScale.upperLimit.

Note that there might exist additional CompuScales with non-point ranges inside a CompuMethod of category "TEXTTABLE", "SCALE_LINEAR_AND_TEXTTABLE", "SCALE_RATIONAL_AND_TEXTTABLE", or BITFIELD_TEXTTABLE , but for those no enumeration literals are generated by the RTE generator.

The RTE generator does not support the use of C enums for DataPrototypes used in application software.

[SWS_Rte_03862] ⌈ The RTE generator shall reject configurations violating the [constr_1244], so where a DataPrototype that is used in an AtomicSwComponentType has set the swDataDefProps.additionalNativeTypeQualifier attribute set to enum. ⌋(*SRS_Rte_00018*)

5.5.5 Range Data Types

For the ApplicationPrimitiveDataType a Range might be specified by referencing a data constraint (dataConstr) giving the lowerLimit and the upperLimit. To allow a Software Component the access to these values two definitions for these values shall be generated.

[SWS_Rte_05051] ⌈ The *Application Types Header File* shall include the definitions of all lowerLimit and upperLimit constants of each ApplicationPrimitiveDataType used by this software component once per ApplicationPrimitiveDataType if the ApplicationPrimitiveDataType is not referenced via different IncludedDataTypeSets. ⌋(*SRS_Rte_00167*)

[SWS_Rte_08402] ⌈ The Application Types Header File shall include the definitions of all lowerLimit and upperLimit constants of each ApplicationPrimitiveDataType used by this software component for each combination of different literalPrefix and ApplicationPrimitiveDataType when the same ImplementationDataType or ApplicationPrimitiveDataType is referenced via different IncludedDataTypeSets. ⌋(*SRS_Rte_00167*)

[SWS_Rte_05052] ⌈ The lowerLimit and upperLimit constants for *ApplicationPrimitiveDataType* referencing a DataConstr shall be generated by RTE generator in the *Application Type Header File* as:

```
#define <prefix><DataType>_LowerLimit <lowerValue><suffix>
#define <prefix><DataType>_UpperLimit <upperValue><suffix>
```

where <DataType> is the name of the ApplicationPrimitiveDataType used by the software component.

<prefix> is the optional literalPrefix attribute defined by the IncludedDataTypeSet referring the AutosarDataType to which the DataConstr belongs.

<lowerValue> and <upperValue> are the values lowerLimit and upperLimit of the dataConstr referenced by the *ApplicationPrimitiveDataType* onto which the corresponding CompuMethod has been applied (see [SWS_Rte_07038]). The values in the macro definitions shall always reflect the closed interval, regardless of the interval type specified by the dataConstr.

<suffix> shall be "U" for unsigned data types and empty for signed data types. ⌋(*SRS_Rte_00167*)

Please note that [SWS_Rte_07196] is not applicable for [SWS_Rte_05052]. Further on it's possible that a DataPrototype using an *ApplicationPrimitiveDataType* might reference additional dataConstr (see [SWS_Rte_07196]). In this case the upperLimit and lowerLimit definitions according [SWS_Rte_05052] do not reflect the real applicable range of the DataPrototype. No macros are generated for DataPrototype specific data constraints.

Please note that the prefix can either be defined that the IncludedDataTypeSet with a literalPrefix attribute references the ApplicationDataType or it references the ImplementationDataType.

Rationale: ApplicationPrimitiveDataType is taken as the basis for the generation of limits (as opposed to take the corresponding ImplementationDataType) because the limits defined on the ImplementationDataType) may be wider than the limits of the ApplicationPrimitiveDataType ((see subsection "Data Types for Single Values" in the AUTOSAR SW-C Template [2]).

[SWS_Rte_08403] ⌈ For AUTOSAR data types which have an invalidValue specified, the Application Types header file shall contain the definition

```
#define InvalidValue_<prefix><DataType> <invalidValue><suffix>
```

where

<prefix> is the optional literalPrefix attribute defined by the Included-DataTypeSet referring the AutosarDataType

<DataType> is the short name of the data type.

<invalidValue> is the value defined as invalidValue for the data type.

<suffix> shall be "U" for unsigned data types and empty for signed data types. ⌋

[SWS_Rte_08416] ⌈ The Application Types Header File shall include the definitions of all invalidValue constants used by this software component for each combination of different literalPrefix and ApplicationPrimitiveDataType when the same ImplementationDataType or ApplicationPrimitiveDataType is referenced via different IncludedDataTypeSets. ⌋*(SRS_Rte_00167)*

5.5.6 Data Types with bitfield conversions

AutosarDataTypes associated with a CompuMethod of category BITFIELD_TEXTTABLE support the concatenation of a value set inside a single scalar variable. Thereby single bits may get a individual (boolean) meaning or a set of bits is used carry an enumeration. Please note that those data types are not mapped to C bit fields rather than to scalars (e.g. uint8). Thereby the RTE Generator provides a set of definitions for the "Bit Mask", "Bit Start Position" and the "Number of Bits" in order to support the usage of the AUTOSAR Bit Handling Routines [30] for those kind of data types. For some operations on a set of bits (the set may contain only 1 bit) those library requires a single contiguous bit field which means that all bits set to 1 in the in the CompuScale.mask attribute value are adjoining, e.g. 0b00010000 or 0b00111100.

[SWS_Rte_07410] ⌈ For each unique CompuScale.shortLabel / CompuScale.mask value pair for a CompuScale which is located in the compuInternalToPhys container of a CompuMethod referenced by an ImplementationDataType or ApplicationPrimitiveDataType according [SWS_Rte_03809] with category BITFIELD_TEXTTABLE the *Application Types Header File* shall contain a definition for the bit field mask

```
#ifndef <prefix><BflMaskLabel>_BflMask
#define <prefix><BflMaskLabel>_BflMask <mask><suffix>
#endif /* <prefix><BflMaskLabel>_BflMask */
```

where
<BflMaskLabel> is the value of the attribute CompuScale.shortLabel
<mask> is the value of the attribute mask
<prefix> is the optional literalPrefix attribute defined by the IncludedDataTypeSet referring the AutosarDataType using the CompuMethod.
<suffix> shall be "U" for unsigned data types and empty for signed data types.
⌋*(SRS_Rte_00167)*

[SWS_Rte_07411] ⌈ For each unique CompuScale.shortLabel / CompuScale.mask value pair for a CompuScale with a single contiguous bit field which is located in the compuInternalToPhys container of a CompuMethod referenced by an ImplementationDataType or ApplicationPrimitiveDataType according [SWS_Rte_03809] with category BITFIELD_TEXTTABLE the *Application Types Header File* shall contain a definition for the bit start position

```
#ifndef <prefix><BflStartPnLabel>_BflPn
#define <prefix><BflStartPnLabel>_BfltPn <BflStartPnNumber><suffix>
#endif /* <prefix><BflStartPnLabel>_BfltPn */
```

where
<BitStartPnLabel> is the value of the attribute CompuScale.shortLabel
<BflStartPnNumber> is the number of the first bit in the attribute value CompuScale.mask which is set to 1. Thereby the bit counting starts from 0 (LSB) to n (MSB).
<prefix> is the optional literalPrefix attribute defined by the IncludedDataTypeSet referring the AutosarDataType using the CompuMethod.
<suffix> shall be "U" for unsigned data types and empty for signed data types.
⌋*(SRS_Rte_00167)*

[SWS_Rte_07412] ⌈ For each unique CompuScale.shortLabel / CompuScale.mask value pair for a CompuScale with a single contiguous bit field which is located in the compuInternalToPhys container of a CompuMethod referenced by an ImplementationDataType or ApplicationPrimitiveDataType according [SWS_Rte_03809] with category BITFIELD_TEXTTABLE the *Application Types Header File* shall contain a definition for the bit field length

```
#ifndef <prefix><BflLengthLabel>_BflLn
#define <prefix><BflLengthLabel>_BflLn <BflLength><suffix>
#endif /* <prefix><BflLengthLabel>_BflLn */
```

where
<BflLengthLabel> is the value of the attribute shortLabel <BflLength> is the number of contiguous bits set to 1 in the attribute value CompuScale.mask. <prefix> is the optional literalPrefix attribute defined by the IncludedDataTypeSet referring the AutosarDataType using the CompuMethod.
<suffix> shall be "U" for unsigned data types and empty for signed data types. ⌋*(SRS_Rte_00167)*

Please note the example in section F.3.

5.6 API Reference

The functions described in this section are organized by the RTE API mapping name used by C and C++ AUTOSAR software-components to access the API. The API mapping hides from the AUTOSAR software-component programmer any need to be aware of the steps taken by the RTE generator to ensure that the generated API functions have unique names.

The instance handle as the first parameter of the API calls is marked as an optional parameter in this section. If an AUTOSAR software-component supports multiple instantiation, the instance handle shall be passed [SWS_Rte_01013].

Note that [SWS_Rte_03806] requires that the instance handle parameter does not exist if the AUTOSAR software-component does not support multiple instantiation.

5.6.1 Rte_Ports

Purpose: Provide an array of the ports of a given interface type and a given provide / require usage that can be accessed by the indirect API.

Signature: **[SWS_Rte_02619]** ⌈

```
Rte_PortHandle_<i>_<R/P>
Rte_[Byps_]Ports_<i>_<R/P>([IN Rte_Instance])
```

Where here <i> is the port interface name and ‘P’ or ‘R’ are literals to indicate provide or require ports respectively. [Byps_] is an optional infix used when component wrapper method for bypass support is enabled for the related software component type (See chapter 4.9.2). ⌋*(SRS_Rte_00051)*

Existence: **[SWS_Rte_02613]** ⌈ An Rte_Ports API shall be created for each interface type and usage by a port in at least one PreCompileTime variant when the indirectAPI attribute of that port is set to true. ⌋*(SRS_Rte_00051)*

Description: The Rte_Ports API provides access to an array of ports for the port oriented API.

[SWS_Rte_03602] ⌈ `Rte_Ports` API shall return an array of ports which contains only those ports for which the indirect API was generated or it shall return an `NULL_PTR` if the port interface is not used in the current `PreCompileTime` variant. ⌋*(SRS_Rte_00051)*

Return Value: Array of port data structures of the corresponding interface type and usage.

Notes: None.

5.6.2 Rte_NPorts

Purpose: Provide the number of ports of a given interface type and provide / require usage that can be accessed through the indirect API.

Signature: **[SWS_Rte_02614]** ⌈

```
uint8
Rte_[Byps_]NPorts_<i>_<R/P>([IN Rte_Instance])
```

Where here `<i>` is the port interface name and 'P' or 'R' are literals to indicate provide or require ports respectively. `[Byps_]` is an optional infix used when component wrapper method for bypass support is enabled for the related software component type (See chapter 4.9.2). ⌋*(SRS_Rte_00051)*

Existence: **[SWS_Rte_02615]** ⌈ An `Rte_NPorts` API shall be created for each interface type and usage by a port in at least one `PreCompileTime` variant when the `indirectAPI` attribute of the port is set to true. ⌋*(SRS_Rte_00051)*

Description: The `Rte_NPorts` API supports access to an array of ports for the port oriented API.

[SWS_Rte_03603] ⌈ The `Rte_NPorts` shall return the number of ports of a given interface and provide / require usage for which the indirect API was generated or 0 if the port interface is not used in the current `PreCompileTime` variant. ⌋*(SRS_Rte_00051)*

Return Value: Number of port data structures of the corresponding interface type and usage.

Notes: None.

5.6.3 Rte_Port

Purpose: Provide access to the port data structure for a single port of a particular software component instance. This allows a software component to extract a sub-group of ports characterized by the same interface in order to iterate over this sub-group.

Signature: **[SWS_Rte_01354]** ⌈

```
Rte_PortHandle_<i>_<R/P>
Rte_[Byps_]Port_<p>([IN Rte_Instance])
```

where `<i>` is the port interface name and `<p>` is the name of the port. `[Byps_]` is an optional infix used when component wrapper method for bypass support is enabled for the related software component type (See chapter 4.9.2). ⌋*(SRS_Rte_00051)*

Existence: **[SWS_Rte_01355]** ⌈ An `Rte_Port` API shall be created for each port of an AUTOSAR SW-C, for which the `indirectAPI` attribute is set to true. ⌋*(SRS_Rte_00051)*

Description: The `Rte_Port` API provides a pointer to a single port data structure, in order to support the indirect API.

Return Value: Pointer to port data structure for the appropriate port.

Notes: None.

5.6.4 Rte_Write

Purpose: Initiate an "explicit" sender-receiver transmission of data elements with "data" semantic (`swImplPolicy` different from `queued`).

Signature: **[SWS_Rte_01071]** ⌈

```
Std_ReturnType
Rte_[Byps_]Write_<p>_<o>([IN Rte_Instance <instance>],
                  IN <data>)
```

Where `<p>` is the port name and `<o>` the `VariableDataPrototype` within the sender-receiver interface categorizing the port. `[Byps_]` is an optional infix used when component wrapper method for bypass support is enabled for the related software component type (See chapter 4.9.2). ⌋*(SRS_BSW_00310, SRS_Rte_00098, SRS_Rte_00028, SRS_Rte_00131)*

Existence: **[SWS_Rte_01280]** ⌈ The presence of a `VariableAccess` in the `dataSendPoint` role for a provided `VariableDataPrototype` with `data semantic`s shall result in the generation of an `Rte_Write` API for the provided `VariableDataPrototype`. ⌋*(SRS_Rte_00051)*

[constr_9015] `Rte_Write` API may only be used by the runnable that describe its usage ⌈ The `Rte_Write` API may only be used by the runnable that contains the corresponding `VariableAccess` in the `dataSendPoint` role ⌋

Description: The Rte_Write API call initiates a sender-receiver communication where the transmission occurs at the point the API call is made (cf. explicit transmission).

The Rte_Write API call includes the IN parameter <data> to pass the data element to write.

The IN parameter <data> is passed by value or reference according to the ImplementationDataType as described in the section 5.2.6.5.

If the IN parameter <data> is passed by reference, the pointer must remain valid until the API call returns.

The RTE generator shall take into account the kind of connected require port which might not be just a variable but also a NV data. The table 4.6 gives an overview of compatibility rules.

Return Value: The return value is used to indicate errors detected by the RTE during execution of the Rte_Write.

- **[SWS_Rte_07820]** ⌈ RTE_E_OK – data passed to communication service successfully. ⌋*(SRS_Rte_00094)*
- **[SWS_Rte_07822]** ⌈ RTE_E_COM_STOPPED – the RTE could not perform the operation because the COM service is currently not available (inter ECU communication only). RTE shall return RTE_E_COM_STOPPED when the corresponding COM service returns COM_SERVICE_NOT_AVAILABLE. ⌋*(SRS_Rte_00094)*
- **[SWS_Rte_02756]** ⌈ RTE_E_SEG_FAULT – a segmentation violation is detected in the handed over parameters to the RTE API as required in [SWS_Rte_02752] and [SWS_Rte_02753]. No transmission is executed. ⌋*(SRS_Rte_00210)*

Notes: The Rte_Write call is used to transmit "data" (swImplPolicy not queued).

[SWS_Rte_07824] ⌈ In case of inter ECU communication, the Rte_Write shall cause an immediate transmission request. ⌋*(SRS_Rte_00028, SRS_Rte_00131)*

Note that depending on the configuration a transmission request may not result in an actual transmission, for example transmission may be rate limited (time-based filtering) and thus dependent on other factors than API calls.

[SWS_Rte_07826] ⌈ In case of inter ECU communication, the Rte_Write API shall return when the signal has been passed to the communication service for transmission. ⌋*(SRS_Rte_00028, SRS_Rte_00131)*

Depending on the communication server the transmission may or may not have been acknowledged by the receiver at the point the API call returns.

[SWS_Rte_02635] ⌈ In case of intra ECU communication, the Rte_Write API call shall return after copying the data to RTE local memory or using IOC buffers. ⌋*(SRS_Rte_00028, SRS_Rte_00131)*

[SWS_Rte_01080] ⌈ If the transmission acknowledgement is enabled, the RTE shall notify component when the transmission is acknowledged or a transmission error occurs. ⌋*(SRS_Rte_00122)*

[SWS_Rte_01082] ⌈ If a provide port typed by a sender-receiver interface has multiple require ports connected (i.e. it has multiple receivers), then the RTE shall ensure that writes to all receivers are independent. ⌋*(SRS_Rte_00028)*

Requirement [SWS_Rte_01082] ensures that an error detected by the RTE when writing to one receiver, e.g. communication is stopped, does not prevent the transmission of this message to other components.

[SWS_Rte_08413] ⌈ If a provide port typed by a sender-receiver interface has multiple require ports connected (i.e. it has multiple receivers), then the RTE shall return RTE_E_OK only if no error at all occurred. ⌋*(SRS_Rte_00028)*

[SWS_Rte_08414] ⌈ In case of multiple faults during a call of Rte_Write the resulting return value shall be derived according to the following priority rules (highest priority first):

1. RTE_E_SEG_FAULT
2. RTE_E_COM_STOPPED

⌋*(SRS_Rte_00028)*

5.6.5 Rte_Send

Purpose: Initiate an “explicit” sender-receiver transmission of data elements with “event” semantic (swImplPolicy equal to queued).

Signature: **[SWS_Rte_01072]** ⌈

```
Std_ReturnType
Rte_[Byps_]Send_<p>_<o>([IN Rte_Instance <instance>],
                        IN <data>,
                        [IN uint16 <length>])
```

Where <p> is the port name and <o> the VariableDataPrototype within the sender-receiver interface categorizing the port.

[Byps_] is an optional infix used when component wrapper method for bypass support is enabled for the related software component type (See chapter 4.9.2). ⌋(*SRS_BSW_00310, SRS_Rte_00141, SRS_Rte_00028, SRS_Rte_00131*)

Existence: **[SWS_Rte_01281]** ⌈ The presence of a VariableAccess in the dataSendPoint role for a provided VariableDataPrototype with event semantics shall result in the generation of an Rte_Send API for the provided VariableDataPrototype. ⌋(*SRS_Rte_00051*)

[SWS_Rte_07813] ⌈ The optional IN parameter <length> of the Rte_Send API shall be generated if the VariableDataPrototype is of type dynamic. ⌋(*SRS_Rte_00190*)

[constr_9016] Rte_Send API may only be used by the runnable that describes its usage ⌈ The Rte_Send API may only be used by the runnable that contains the corresponding VariableAccess in the dataSendPoint role ⌋

Description: The Rte_Send API call initiates a sender-receiver communication where the transmission occurs at the point the API call is made (cf. explicit transmission).

The Rte_Send API call includes the IN parameter <data> to pass the data element to send.

The IN parameter <data> is passed by value or reference according to the ImplementationDataType as described in the section 5.2.6.5.

If the IN parameter <data> is passed by reference, the pointer must remain valid until the API call returns.

If the VariableDataPrototype is of type dynamic, the Rte_Send API call includes the IN parameter <length> to pass the number of elements in the data element to send.

The RTE generator has to take into account the kind of connected require port which might not be just a variable but also a NV data. The table 4.6 gives an overview of compatibility rules.

Return Value: The return value is used to indicate errors detected by the RTE during execution of the Rte_Send.

- **[SWS_Rte_07821]** ⌈ RTE_E_OK – data passed to communication service successfully. ⌋(*SRS_Rte_00094*)
- **[SWS_Rte_07823]** ⌈ RTE_E_COM_STOPPED – the RTE could not perform the operation because the COM service is currently not available (inter ECU communication only). RTE shall return

RTE_E_COM_STOPPED when the corresponding COM service returns COM_SERVICE_NOT_AVAILABLE. ⌋(SRS_Rte_00094)

- **[SWS_Rte_02634]** ⌈ RTE_E_LIMIT – an 'event' has been discarded due to a full queue by one of the ECU local receivers (intra ECU communication only). ⌋(SRS_Rte_00143)
- **[SWS_Rte_02754]** ⌈ RTE_E_SEG_FAULT – a segmentation violation is detected in the handed over parameters to the RTE API as required in [SWS_Rte_02752] and [SWS_Rte_02753]. No transmission is executed. ⌋(SRS_Rte_00210)

Notes: The Rte_Send call is used to transmit "events" (swImplPolicy = queued).

[SWS_Rte_07825] ⌈ In case of inter ECU communication, the Rte_Send shall cause an immediate transmission request. ⌋(SRS_Rte_00028, SRS_Rte_00131)

Note that depending on the configuration a transmission request may not result in an actual transmission, for example transmission may be rate limited (time-based filtering) and thus dependent on other factors than API calls.

[SWS_Rte_07827] ⌈ In case of inter ECU communication, the Rte_Send API shall return when the signal has been passed to the communication service for transmission. ⌋(SRS_Rte_00028, SRS_Rte_00131)

Depending on the communication server the transmission may or may not have been acknowledged by the receiver at the point the API call returns.

[SWS_Rte_02633] ⌈ In case of intra ECU communication, the Rte_Send API call shall return after attempting to enqueue the data in the IOC or RTE internal queues. ⌋(SRS_Rte_00028, SRS_Rte_00131)

If the transmission acknowledgement is enabled, the RTE has to notify component when the transmission is acknowledged or a transmission error occurs. [SWS_Rte_01080]

If a provide port typed by a sender-receiver interface has multiple require ports connected (i.e. it has multiple receivers), then the RTE shall ensure that writes to all receivers are independent. [SWS_Rte_01082]

Requirement [SWS_Rte_01082] ensures that an error detected by the RTE when writing to one receiver, e.g. an overflow in one component's queue, does not prevent the transmission of this message to other components.

If a provide port typed by a sender-receiver interface has multiple require ports connected (i.e. it has multiple receivers), then the RTE shall return RTE_E_OK only if no error at all occurred. [SWS_Rte_08413]

[SWS_Rte_08415] ⌈ In case of multiple faults during a call of Rte_Send the resulting return value shall be derived according to the following priority rules (highest priority first):

1. RTE_E_SEG_FAULT
2. RTE_E_LIMIT (only in case of Intra-ECU communication)
3. RTE_E_COM_STOPPED

⌋*(SRS_Rte_00028)*

5.6.6 Rte_Switch

Purpose: Initiate a mode switch. The Rte_Switch API call is used for 'explicit' sending of a mode switch notification.

Signature: **[SWS_Rte_02631]** ⌈

```
Std_ReturnType
Rte_[Byps_]Switch_<p>_<o>([IN Rte_Instance <instance>],
                          IN <mode>)
```

Where <p> is the port name and <o> the *ModeDeclarationGroupPrototype* within the ModeSwitchInterface categorizing the port. [Byps_] is an optional infix used when component wrapper method for bypass support is enabled for the related software component type (See chapter 4.9.2). ⌋*(SRS_BSW_00310, SRS_Rte_00143, SRS_Rte_00028, SRS_Rte_00131)*

Existence: **[SWS_Rte_02632]** ⌈The existence of a *ModeSwitchPoint* shall result in the generation of a Rte_Switch API. ⌋*(SRS_Rte_00051)*

[constr_9017] Rte_Switch API may only be used by the runnable that describes its usage ⌈ The Rte_Switch API may only be used by the runnable that contains the corresponding *ModeSwitchPoint* ⌋

Description: The Rte_Switch triggers a mode switch for all connected require ModeDeclarationGroupPrototypes.

The Rte_Switch API call includes exactly one IN parameter for the next mode <mode>. The IN parameter <mode> is passed by value according to the ImplementationDataType on which the *ModeDeclarationGroup* is mapped. The type name shall be equal to the shortName of the ImplementationDataType.

Return Value: The return value is used to indicate errors detected by the RTE during execution of the Rte_Switch call.

- **[SWS_Rte_02674]** ⌈ RTE_E_OK – data passed to service successfully. ⌋*(SRS_Rte_00094)*
- **[SWS_Rte_02675]** ⌈ RTE_E_LIMIT – a mode switch has been discarded by the receiving partition due to a full queue. ⌋*(SRS_Rte_00143)*

Notes: Rte_Switch is restricted to ECU local communication.

If a mode instance is currently involved in a transition then the Rte_Switch API will attempt to queue the request and return [SWS_Rte_02667]. However if no transition is in progress for the mode instance, the mode disablings and the activations of OnEntry, OnTransition, and OnExit ExecutableEntities for this mode instance are executed before the Rte_Switch API returns [SWS_Rte_02665].

Note that the mode switch might be discarded when the queue is full and a mode transition is in progress, see [SWS_Rte_02675].

5.6.7 Rte_Invalidate

Purpose: Invalidate a data element for an "explicit" sender-receiver transmission.

Signature: **[SWS_Rte_01206]** ⌈

```
Std_ReturnType
Rte_[Byps_]Invalidate_<p>_<o>([IN Rte_Instance <instance>])
```

Where <p> is the port name and <o> the VariableDataPrototype within the sender-receiver interface categorizing the port. [Byps_] is an optional infix used when component wrapper method for bypass support is enabled for the related software component type (See chapter 4.9.2). ⌋*(SRS_BSW_00310, SRS_Rte_00078)*

Existence: **[SWS_Rte_01282]** ⌈ An Rte_Invalidate API shall be created for any VariableAccess in the dataSendPoint role that references a provided VariableDataPrototype which associated InvalidationPolicy is set to keep or replace. ⌋*(SRS_Rte_00051, SRS_Rte_00078)*

[constr_9018] Rte_Invalidate API may only be used by the runnable that describe its usage ⌈ The Rte_Invalidate API may only be used by the runnable that contains the corresponding VariableAccess in the dataSendPoint role ⌋

Description: The Rte_Invalidate API takes no parameters other than the instance handle – the return value is used to indicate the success, or otherwise, of the API call to the caller.

[SWS_Rte_01231] ⌈ When COM is used for communication and the VariableDataPrototype is primitive the COM API function Com_InvalidateSignal shall be called for invalidation. ⌋*(SRS_Rte_00019, SRS_Rte_00078)*

[SWS_Rte_05063] ⌈ When COM is used for communication and the VariableDataPrototype is composite the COM API function Com_InvalidateSignalGroup shall be called for invalidation. ⌋*(SRS_Rte_00019, SRS_Rte_00078)*

The behavior required when COM is not used for communication is described in Section 4.3.1.8.

Return Value: The return value is used to indicate the "OK" status or errors detected by the RTE during execution of the Rte_Invalidate call.

- **[SWS_Rte_01207]** ⌈ RTE_E_OK – No error occurred. ⌋*(SRS_Rte_00094)*
- **[SWS_Rte_01339]** ⌈ RTE_E_COM_STOPPED – the RTE could not perform the operation because the COM service is currently not available (inter ECU communication only). RTE shall return RTE_E_COM_STOPPED when the corresponding COM service returns COM_SERVICE_NOT_AVAILABLE. ⌋*(SRS_Rte_00094)*

Notes: The API name includes an identifier <p>_<o> that is formed from the port and operation item names. See Section 5.2.6.4 for details on the naming convention.

The communication service configuration determines whether the signal receiver(s) receive an "invalid signal" notification or whether the invalidated signal is silently replaced by the signal's initial value.

5.6.8 Rte_Feedback

Purpose: Provide access to acknowledgement notifications for explicit sender-receiver communication and to pass error notification to senders.

Signature: **[SWS_Rte_01083]** ⌈

```
Std_ReturnType
Rte_[Byps_]Feedback_<p>_<o>([IN Rte_Instance <instance>])
```

Where <p> is the port name and <o> the VariableDataPrototype within the sender-receiver interface categorizing the port. [Byps_] is an optional infix used when component wrapper method for bypass support is enabled for the related software component type (See chapter 4.9.2). ⌋(SRS_BSW_00310, SRS_Rte_00122)

Existence: **[SWS_Rte_01283]** ⌈ Acknowledgement is enabled for a provided VariableDataPrototype by the existence of a TransmissionAcknowledgementRequest in the SenderComSpec. ⌋(SRS_Rte_00051, SRS_Rte_00122)

[SWS_Rte_01284] ⌈ A blocking Rte_Feedback API shall be generated for a provided VariableDataPrototype if acknowledgement is enabled and a WaitPoint references a DataSendCompletedEvent that in turn references the VariableAccess which in turn references the VariableDataPrototype. ⌋(SRS_Rte_00051, SRS_Rte_00122)

[SWS_Rte_07850] ⌈ A blocking Rte_Feedback API shall block when a transmission of the related VariableDataPrototype is ongoing. ⌋(SRS_Rte_00051, SRS_Rte_00122)

[SWS_Rte_07851] ⌈ A blocking Rte_Feedback API shall return:

- if the sender port is not connected or
- if the calling runnable runs in an exclusive area or
- if no transmission of the related VariableDataPrototype is ongoing or
- when the wait point timeout occurs or
- when the related DataSendCompletedEvent is triggered.

⌋(SRS_Rte_00051, SRS_Rte_00122)

[SWS_Rte_01285] ⌈ A non-blocking Rte_Feedback API shall be generated for a provided VariableDataPrototype if acknowledgement is enabled and a VariableAccess in the dataSendPoint role references the VariableDataPrototype but no WaitPoint references the DataSendCompletedEvent that references the VariableAccess which in turn references the VariableDataPrototype. ⌋(SRS_Rte_00051, SRS_Rte_00122)

Please note that a non-blocking Rte_Feedback API does not require the existence of a DataSendCompletedEvent. If the DataSendCompletedEvent exists it can be used to trigger the execution of a RunnableEntity in which the non-blocking Rte_Feedback API function may be called.

[SWS_Rte_01286] ⌈ If acknowledgement is enabled for a provided VariableDataPrototype and a DataSendCompletedEvent references a runnable entity as well as the VariableAccess which in turn references the VariableDataPrototype, the runnable entity shall be activated when the transmission acknowledgement occurs or when a timeout was detected by the RTE. [SWS_Rte_01137]. ⌋(*SRS_Rte_00051, SRS_Rte_00122*)

Requirement [SWS_Rte_01286] merely affects when the runnable is activated – an API call should still be created, according to requirement [SWS_Rte_01285] to actually read the data.

[SWS_Rte_01287] ⌈ A `DataSendCompletedEvent` that references a `RunnableEntity` and is referenced by a `WaitPoint` shall be an invalid configuration which is rejected by the RTE generator. ⌋(*SRS_Rte_00051, SRS_Rte_00122, SRS_Rte_00018*)

[constr_9019] `Rte_Feedback` API may only be used by the runnable that describe its usage ⌈ A blocking `Rte_Feedback` API may only be used by the runnable that contains the corresponding `WaitPoint` ⌋

[SWS_Rte_07634] ⌈ A call to `Rte_Feedback` shall not change the status returned by `Rte_Feedback`. ⌋(*SRS_Rte_00122*)

The `Rte_Feedback` API return value is only changed when a new transmission is requested (`Rte_Send` or `Rte_Write`) or when the notification from COM is received.

[SWS_Rte_07635] ⌈ After a `Rte_Send` or `Rte_Write` transmission request, only the first notification from COM shall be taken into account for a given Signal or SignalGroup. ⌋(*SRS_Rte_00122*)

[SWS_Rte_07635] is needed in case of cyclic transmission which could result in multiple transmissions with different status.

Description: The `Rte_Feedback` API takes no parameters other than the instance handle – the return value is used to indicate the acknowledgement status to the caller.

The `Rte_Feedback` API applies only to explicit sender-receiver communication.

Return Value: The return value is used to indicate the status of the transmission and errors detected by the RTE.

- **[SWS_Rte_01084]** ⌈ `RTE_E_NO_DATA` – No acknowledgments or error notifications were received from COM when the `Rte_Feedback` API was called (non-blocking call) or when the `WaitPoint` timeout expired (blocking call). ⌋(*SRS_Rte_00094, SRS_Rte_00122*)
- `RTE_E_COM_STOPPED` – returned in one of these cases:
  - **[SWS_Rte_07636]** ⌈ (Inter-ECU communication only) The last transmission was rejected (when the

Rte_Send or Rte_Write API was called), with an RTE_E_COM_STOPPED return code. ⌋(SRS_Rte_00094, SRS_Rte_00122)

  – **[SWS_Rte_03774]** ⌈ (Inter-ECU communication only) An error notification from COM was received before any timeout notification. ⌋(SRS_Rte_00094, SRS_Rte_00122)

- **[SWS_Rte_07637]** ⌈ RTE_E_TIMEOUT – (Inter-ECU and Inter-Partition only) A timeout notification was received from COM or IOC before any error notification. ⌋(SRS_Rte_00094, SRS_Rte_00122)
- **[SWS_Rte_01086]** ⌈ RTE_E_TRANSMIT_ACK – In case of inter-ECU communication, a transmission acknowledgment was received from COM; or in case of intra-ECU communication, even if a queue overflow occurred. ⌋(SRS_Rte_00094, SRS_Rte_00122)
- RTE_E_UNCONNECTED – Indicates that the sender port is not connected [SWS_Rte_01344].
- **[SWS_Rte_02740]** ⌈ RTE_E_IN_EXCLUSIVE_AREA – Used only for the blocking API. RTE_E_IN_EXCLUSIVE_AREA indicates that the runnable can not enter wait, as one of the ExecutableEntitys in the call stack of this task is currently in an exclusive area, see [SWS_Rte_02739]. - In a properly configured system, this error should not occur. The check can be disabled according to [SWS_Rte_08318]. ⌋(SRS_Rte_00092, SRS_Rte_00046, SRS_Rte_00032)
- **[SWS_Rte_08318]** ⌈ If RteInExclusiveAreaCheckEnabled is set to *false* the RTE generator shall omit the check and return of [SWS_Rte_02740]. ⌋(SRS_Rte_00092, SRS_Rte_00046, SRS_Rte_00032)

The RTE_E_NO_DATA, RTE_E_TRANSMIT_ACK and RTE_E_UNCONNECTED return values are not considered to be an error but rather indicates correct operation of the API call.

[SWS_Rte_07652] ⌈ The initial return value of the Rte_Feedback API, before any attempt to write some data shall be RTE_E_TRANSMIT_ACK. ⌋(SRS_Rte_00094, SRS_Rte_00122, SRS_Rte_00128, SRS_Rte_00185)

[SWS_Rte_08075] ⌈ In case of multiple faults during a call of Rte_Feedback the resulting return value shall be derived according to the following priority rules (highest priority first): (1) RTE_E_UNCONNECTED, (2) RTE_E_IN_EXCLUSIVE_AREA, (3)

RTE_E_TIMEOUT, (4) RTE_E_COM_STOPPED, (5) RTE_E_NO_DATA, (6) RTE_E_TRANSMIT_ACK. ⌋*(SRS_Rte_00122)*

Notes: If multiple transmissions on the same port/element are outstanding it is not possible to determine which is acknowledged first. If this is important, transmissions should be call serialized with the next occurring only when the previous transmission has been acknowledged or has timed out.

A transmission acknowledgment (or error and timeout) notification is not always provided by COM (the bus or PDU Router may not support transmission acknowledgment for this PDU, or COM may not be configured to perform transmission deadline monitoring).

In case of a blocking Rte_Feedback the value of the WaitPoint timeout depends on the timeout defined at the COM level.

5.6.9 Rte_SwitchAck

Purpose: Provide access to mode switch completed acknowledgements and error notifications to mode managers.

Signature: **[SWS_Rte_02725]** ⌈

```
Std_ReturnType
Rte_[Byps_]SwitchAck_<p>_<o>([IN Rte_Instance <instance>])
```

Where <p> is the port name and <o> the ModeDeclarationGroupPrototype within the ModeSwitchInterface categorizing the port. [Byps_] is an optional infix used when component wrapper method for bypass support is enabled for the related software component type (See chapter 4.9.2). ⌋*(SRS_BSW_00310, SRS_Rte_00122)*

Existence: **[SWS_Rte_02676]** ⌈ Acknowledgement is enabled for a provided ModeDeclarationGroupPrototype by the existence of a ModeSwitchedAckRequest in the ModeSwitchSenderComSpec. ⌋*(SRS_Rte_00051, SRS_Rte_00122)*

[SWS_Rte_02677] ⌈ A blocking Rte_SwitchAck API shall be generated for a provided ModeDeclarationGroupPrototype if acknowledgement is enabled and a WaitPoint references a ModeSwitchedAckEvent that in turn references the ModeDeclarationGroupPrototype. ⌋*(SRS_Rte_00051, SRS_Rte_00122)*

[SWS_Rte_07846] ⌈ A blocking Rte_SwitchAck API shall block when a mode switch in the related mode machine instance is ongoing. ⌋*(SRS_Rte_00122, SRS_Rte_00092)*

[SWS_Rte_07847] ⌈ A blocking Rte_SwitchAck API shall return:

- if the mode machine instance behaves as unconnected or
- if the calling runnable runs in an exclusive area or
- if no mode switch in the related mode machine instance is ongoing or
- when the wait point timeout occurs or
- when the related ModeSwitchedAckEvent is triggered.

⌋(*SRS_Rte_00122, SRS_Rte_00092, SRS_Rte_00139*)

[SWS_Rte_02678] ⌈ A non-blocking Rte_SwitchAck API shall be generated for a provided ModeDeclarationGroupPrototype if acknowledgement is enabled but no WaitPoint references a ModeSwitchedAckEvent that references the ModeDeclarationGroupPrototype.

Please note that a non-blocking API does not require the existence of a ModeSwitchedAckEvent. If the ModeSwitchedAckEvent exists it can be used to trigger the execution of a RunnableEntity in which the non-blocking API function may be called. ⌋(*SRS_Rte_00051, SRS_Rte_00122*)

[constr_9020] The blocking Rte_SwitchAck API may only be used by the runnable that describes its usage. ⌈ A blocking Rte_SwitchAck API must only be used by the runnable that contains the corresponding WaitPoint ⌋

Description: The Rte_SwitchAck API takes no parameters other than the instance handle – the return value is used to indicate the acknowledgement status to the caller.

Return Value: The return value is used to indicate the status of a mode switch and errors detected by the RTE.

- **[SWS_Rte_02727]** ⌈ RTE_E_NO_DATA – (non-blocking read) The mode switch is still in progress. ⌋(*SRS_Rte_00094, SRS_Rte_00122*)
- **[SWS_Rte_02728]** ⌈ RTE_E_TIMEOUT – The configured timeout exceeds before the mode transition was completed. ⌋(*SRS_Rte_00094, SRS_Rte_00210*)
- **[SWS_Rte_03853]** ⌈ RTE_E_TIMEOUT – The partition of the mode users is stopped or restarting or has been restarted while the mode switch was requested. ⌋(*SRS_Rte_00094, SRS_Rte_00210*)

- **[SWS_Rte_02729]** ⌈ RTE_E_TRANSMIT_ACK – The mode switch has been completed (see [SWS_Rte_02587]). ⌋*(SRS_Rte_00094, SRS_Rte_00122)*
- **[SWS_Rte_07659]** ⌈ RTE_E_UNCONNECTED – Indicates that the mode provider port is not connected. ⌋*(SRS_Rte_00094, SRS_Rte_00122, SRS_Rte_00139)*
- **[SWS_Rte_02741]** ⌈ RTE_E_IN_EXCLUSIVE_AREA – Used only for the blocking API. RTE_E_IN_EXCLUSIVE_AREA indicates that the runnable can not enter wait, as one of the ExecutableEntitys in the call stack of this task is currently in an exclusive area, see [SWS_Rte_02739]. - In a properly configured system, this error should not occur. The check can be disabled according to [SWS_Rte_08319]. ⌋*(SRS_Rte_00092, SRS_Rte_00046, SRS_Rte_00032)*
- **[SWS_Rte_08319]** ⌈ If RteInExclusiveAreaCheckEnabled is set to *false* the RTE generator shall omit the check and return of [SWS_Rte_02741]. ⌋*(SRS_Rte_00092, SRS_Rte_00046, SRS_Rte_00032)*

The RTE_E_TRANSMIT_ACK return value is not considered to be an error but rather indicates correct operation of the API call.

When RTE_E_NO_DATA occurs, a component is free to re-invoke Rte_SwitchAck and thus repeat the attempt to read the status of the mode switch.

[SWS_Rte_07848] ⌈ The initial return value of the Rte_SwitchAck API before any attempt to switch a mode shall be RTE_E_TRANSMIT_ACK. ⌋*(SRS_Rte_00094, SRS_Rte_00122)*

[SWS_Rte_07849] ⌈ In case of multiple faults during a call of Rte_SwitchAck the resulting return value shall be derived according to the following priority rules (highest priority first): (1) RTE_E_UNCONNECTED, (2) RTE_E_IN_EXCLUSIVE_AREA, (3) RTE_E_TIMEOUT, (4) RTE_E_NO_DATA, (5) RTE_E_TRANSMIT_ACK. ⌋*(SRS_Rte_00094, SRS_Rte_00122)*

Notes: If multiple mode switches of the same mode machine instance are outstanding, it is not possible to determine which is acknowledged first. If this is important, switches should be serialized with the next switch occurring only when the previous switch has been acknowledged. The queue length should be 1.

5.6.10 Rte_Read

Purpose: Performs an "explicit" read on a sender-receiver communication data element with "data" semantics (swImplPolicy != queued). By compatibility, the port may also have a ParameterInterface or a NvDataInterface. The Rte_Read API is used for explicit read by argument.

Signature: **[SWS_Rte_01091]** ⌈

```
Std_ReturnType
Rte_[Byps_]Read_<p>_<o>([IN Rte_Instance <instance>],
                        OUT <data>))
```

Where <p> is the port name and <o> the VariableDataPrototype within the sender-receiver interface categorizing the port. [Byps_] is an optional infix used when component wrapper method for bypass support is enabled for the related software component type (See chapter 4.9.2). ⌋*(SRS_BSW_00310, SRS_Rte_00141, SRS_Rte_00028, SRS_Rte_00131)*

Existence: **[SWS_Rte_01289]** ⌈ A non-blocking Rte_Read API shall be generated if a VariableAccess in the dataReceivePointByArgument role references a required VariableDataPrototype with 'data' semantics. ⌋*(SRS_Rte_00051)*

[SWS_Rte_07396] ⌈ The RTE shall ensure that direct explicit read accesses will not deliver undefined data item values. In case there may be an explicit read access before the first data reception an initial value shall be provided as the result of this explicit read access. ⌋*(SRS_Rte_00051, SRS_Rte_00183)*

A WaitPoint cannot reference a DataReceivedEvent that in turn references a required VariableDataPrototype with 'data' semantics shall be considered an invalid configuration (see [SWS_Rte_03018]). Hence there are no blocking Rte_Read API.

[constr_9021] Rte_Read API may only be used by the runnable that describe its usage ⌈ The Rte_Read API may only be used by the runnable that contains the corresponding VariableAccess in the dataReceivePointByArgument role ⌋

[SWS_Rte_01313] ⌈ A DataReceivedEvent that references a runnable entity and is referenced by a WaitPoint shall be an invalid configuration. ⌋*(SRS_Rte_00051, SRS_Rte_00018)*

The RTE generator shall take into account the kind of provide port which might not be just a variable but also a Parameter (fixed, const or standard), a standard sender (i.e. a variable) or a NV data. The table 4.6 gives an overview of compatibility rules.

Description: The Rte_Read API call includes the OUT parameter <data> to pass back the received data.

The pointer to the OUT parameter <data> must remain valid until the API call returns.

Return Value: The return value is used to indicate errors detected by the RTE during execution of the Rte_Read API call or errors detected by the communication system.

- **[SWS_Rte_01093]** ⌈ RTE_E_OK – data read successfully. ⌋(*SRS_Rte_00094*)
- **[SWS_Rte_02626]** ⌈ RTE_E_INVALID – data element invalid. ⌋(*SRS_Rte_00078*)
- **[SWS_Rte_02703]** ⌈ RTE_E_MAX_AGE_EXCEEDED – data element outdated. This Overlayed Error can be combined with any of the above error codes. ⌋(*SRS_Rte_00147*)
- **[SWS_Rte_07643]** ⌈ RTE_E_NEVER_RECEIVED – No data received since system start or partition restart. ⌋(*SRS_Rte_00184, SRS_Rte_00224*)
- **[SWS_Rte_01371]** ⌈ RTE_E_OUT_OF_RANGE – data element out of range. ⌋(*SRS_Rte_00180*)
- RTE_E_UNCONNECTED – Indicates that the receiver port is not connected [SWS_Rte_01330].

Notes: The API name includes an identifier <p>_<o> that indicates the read access point name and is formed from the port and operation item names. See section 5.2.6.4 for details on the naming convention.

5.6.11 Rte_DRead

Purpose: Performs an “explicit” read on a sender-receiver communication data element with “data” semantics (swImplPolicy != queued). By compatibility, the port may also have a ParameterInterface or a NvDataInterface. The Rte_DRead API is used for explicit read by value.

Signature: **[SWS_Rte_07394]** ⌈

```
<return>
Rte_[Byps_]DRead_<p>_<o>([IN Rte_Instance <instance>])
```

Where <p> is the port name and <o> the VariableDataPrototype within the sender-receiver interface categorizing the port. [Byps_] is an optional infix used when component wrapper method for bypass support is enabled for the related software component type (See chapter 4.9.2). ⌋*(SRS_BSW_00310, SRS_Rte_00141, SRS_Rte_00028, SRS_Rte_00131, SRS_Rte_00183)*

Existence: **[SWS_Rte_07395]** ⌈ A non-blocking Rte_DRead API shall be generated if a VariableAccess in the dataReceivePointByValue role references a required VariableDataPrototype with 'data' semantics. This requirement is applicable only for primitive data types. ⌋*(SRS_Rte_00051, SRS_Rte_00183)*

The RTE shall ensure that direct explicit read accesses will not deliver undefined data item values. In case there may be an explicit read access before the first data reception an initial value has to be provided as the result of this explicit read access. [SWS_Rte_07396]

A WaitPoint cannot reference a DataReceivedEvent that in turn references a required VariableDataPrototype with 'data' semantics. Such a configuration has to be considered as invalid (see [SWS_Rte_03018]). Hence there are no blocking Rte_DRead API.

[constr_9022] Rte_DRead API may only be used by the runnable that describe its usage ⌈ The Rte_DRead API may only be used by the runnable that contains the corresponding VariableAccess in the dataReceivePointByValue role ⌋

A DataReceivedEvent that references a runnable entity and is referenced by a WaitPoint shall be an invalid configuration. [SWS_Rte_01313]

The RTE generator shall take into account the kind of provide port which might not be just a variable but also a Parameter (fixed, const or standard), a standard sender (i.e. a variable) or a NV data. The table 4.6 gives an overview of compatibility rules.

Description: The Rte_DRead API returns the received data as a return value.

Return Value: The Rte_DRead return value provide access to the data value of the VariableDataPrototype.

The return type of Rte_DRead is dependent on the ImplementationDataType of the VariableDataPrototype. Thus the component does not need to use type casting to convert access to the VariableDataPrototype data.

For details of the <return> value definition see section 5.2.6.6.

Please note that the Rte_DRead API only supports VariableDataPrototypes typed by a Primitive Implementation Data Type or Redefinition Implementation Data Type redefining a Primitive Implementation Data Type.

Notes: The API name includes an identifier `<p>_<o>` that indicates the read access point name and is formed from the port and operation item names. See section 5.2.6.4 for details on the naming convention.

5.6.12 Rte_Receive

Purpose: Performs an "explicit" read on a sender-receiver communication data element with "event" semantics (`swImplPolicy` = `queued`).

[SWS_Rte_01092] ⌈

```
Std_ReturnType
Rte_[Byps_]Receive_<p>_<o>([IN Rte_Instance <instance>],
                          OUT <data>,
                          [OUT uint16 <length>])
```

Where `<p>` is the port name and `<o>` the data element within the sender-receiver interface categorizing the port. `[Byps_]` is an optional infix used when component wrapper method for bypass support is enabled for the related software component type (See chapter 4.9.2). ⌋*(SRS_BSW_00310, SRS_Rte_00141, SRS_Rte_00028, SRS_Rte_00131)*

Existence: **[SWS_Rte_01288]** ⌈ A non-blocking `Rte_Receive` API shall be generated if a `VariableAccess` in the `dataReceivePointByArgument` role references a required `VariableDataPrototype` with 'event' semantics. ⌋*(SRS_Rte_00051)*

[SWS_Rte_07638] ⌈ The RTE Generator shall reject configurations were a `VariableDataPrototype` with 'event' semantics is referenced by a `VariableAccess` in the `dataReceivePointByValue` role. ⌋*(SRS_Rte_00018)*

[SWS_Rte_07814] ⌈ The optional OUT parameter `<length>` of the `Rte_Receive` API shall be generated if the `VariableDataPrototype` is of type dynamic. ⌋*(SRS_Rte_00190)*

[SWS_Rte_01290] ⌈ A blocking `Rte_Receive` API shall be generated if a `VariableAccess` in the `dataReceivePointByArgument` role references a required `VariableDataPrototype` with 'event' semantics that is, in turn, referenced by a `DataReceivedEvent` and the `DataReceivedEvent` is referenced by a `WaitPoint`. ⌋*(SRS_Rte_00051)*

[constr_9023] `Rte_Receive` API may only be used by the runnable that describe its usage ⌈ The `Rte_Receive` API may only be used by the runnable that contains the corresponding `VariableAccess` in the `dataReceivePointByArgument` role ⌋

A DataReceivedEvent that references a runnable entity and is referenced by a WaitPoint has to be treated as an invalid configuration. [SWS_Rte_01313]

Description: The Rte_Receive API call includes the OUT parameter <data> to pass back the received data element.

If the VariableDataPrototype is of type dynamic, the Rte_Receive API call include the OUT parameter <length> to pass back the number of elements in the received data element.

The pointers to the OUT parameters must remain valid until the API call returns.

[SWS_Rte_07673] ⌈ In case return value is RTE_E_NO_DATA, RTE_E_TIMEOUT, RTE_E_UNCONNECTED or RTE_E_IN_EXCLUSIVE_AREA, the OUT parameters shall remain unchanged. ⌋*(SRS_Rte_00094, SRS_Rte_00141)*

Return Value: The return value is used to indicate errors detected by the RTE during execution of the Rte_Receive API call or errors detected by the communication system.

- **[SWS_Rte_02598]** ⌈ RTE_E_OK – data read successfully. ⌋*(SRS_Rte_00094)*
- **[SWS_Rte_01094]** ⌈ RTE_E_NO_DATA – (explicit non-blocking read) no events were received and no other error occurred when the read was attempted. ⌋*(SRS_Rte_00094)*
- **[SWS_Rte_01095]** ⌈ RTE_E_TIMEOUT – (explicit blocking read) no events were received and no other error occurred when the read was attempted. ⌋*(SRS_Rte_00094, SRS_Rte_00069)*
- **[SWS_Rte_02572]** ⌈ RTE_E_LOST_DATA – Indicates that some incoming data has been lost due to an overflow of the receive queue or due to an error of the underlying communication layers. This is not an error of the data returned in the parameters. This Overlayed Error can be combined with any of the above. ⌋*(SRS_Rte_00107, SRS_Rte_00110, SRS_Rte_00094)*
- RTE_E_UNCONNECTED – Indicates that the receiver port is not connected [SWS_Rte_01331].

  Unlike RTE_E_NO_DATA, there is no need to retry receiving an event in this case.
- **[SWS_Rte_02743]** ⌈ RTE_E_IN_EXCLUSIVE_AREA – Used only for the blocking API. RTE_E_IN_EXCLUSIVE_AREA indicates that the runnable can not enter wait, as one of the ExecutableEntitys in the call stack of this task is currently in an exclusive area, see [SWS_Rte_02739]. - In a properly con-

figured system, this error should not occur. The check can be disabled according to [SWS_Rte_08320]. ⌋*(SRS_Rte_00092, SRS_Rte_00046, SRS_Rte_00032)*

- **[SWS_Rte_08320]** ⌈ If `RteInExclusiveAreaCheckEnabled` is set to *false* the RTE generator shall omit the check and return of [SWS_Rte_02743]. ⌋*(SRS_Rte_00092, SRS_Rte_00046, SRS_Rte_00032)*

The `RTE_E_NO_DATA`, `RTE_E_TIMEOUT` and `RTE_E_UNCONNECTED` return values are not considered to be errors but rather indicate correct operation of the API call.

Notes: The API name includes an identifier `<p>_<o>` that indicates the read access point name and is formed from the port and operation item names. See Section 5.2.6.4 for details on the naming convention.

5.6.13 Rte_Call

Purpose: Initiate a client-server communication.

Signature: **[SWS_Rte_01102]** ⌈

```
Std_ReturnType
Rte_[Byps_]Call_<p>_<o>([IN Rte_Instance <instance>],
                        [IN|IN/OUT|OUT] <data_1>...
                        [IN|IN/OUT|OUT] <data_n>)
```

Where `<p>` is the port name and `<o>` the operation within the client-server interface categorizing the port. `[Byps_]` is an optional infix used when component wrapper method for bypass support is enabled for the related software component type (See chapter 4.9.2). ⌋*(SRS_BSW_00310, SRS_Rte_00029)*

Existence: **[SWS_Rte_01293]** ⌈ A synchronous `Rte_Call` API shall be generated if a `SynchronousServerCallPoint` references a required `ClientServerOperation`. ⌋*(SRS_Rte_00051, SRS_Rte_00111)*

[SWS_Rte_01294] ⌈ An asynchronous `Rte_Call` API shall be generated if an `AsynchronousServerCallPoint` references a required `ClientServerOperation`. ⌋*(SRS_Rte_00051, SRS_Rte_00111)*

A configuration that includes both synchronous and asynchronous ServerCallPoints for a given `ClientServerOperation` is invalid ([SWS_Rte_03014]).

[constr_9024] `Rte_Call` API may only be used by the runnable that describe its usage ⌈ The `Rte_Call` API may only be used by the runnable that contains the corresponding `ServerCallPoint` ⌋

Description: Client function to initiate client-server communication. The Rte_Call API is used for both synchronous and asynchronous calls.

The Rte_Call API includes zero or more IN, IN/OUT and OUT parameters.

[SWS_Rte_06639] ⌈ IN/OUT parameters are passed by value when they are "Primitive Implementation Data Type"s and the call is asynchronous. ⌋*(SRS_Rte_00051, SRS_Rte_00111)*

Rational: In case of an asynchronous call, the IN/OUT parameters are only IN parameters.

The IN, IN/OUT and OUT parameters are passed by value or reference according to the ImplementationDataType as described in the section 5.2.6.5.

The pointers to all parameters passed by reference must remain valid until the API call returns.

Return Value: **[SWS_Rte_01103]** ⌈ The return value shall be used to indicate infrastructure errors detected by the RTE during execution of the Rte_Call call and, for synchronous communication, infrastructure and application errors during execution of the server. ⌋*(SRS_Rte_00094, SRS_Rte_00123, SRS_Rte_00124)*

- **[SWS_Rte_01104]** ⌈ RTE_E_OK – The API call completed successfully. ⌋*(SRS_Rte_00094)*

  Note: This means that RTE_E_OK is returned when neither an infrastructure error nor an overlay error occurred at the invocation of the server runnable and the invoked server runnable was returning a value equal to E_OK.

- **[SWS_Rte_01105]** ⌈ RTE_E_LIMIT – The client has multiple outstanding asynchronous client-server invocations of the same operation in the same port. The server invocation shall be discarded, the buffers of the return parameters shall not be modified (see also [SWS_Rte_02658]). ⌋*(SRS_Rte_00094, SRS_Rte_00079)*

- **[SWS_Rte_08727]** ⌈ RTE_E_SERIALIZATION_LIMIT – The RTE is not able to allocate the buffer needed to serialize the data. ⌋*(SRS_Rte_00094, SRS_Rte_00091)*

- **[SWS_Rte_08728]** ⌈ RTE_E_SERIALIZATION_ERROR – The return value of the serialization call or the deserialization call (in case of a synchronous Rte_Call) was not equal to E_OK. ⌋*(SRS_Rte_00094, SRS_Rte_00091)*

- **[SWS_Rte_01106]** ⌈ RTE_E_COM_STOPPED – the RTE could not perform the operation because the COM service is currently not available (inter ECU communication only). RTE shall return RTE_E_COM_STOPPED when the corresponding COM service returns COM_SERVICE_NOT_AVAILABLE. The buffers of the return parameters shall not be modified. ⌋*(SRS_Rte_00094)*

- **[SWS_Rte_01107]** ⌈ RTE_E_TIMEOUT – (synchronous inter-task and inter-ECU only) No reply was received within the configured timeout. The buffers of the return parameters shall not be modified. ⌋*(SRS_Rte_00094, SRS_Rte_00069)*

- RTE_E_UNCONNECTED – Indicates that the client port is not connected [SWS_Rte_01334].

- **[SWS_Rte_02744]** ⌈ RTE_E_IN_EXCLUSIVE_AREA – Used only for the blocking API. RTE_E_IN_EXCLUSIVE_AREA indicates that the runnable can not enter wait, as one of the ExecutableEntitys in the call stack of this task is currently in an exclusive area, see [SWS_Rte_02739]. - In a properly configured system, this error should not occur. The check can be disabled according to [SWS_Rte_08321]. ⌋*(SRS_Rte_00092, SRS_Rte_00046, SRS_Rte_00032)*

- **[SWS_Rte_08321]** ⌈ If RteInExclusiveAreaCheckEnabled is set to *false* the RTE generator shall omit the check and return of [SWS_Rte_02744]. ⌋*(SRS_Rte_00092, SRS_Rte_00046, SRS_Rte_00032)*

- **[SWS_Rte_02755]** ⌈ RTE_E_SEG_FAULT – a segmentation violation is detected in the handed over parameters to the RTE API as required in [SWS_Rte_02752] and [SWS_Rte_02753]. No transmission is executed. ⌋*(SRS_Rte_00210)*

- **[SWS_Rte_02577]** ⌈ The application error (synchronous client-server) from a server shall only be returned if none of the above infrastructure errors (other than RTE_E_OK) have occurred. ⌋*(SRS_Rte_00123)*

Note that the RTE_E_OK return value indicates that the Rte_Call API call completed successfully. In case of a synchronous client server call it also indicates successful processing of the request by the server.

An asynchronous server invocation is considered to be outstanding until either the client retrieved the result successfully, a timeout was detected by the RTE in inter-ECU and inter-partition communication or the server runnable has terminated after a timeout was detected in intra-ECU communication.

When the RTE_E_TIMEOUT error occurs, RTE shall discard any subsequent responses to that request, (see [SWS_Rte_02657]).

Notes: **[SWS_Rte_01109]** ⌈ The interface operation's OUT parameters shall be omitted for an *asynchronous* call. ⌋*(SRS_Rte_00029, SRS_Rte_00079)*

In case of asynchronous communication:

- the `Rte_Call` only includes IN and IN/OUT parameters.
- the `Rte_Result` only includes IN/OUT and OUT parameters to collect the result of the server call.
- the IN/OUT parameters provided during the `Rte_Call` can be a different addresse than the IN/OUT parameter passed during the `Rte_Result`.

5.6.14 Rte_Result

Purpose: Get the result of an asynchronous client-server call.

Signature: **[SWS_Rte_01111]** ⌈

```
Std_ReturnType
Rte_[Byps_]Result_<p>_<o>([IN Rte_Instance <instance>],
                          [IN/OUT|OUT <param 1>]...
                          [IN/OUT|OUT <param n>])
```

Where `<p>` is the port name and `<o>` the operation within the client-server interface categorizing the port. `[Byps_]` is an optional infix used when component wrapper method for bypass support is enabled for the related software component type (See chapter 4.9.2). ⌋*(SRS_BSW_00310)*

The signature can include zero or more IN/OUT and OUT parameters depending on the signature of the operation in the client-server interface.

Existence: **[SWS_Rte_01296]** ⌈ A non-blocking `Rte_Result` API shall be generated if an `AsynchronousServerCallResultPoint` exists for the specific `RunnableEntity` and this `AsynchronousServerCallResultPoint` references an `AsynchronousServerCallPoint` which according to [SWS_Rte_01294] leads to the generation of an asynchronous `Rte_Call` API but no `WaitPoint` (of the `RunnableEntity`) references an `AsynchronousServerCallReturnsEvent` that references the `AsynchronousServerCallResultPoint`. ⌋*(SRS_Rte_00051)*

Please note that a non-blocking `Rte_Result` API does not require the existence of a `AsynchronousServerCallReturnsEvent`. If the `AsynchronousServerCallReturnsEvent` exists it can be used to trigger the execution of a RunnableEntity in which the non-blocking Rte_Result API function may be called.

[SWS_Rte_01297] ⌈ A blocking Rte_Result API shall be generated if an AsynchronousServerCallResultPoint exists for the specific RunnableEntity and this AsynchronousServerCallResultPoint references an AsynchronousServerCallPoint which according to [SWS_Rte_01294] leads to the generation of an asynchronous Rte_Call API and a WaitPoint (of the RunnableEntity) references an AsynchronousServerCallReturnsEvent that references the AsynchronousServerCallResultPoint. ⌋*(SRS_Rte_00051)*

[constr_9025] Blocking Rte_Result API may only be used by the runnable that describe the WaitPoint ⌈ The blocking Rte_Result API may only be used by the runnable that contains the corresponding WaitPoint ⌋

[SWS_Rte_01298] ⌈ If an AsynchronousServerCallReturnsEvent references a RunnableEntity and a required ClientServerOperation, the RunnableEntity shall be activated when the operation's result is available or when a timeout was detected by the RTE [SWS_Rte_01133]. ⌋*(SRS_Rte_00051)*

Requirement [SWS_Rte_01298] merely affects when the runnable is activated – an API call should still be created to actually read the reply based on requirement [SWS_Rte_01296].

[SWS_Rte_01312] ⌈ An AsynchronousServerCallReturnsEvent that references a runnable entity and is referenced by a WaitPoint is invalid. ⌋*(SRS_Rte_00051)*

Description: The Rte_Result API is used by a client to collect the result of an *asynchronous* client-server communication.

The Rte_Result API includes zero or more IN/OUT and OUT parameters to pass back results.

The pointers to all parameters passed by reference must remain valid until the API call returns.

Return Value: The return value is used to indicate errors from either the Rte_Result call itself or communication errors detected before the API call was made.

- **[SWS_Rte_01112]** ⌈ RTE_E_OK – The API call completed successfully. ⌋*(SRS_Rte_00094)*

  Note: This means that RTE_E_OK is returned when neither an infrastructure error nor an overlay error occurred at the invocation of the server runnable and the invoked server runnable was returning a value equal to E_OK.

- **[SWS_Rte_08729]** ⌈ RTE_E_SERIALIZATION_ERROR – The return value of the deserialization call was not equal to E_OK. ⌋*(SRS_Rte_00094, SRS_Rte_00091)*
- **[SWS_Rte_01113]** ⌈ RTE_E_NO_DATA – (non-blocking read) The server's result is not available but no other error occurred within the API call or the server was not called since Rte_Start or the restart of the Partition. The buffers for the IN/OUT and OUT parameters shall not be modified. ⌋*(SRS_Rte_00094)*
- **[SWS_Rte_08301]** ⌈ RTE_E_NO_DATA – (non-blocking read) The previous Rte_Call returned an RTE_E_SEG_FAULT, RTE_E_SERIALIZATION_LIMIT or RTE_E_SERIALIZATION_ERROR. ⌋*(SRS_Rte_00094)*
- **[SWS_Rte_01114]** ⌈ RTE_E_TIMEOUT – The server's result is not available within the specified timeout but no other error occurred within the API call. The buffers for the IN/OUT and OUT parameters shall not be modified. ⌋*(SRS_Rte_00094, SRS_Rte_00069)*
- **[SWS_Rte_03606]** ⌈ RTE_E_COM_STOPPED – the RTE could not perform the operation because the COM service is currently not available (inter ECU communication only). RTE shall return RTE_E_COM_STOPPED when the corresponding COM service returns COM_SERVICE_NOT_AVAILABLE. The server's result has *not* been successfully retrieved from the communication service. The buffers of the return parameters shall not be modified. ⌋*(SRS_Rte_00094)*
- RTE_E_UNCONNECTED – Indicates that the client port is not connected [SWS_Rte_01333].
- **[SWS_Rte_02745]** ⌈ RTE_E_IN_EXCLUSIVE_AREA – Used only for the blocking API. RTE_E_IN_EXCLUSIVE_AREA indicates that the runnable can not enter wait, as one of the ExecutableEntitys in the call stack of this task is currently in an exclusive area, see [SWS_Rte_02739]. - In a properly configured system, this error should not occur. The check can be disabled according to [SWS_Rte_08322]. ⌋*(SRS_Rte_00092, SRS_Rte_00046, SRS_Rte_00032)*
- **[SWS_Rte_08322]** ⌈ If RteInExclusiveAreaCheckEnabled is set to *false* the RTE generator shall omit the check and return of [SWS_Rte_02745]. ⌋*(SRS_Rte_00092, SRS_Rte_00046, SRS_Rte_00032)*

[SWS_Rte_02746] ⌈ Rte_Result shall not return RTE_E_IN_EXCLUSIVE_AREA, if the wait is resolved by a mapping of the server runnable to a task with higher priority on the same core. ⌋(SRS_Rte_00092, SRS_Rte_00046, SRS_Rte_00032)

- **[SWS_Rte_08302]** ⌈ RTE_E_SEG_FAULT – a segmentation violation is detected in the handed over parameters to the RTE API as required in [SWS_Rte_02752] and [SWS_Rte_02753]. No transmission is executed. ⌋(*SRS_Rte_00094*)
- **[SWS_Rte_02578]** ⌈ Application Errors – The error code of the server shall only be returned, if none of the above infrastructure errors or indications have occurred. ⌋(*SRS_Rte_00094, SRS_Rte_00123*)

The RTE_E_NO_DATA, RTE_E_TIMEOUT, and RTE_E_UNCONNECTED return values are not considered to be errors but rather indicate correct operation of the API call.

When the RTE_E_TIMEOUT error occurs, RTE has to discard any subsequent responses to that request, (see [SWS_Rte_02657]).

When RTE_E_NO_DATA occurs, a component is free to invoke Rte_Result again and thus repeat the attempt to read the server's result.

Notes: The API name includes an identifier <p>_<o> that indicates the read access point name and is formed from the port and operation item names. See Section 5.2.6.4 for details on the naming convention.

If a AsynchronousServerCallPoint exists which is not referenced by a WaitPoint, a non-blocking Rte_Result API shall be generated. In this case Rte_Result has to return RTE_E_NO_DATA until the timeout expires and RTE_E_TIMEOUT afterwards.

5.6.15 Rte_Pim

Purpose: Provide access to the defined per-instance memory (section) of a software component.

Signature: **[SWS_Rte_01118]** ⌈

```
<type>/<return reference>
Rte_[Byps_]Pim_<name>([IN Rte_Instance <instance>])
```

Where <name> is the (short) name of the per-instance name. [Byps_] is an optional infix used when component wrapper method for bypass support is enabled for the related software component type (See chapter 4.9.2). ⌋(*SRS_BSW_00310, SRS_Rte_00075*)

Existence: **[SWS_Rte_01299]** ⌈ An `Rte_Pim` API shall be created for each defined `PerInstanceMemory` or `arTypedPerInstanceMemory` within the AUTOSAR software-component (description). ⌋*(SRS_Rte_00051)*

Description: The `Rte_Pim` API provides access to the per-instance memory (section) defined in the context of a `SwcInternalBehavior` of a software-component description.

Return Value: **[SWS_Rte_01119]** ⌈ The API returns a typed reference (in C a typed pointer) to the per-instance memory. ⌋*(SRS_Rte_00051, SRS_Rte_00075)*

Notes: For a `'C' typed PerInstanceMemory`, the name of the return type `<type>` has to be defined in the `type` attribute of the `PerInstanceMemory`. The type itself is defined using the `typeDefinition` attribute of the `PerInstanceMemory`. It is assumed that this attribute contains a string that represents a C type definition (typedef) in valid C syntax (see [SWS_Rte_02304] and [SWS_Rte_07133]). For an `arTypedPerInstanceMemory` the `<return reference>` is defined by the associated `AutosarDataType` (see [SWS_Rte_07161]). For details of the `<return reference>` definition see section 5.2.6.7.

5.6.16 Rte_CData

Purpose: Provide access to the calibration parameter an AUTOSAR software-component defined internally. The `ParameterDataPrototype` in the role `perInstanceParameter` or `sharedParameter` is used to define software component internal calibration parameters. Internal because the `ParameterDataPrototype` cannot be reused outside the software-component. Access is read-only. It can be configured for each calibration parameter individually if it is shared by all instances of an AUTOSAR software-component or if each instance has an own data value associated with it.

Signature: **[SWS_Rte_01252]** ⌈

```
<return>
Rte_[Byps_]CData_<name>([IN Rte_Instance <instance>])
```

Where `<name>` is the calibration parameter name. `[Byps_]` is an optional infix used when component wrapper method for bypass support is enabled for the related software component type (See chapter 4.9.2). ⌋*(SRS_BSW_00310, SRS_Rte_00155)*

Existence: **[SWS_Rte_01300]** ⌈ An `Rte_CData` API shall be generated if a `ParameterAccess` references a `ParameterDataPrototype` in the role `perInstanceParameter` or `sharedParameter` within

the SwcInternalBehavior of an AUTOSAR software-component.
⌋*(SRS_Rte_00051, SRS_Rte_00155)*

Description: The Rte_CData API provides access to the defined calibration parameter within a software-component. The actual data values for a software-component instance may be set after component compilation.

Return Value: The Rte_CData return value provide access to the data value of the ParameterDataPrototype in the role perInstanceParameter or sharedParameter.

The return type of Rte_CData is dependent on the ImplementationDataType of the ParameterDataPrototype and can either be a value or a pointer to the location where the value can be accessed. Thus the component does not need to use type casting to convert access to the ParameterDataPrototype data.

For details of the <return> value definition see section 5.2.6.6.

[SWS_Rte_03927] ⌈ If a ParameterDataPrototype is aggregated by an SwcInternalBehavior in the role of sharedParameter, the return value of the corresponding Rte_CData API shall provide access to the calibration parameter value common to all instances of the AtomicSwComponentType. ⌋*(SRS_Rte_00051, SRS_Rte_00155)*

[SWS_Rte_03952] ⌈ If a ParameterDataPrototype is aggregated by an SwcInternalBehavior in the role of perInstanceParameter, the return value of the corresponding Rte_CData API shall provide access to the calibration parameter value specific to the instance of the AtomicSwComponentType. ⌋*(SRS_Rte_00051, SRS_Rte_00155)*

Notes: None.

5.6.17 Rte_Prm

Purpose: Provide access to the parameters defined by an AUTOSAR ParameterSwComponentType. Access is read-only.

Signature: **[SWS_Rte_03928]** ⌈

```
<return>
Rte_[Byps_]Prm_<p>_<o>([IN Rte_Instance <instance>])
```

Where <p> is the port name and <o> is the name of the ParameterDataPrototype within the ParameterInterface categorizing the port. [Byps_] is an optional infix used when component wrapper method for bypass support is enabled for the related software component type (See chapter 4.9.2). ⌋*(SRS_BSW_00310, SRS_Rte_00155)*

Existence: **[SWS_Rte_03929]** ⌈ A Rte_Prm API shall be generated if a ParameterAccess references a ParameterDataPrototype in a require PortPrototype. ⌋*(SRS_BSW_00310, SRS_Rte_00155)*

Description: The Rte_Prm API provides access to the defined parameter within a ParameterSwComponentType.

In the case of a standard parameter (swImplPolicy = standard), i.e. a calibration, the actual data values for a ParameterSwComponentType instance may be set after ParameterSwComponentType compilation.

In the case of fixed parameter or constant parameter, the value is set during compilation time.

Return Value: **[SWS_Rte_03930]** ⌈ For primitive data types, the Rte_Prm API shall return the parameter value. For composite data types, the Rte_Prm API shall return a reference (in C, a pointer) to the parameter, which shall be const. With fixed parameters, only primitive data is possible.

The return type of Rte_Prm is specified by the ImplementationDataType associated to the ParameterDataPrototype. Thus the component does not need to use type casting to access the calibration parameter. ⌋*(SRS_Rte_00051, SRS_Rte_00155, SRS_Rte_00171)* The Rte_Prm return value provide access to the data value of the ParameterDataPrototype.

The return type of Rte_Prm is dependent on the ImplementationDataType of the ParameterDataPrototype and can either be a value or a pointer to the location where the value can be accessed. Thus the component does not need to use type casting to convert access to the ParameterDataPrototype data.

For details of the <return> value definition see section 5.2.6.6.

Notes: The Rte_Prm API should not be used within a pre-compilation directive, e.g. #if. For such case, the coder should use the Rte_SysCon definitions which are dedicated to variant handling.

5.6.18 Rte_IRead

Purpose: Provide **read** access to the VariableDataPrototype referenced by VariableAccess in the dataReadAccess role.

Signature: **[SWS_Rte_03741]** ⌈

```
<return>
Rte_[Byps_]IRead_<re>_<p>_<o>([IN Rte_Instance])
```

Where `<re>` is the runnable entity name, `<p>` the port name and `<o>` the `VariableDataPrototype` name. `[Byps_]` is an optional infix used when component wrapper method for bypass support is enabled for the related software component type (See chapter 4.9.2).
⌋*(SRS_BSW_00310, SRS_Rte_00128)*

Existence: **[SWS_Rte_01301]** ⌈ An `Rte_IRead` API shall be created for a required `VariableDataPrototype` if the `RunnableEntity` has a `VariableAccess` in the `dataReadAccess` role referring to this `VariableDataPrototype`. ⌋*(SRS_Rte_00051)*

Description: The `Rte_IRead` API provides access to the `VariableDataPrototype`s declared as accessed by a runnable using `VariableAccess`es in the `dataReadAccess` role. As the APIcan also be used in context of category 1A runnables an implementation has to ensure finite and constant execution times.

No error information is provided by this API. If required, the error status can be picked up with a separate API, see 5.6.22

The data value can always be read. To provide the required consistency the API provides access to a *copy* of the data data element for which it's guaranteed that it never changes during the actual execution of the runnable entity.

Implicit data read access by a SW-C should always return defined data.

[SWS_Rte_01268] ⌈ The RTE shall ensure that implicit read accesses will not deliver undefined data item values. ⌋*(SRS_Rte_00108, SRS_Rte_00051, SRS_Rte_00128)*

In case where there may be an implicit read access before the first data reception an initial value has to be provided as the result of this implicit read access.

Return Value: The `Rte_IRead` return value provide access to the data value of the `VariableDataPrototype`.

The return type of `Rte_IRead` is dependent on the `ImplementationDataType` of the `VariableDataPrototype` and can either be a value or a pointer to the location where the value can be accessed. Thus the component does not need to use type casting to convert access to the `VariableDataPrototype` data.

For details of the `<return>` value definition see section 5.2.6.6.

Notes: None.

5.6.19 Rte_IWrite

Purpose: Provide **write** access to the `VariableDataPrototype`s referenced by `VariableAccess`es in the `dataWriteAccess` role.

Signature: **[SWS_Rte_03744]** ⌈

```
void
Rte_[Byps_]IWrite_<re>_<p>_<o>([IN RTE_Instance],
                               IN <data>)
```

Where `<re>` is the runnable entity name, `<p>` the port name and `<o>` the `VariableDataPrototype` name. `[Byps_]` is an optional infix used when component wrapper method for bypass support is enabled for the related software component type (See chapter 4.9.2). ⌋*(SRS_BSW_00310, SRS_Rte_00129)*

Existence: **[SWS_Rte_01302]** ⌈ An `Rte_IWrite` API shall be created for a provided `VariableDataPrototype` if the RunnableEntity has a `VariableAccess` in the `dataWriteAccess` role referring to this `VariableDataPrototype`. ⌋*(SRS_Rte_00051)*

Description: The `Rte_IWrite` API provides write access to the `VariableDataPrototype`s declared as accessed by a runnable using `VariableAccess`es in the `dataWriteAccess` role. The API function is guaranteed to be have constant execution time and therefore can also be used within category 1A runnable entities.

No access error information is required for the user – the value can always be written. To provide the required write-back semantics the RTE only makes written values available to other entities after the writing runnable entity has terminated.

[SWS_Rte_03746] ⌈ The `Rte_IWrite` API call includes the IN parameter `<data>` to pass the data element to write. ⌋*(SRS_Rte_00051, SRS_Rte_00129)*

The IN parameter `<data>` is passed by value or reference according to the `ImplementationDataType` as described in the section 5.2.6.5.

If the IN parameter `<data>` is passed by reference, the pointer must remain valid until the API call returns.

Return Value: None.

Notes: None.

5.6.20 Rte_IWriteRef

Purpose: Provide a reference to the VariableDataPrototype referenced by a VariableAccess in the dataWriteAccess role.

Signature: **[SWS_Rte_05509]** ⌈

```
<return reference>
Rte_[Byps_]IWriteRef_<re>_<p>_<o>([IN RTE_Instance])
```

Where <re> is the runnable entity name, <p> the port name and <o> the VariableDataPrototype name. [Byps_] is an optional infix used when component wrapper method for bypass support is enabled for the related software component type (See chapter 4.9.2).
⌋*(SRS_BSW_00310, SRS_Rte_00129)*

Existence: **[SWS_Rte_05510]** ⌈ An Rte_IWriteRef API shall be created for a provided VariableDataPrototype if the RunnableEntity has a VariableAccess in the dataWriteAccess role referring to this VariableDataPrototype. ⌋*(SRS_Rte_00051)*

Description: The Rte_IWriteRef API returns a reference to the VariableDataPrototypes declared as accessed by a runnable using VariableAccesses in the dataWriteAccess role. The reference can be used by the runnable to directly update the corresponding data elements. This is especially useful for data elements of Structure Implementation Data Type or Array Implementation Data Type. The API function is guaranteed to be have constant execution time and therefore can also be used within category 1A runnable entities.

No error information is required for the user. To provide the required write-back semantics the RTE only makes written values available to other entities after the writing runnable entity has terminated.

[constr_9026] Rte_IWriteRef may not return values written in previous executions ⌈ The reference returned by Rte_IWriteRef shall not be used by the runnables for reading the value previously written. ⌋

The rationale for [constr_9026] is that Rte_IWriteRef has a write semantic. Also, in case of an unconnected port, the written data shall be discarded (similarly to [SWS_Rte_01347]), and implementations may return a reference to the same buffer for all Rte_IWriteRef of unconnected provide ports.

Return Value: The Rte_IWriteRef return value provide access to the data write buffer of the VariableDataPrototype.

[SWS_Rte_05511] ⌈ Rte_IWriteRef returns a reference to the corresponding VariableDataPrototype. ⌋*(SRS_Rte_00051)*

The return reference type of Rte_IWriteRef is dependent on the ImplementationDataType of the VariableDataPrototype and is a pointer to the location where the value can be accessed. Thus the component does not need to use type casting to convert access to the VariableDataPrototype data.

For details of the <return reference> definition see section 5.2.6.7.

Notes: None.

5.6.21 Rte_IInvalidate

Purpose: Invalidate a VariableDataPrototype referenced by a VariableAccess in the dataWriteAccess role.

Signature: **[SWS_Rte_03800]** ⌈

```
void
Rte_[Byps_]IInvalidate_<re>_<p>_<o>([IN Rte_Instance <instance>])
```

Where <re> is the runnable entity name, <p> the port name and <o> the VariableDataPrototype name. [Byps_] is an optional infix used when component wrapper method for bypass support is enabled for the related software component type (See chapter 4.9.2).
⌋*(SRS_BSW_00310, SRS_Rte_00078)*

Existence: **[SWS_Rte_03801]** ⌈ An Rte_IInvalidate API shall be created for a provided VariableDataPrototype if the RunnableEntity has VariableAccesses in the dataWriteAccess role referring to this VariableDataPrototype and the associated InvalidationPolicy of the VariableDataPrototype is set to keep or replace. ⌋*(SRS_Rte_00051, SRS_Rte_00078)*

Description: The Rte_IInvalidate API takes no parameters other than the instance handle – the return value is used to indicate the success, or otherwise, of the API call to the caller.

[SWS_Rte_03802] ⌈ In case of a primitive VariableDataPrototype the Rte_IInvalidate shall be implemented as a macro that writes the invalidValue to the buffer. ⌋*(SRS_Rte_00078)*

[SWS_Rte_05064] ⌈ In case of a composite VariableDataPrototype the Rte_IInvalidate shall be implemented as a macro that writes the invalidValue of every primitive part of the composition to the buffer. ⌋*(SRS_Rte_00078)*

[SWS_Rte_03778] ⌈ If Rte_IInvalidate is followed by an Rte_IWrite call for the same VariableDataPrototype or vice versa, the RTE shall use the last value written before the runnable entity terminates (last-is-best semantics). ⌋*(SRS_Rte_00078)*

[SWS_Rte_03778] states that an Rte_IWrite overrules an Rte_IInvalidate call if it occurs after the Rte_IInvalidate, since Rte_IWrite overwrites the contents of the internal buffer for the data element prototype before it is made known to other runnable entities.

Return Value: None.

Notes: The communication service configuration determines whether the signal receiver(s) receive an "invalid signal" notification or whether the invalidated signal is silently replaced by the signal's initial value.

5.6.22 Rte_IStatus

Purpose: Provide the error status of a VariableDataPrototype referenced by a VariableAccess in the dataReadAccess role.

Signature: **[SWS_Rte_02599]** ⌈

```
Std_ReturnType
Rte_[Byps_]IStatus_<re>_<p>_<o>([IN Rte_Instance])
```

Where <re> is the runnable entity name, <p> the port name and <o> the VariableDataPrototype name. [Byps_] is an optional infix used when component wrapper method for bypass support is enabled for the related software component type (See chapter 4.9.2). ⌋*(SRS_Rte_00147, SRS_Rte_00078)*

Existence: **[SWS_Rte_02600]** ⌈ An Rte_IStatus API shall be created for a required VariableDataPrototype if a RunnableEntity has a VariableAccess in the dataReadAccess role referring to this VariableDataPrototype, and

- if at the RPortPrototype or PRPortPrototype a Non-queuedReceiverComSpec with either
  - the attribute aliveTimeout set to a value greater than zero **and/or**
  - the attribute handleNeverReceived set to TRUE **and/or**
  - the attribute handleOutOfRange not set to none

  and/or
- if at the SenderReceiverInterface classifying the RPort-Prototype or PRPortPrototype an InvalidationPolicy set to keep

is specified for this VariableDataPrototype. ⌋(*SRS_Rte_00147*, *SRS_Rte_00078*)

[constr_9027] Rte_IStatus API shall only be used by a RunnableEntity describing an access to the data or which is triggered by an error event related to this data ⌈ The Rte_IStatus API shall only be used by a RunnableEntity that either has a VariableAccess in the dataReadAccess role referring to the VariableDataPrototype or is triggered by a DataReceiveErrorEvent referring to the VariableDataPrototype. ⌋

Description: The Rte_IStatus API provides access to the current status of the data elements declared as accessed by a runnable using a VariableAccess in the dataReadAccess role. The API function is guaranteed to be have constant execution time and therefore can also be used within category 1A runnable entities.

To provide the required consistency access by a runnable is to a *copy* of the status together with the data that is guaranteed never to be modified by the RTE during the lifetime of the runnable entity.

Return Value: The return value is used to indicate errors detected by the communication system.

- **[SWS_Rte_02602]** ⌈ RTE_E_OK – no errors. ⌋(*SRS_Rte_00094*)
- **[SWS_Rte_02603]** ⌈ RTE_E_INVALID – data element invalid. ⌋(*SRS_Rte_00078*)
- **[SWS_Rte_02604]** ⌈ RTE_E_MAX_AGE_EXCEEDED – data element outdated. This Overlayed Error can be combined with any of the above error codes. ⌋(*SRS_Rte_00147*)
- **[SWS_Rte_07644]** ⌈ RTE_E_NEVER_RECEIVED – No data received since system start or partition restart. ⌋(*SRS_Rte_00184*, *SRS_Rte_00224*)
- **[SWS_Rte_01372]** ⌈ RTE_E_OUT_OF_RANGE – data element out of range. ⌋(*SRS_Rte_00180*)
- RTE_E_UNCONNECTED – Indicates that the receiver port is not connected [SWS_Rte_03785].

Notes: None.

5.6.23 Rte_IrvIRead

Purpose: Provide **read** access to the *InterRunnableVariables with implicit* behavior of an AUTOSAR SW-C.

Signature: **[SWS_Rte_03550]** ⌈

```
<return>
Rte_[Byps_]IrvIRead_<re>_<o>([IN RTE_Instance <instance>])
```

Where `<re>` is the name of the runnable entity the API might be used in, `<o>` is the name of the `VariableDataPrototype` in role `implicitInterRunnableVariable`. `[Byps_]` is an optional infix used when component wrapper method for bypass support is enabled for the related software component type (See chapter 4.9.2).
⌋*(SRS_BSW_00310, SRS_Rte_00142)*

Existence: **[SWS_Rte_01303]** ⌈ An `Rte_IrvIRead` API shall be created for each `VariableAccess` in role `readLocalVariable` to an `implicitInterRunnableVariable`. ⌋*(SRS_Rte_00051, SRS_Rte_00142)*

Description: The `Rte_IrvIRead` API provides read access to the defined InterRunnableVariables with *implicit* behavior within a component description.

The return value is used to deliver the requested data value. The return value is not required to pass error information to the user because no inter-ECU communication is involved and there will always be a readable value present.

Return Value: The `Rte_IrvIRead` return value provide access to the data value of the InterRunnableVariable.

The return type of `Rte_IrvIRead` is dependent on the `ImplementationDataType` of the InterRunnableVariable and can either be a value or a pointer to the location where the value can be accessed. Thus the component does not need to use type casting to convert access to the InterRunnableVariable data.

For details of the `<return>` value definition see section 5.2.6.6.

Notes: The runnable entity name in the signature allows runnable context specific optimizations.

The concept of InterRunnableVariables is explained in section 4.2.5.6. More details about InterRunnableVariables with *implicit* behavior is explained in section 4.2.5.6.1.

5.6.24 Rte_IrvIWrite

Purpose: Provide **write** access to the *InterRunnableVariables with implicit behavior* of an AUTOSAR SW-C.

Signature: **[SWS_Rte_03553]** ⌈

```
void
Rte_[Byps_]IrvIWrite_<re>_<o>([IN RTE_Instance <instance>],
                               IN <data>)
```

Where `<re>` is the name of the `RunnableEntity` the API might be used in, `<o>` is the name of the `VariableDataPrototype` in the role `implicitInterRunnableVariable` to access and `<data>` is the placeholder for the data the InterRunnableVariable shall be set to. `[Byps_]` is an optional infix used when component wrapper method for bypass support is enabled for the related software component type (See chapter 4.9.2). ⌋*(SRS_BSW_00310, SRS_Rte_00142)*

Existence: **[SWS_Rte_01304]** ⌈ An `Rte_IrvIWrite` API shall be created for each `VariableAccess` in role `writtenLocalVariable` to an `implicitInterRunnableVariable`. ⌋*(SRS_Rte_00142, SRS_Rte_00051)*

Description: The `Rte_IrvIWrite` API provides write access to the InterRunnableVariables with *implicit* behavior within a component description. The runnable entity name in the signature allows runnable context specific optimizations.

The data given by `Rte_IrvIWrite` is dependent on the InterRunnableVariable data type. Thus the component does not need to use type casting to write the InterRunnableVariable.

The return value is unused. The return value is not required to pass error information to the user because no inter-ECU communication is involved and the value can always be written.

The IN parameter `<data>` is passed by value or reference according to the `ImplementationDataType` as described in the section 5.2.6.5.

Return Value: None.

Notes: The runnable entity name in the signature allows runnable context specific optimizations.

The concept of InterRunnableVariables is explained in section 4.2.5.6. Further details about InterRunnableVariables with *implicit* behavior are explained in Section 4.2.5.6.1.

5.6.25 Rte_IrvRead

Purpose: Provide **read** access to the *InterRunnableVariables with explicit behavior* of an AUTOSAR SW-C.

Signature: **[SWS_Rte_03560]** ⌈

primitive type signature:

```
<return>
Rte_[Byps_]IrvRead_<re>_<o>([IN RTE_Instance <instance>])
```

complex type signature:

```
void
Rte_[Byps_]IrvRead_<re>_<o>([IN RTE_Instance <instance>],
                                 OUT <data>)
```

Where <re> is the name of the runnable entity the API might be used in, <o> is the name of the InterRunnableVariables. [Byps_] is an optional infix used when component wrapper method for bypass support is enabled for the related software component type (See chapter 4.9.2).

The complex type signature is used, if the ImplementationDataType of the InterRunnableVariable resolves to Array Implementation Data Type or Structure Implementation Data Type, otherwise the primitive type signature is used.
⌋(*SRS_BSW_00310, SRS_Rte_00142*)

Existence: **[SWS_Rte_01305]** ⌈ An Rte_IrvRead API shall be created for each read InterRunnableVariable using explicit access.

⌋(*SRS_Rte_00142, SRS_Rte_00051*)

Description: The Rte_IrvRead API provides read access to the defined InterRunnableVariables with *explicit* behavior within a component description.

The return value is not required to pass error information to the user because no inter-ECU communication is involved and there will always be a readable value present.

For the primitive type signature, the return value is used to deliver the requested data value. For the complex type signature, the return value is void.

For the complex type signature, the Rte_IrvRead API call includes the OUT parameter <data> to pass back the received data. The OUT parameter <data> is typed as reference (pointer) to the type of the InterRunnableVariable. The pointer to the OUT parameter <data> must remain valid until the API call returns.

Return Value: The Rte_IrvRead return value provide access to the data value of the InterRunnableVariable.

The return type of Rte_IrvRead is dependent on the ImplementationDataType of the InterRunnableVariable. Thus the component does not need to use type casting to convert access to the InterRunnableVariable data.

For details of the <return> value definition see section 5.2.6.6.

Please note that the Rte_IrvRead API Signature only has a return value if the InterRunnableVariable is typed by a Primitive Implementation Data Type or Redefinition Implementation Data Type redefining a Primitive Implementation Data Type.

[SWS_Rte_03562] ⌈ For the primitive type signature, the Rte_IrvRead call shall return the value of the accessed InterRunnableVariable. ⌋*(SRS_Rte_00142, SRS_Rte_00051)*

For complex type signature, the Rte_IrvRead call does not return any value (void).

Notes: The runnable entity name in the signature allows runnable context specific optimizations.

The concept of InterRunnableVariables is explained in section 4.2.5.6. Further details about InterRunnableVariables with *explicit* behavior are explained in Section 4.2.5.6.2.

5.6.26 Rte_IrvWrite

Purpose: Provide **write** access to the *InterRunnableVariables with explicit behavior* of an AUTOSAR SW-C.

Signature: **[SWS_Rte_03565]** ⌈

```
void
Rte_[Byps_]IrvWrite_<re>_<o>([IN RTE_Instance <instance>],
                             IN <data>)
```

Where <re> is the name of the runnable entity the API might be used in, <o> is the name of the InterRunnableVariable to access and <data> is the placeholder for the data the InterRunnableVariable shall be set to. [Byps_] is an optional infix used when component wrapper method for bypass support is enabled for the related software component type (See chapter 4.9.2). ⌋*(SRS_BSW_00310, SRS_Rte_00142)*

Existence: **[SWS_Rte_01306]** ⌈ An Rte_IrvWrite API shall be created for each written InterRunnableVariable using explicit access. ⌋*(SRS_Rte_00142, SRS_Rte_00051)*

Description: The Rte_IrvWrite API provides write access to the InterRunnableVariables with *explicit* behavior within a component description.

The return value is unused. The return value is not required to pass error information to the user because no inter-ECU communication is involved and the value can always be written.

[SWS_Rte_03567] ⌈ The `Rte_IrvWrite` API call include the IN parameter `<data>` to pass the data element to write. ⌋*(SRS_Rte_00142, SRS_Rte_00051)*

The IN parameter `<data>` is passed by value or reference according to the `ImplementationDataType` as described in the section 5.2.6.5.

If the IN parameter `<data>` is passed by reference, the pointer must remain valid until the API call returns.

Return Value: None.

Notes: The runnable entity name in the signature allows runnable context specific optimizations.

The concept of InterRunnableVariables is explained in section 4.2.5.6. Further details about InterRunnableVariables with *explicit* behavior are explained in Section 4.2.5.6.2.

5.6.27 Rte_Enter

Purpose: Enter an exclusive area.

Signature: **[SWS_Rte_01120]** ⌈

```
void
Rte_[Byps_]Enter_<name>([IN Rte_Instance <instance>])
```

Where `<name>` is the exclusive area name. `[Byps_]` is an optional infix used when component wrapper method for bypass support is enabled for the related software component type (See chapter 4.9.2). ⌋*(SRS_BSW_00310, SRS_Rte_00046, SRS_Rte_00115)*

Existence: **[SWS_Rte_01307]** ⌈ An `Rte_Enter` API shall be created for each ExclusiveArea that is declared and which has an *canEnterExclusiveArea* association. ⌋*(SRS_Rte_00115, SRS_Rte_00051)*

Description: The `Rte_Enter` API call is invoked by an AUTOSAR software-component to define the start of an exclusive area.

Return Value: None.

Notes: The RTE is not required to support nested invocations of `Rte_Enter` for the same exclusive area.

[SWS_Rte_01122] ⌈ The RTE shall permit calls to `Rte_Enter` and `Rte_Exit` to be nested as long as different exclusive areas are exited in the reverse order they were entered. ⌋(*SRS_Rte_00046*, *SRS_Rte_00032*, *SRS_Rte_00115*)

[constr_9028] `Rte_Enter` and `Rte_Exit` API may only be used by runnables describing its usage ⌈ The `Rte_Enter` and `Rte_Exit` API may only be used by *Runnable Entities* that contain a corresponding *canEnterExclusiveArea* association ⌋

[constr_9029] Nested call of `Rte_Enter` and `Rte_Exit` is restricted ⌈ The `Rte_Enter` and `Rte_Exit` API may only be called nested if different exclusive areas are invoked; in this case exclusive areas shall exited in the reverse order they were entered. ⌋

Within the AUTOSAR OS an attempt to lock a resource cannot fail because the lock is already held. The lock attempt can only fail due to configuration errors (e.g. caller not declared as accessing the resource) or invalid handle. Therefore the return type from this function is `void`.

5.6.28 Rte_Exit

Purpose: Leave an exclusive area.

Signature: **[SWS_Rte_01123]** ⌈

```
void
Rte_[Byps_]Exit_<name>([IN Rte_Instance <instance>])
```

Where `<name>` is the exclusive area name. `[Byps_]` is an optional infix used when component wrapper method for bypass support is enabled for the related software component type (See chapter 4.9.2). ⌋(*SRS_BSW_00310*, *SRS_Rte_00046*, *SRS_Rte_00051*)

Existence: **[SWS_Rte_01308]** ⌈ An `Rte_Exit` API shall be created for each ExclusiveArea that is declared and which has an *canEnterExclusiveArea* association. ⌋(*SRS_Rte_00115*, *SRS_Rte_00051*)

Description: The `Rte_Exit` API call is invoked by an AUTOSAR software-component to define the end of an exclusive area.

Return Value: None.

Notes: The RTE is not required to support nested invocations of `Rte_Exit` for the same exclusive area.

Requirement [SWS_Rte_01122] permits calls to `Rte_Enter` and `Rte_Exit` to be nested as long as different exclusive areas are exited in the reverse order they were entered.

5.6.29 Rte_Mode

There exist two versions of the Rte_Mode API. Depending on the attribute enhancedModeApi in the *software component description* there shall be provided different versions of this API (see also 5.6.30).

Purpose: Provides the currently active mode of a mode switch port.

Signature: **[SWS_Rte_02628]** ⌈

```
<return>
Rte_[Byps_]Mode_<p>_<o>([IN Rte_Instance <instance>])
```

Where <p> is the port name, and <o> the ModeDeclarationGroupPrototype name within the ModeSwitchInterface categorizing the port. [Byps_] is an optional infix used when component wrapper method for bypass support is enabled for the related software component type (See chapter 4.9.2). ⌋*(SRS_Rte_00144)*

Existence: **[SWS_Rte_02629]** ⌈ If a ModeAccessPoint exists and if the attribute enhancedModeApi of the ModeSwitchSenderComSpec resp. ModeSwitchReceiverComSpec is set to *false* or does not exist a Rte_Mode API according to [SWS_Rte_02628] shall be generated. ⌋*(SRS_Rte_00147, SRS_Rte_00078)*

[constr_9030] Rte_Mode API may only be used by the runnable that describe its usage ⌈ The Rte_Mode API may only be used by the runnable that contains the corresponding ModeAccessPoint ⌋

Description: The Rte_Mode API tells the AUTOSAR software-component which mode of a ModeDeclarationGroup of a given port is currently active. This is the information that the RTE uses for the ModeDisablingDependency's. A new mode will not be indicated immediately after the reception of a mode switch notification from a mode manager, see section 4.4.4. During mode transitions, i.e. during the execution of runnables that are triggered on exiting one mode or on entering the next mode, overlapping mode disablings of two modes are active. In this case, the Rte_Mode will return RTE_TRANSITION_<ModeDeclarationGroup>.

The Rte_Mode will return the same mode for all mode switch ports that are connected to the same mode switch port of the mode manager (see [SWS_Rte_02630]).

It is supported to have ModeAccessPoint(s) referring the provide mode switch ports of the mode manager to provide access for the mode manager on the information that the RTE uses for the ModeDisablingDependency's.

Return Value: The return type of Rte_Mode is dependent on the ImplementationDataType of the ModeDeclarationGroup. It shall return the value of the ModeDeclarationGroupPrototype. The type name shall be equal to the shortName of the Implementation-DataType.

The return value of the Rte_Mode is used to inform the caller about the current mode of the mode machine instance. The Rte_Mode API shall return the following values:

[SWS_Rte_07666] ⌈ During a transition of the mode machine instance, Rte_Mode shall return RTE_TRANSITION_<ModeDeclarationGroup>, where <ModeDeclarationGroup> is the short name of the ModeDeclarationGroup. ⌋*(SRS_Rte_00144)*

[SWS_Rte_02660] ⌈ When the mode machine instance is in a defined mode, Rte_Mode shall return RTE_MODE_<ModeDeclarationGroup>_<ModeDeclaration>, where <ModeDeclarationGroup> is the short name of the ModeDeclarationGroup and <ModeDeclaration> is the short name of the currently active ModeDeclaration. ⌋*(SRS_Rte_00144)*

[SWS_Rte_06742] ⌈ The API Rte_Mode shall return the value RTE_TRANSITION_<ModeDeclarationGroup> for a mode machine instance assigned to the RTE ([SWS_Rte_07533]) until the RTE has been initialized and where <ModeDeclarationGroup> is the short name of the ModeDeclarationGroup. ⌋*(SRS_Rte_00144)*

[SWS_Rte_06781] ⌈ If modeManagerErrorBehavior.errorReactionPolicy is set to defaultMode the API Rte_Mode shall return the value RTE_TRANSITION_<ModeDeclarationGroup> for a mode machine instance while the partition of the mode users is stopped or restarting and until the RTE dequeues the next mode switch notifications.
<ModeDeclarationGroup> is the short name of the ModeDeclarationGroup. ⌋*(SRS_Rte_00144)* This indicates a transition and therefore the behavior is identical as during the initialization of the RTE (see [SRS_Rte_00144]).

[SWS_Rte_06782] ⌈ If the modeManagerErrorBehavior.errorReactionPolicy is set to lastMode, the API enhanced Rte_Mode shall return the value RTE_MODE_<ModeDeclarationGroup>_<ModeDeclaration> of the last mode for a mode machine instance while the partition of the mode users is stopped or restarting and until the RTE dequeues the next mode switch notifications.
<ModeDeclarationGroup> is the short name of the ModeDeclarationGroup. ⌋*(SRS_Rte_00144)* This indicates a stable mode during the re-initialization of the partition until the RTE is

capable to dequeue the first mode switch notification after the partition restart.

[SWS_Rte_06743] ⌈ The Rte_Mode API shall return the values according [SWS_Rte_07666] and [SWS_Rte_02660] for a common mode machine instance already after initialization of the *Basic Software Scheduler*. ⌋*(SRS_Rte_00144)*

In inter partition mode management, RTE on the mode manager sided partition might not have direct access to the state variables of the mode machine instance.

[SWS_Rte_02732] ⌈ In inter partition mode management, the return value of the Rte_Mode API to the mode manager shall be consistent with the start of a transition by the Rte_Switch API and the inter partition communication of the ModeSwitchedAckEvent. ⌋*(SRS_Rte_00144, SRS_Rte_00210)*

Notes: The Rte_Mode API may already indicate the next ModeDeclaration, before the mode manager has picked up the ModeSwitchedAckEvent with the Rte_SwitchAck. This is not in contradiction to [SWS_Rte_02732].

[SWS_Rte_06744] ⌈ The RTE shall support calls of Rte_Mode after initialization of the *Basic Software Scheduler* but before the RTE is initialized. ⌋*(SRS_Rte_00144)*

5.6.30 Enhanced Rte_Mode

Purpose: Provides the currently active mode of a mode switch port. If the mode machine instance is in transition additionally the values of the previous and the next mode are provided.

Signature: **[SWS_Rte_08500]** ⌈

```
<return>
Rte_[Byps_]Mode_<p>_<o>([IN Rte_Instance <instance>,]
                        OUT <previousmode>,
                        OUT <nextmode>)
```

Where <p> is the port name, and <o> the ModeDeclarationGroupPrototype name within the ModeSwitchInterface categorizing the port. [Byps_] is an optional infix used when component wrapper method for bypass support is enabled for the related software component type (See chapter 4.9.2). ⌋*(SRS_Rte_00144)*

Existence: **[SWS_Rte_08501]** ⌈ The existence of a ModeAccessPoint given that the attribute *enhancedModeApi* of the ModeSwitchSenderComSpec resp. ModeSwitchReceiverComSpec is set to *true* shall result in the generation of a Rte_Mode API according to [SWS_Rte_08500]. ⌋*(SRS_Rte_00147, SRS_Rte_00078)*

[constr_9031] Rte_Mode API may only be used by the runnable that describe its usage ⌈ The Rte_Mode API may only be used by the runnable that contains the corresponding ModeAccessPoint ⌋

Description: The Rte_Mode API tells the AUTOSAR software-component which mode of a ModeDeclarationGroup of a given port is currently active. This is the information that the RTE uses for the ModeDisablingDependency's. A new mode will not be indicated immediately after the reception of a mode switch notification from a mode manager, see section 4.4.4. During mode transitions, i.e. during the execution of runnables that are triggered on exiting one mode or on entering the next mode, overlapping mode disablings of two modes are active. In this case, the Rte_Mode will return RTE_TRANSITION_<ModeDeclarationGroup>. The parameter <previousmode> than contains the mode currently being left,the parameter <nextmode> the mode being entered.

The Rte_Mode will return the same mode for all mode switch ports that are connected to the same mode switch port of the mode manager (see [SWS_Rte_02630]).

It is supported to have ModeAccessPoint(s) referring the provided mode switch ports of the mode manager to provide access for the mode manager on the information that the RTE uses for the ModeDisablingDependency's.

Return Value: The return type of Rte_Mode is dependent on the ImplementationDataType of the ModeDeclarationGroup. It shall return the value of the ModeDeclarationGroupPrototype. The type name shall be equal to the shortName of the ImplementationDataType. The return value of the Rte_Mode and the parameters <previousmode> and <nextmode> are used to inform the caller about the current mode of the mode machine instance.

[SWS_Rte_08504] ⌈ During a transition of a mode machine instance Rte_Mode shall return the following values

- the return value shall be RTE_TRANSITION_<ModeDeclarationGroup>,
- <previousmode> shall contain the value of the RTE_MODE_<ModeDeclarationGroup>_<ModeDeclaration> of the mode being left,
- <nextmode> shall contain the RTE_MODE_<ModeDeclarationGroup>_<ModeDeclaration> of the mode being entered,

where `<ModeDeclarationGroup>` is the short name of the ModeDeclarationGroup and `<ModeDeclaration>` is the short name of the ModeDeclaration. ⌋*(SRS_Rte_00144, SRS_Rte_00210)*

[SWS_Rte_08505] ⌈ When the mode machine instance is in a defined mode, Rte_Mode shall return the following values

- the return value shall contain the value of RTE_MODE_<ModeDeclarationGroup>_<ModeDeclaration>,
- <previousmode> shall contain the value of the RTE_MODE_<ModeDeclarationGroup>_<ModeDeclaration>
- <nextmode> shall contain the RTE_MODE_<ModeDeclarationGroup>_<ModeDeclaration>

where <ModeDeclarationGroup> is the short name of the ModeDeclarationGroup and <ModeDeclaration> is the short name of the currently active ModeDeclaration. ⌋*(SRS_Rte_00144)*

[SWS_Rte_06745] ⌈ The API enhanced Rte_Mode shall return the following values for a mode machine instance assigned to the RTE ([SWS_Rte_07533]) until the RTE has been initialized:

- the return value shall be RTE_TRANSITION_<ModeDeclarationGroup>,
- <previousmode> shall contain the value of the RTE_MODE_<ModeDeclarationGroup>_<ModeDeclaration> of the initialMode of the ModeDeclarationGroup
- <nextmode> shall contain the value of the RTE_MODE_<ModeDeclarationGroup>_<ModeDeclaration> of the initialMode of the ModeDeclarationGroup

where <ModeDeclarationGroup> is the short name of the ModeDeclarationGroup. ⌋*(SRS_Rte_00144)*

[SWS_Rte_06783] ⌈ If modeManagerErrorBehavior.errorReactionPolicy is set to defaultMode the API enhanced Rte_Mode shall return the following values for a mode machine instance while the partition of the mode users is stopped or restarting and until the RTE dequeues the next mode switch notifications.

- the return value shall be RTE_TRANSITION_<ModeDeclarationGroup>,
- <previousmode> shall contain the value of the RTE_MODE_<ModeDeclarationGroup>_<ModeDeclaration> of the modeUserErrorBehavior.defaultMode of the ModeDeclarationGroup

- `<nextmode>` shall contain the value of the `RTE_MODE_<ModeDeclarationGroup>_<ModeDeclaration>` of the `modeUserErrorBehavior.defaultMode` of the `ModeDeclarationGroup`

where `<ModeDeclarationGroup>` is the short name of the `ModeDeclarationGroup`. ⌋*(SRS_Rte_00144)* This indicates a transition from and to the `defaultMode`. If the `defaultMode` is identical to the `initialMode` the behavior is identical as during the initialization of the RTE (see [SRS_Rte_00144]).

[SWS_Rte_06784] ⌈ If the `modeManagerErrorBehavior.errorReactionPolicy` is set to `lastMode`, the API enhanced `Rte_Mode` shall return the following values for a `mode machine instance` while the partition of the mode users is stopped or restarting and until the RTE dequeues the next `mode switch notifications`.

- the return value shall be `RTE_MODE_<ModeDeclarationGroup>_<ModeDeclaration>` of the last mode,
- `<previousmode>` shall contain the value of the `RTE_MODE_<ModeDeclarationGroup>_<ModeDeclaration>` of the last mode
- `<nextmode>` shall contain the value of the `RTE_MODE_<ModeDeclarationGroup>_<ModeDeclaration>` of the last mode

where `<ModeDeclarationGroup>` is the short name of the `ModeDeclarationGroup`. ⌋*(SRS_Rte_00144)* This indicates a stable mode during the re-initialization of the partition until the RTE is capable to dequeue the first `mode switch notification` after the partition restart.

[SWS_Rte_06746] ⌈ The enhanced `Rte_Mode` API shall return the values according [SWS_Rte_08504] and [SWS_Rte_08505] for a `common mode machine instance` already after initialization of the *Basic Software Scheduler*. ⌋*(SRS_Rte_00144)*

In inter partition mode management, RTE on the `mode manager` sided partition might not have direct access to the state variables of the `mode machine instance`.

[SWS_Rte_08506] ⌈ In inter partition mode management, the return value and the contents of the parameters `<previousmode>` and `<nextmode>` of the `Rte_Mode` API to the `mode manager` shall be consistent with the start of a transition by the `Rte_Switch` API and the inter partition communication of the ModeSwitchedAckEvent. ⌋(SRS_Rte_00144, SRS_Rte_00210)

Notes: The Rte_Mode API may already indicate the next ModeDeclaration, before the mode manager has picked up the ModeSwitchedAckEvent with the Rte_SwitchAck. This is not in contradiction to [SWS_Rte_02732].

[SWS_Rte_06747] ⌈ The RTE shall support calls of the enhanced Rte_Mode after initialization of the *Basic Software Scheduler* but before the RTE is initialized. ⌋(*SRS_Rte_00144*)

5.6.31 Rte_Trigger

Purpose: Raise a external trigger of a trigger port.

Signature: **[SWS_Rte_07200]** ⌈
signature without queuing support:

```
void
Rte_[Byps_]Trigger_<p>_<o>([IN Rte_Instance <instance>])
```

signature with queuing support:

```
Std_ReturnType
Rte_[Byps_]Trigger_<p>_<o>([IN Rte_Instance <instance>])
```

Where <p> is the port name and <o> the Trigger within the trigger interface categorizing the port. [Byps_] is an optional infix used when component wrapper method for bypass support is enabled for the related software component type (See chapter 4.9.2).

The signature for queuing support shall be generated by the RTE generator if the swImplPolicy of the associated Trigger is set to queued. ⌋(*SRS_Rte_00162*)

Existence: **[SWS_Rte_07201]** ⌈ The existence of a ExternalTriggeringPoint shall result in the generation of a Rte_Trigger API. ⌋(*SRS_Rte_00162*)

[constr_9032] Rte_Trigger API may only be used by the runnable that describe its usage ⌈ The Rte_Trigger API may only be used by the runnable that contains the corresponding ExternalTriggeringPoint. ⌋

Description: The Rte_Trigger API triggers an execution for all runnables whose ExternalTriggerOccurredEvent is associated to the Trigger.

Return Value: None in case of signature without queuing support.

[SWS_Rte_06720] ⌈ The Rte_Trigger API shall return the following values:

- RTE_E_OK if the trigger was successfully queued or if no queue is configured
- RTE_E_LIMIT if the trigger was not queued because the maximum queue size is already reached.

in the case of signature with queuing support. ⌋*(SRS_Rte_00235)*

5.6.32 Rte_IrTrigger

Purpose: Raise a internal trigger to activate Runnable entities of the same software component instance.

Signature: **[SWS_Rte_07203]** ⌈
signature without queuing support:

```
void
Rte_[Byps_]IrTrigger_<re>_<o>([IN Rte_Instance <instance>])
```

signature with queuing support:

```
Std_ReturnType
Rte_[Byps_]IrTrigger_<re>_<o>([IN Rte_Instance <instance>])
```

Where <re> is the name of the runnable entity the API might be used in and <o> is the name of the InternalTriggeringPoint. [Byps_] is an optional infix used when component wrapper method for bypass support is enabled for the related software component type (See chapter 4.9.2).

The signature for queuing support shall be generated by the RTE generator if the swImplPolicy of the associated InternalTriggeringPoint is set to queued. ⌋*(SRS_Rte_00163)*

Existence: **[SWS_Rte_07204]** ⌈ The existence of a InternalTriggeringPoint shall result in the generation of a Rte_IrTrigger API. ⌋*(SRS_Rte_00163)*

[constr_9033] Rte_IrTrigger API may only be used by the runnable that describe its usage ⌈ The Rte_IrTrigger API may only be used by the runnable that contains the corresponding InternalTriggeringPoint. ⌋

Description: The Rte_IrTrigger triggers an execution for all runnables whose InternalTriggerOccurredEvent is associated to the InternalTriggeringPoint.

Return Value: None in case of signature without queuing support.

[SWS_Rte_06721] ⌈ The Rte_Trigger API shall return the following values:

- RTE_E_OK if the trigger was successfully queued or if no queue is configured
- RTE_E_LIMIT if the trigger was not queued because the maximum queue size is already reached.

in the case of signature with queuing support. ⌋*(SRS_Rte_00235)*

Notes: None.

5.6.33 Rte_IFeedback

Purpose: Provide access to acknowledgement notifications for implicit sender receiver communication and to pass error notification to senders.

Signature: **[SWS_Rte_07367]** ⌈

```
Std_ReturnType
Rte_[Byps_]IFeedback_<re>_<p>_<o> ([IN RTE_Instance <instance>])
```

Where `<re>` is the runnable entity name, `<p>` the port name and `<o>` the VariableDataPrototype within the sender-receiver interface categorizing the port. `[Byps_]` is an optional infix used when component wrapper method for bypass support is enabled for the related software component type (See chapter 4.9.2). ⌋*(SRS_BSW_00310, SRS_Rte_00122, SRS_Rte_00129, SRS_Rte_00185)*

Existence: Note: according to [SWS_Rte_01283], acknowledgment is enabled for a provided VariableDataPrototype by the existence of a TransmissionAcknowledgementRequest in the SenderComSpec.

[SWS_Rte_07646] ⌈ An Rte_IFeedback API shall be created for a provided VariableDataPrototype if acknowledgment is enabled and the RunnableEntity has a VariableAccess in the dataWriteAccess role referring to this VariableDataPrototype. ⌋*(SRS_Rte_00122, SRS_Rte_00129, SRS_Rte_00185)*

[SWS_Rte_07647] ⌈ An Rte_IFeedback API shall be created for a provided VariableDataPrototype if acknowledgment is enabled and a DataWriteCompletedEvent references the RunnableEntity as well as the VariableAccess which in turn references the VariableDataPrototype. ⌋*(SRS_Rte_00122, SRS_Rte_00129, SRS_Rte_00185)*

[SWS_Rte_07648] ⌈ If acknowledgment is enabled for a provided VariableDataPrototype and a DataWriteCompletedEvent references a runnable entity as well as the VariableAccess which

in turn references the VariableDataPrototype, the runnable entity shall be activated when the transmission acknowledgment occurs or when a timeout was detected by the RTE. See [SWS_Rte_07379].
⌋*(SRS_Rte_00122, SRS_Rte_00129, SRS_Rte_00185)*

[constr_9000] Rte_IFeedback API may only be used by the RunnableEntitys that describe its usage ⌈ The Rte_IFeedback API shall only be used by a RunnableEntity that either has a VariableAccess in the dataWriteAccess role referring to the VariableDataPrototype or is triggered by a DataWriteCompletedEvent referring to the VariableAccess which in turn references the VariableDataPrototype. ⌋

Description: The Rte_IFeedback API takes no parameters other than the instance handle – the return value is used to indicate the acknowledgment status to the caller.

The Rte_IFeedback API applies only to implicit sender-receiver communication.

The Rte_IFeedback API provides access to the transmission feedback of the data elements, declared as sent by a runnable using a VariableAccess in the dataWriteAccess role, and sent after the previous invocation of the runnable. The API function is guaranteed to be have constant execution time and therefore can also be used within category 1A runnable entities.

The required consistency access by a runnable can be provided by copying of the status before the execution of the runnable so that it cannot be modified by the RTE during the lifetime of the runnable entity.

Return Value: The return value is used to indicate the "status" status and errors detected by the RTE during execution of the Rte_IFeedback call.

- **[SWS_Rte_07374]** ⌈ RTE_E_NO_DATA – No acknowledgments or error notifications were received from COM when the runnable entity was started. ⌋*(SRS_Rte_00094, SRS_Rte_00122, SRS_Rte_00129, SRS_Rte_00185)*
- **[SWS_Rte_07375]** ⌈ RTE_E_COM_STOPPED – (Inter-ECU communication only) The last transmission was rejected (when the local buffer was sent), with an RTE_E_COM_STOPPED return code or an error notification was received from COM before any timeout notification. ⌋*(SRS_Rte_00094, SRS_Rte_00122, SRS_Rte_00129, SRS_Rte_00185)*
- **[SWS_Rte_07650]** ⌈ RTE_E_TIMEOUT – (Inter-ECU only) A timeout notification was received from COM before

any error notification. ⌋(*SRS_Rte_00094, SRS_Rte_00122, SRS_Rte_00129, SRS_Rte_00185*)

- **[SWS_Rte_07376]** ⌈ `RTE_E_TRANSMIT_ACK` – A transmission acknowledgment was received. This error code is valid for both inter-ECU and intra-ECU communication. ⌋(*SRS_Rte_00094, SRS_Rte_00122, SRS_Rte_00129, SRS_Rte_00185*)
- **[SWS_Rte_07660]** ⌈ `RTE_E_UNCONNECTED` – Indicates that the sender port is not connected. ⌋(*SRS_Rte_00094, SRS_Rte_00122, SRS_Rte_00129, SRS_Rte_00185, SRS_Rte_00139*)

The `RTE_E_NO_DATA`, `RTE_E_TRANSMIT_ACK` and `RTE_E_UNCONNECTED` return values are not considered to be an error but rather indicates correct operation of the API call.

[SWS_Rte_07651] ⌈ The initial return value of the `Rte_IFeedback` API, when the runnable entity is executed before any attempt to write some data shall be `RTE_E_TRANSMIT_ACK`. ⌋(*SRS_Rte_00094, SRS_Rte_00122, SRS_Rte_00129, SRS_Rte_00185*)

[SWS_Rte_08074] ⌈ In case of multiple faults during a call of `Rte_IFeedback` the resulting return value shall be derived according to the following priority rules (highest priority first): (1) `RTE_E_UNCONNECTED`, (2) `RTE_E_IN_EXCLUSIVE_AREA`, (3) `RTE_E_TIMEOUT`, (4) `RTE_E_COM_STOPPED`, (5) `RTE_E_NO_DATA`, (6) `RTE_E_TRANSMIT_ACK`. ⌋(*SRS_Rte_00122*)

Notes: See the notes for the `Rte_Feedback` API in section 5.6.8.

5.6.34 Rte_IsUpdated

Purpose: Provide access to the update flag for an explicit receiver.

Signature: **[SWS_Rte_07390]** ⌈

```
boolean
Rte_[Byps_]IsUpdated_<p>_<o> ([IN RTE_Instance <instance>])
```

Where `<p>` is the port name and `<o>` the `VariableDataPrototype` within the sender-receiver interface categorizing the port. `[Byps_]` is an optional infix used when component wrapper method for bypass support is enabled for the related software component type (See chapter 4.9.2). ⌋(*SRS_BSW_00310, SRS_Rte_00179*)

Existence: **[SWS_Rte_07391]** ⌈ An `Rte_IsUpdated` API shall be created for a required `VariableDataPrototype` if a `RunnableEntity` has a `VariableAccess` in the `dataReceivePointByArgument` or dataReceivePointByValue role referring to the VariableDataPrototype and the enableUpdate attribute is enabled in the NonqueuedReceiverComSpec of the VariableDataPrototype. ⌋(SRS_Rte_00179)

[constr_9034] Rte_IsUpdated API may only be used by the runnable that describe the access to the corresponding data ⌈ The Rte_IsUpdated API may only be used by the runnable that contains the corresponding VariableAccess in the dataReceivePointByArgument or dataReceivePointByValue role. ⌋

Description: The Rte_IsUpdated API takes no parameters other than the instance handle – the return value is used to indicate if the VariableDataPrototype has been updated or not.

The Rte_IsUpdated API applies only to sender-receiver communication.

Return Value: The return value is used to indicate if the VariableDataPrototype has been updated or not.

- **[SWS_Rte_07392]** ⌈ TRUE – DataElement updated since last read. ⌋(*SRS_Rte_00094*, *SRS_Rte_00179*)
- **[SWS_Rte_07393]** ⌈ FALSE – DataElement not updated since last read. ⌋(*SRS_Rte_00094*, *SRS_Rte_00179*)

Notes: None.

5.6.35 Rte_PBCon

Purpose: Provide access to the individual post-build artifacts of a VariationPointProxy for SWCs of a system containing different variants.

Signature: **[SWS_Rte_08066]** ⌈

```
<return>
Rte_[Byps_]PBCon_<vpp> ()
```

Where <vpp> is the shortName of the VariationPointProxy. [Byps_] is an optional infix used when component wrapper method for bypass support is enabled for the related software component type (See chapter 4.9.2). ⌋(*SRS_Rte_00191*)

Existence: **[SWS_Rte_08067]** ⌈ A Rte_PBCon API shall be generated, if a PostBuildVariantCriterion or at least one PostBuildVariantCondition is defined for the VariationPointProxy. ⌋(*SRS_Rte_00191*)

Description: Depending on the category of the VariationPointProxy (see Software Component Template [2]), the Rte_PBCon API provides either access to the PostBuildVariantCriterion or to the result of the evaluation of the PostBuildVariantConditions against the PostBuildVariantCriterion.

Return Value: **[SWS_Rte_08068]** ⌈ For VariationPointProxys of category VALUE the return value of Rte_PBCon shall be an integer value yielding from the VariationPointProxy.postBuildValueAccess.

The return type of Rte_PBCon shall be in this case conform with the ImplementationDataType defined by VariationPointProxy.implementationDataType. ⌋*(SRS_Rte_00191)*

[SWS_Rte_08069] ⌈ For VariationPointProxys of category CONDITION the return value of Rte_PBCon shall be the result of the evaluated expression $PBExp$:$\bigwedge_{PBVarCon}$(VariationPointProxy.postBuildValueAccess $=$ PostBuildVariantCondition.value), where *PBVarCon* is the set of all postBuildVariantConditions of the VariationPointProxy. If a pre-build condition is defined in addition the return value shall be the result of the evaluated expression $PPBExp$:VariationPointProxy.conditionAccess$\bigwedge PBExp$.

The return type of Rte_PBCon shall be in this case the Platform Type boolean. ⌋*(SRS_Rte_00191)*

Notes: **[SWS_Rte_08070]** ⌈ For VariationPointProxys of category CONDITION that are using both conditionAccess and postBuildVariantCondition the RTE shall ensure in Rte_PBCon that pre-build conditions have precedence over post-build conditions. ⌋*(SRS_Rte_00191)*

More details regarding Rte_PBCon API can be found in section 4.7.5.

5.7 Runnable Entity Reference

An AUTOSAR component defines one or more “runnable entities”. A runnable entity is a piece of code with a single entry point and an associate set of data. A software-component description provides definitions for each runnable entity within the software-component.

For components implemented using C or C++ the entry point of a runnable entity is implemented by a function with global scope defined within a software-component’s source code. The following sections consider the function signature and prototype.

5.7.1 Signature

The definition of all runnable entities, whatever the `RTEEvent` that triggers their execution, follows the same basic form.

[SWS_Rte_01126] ⌈

```
<void|Std_ReturnType> [Byps_]<prefix><name>([IN Rte_Instance <instance>],
      [IN Rte_ActivatingEvent_<name> <activation>],
                    [role parameters])
```

Where `<name>` [8] is the `symbol` describing the runnable's entry point and `<prefix>` is the optional `SymbolProps.symbol` attribute of the `AtomicSwComponentType` owning the `RunnableEntity`, i.e. `<prefix>` will only appear if the attribute `SymbolProps.symbol` exists. The usage of `Rte_ActivatingEvent` is optional and defined in [SWS_Rte_08051]. The definition of the *role parameters* is defined in Section 5.7.3. `[Byps_]` is an optionnal infix used when component wrapper method for bypass support is enabled for the related software component type (See chapter 4.9.2).
⌋*(SRS_Rte_00031, SRS_Rte_00011, SRS_Rte_00238)*

Section 5.2.6.4 contains details on a recommended naming conventions for runnable entities based on the `RTEEvent` that triggers the runnable entity. The recommended naming convention makes explicit the functions that implement runnable entities as well as clearly associating the runnable entity and the applicable data element or operation.

5.7.2 Entry Point Prototype

The RTE determines the required role parameters, and hence the prototype of the entry point, for a runnable entity based on information in the input information. The entry point defined in the component source *must* be compatible with the parameters passed by the RTE when the runnable entity is triggered by the RTE and therefore the RTE generator is required to emit a prototype for the function.

[SWS_Rte_01132] ⌈ The RTE generator shall emit a prototype for the runnable entity's entry point in the *Application Header File*. ⌋*(SRS_Rte_00087, SRS_Rte_00051, SRS_Rte_00031)*

The prototype for a function implementing the entry point of a runnable entity is emitted for both "RTE Contract" and "RTE Generation" phases. The function name for the prototype is the runnable entity's entry point. The prototype of the entry point function includes the runnable entity's instance handle and its role parameters, see Figure 5.2.

[8] Runnable entities have two "names" associated with them in the AUTOSAR Software Component Template; the runnable's identifier and the entry point's symbol. The identifier is used to reference the runnable entity within the input data and the symbol used within code to identify the runnable's implementation. In the context of a prototype for a runnable entity, "name" is the runnable entity's entry point symbol.

[SWS_Rte_07194] ⌈ The RTE Generator shall wrap each RunnableEntity's *Entry Point Prototype* in the *Application Header File* with the *Memory Mapping* and *Compiler Abstraction* macros.

```
#define [Byps_]<c>_START_SEC_<sadm>
#include "[Byps_]<c>_MemMap.h"

FUNC(<void|Std_ReturnType>, <c>_<sadm>)  [Byps_]<prefix><name> (
                            [IN Rte_Instance <instance>],
                            [IN Rte_ActivatingEvent_<name> <activation>],
                            [role parameters]);

#define [Byps_]<c>_STOP_SEC_<sadm>
#include "[Byps_]<c>_MemMap.h"
```

where <c> is the shortName of the software component type,

<sadm> is the shortName of the referred swAddrMethod.

<prefix> is the optional SymbolProps.symbol attribute of the AtomicSwComponentType owning the RunnableEntity, i.e. <prefix> will only appear if the attribute SymbolProps.symbol exists.

<name> is the attribute symbol describing the RunnableEntity's entry point.

The usage of Rte_ActivatingEvent is optional and defined in [SWS_Rte_08051]. The definition of the *role parameters* is defined in Section 5.7.3. The Memory Mapping macros could wrap several *Entry Point Prototype* if these are referring to the same swAddrMethod. If RunnableEntity does not refer a swAddrMethod the <sadm> is set to default CODE. [Byps_] is an optionnal infix used when component wrapper method for bypass support is enabled for the related software component type (See chapter 4.9.2). ⌋*(SRS_Rte_00148, SRS_Rte_00149, SRS_Rte_00238, SRS_Rte_00011)*

[SWS_Rte_06531] ⌈ The RTE Generator shall wrap each *Entry Point Prototype* in the *Application Header File* of a variant existent RunnableEntity if the variability shall be implemented. ⌋*(SRS_Rte_00201)*

```
#if (<condition>)

<Entry Point Prototype>

#endif
```

where condition is the *Condition Value Macro* of the VariationPoint relevant for the variant existence of the RunnableEntity (see table 4.18), Entry Point Prototype is the code according an invariant *Entry Point Prototype* (see also [SWS_Rte_01131], [SWS_Rte_07177], [SWS_Rte_02512], [SWS_Rte_01133], [SWS_Rte_01359], [SWS_Rte_01166], [SWS_Rte_01135], [SWS_Rte_01137], [SWS_Rte_07207], [SWS_Rte_07208], [SWS_Rte_07379]).

[SWS_Rte_01016] ⌈ The function implementing the entry point of a runnable entity shall define an instance handle as the first formal parameter if and only if the

software component's supportsMultipleInstantiation attribute is set to TRUE.
⌋(*SRS_Rte_00011, SRS_Rte_00031*)

The RTE will ensure that when the runnable entity is triggered the instance handle parameter indicates the correct component instance. The remaining parameters passed to the runnable entity depend on the RTEEvent that triggers execution of the runnable entity.

Due to the global name space of a C Linker Locater symbols of RunnableEntitys have to be unique in the ECU. When AtomicSwComponentTypes of several vendors are integrated in the same ECU name clashes might occur if the same symbol is accidentally used twice. To ease the dissolving of name clashes the RTE supports an abstraction of the RunnableEntity symbol in the implementation of the software component.

[SWS_Rte_06713] ⌈ The RTE generator shall emit for each RunnableEntity a define for a symbolic name of the RunnableEntity.

```
#define RTE_RUNNABLE_<name> <prefix><symbol>
```

where <name> is the shortName of the RunnableEntity,

<prefix> is the optional SymbolProps.symbol attribute of the AtomicSwComponentType owning the RunnableEntity.

<symbol> is the attribute symbol describing the RunnableEntity's entry point.

⌋(*SRS_Rte_00087, SRS_Rte_00051, SRS_Rte_00031*)

This symbolic name of the RunnableEntity can be used as follows in the software component implementation.

Example 5.29

For software component "'HugeSwc"' with a runnable "'FOO"' where the SymbolProps.symbol is set to "'TinySwc"' the *Application Header File* contains the definition:

```
/* Application Header File of HugeSwc*/
#define RTE_RUNNABLE_FOO TinySwcfoo
```

This can be used in the software components c file for the definition of the runnable:

```
/* software component c file */
RTE_RUNNABLE_FOO()
{
  /* The algorithm of foo */
  return;
}
```

A change of the SymbolProps.symbol valued would have no effect on the c implementation of the software component but it would change the contract and the used labels in a object code delivery.

In case that the RunnableEntity is mapped to BswModuleEntity the RTE Generator has to additionally respect the definitions in 6.3.2.3.4.

5.7.3 Role Parameters

The *role parameters* are optional and their presence and types depend on the RTEEvent that triggers the execution of the runnable entity. The role parameters that are necessary for each triggering RTEEvent are defined in Section 5.7.5.

[SWS_Rte_06703] ⌈ The RTE Generator shall name role parameters according to the value of the symbol attribute of RunnableEntityArguments if RunnableEntityArguments are defined for the related RunnableEntity and if no mapping to a BswModuleEntry is defined. ⌋*(SRS_Rte_00087)*

[SWS_Rte_06704] ⌈ The RTE Generator shall name role parameters according to the shortName of the SwServiceArgs of the mapped BswModuleEntry if a mapping of the RunnableEntity to a BswModuleEntry is defined. ⌋*(SRS_Rte_00087)*

Please note that RunnableEntityArguments defined for a RunnableEntity which is mapped to a BswModuleEntry are irrelevant.

[SWS_Rte_06705] ⌈ The RTE Generator shall generate nameless role parameters if neither RunnableEntityArguments nor a mapping to a BswModuleEntry is defined for the RunnableEntity. ⌋*(SRS_Rte_00087)*

Further details about the mapping of RunnableEntitys and BswModuleEntry can be found section "'Synchronization with a Corresponding SWC"' of the document [9]

5.7.4 Return Value

A function in C or C++ is required to have a return type. The RTE only uses the function return value to return application error codes of a server operation.

[SWS_Rte_01130] ⌈ A function implementing a runnable entity entry point shall only have the return type Std_ReturnType, if the runnable entity represents a server operation and the AUTOSAR interface description of that client server communication lists potential application errors. All other functions implementing a runnable entity entry point shall have a return type of void. ⌋*(SRS_Rte_00124, SRS_Rte_00031)*

Note: If the potential application errors include RTE_E_OK, this shall also lead to a return type of Std_ReturnType.

[constr_9045] The upper two bits of the of the server return value are reserved ⌈ Only the least significant six bit of the return value of a server runnable shall be used by the application to indicate an error. The upper two bit shall be zero. ⌋

See also [SWS_Rte_02573].

5.7.5 Triggering Events

The RTE is the *sole* entity that can trigger the execution of a runnable entity. The RTE triggers runnable entities in response to different `RTEEvent`s.

The most basic `RTEEvent` that can trigger a runnable entity is the `TimingEvent` that causes a runnable entity to be periodically triggered by the RTE. In contrast, the remaining `RTEEvent`s that can trigger runnable entities all occur as a result of communication activity or as a result of mode switches.

The following subsections describe the conditions that can trigger execution of a runnable entity. For each triggering event the signature of the function (the "entry point") that implements the runnable entity is defined. The signature definition includes two classes of parameters for each function;

1. The instance handle – the parameter type is always `Rte_Instance`. ([SWS_Rte_01016])
2. The role parameters – used to pass information required by the runnable entity as a consequence of the triggering condition. The presence (and number) of role parameters depends solely on the triggering condition.

5.7.5.1 TimingEvent

Purpose: Trigger a runnable entity periodically at a rate defined within the software-component description.

Signature: **[SWS_Rte_01131]** ⌈

```
void <name>([IN Rte_Instance <instance>])
```

⌋*(SRS_Rte_00072)*

5.7.5.2 BackgroundEvent

Purpose: A recurring `RTEEvent` which is used to perform background activities. It is similar to a `TimingEvent` but has no fixed time period and is activated only with low priority.

Signature: **[SWS_Rte_07177]** ⌈

```
void <name>([IN Rte_Instance <instance>])
```

⌋*(SRS_Rte_00072)*

5.7.5.3 SwcModeSwitchEvent

Purpose: Trigger of a runnable entity as a result of a mode switch. See also sections 4.4.4 and 4.4.7 for reference.

Signature: **[SWS_Rte_02512]** ⌈

```
void <name>([IN Rte_Instance <instance>])
```

⌋*(SRS_Rte_00072, SRS_Rte_00143)*

5.7.5.4 AsynchronousServerCallReturnsEvent

Purpose: Triggers a runnable entity used to “collect” the result and status information of an asynchronous client-server operation.

Signature: **[SWS_Rte_01133]** ⌈

```
void <name>([IN Rte_Instance <instance>])
```

⌋*(SRS_Rte_00072, SRS_Rte_00029, SRS_Rte_00079)*

Notes: The runnable entity triggered by an `AsynchronousServerCallReturnsEvent` `RTEEvent` should use the `Rte_Result` API to actually receive the result and the status of the server operation.

5.7.5.5 DataReceiveErrorEvent

Purpose: Triggers a runnable entity used to “collect” the error status of a `data element` with “data” semantics on the receiver side.

Signature: **[SWS_Rte_01359]** ⌈

```
void <name>([IN Rte_Instance <instance>])
```

⌋*(SRS_Rte_00072, SRS_Rte_00029, SRS_Rte_00079)*

Notes: The runnable entity triggered by a `DataReceiveErrorEvent` `RTEEvent` should use the `Rte_IStatus` API to actually read the status.

5.7.5.6 OperationInvokedEvent

Purpose: An `RTEEvent` that causes the RTE to trigger a runnable entity whose entry point provides an implementation for a client-server operation. This event occurs in response to a received request from a client to execute the operation.

Signature: **[SWS_Rte_01166]** ⌈

```
<void|Std_ReturnType> <name>
                    ([IN Rte_Instance <instance>],
```

```
[IN             <portDefArg 1>, ...
 IN             <portDefArg n>],
[IN|INOUT|OUT] <param 1>, ...
[IN|INOUT|OUT] <param n>)
```

Where <portDefArg 1>, ..., <portDefArg n> represent the port-defined argument values (see Section 4.3.2.4) and <param 1>, ... <param n> indicates the operation IN, INOUT and OUT parameters. ⌋*(SRS_Rte_00029, SRS_Rte_00079, SRS_Rte_00072, SRS_Rte_00152)*

The data type of each port defined argument is taken from the software component template, as defined in valueType.

Note that the port-defined argument values are optional, depending upon the server's internal behavior.

[SWS_Rte_07023] ⌈ The operation parameters <param 1>, ... <param n> are the specified ArgumentDataPrototypes of the ClientServerOperation that is associated with the OperationInvokedEvent. The operation parameters shall be ordered according to the ClientServerOperation's ordered list of the ArgumentDataPrototypes. ⌋*(SRS_Rte_00029, SRS_Rte_00079, SRS_Rte_00072)*

[SWS_Rte_07024] ⌈ If the ServerArgumentImplPolicy is set to useArgumentType the data type of the <param> is derived from the ArgumentDataPrototype's ImplementationDataType. ⌋*(SRS_Rte_00029, SRS_Rte_00079, SRS_Rte_00072)*

In case of [SWS_Rte_07024] the RunnableEntitys parameter are equally typed as the parameter for the Rte_Call API described in section 5.2.6.5

[SWS_Rte_07025] ⌈ If the ServerArgumentImplPolicy is set to useArrayBaseType the data type of the <param> is derived from the ArgumentDataPrototype's ImplementationDataTypeElement specifying the base type of the array. ⌋*(SRS_Rte_00029, SRS_Rte_00079, SRS_Rte_00072)*

ServerArgumentImplPolicy is set to useArrayBaseType is only applicable in case of ArgumentDataPrototype's which data type is of category ARRAY.

Please note that [SWS_Rte_07025] results in the same C Data Type as described in [SWS_Rte_07024] since the Rte API is typed with the base type of the array as specified in [SWS_Rte_05107], [SWS_Rte_05108] and [SWS_Rte_05109], in section 5.2.6.5.

[SWS_Rte_07026] ⌈ The RTE-Generator shall reject configurations violating [constr_1297]. ⌋*(SRS_Rte_00029, SRS_Rte_00079, SRS_Rte_00072, SRS_Rte_00018)*

[SWS_Rte_07027] ⌈ If the `ServerArgumentImplPolicy` is set to `useVoid` the data type of the `<param>` is set to `void *` for any kind of data type. ⌋*(SRS_Rte_00029, SRS_Rte_00079, SRS_Rte_00072)*

[SWS_Rte_08800] ⌈ It is considered an invalid configuration if `ServerArgumentImplPolicy` uses `void` in case of primitive `IN` arguments. See [constr_1286] in Software Component Template specification. ⌋*(SRS_Rte_00079, SRS_Rte_00018)*

[SWS_Rte_05193] ⌈ If the `serverArgumentImplPolicy` is set to `useArrayBaseType` or `useVoid` the RTE shall cast the arguments passed by `Rte_Call()` and `Rte_Result()` to the data types defined by the runnable entity prototype. ⌋*(SRS_Rte_00029, SRS_Rte_00079, SRS_Rte_00072)*

Return Value: If the AUTOSAR interface description of the client server communication lists possible error codes, these are returned by the function using the return type `Std_ReturnType`. If no error codes are defined for this interface, the return type shall be `void` (see [SWS_Rte_01130]).

This means that even if a runnable entity implementing a server "only" returns `E_OK`, application errors have to be defined. Else the return types do not match.

5.7.5.7 DataReceivedEvent

Purpose: A runnable entity triggered by the RTE to receive and process a signal received on a sender-receiver interface.

Signature: **[SWS_Rte_01135]** ⌈

```
void <name>([IN Rte_Instance <instance>])
```

⌋*(SRS_Rte_00072, SRS_Rte_00028, SRS_Rte_00131, SRS_Rte_00107)*

Notes: The data or event is not passed as an additional parameter. Instead, the previously described reception API should be used to access the data/event. This approach permits the same signature for runnables that are triggered by time (`TimingEvent`) or data reception.

Caution: For intra-ECU communication, the `DataReceivedEvent` is fired after each completed write operation to the shared data. In case of implicit access, write operation is considered to be completed when the runnable ends. While for inter-ECU communication, the DataReceivedEvent is fired by the RTE after a callback from COM due to data reception. Over a physical network, 'data' is commonly transmitted periodically and hence not only will the latency and jitter of DataReceivedEvents vary depending on whether a configuration uses intra or inter-ECU communication, but also the number and frequency of these RTEEvents may change significantly. This means that a TimingEvent should be used to periodically activation of a runnable rather than relying on the periodic transmission of data.

5.7.5.8 DataSendCompletedEvent

Purpose: A runnable entity triggered by the RTE to receive and process transmit acknowledgment notifications.

Signature: **[SWS_Rte_01137]** ⌈

```
void <name>([IN Rte_Instance <instance>])
```

⌋*(SRS_Rte_00072, SRS_Rte_00122, SRS_Rte_00107)*

Notes: The runnable entity triggered by a DataSendCompletedEvent RTEEvent should use the Rte_Feedback API to actually receive the status of the acknowledgment.

5.7.5.9 ModeSwitchedAckEvent

Purpose: A runnable entity triggered by the RTE to receive and process mode switched acknowledgment notifications.

Signature: **[SWS_Rte_02758]** ⌈

```
void <name>([IN Rte_Instance <instance>])
```

⌋*(SRS_Rte_00072, SRS_Rte_00122, SRS_Rte_00107)*

Notes: The runnable entity triggered by an ModeSwitchedAckEvent should use the Rte_SwitchAck API to actually receive the status of the acknowledgment.

5.7.5.10 SwcModeManagerErrorEvent

Purpose: A runnable entity triggered by the RTE to react on errors occurring during mode handling.

Signature: **[SWS_Rte_06771]** ⌈

```
void <name>([IN Rte_Instance <instance>])
```

⌋(SRS_Rte_00072, SRS_Rte_00122, SRS_Rte_00107)

Notes: –

5.7.5.11 ExternalTriggerOccurredEvent

Purpose: A runnable entity triggered by the RTE at the occurrence of an external event.

Signature: **[SWS_Rte_07207]** ⌈

```
void <name>([IN Rte_Instance <instance>])
```

⌋(SRS_Rte_00162, SRS_Rte_00072)

Notes: –

5.7.5.12 InternalTriggerOccurredEvent

Purpose: A runnable entity triggered by the RTE by an inter runnable trigger.

Signature: **[SWS_Rte_07208]** ⌈

```
void <name>([IN Rte_Instance <instance>])
```

⌋(SRS_Rte_00163, SRS_Rte_00072)

Notes: –

5.7.5.13 DataWriteCompletedEvent

Purpose: A runnable entity triggered by the RTE to receive and process transmit acknowledgment notifications for implicit communication.

Signature: **[SWS_Rte_07379]** ⌈

```
void <name>([IN Rte_Instance <instance>])
```

⌋(SRS_Rte_00072, SRS_Rte_00122, SRS_Rte_00185)

Notes: The runnable entity triggered by a DataWriteCompletedEvent RTEEvent should use the Rte_IFeedback API to actually receive the status of the acknowledgment.

5.7.5.14 InitEvent

Purpose: A runnable entity triggered by the RTE for initialization.

Signature: **[SWS_Rte_06748]** ⌈

```
void <name>([IN Rte_Instance <instance>])
```

⌋(SRS_Rte_00072, SRS_Rte_00240)

Notes: The runnable entity triggered by a InitEvent RTEEvent is supposed to be used for initialization purposes, i.e. for starting and restarting a partition. It is not guaranteed that all RunnableEntitys referenced by this InitEvent are executed before the 'regular' RunnableEntitys are executed for the first time.

5.7.6 Reentrancy

A runnable entity is declared within a software-component type. The RTE ensures that concurrent activation of same instance of a runnable entity is only allowed if the runnables attribute "canBeInvokedConcurrently" is set to TRUE (see Section 4.2.6).

When a software-component is multiple instantiated each separate instance has its own instance of the runnable entities in the software-component. Whilst instances of a software-component are independent, the runnable entities instances share the same code ([SWS_Rte_03015]).

Example 5.30

Consider a component `c1` with runnable entity `re1` and entry point `ep` that is instantiated twice on the same ECU.

The two instances of `c1` each has a separate *instance* of `re1`. Software-component instances are scheduled independently and therefore each instance of `re1` could be concurrently executing `ep`.

The potential for concurrent execution of runnable entities when multiple instances of a software-component are created means that each entry point should be reentrant.

5.8 RTE Lifecycle API Reference

This section documents the API functions used to start and stop the RTE. RTE Lifecycle API functions are not invoked from AUTOSAR software-components – instead they are invoked from other basic software module(s).

5.8.1 Rte_Start

Purpose: Initialize the RTE itself.

Signature: **[SWS_Rte_02569]** ⌈

```
Std_ReturnType Rte_Start(void)
```

⌋(*SRS_BSW_00310*, *SRS_Rte_00116*)

Existence: **[SWS_Rte_01309]** ⌈ The Rte_Start API is always created. ⌋(*SRS_Rte_00051*)

Description: Rte_Start is intended to allocate and initialise system resources and communication resources used by the RTE.

[constr_9035] Rte_Start shall be called only once ⌈ Rte_Start shall be called only once by the EcuStateManager from trusted OS context on a core after the basic software modules required by RTE are initialized. ⌋

These modules include:

- OS
- COM
- memory services

The Rte_Start API shall not be invoked from AUTOSAR software components.

[constr_9036] Rte_Start API may only be used after call of SchM_Init ⌈ The Rte_Start API may only be used after the *Basic Software Scheduler* is initialized (after termination of the SchM_Init). ⌋

[constr_9037] Rte_Start API shall be called on every core ⌈ The Rte_Start API shall be called on every core that hosts AUTOSAR software-components of the ECU. ⌋

[SWS_Rte_02585] ⌈ Rte_Start shall return within finite execution time – it must not enter an infinite loop. ⌋(*SRS_Rte_00116*)

Rte_Start may be implemented as a function or a macro.

Return Value: If the allocation of a resource fails, Rte_Start shall return with an error.

- **[SWS_Rte_01261]** ⌈ RTE_E_OK – No error occurred. ⌋(*SRS_Rte_00094*)
- **[SWS_Rte_01262]** ⌈ RTE_E_LIMIT – An internal limit has been exceeded. The allocation of a required resource has failed. ⌋(*SRS_Rte_00094*)

Notes: Rte_Start is declared in the lifecycle header file Rte_Main.h. The initialization of AUTOSAR software-components takes place after the termination of Rte_Start and is triggered by a mode change event on entering run state.

5.8.2 Rte_Stop

Purpose: finalize the RTE itself

Signature: **[SWS_Rte_02570]** ⌈

```
Std_ReturnType Rte_Stop(void)
```

⌋(*SRS_Rte_00116*)

Existence: **[SWS_Rte_01310]** ⌈ The Rte_Stop API is always created. ⌋(*SRS_Rte_00051*)

Description: Rte_Stop is used to finalize the RTE on the core it is called. This service releases all system and communication resources allocated by the RTE on that core.

[constr_9038] Rte_Stop shall be called before BSW shutdown ⌈ Rte_Stop shall be called by the EcuStateManager before the basic software modules required by RTE are shut down. ⌋

These modules include:

- OS
- COM
- memory services

Rte_Stop shall be called from trusted context and not by an AUTOSAR software component.

[SWS_Rte_02584] ⌈ Rte_Stop shall return within finite execution time. ⌋(*SRS_Rte_00116*)

Rte_Stop may be implemented as a function or a macro.

Return Value:

- **[SWS_Rte_01259]** ⌈ RTE_E_OK – No error occurred. ⌋(*SRS_Rte_00094*)
- **[SWS_Rte_01260]** ⌈ RTE_E_LIMIT – a resource could not be released. ⌋(*SRS_Rte_00094*)

Notes: Rte_Stop is declared in the lifecycle header file Rte_Main.h.

5.8.3 Rte_PartitionTerminated

Purpose: Indicate to the RTE that a partition is going to be terminated, and the communication with the Partition shall be ignored.

Signature: **[SWS_Rte_07330]** ⌈

```
void Rte_PartitionTerminated_<PID>(void)
```

⌋(*SRS_Rte_00223*)

Where <PID> is the name of the EcucPartition according to the ECU Configuration Description [5].

Existence: **[SWS_Rte_07331]** ⌈ An Rte_PartitionTerminated API shall be created for every Partition. ⌋*(SRS_Rte_00223)*

Description: Rte_PartitionTerminated is intended to notify the RTE that a given partition is terminated or is being restarted.

[constr_9039] Rte_PartitionTerminated shall be called only once ⌈ Rte_PartitionTerminated shall be called only once by the ProtectionHook. ⌋

Rte_PartitionTerminated may be implemented as a function or a macro.

[SWS_Rte_07334] ⌈ The treatments in Rte_PartitionTerminated shall be restricted to the ones allowed in the context of a ProtectionHook. ⌋*(SRS_Rte_00223)*

Since Rte_PartitionTerminated is called from the ProtectionHook context, it should be as fast as possible. Moreover, it cannot be assumed any more that partition local data including RTE data is consistent. Therefore, actions should be limited to setting a flag. Actual cleanup needs to be deferred to another task.

The notification provided by Rte_PartitionTerminated can be used later by the RTE to immediately return an error status when SW-Cs of other partitions tries to communicate with the stopped partition. See [SWS_Rte_02710] and [SWS_Rte_02709].

[SWS_Rte_07335] ⌈ Terminating an already terminated Partition shall be ignored. ⌋*(SRS_Rte_00223)*

Return Value: None.

Notes: Rte_PartitionTerminated is declared in the lifecycle header file Rte_Main.h.

5.8.4 Rte_PartitionRestarting

Purpose: Indicate to the RTE that a Partition is going to be restarted and that the communication with the Partition shall be ignored.

Signature: **[SWS_Rte_07620]** ⌈

```
void Rte_PartitionRestarting_<PID>(void)
```

Where <PID> is the name of the EcucPartition according to the ECU Configuration Description [5]. ⌋*(SRS_Rte_00223)*

Existence: **[SWS_Rte_07619]** ⌈ An Rte_PartitionRestarting API shall be created for any Partition which can be restarted (i.e. a Partition whose PartitionCanBeRestarted parameter is enabled). ⌋*(SRS_Rte_00223)*

Description: Rte_PartitionRestarting is intended to notify the RTE that a given partition is being restarted. As Rte_PartitionTerminated, Rte_PartitionRestarting indicates that the communication with the partition shall be ignored, but in case of Rte_PartitionRestarting, the partition may be restarted later in the ECU lifecycle.

[constr_9040] Rte_PartitionRestarting shall be called only onc ⌈ Rte_PartitionRestarting shall be called only once by the ProtectionHook. ⌋

Rte_PartitionRestarting may be implemented as a function or a macro.

[SWS_Rte_07617] ⌈ The treatments in Rte_PartitionRestarting shall be restricted to the ones allowed in the context of a ProtectionHook. ⌋*(SRS_Rte_00223)*

Since Rte_PartitionRestarting is called from the ProtectionHook context, it should be as fast as possible. It should be limited to setting a flag. Actual cleanup should be deferred to another task.

[SWS_Rte_07622] ⌈ Restarting an already terminated Partition or restarting a Partition during an ongoing restart shall be ignored. ⌋*(SRS_Rte_00223)*

Return Value: None.

Notes: Rte_PartitionRestarting is declared in the lifecycle header file Rte_Main.h.

5.8.5 Rte_RestartPartition

Purpose: Initialize the RTE resources allocated for a partition.

Signature: **[SWS_Rte_07188]** ⌈

```
Std_ReturnType Rte_RestartPartition_<PID>(void)
```

Where <PID> is the name of the EcucPartition according to the ECU Configuration Description [5]. ⌋*(SRS_Rte_00224)*

Existence: **[SWS_Rte_07336]** ⌈ An Rte_RestartPartition API shall be created for any Partition which can be restarted (i.e. a Partition whose PartitionCanBeRestarted parameter is enabled). ⌋*(SRS_Rte_00224)*

Description: Rte_RestartPartition is intended to notify the RTE that a given partition will be restarted.

[constr_9041] Rte_RestartPartition shall be called from RestartTask ⌈ Rte_RestartPartition shall be called only in the context of the RestartTask of the given partition. ⌋

[SWS_Rte_07338] ⌈ Rte_RestartPartition shall return within finite execution time – it must not enter an infinite loop. ⌋*(SRS_Rte_00224)*

Rte_RestartPartition may be implemented as a function or a macro.

[SWS_Rte_07339] ⌈ The Rte_RestartPartition shall restore an initial RTE environment for the partition and re-activate communication with this partition. ⌋*(SRS_Rte_00224)*

This includes:

- signal initial values,
- modes,
- queued events,
- sequence counters.

[SWS_Rte_07340] ⌈ Rte_RestartPartition shall be ignored if the given partition was not stopped before (with Rte_PartitionTerminated or Rte_PartitionRestarting). ⌋*(SRS_Rte_00224)*

Return Value: If the allocation of a resource fails, Rte_RestartPartition shall return with an error.

- **[SWS_Rte_07341]** ⌈ RTE_E_OK – No error occurred. ⌋*(SRS_Rte_00224)*
- **[SWS_Rte_07342]** ⌈ RTE_E_LIMIT – An internal limit has been exceeded. The allocation of a required resource has failed. ⌋*(SRS_Rte_00224)*

Notes: Rte_RestartPartition is declared in the lifecycle header file Rte_Main.h.

5.8.6 Rte_Init

Purpose: Schedules RunnableEntitys for initialization purpose.

Signature: **[SWS_Rte_06749]** ⌈

```
void Rte_Init_<InitContainer>(void)
```

Where <InitContainer> is the short name of the RteInitializationRunnableBatch container. ⌋(*SRS_Rte_00240*)

Existence: **[SWS_Rte_06750]** ⌈ An Rte_Init API shall be created for each RteInitializationRunnableBatch container. ⌋(*SRS_Rte_00240*)

Description: Rte_Init is intended schedule RunnableEntitys for initialization purpose which are mapped to the related RteInitializationRunnableBatch container.

[SWS_Rte_06751] ⌈ An Rte_Init API shall invoke the RunnableEntitys which are associated with an RTEEvent mapped to the related RteInitializationRunnableBatch container in the order defined by the RtePositionInTask parameters. ⌋(*SRS_Rte_00240*)

[SWS_Rte_06752] ⌈ Rte_Init shall return within finite execution time – it must not enter an infinite loop. ⌋(*SRS_Rte_00240*)

[SWS_Rte_06753] ⌈ Rte_Init shall be implemented as a function. ⌋(*SRS_Rte_00240*)

[constr_9060] Rte_Init API may only be used after call of Rte_Start ⌈ The Rte_Init API may only be used after the *RTE* is initialized (after termination of the Rte_Start). ⌋

Return Value: none

Notes: Rte_Init is declared in the lifecycle header file Rte_Main.h.

5.8.7 Rte_StartTiming

Purpose: Starts the triggering of recurrent events.

Signature: **[SWS_Rte_06754]** ⌈

```
void Rte_StartTiming(void)
```

⌋(*SRS_Rte_00240*)

Existence: **[SWS_Rte_06755]** ⌈ An Rte_StartTiming API shall be created if any Rte_Init API is created. ⌋(*SRS_Rte_00240*)

Description: Rte_StartTiming API is intended to release the activation of RunnableEntitys triggered by TimingEvents and BackgroundEvents after the last call of a Rte_Init function.

[SWS_Rte_06756] ⌈ Rte_StartTiming API shall release the activation of RunnableEntitys triggered by TimingEvents and BackgroundEvents. ⌋(*SRS_Rte_00240*)

See as well [SWS_Rte_06759] and [SWS_Rte_06760].

[SWS_Rte_06757] ⌈ `Rte_StartTiming` shall return within finite execution time – it must not enter an infinite loop. ⌋*(SRS_Rte_00240)*

[SWS_Rte_06758] ⌈ `Rte_StartTiming` shall be implemented as a function. ⌋*(SRS_Rte_00240)*

[constr_9061] `Rte_StartTiming` API may only be used after call of `Rte_Start` ⌈ The `Rte_StartTiming` API may only be used after the *RTE* is initialized (after termination of the `Rte_Start`). ⌋

Return Value: none

Notes: `Rte_StartTiming` is declared in the lifecycle header file `Rte_Main.h`.

5.9 RTE Call-backs Reference

This section documents the call-backs that are generated by the RTE that must be invoked by other components, such as the communication service, and therefore must have a well-defined name and semantics.

[SWS_Rte_01165] ⌈ A call-back implementation created by the RTE generator is not permitted to block. ⌋*(SRS_Rte_00022)*

Requirement [SWS_Rte_01165] serves to constrain RTE implementations so that all implementations can work with all basic software.

5.9.1 RTE-COM Message Naming Conventions

The COM signals used for communication are defined in the input information provided by Com.

[SWS_Rte_03007] ⌈ The RTE shall initiate an inter-ECU transmission using the COM API with the handle id of the corresponding COM signal for primitive data element `SenderReceiverToSignalMapping`. ⌋*(SRS_Rte_00019)*

[SWS_Rte_03008] ⌈ The RTE shall initiate an inter-ECU transmission using the COM API with the handle id of the corresponding COM signal group for composite data elements or operation arguments `SenderReceiverToSignalGroupMapping`. ⌋*(SRS_Rte_00019)*

5.9.2 Communication Service Call-backs

Purpose: Implement the call-back functions that AutoSAR COM invokes as a result of inter-ECU communication, where:

- A data item/event is ready for reception by a receiver.
- A transmission acknowledgment shall be routed to a sender.
- An operation shall be invoked by a server.
- The result of an operation is ready for reading by a client.

Signature: **[SWS_Rte_03000]** ⌈

```
void <CallbackRoutineName> (void);
```

⌋ *(SRS_Rte_00019)*

Where `<CallbackRoutineName>` is the name of the call-back function (refer to Section 5.9.1 for details on the naming convention).

Description: Prototypes for the call-back `<CallbackRoutineName>` provided by AutoSAR COM.

Return Value: No return value : `void`

5.9.3 Naming convention of Communication Callbacks

In the following table, the naming convention of `<CallBackRoutineName>` are defined:

Calling Situation	callbackRoutineName	Comments
A primitive data item/event is ready for reception by a receiver.	**[SWS_Rte_03001]** ⌈ `Rte_COMCbk_<sn>` ⌋ *(SRS_Rte_00019)*	`<sn>` is the name of the COM signal. This callback function indicates that the signal of the primitive data item/event is ready for reception. Configured in Com: `ComNotification` [ECUC_Com_00498] as part of `ComSignal`

Calling Situation	callbackRoutineName	Comments
A transmission acknowledgment of a primitive data item/event shall be routed to a sender.	**[SWS_Rte_03002]** ⌈ `Rte_COMCbkTAck_<sn>` ⌋*(SRS_Rte_00019, SRS_Rte_00122)*	"TAck" is literal text indicating transmission acknowledgment. This callback function indicates that the signal of the primitive data item/event is already handed over by COM to the PDU router. Configured in Com: `ComNotification` [ECUC_Com_00498] as part of `ComSignal`
A transmission error notificatoin of a primitive data item/event shall be routed to a sender.	**[SWS_Rte_03775]** ⌈ `Rte_COMCbkTErr_<sn>` ⌋*(SRS_Rte_00019, SRS_Rte_00122)*	"TErr" is literal text indicating transmission error. This callback function indicates that an error occurred when the signal of the primitive data item/event was handed over by COM to the PDU router. Configured in Com: `ComErrorNotification` [ECUC_Com_00499] as part of `ComSignal`
A signal invalidation of a primitive data item shall be routed to a receiver.	**[SWS_Rte_02612]** ⌈ `Rte_COMCbkInv_<sn>` ⌋*(SRS_Rte_00019, SRS_Rte_00122)*	"Inv" is literal text indicating signal invalidation. This callback function indicates that COM has received a signal and parsed it as "invalid". Configured in Com: `ComInvalidNotification` [ECUC_COM_00315] as part of `ComSignal`
A signal of a primitive data item is outdated. No new data is available.	**[SWS_Rte_02610]** ⌈ `Rte_COMCbkRxTOut_<sn>` ⌋*(SRS_Rte_00019, SRS_Rte_00147)*	"RxTOut" is literal text indicating reception signal time out. This callback function indicates that the `aliveTimeout` after the last successful reception of the signal of the primitive data item/event has expired (`data element outdated`). Configured in Com: `ComTimeoutNotification` [ECUC_Com_00552] as part of `ComSignal`

Calling Situation	callbackRoutineName	Comments
Transmission has failed and timed out for a primitive data item.	**[SWS_Rte_05084]** ⌈ `Rte_COMCbkTxTOut_<sn>` ⌋(*SRS_Rte_00019, SRS_Rte_00122*)	"TxTOut" is literal text indicating transmission failure and time out. This callback function indicates that the `timeout` of `TransmissionAcknowledgementRequest` for sending the signal of the primitive data item/event has expired. Configured in Com: `ComTimeoutNotification` [ECUC_Com_00552] as part of `ComSignal`
A composite data item/event or the arguments of an operation is ready for reception by a receiver.	**[SWS_Rte_03004]** ⌈ `Rte_COMCbk_<sg>` ⌋(*SRS_Rte_00019*)	`<sg>` is the name of the COM signal group, which contains all the signals of the composite data item/event or an operation. This callback function indicates that the signals of the composite data item/event or the arguments of an operation are ready for reception. Configured in Com: `ComNotification` [ECUC_Com_00498] as part of `ComSignalGroup`
A transmission acknowledgment of a composite data item/event shall be routed to a sender.	**[SWS_Rte_03005]** ⌈ `Rte_COMCbkTAck_<sg>` ⌋(*SRS_Rte_00019, SRS_Rte_00122*)	"TAck" is literal text indicating transmission acknowledgment. This callback function indicates that the signals of the composite data item/event is already handed over by COM to the PDU router. Configured in Com: `ComNotification` [ECUC_Com_00498] as part of `ComSignalGroup`

Calling Situation	callbackRoutineName	Comments
A transmission error notificatoin of a composite data item/event shall be routed to a sender.	**[SWS_Rte_03776]** ⌈ Rte_COMCbkTErr_<sg> ⌋(*SRS_Rte_00019, SRS_Rte_00122*)	"TErr" is literal text indicating transmission error. This callback function indicates that an error occurred when the signal of the composite data item/event was handed over by COM to the PDU router. Configured in Com: ComErrorNotification [ECUC_Com_00499] as part of ComSignalGroup
A signal group invalidation of a composite data item shall be routed to a receiver.	**[SWS_Rte_05065]** ⌈ Rte_COMCbkInv_<sg> ⌋(*SRS_Rte_00019, SRS_Rte_00122*)	"Inv" is literal text indicating signal group invalidation. This callback function indicates that COM has received a signal group and parsed it as "invalid". Configured in Com: ComInvalidNotification [ECUC_Com_00315] as part of ComSignalGroup
A signal group of a composite data item is outdated. No new data is available.	**[SWS_Rte_02611]** ⌈ Rte_COMCbkRxTOut_<sg> ⌋(*SRS_Rte_00019, SRS_Rte_00147*)	"RxTOut" is literal text indicating reception signal time out. This callback function indicates that the aliveTimeout after the last successful reception of the signal group carrying the composite data item has expired (data element outdated). Configured in Com: ComTimeoutNotification [ECUC_Com_00552] as part of ComSignalGroup

Calling Situation	callbackRoutineName	Comments
Transmission has failed and timed out for a composite data item.	**[SWS_Rte_05085]** ⌈ `Rte_COMCbkTxTOut_<sg>` ⌋(*SRS_Rte_00019*, *SRS_Rte_00122*)	"TxTOut" is literal text indicating transmission failure and time out. This callback function indicates that the `timeout` of `TransmissionAcknowledgementRequest` for sending the signal group of the composite data item/event has expired. Configured in Com: `ComTimeoutNotification` [ECUC_Com_00552] as part of `ComSignalGroup`

Table 5.5: RTE COM Callback Function Naming Conventions

Where:

- `<sn>` is a COM signal name.
- `<sg>` is a COM signal group name.

5.9.4 NVM Service Call-backs

5.9.4.1 Rte_SetMirror

Purpose: Warranty the consistency of the `VariableDataPrototype`s contained in a `NvBlockSwComponentType`, when the associated NVM block is read and copied to the `VariableDataPrototype`s storage locations.

Signature: **[SWS_Rte_07310]** ⌈

```
Std_ReturnType
Rte_SetMirror_<b>_<d> (const void *NVMBuffer)
```

⌋(*SRS_Rte_00178*)

Where `<b>` is the `SwComponentPrototype`'s name of the `NvBlockSwComponentType` and `<d>` is the `NvBlockDescriptor` name.

Existence: **[SWS_Rte_07311]** ⌈ An `Rte_SetMirror` API shall be created for each instance of a `NvBlockDescriptor`. ⌋(*SRS_Rte_00178*)

Description: The `Rte_SetMirror` API copies the values of the `VariableDataPrototype`s contained in a `NvBlockDescriptor` from a NVM internal buffer to their locations in the RTE.

[SWS_Rte_07312] ⌈ The `Rte_SetMirror` API shall copy the specified buffer to the `NvBlockDescriptor`'s `ramBlock`, according to the NvBlockDescriptor's NvBlockDataMapping. ⌋(SRS_Rte_00177)

The RTE is responsible for ensuring the data consistency, see section 4.2.5 In particular for the NvBlockDescriptor, the Sender-Receiver ports, the Rte_SetMirror, and Rte_GetMirror may access concurrently the same VariableDataPrototypes.

[SWS_Rte_07319] ⌈ The Rte_SetMirror API shall be callable before the Rte is started (with Rte_Start), and can rely on a running OS. ⌋(SRS_Rte_00178)

Return Value: The NVM module uses the return value of the Rte_SetMirror API to check if the copy was successful. In case of failure, the NVM may retry later.

[SWS_Rte_07602] ⌈ The Rte_SetMirror API shall return E_OK if the copy is successful. ⌋(SRS_Rte_00178)

[SWS_Rte_07613] ⌈ The Rte_SetMirror API shall return E_NOT_OK if the copy could not be performed. ⌋(SRS_Rte_00178)

Notes: The NVM shall be configured to use this function when ReadBlock requests are processed (see NvmWriteRamBlockFromNvm in [21]).

5.9.4.2 Rte_GetMirror

Purpose: Warranty the consistency of the VariableDataPrototypes contained in a NvBlockSwComponentType, when their values are written to the NVRAM device by the NVM.

Signature: **[SWS_Rte_07315]** ⌈

```
Std_ReturnType
Rte_GetMirror_<b>_<d> (void *NVMBuffer)
```

⌋(SRS_Rte_00178)

Where <b> is the SwComponentPrototype's name of the NvBlockSwComponentType and <d> is the NvBlockDescriptor name.

Existence: **[SWS_Rte_07316]** ⌈ An Rte_GetMirror API shall be created for each instance of a NvBlockDescriptor. ⌋(SRS_Rte_00178)

Description: The Rte_GetMirror API copies the values of the VariableDataPrototypes contained in a NvBlockDescriptor to a specified NVM internal buffer.

[SWS_Rte_07317] ⌈ The Rte_GetMirror API shall copy the NvBlockDescriptor's ramBlock to the specified buffer, according to the NvBlockDescriptor's NvBlockDataMapping. ⌋(*SRS_Rte_00177*)

The RTE is responsible for ensuring the data consistency, see section 4.2.5 In particular for the NvBlockDescriptor, the Sender-Receiver ports, the Rte_SetMirror, and Rte_GetMirror may access concurrently the same VariableDataPrototypes.

[SWS_Rte_07350] ⌈ The Rte_GetMirror API shall be callable after the Rte is stopped (with Rte_Stop), and can rely on a running OS. ⌋(*SRS_Rte_00178*)

Return Value: The NVM module uses the return value of the Rte_GetMirror API to check if the copy was successful. In case of failure, the NVM may retry later.

[SWS_Rte_07601] ⌈ The Rte_GetMirror API shall return E_OK if the copy is successful. ⌋(*SRS_Rte_00178*)

[SWS_Rte_07614] ⌈ The Rte_GetMirror API shall return E_NOT_OK if the copy could not be performed. ⌋(*SRS_Rte_00178*)

Notes: The NVM shall be configured to use this function when WriteBlock requests are processed (see NvmWriteRamBlockToNvm in [21]).

5.9.4.3 Rte_NvMNotifyJobFinished

Purpose: Forward notifications back to the SW-Cs.

Signature: **[SWS_Rte_07623]** ⌈

```
Std_ReturnType
Rte_NvMNotifyJobFinished_<b>_<d> (
    uint8 ServiceId,
    NvM_RequestResultType JobResult)
```

⌋(*SRS_Rte_00228*)

Where <b> is the SwComponentPrototype's name of the NvBlockSwComponentType and <d> is the NvBlockDescriptor name.

Existence: **[SWS_Rte_07624]** ⌈ An Rte_NvMNotifyJobFinished API shall be created for each instance of a NvBlockDescriptor. ⌋(*SRS_Rte_00228*)

Description: The Rte_NvMNotifyJobFinished receives the notification from the NvM when a job is finished and forward it to the SW-C.

[SWS_Rte_07625] ⌈ The Rte_NvMNotifyJobFinished API shall call the servers referenced by RoleBasedPortAssignment with a NvMNotifyJobFinished role which are aggregated to the NvBlockDescriptor. ⌋*(SRS_Rte_00228)*

[SWS_Rte_07671] ⌈ The Rte_NvMNotifyJobFinished API shall return without any action when the RTE is not started, when the RTE is stopped, or when the partition containing the NvBlockSwComponentType is terminated or restarting. ⌋*(SRS_Rte_00228)*

Return Value: **[SWS_Rte_07626]** ⌈ The Rte_NvMNotifyJobFinished API shall return E_OK. ⌋*(SRS_Rte_00228)*

Notes: The NVM shall be configured to use this function (see NvmSingleBlockCallback in [21]).

5.9.4.4 Rte_NvMNotifyInitBlock

Purpose: Indicate to the SW-Cs that initialization of the Mirror is requested by the NvM.

Signature: **[SWS_Rte_07627]** ⌈

```
Std_ReturnType
Rte_NvMNotifyInitBlock_<b>_<d> (void)
```

⌋*(SRS_Rte_00228)*

Where <b> is the SwComponentPrototype's name of the NvBlockSwComponentType and <d> is the NvBlockDescriptor name.

Existence: **[SWS_Rte_07628]** ⌈ An Rte_NvMNotifyInitBlock API shall be created for each instance of a NvBlockDescriptor. ⌋*(SRS_Rte_00228)*

Description: The Rte_NvMNotifyInitBlock API receives the notification from the NvM when initialization of the mirror is requested.

[SWS_Rte_07629] ⌈ If the NvBlockDescriptor is configured with a romBlock initValue, this initValue shall be copied into the NvBlockDescriptor's mirror before calling any SW-C server. ⌋*(SRS_Rte_00228)*

[SWS_Rte_07630] ⌈ The Rte_NvMNotifyInitBlock API shall call the servers referenced by RoleBasedPortAssignment with a NvMNotifyInitBlock role which are aggregated to the NvBlockDescriptor. ⌋*(SRS_Rte_00228)*

[SWS_Rte_07672] ⌈ The Rte_NvMNotifyInitBlock API shall return without any action when the RTE is not started, when the RTE is stopped, or when the partition containing the NvBlockSwComponentType is terminated or restarting. ⌋*(SRS_Rte_00228)*

Due to [SWS_Rte_07672], a block selected in the NVRAM Manager [21] as read during NvM_ReadAll should not be configured with its NvmInitBlockCallback set to a Rte_NvMNotifyInitBlock API.

Return Value: **[SWS_Rte_07631]** ⌈ The Rte_NvMNotifyInitBlock API shall return E_OK. ⌋*(SRS_Rte_00228)*

Notes: The NVM shall be configured to use this function (see InitBlockCallbackFunction in [21]).

5.10 Expected interfaces

5.10.1 Expected Interfaces from Com

The specification of the RTE requires the usage of the following COM API functions.

Com API function	Context
Com_SendSignal	to transmit a data element of primitive type using COM.
Com_SendDynSignal	to transmit a data element of primitive dynamic type uint8[n] using COM.
Com_ReceiveSignal	to retrieve the new value of a data element of primitive type from COM.
Com_ReceiveDynSignal	to retrieve the new value of a data element of primitive dynamic type uint[8] from COM.
Com_UpdateShadowSignal (deprecated)	to update a primitive element of a data element of composite type in preparation for sending the composite type using COM.
Com_SendSignalGroup	to initiate sending of a data element of composite type using COM.
Com_ReceiveSignalGroup	to retrieve the new value of a data element of composite type from COM.
Com_ReceiveShadowSignal (deprecated)	to retrieve the new value of a primitive element of a data element of composite type from COM.
Com_InvalidateSignal	to invalidate a data element of primitive type using COM.
Com_InvalidateSignalGroup	to invalidate a whole signal group using COM.

Table 5.6: COM API functions used by the RTE

Please note that [SWS_Rte_02761] may require to access COM through the use of call trusted function in a partitioned system.

5.10.2 Expected Interfaces from Os

The usage of APIs provided by the Os module [4] is up to the implementation of a specific RTE Generator, System description and Ecu configuration. In general a RTE may utilize any standardized API. Therefore no dedicated list of expected APIs is specified here.

In case of multi-core the RTE may utilize the *IOC*-Module [4] to implement the inter-core communication. The *IOC*-Module is specified to be part of the Os. Therefore no specific APIs are listed here.

5.10.3 Expected Interfaces for Serialization

Serialization transforms multiple arguments into an opaque byte array, which is transferred. The receiver of the byte array deserializes it back into the arguments.

The serializer and deserializer functions are not generated by the RTE but called when needed.

5.10.3.1 Serialization

Purpose: The purpose of the serialize function is to transform multiple arguments into an opaque byte array, which is transferred over the communication system. How this is done is up to the implementation of the serializer.

Signature: **[constr_9065] Signature of Serializer** ⌈

```
Std_ReturnType
<name>(
    IN const Rte_Cs_TransactionHandleType *TransactionHandle,
    OUT uint8 *buffer,
    OUT uint16 *bufferLength,
    [IN Std_ReturnType returnValue,]
    [IN <data_1>,]...
    [IN <data_n>]
)
```

⌋

Where here `<name>` is the `shortName` of the `BswModuleEntry` referred as serializer by `RteSerializerBswModuleEntryRef` of `RteSerializedSignal`.

Existence: **[constr_9066] A BswModuleEntry representing a serializer shall comply to a serializer's signature** ⌈ A `BswModuleEntry` which is referred by a `SerializerBswModuleEntryRef` of a

ClientServerToSignalMapping of a client has to comply with [constr_9065]. ⌋

[SWS_Rte_08761] RTE shall provide a buffer for serialization ⌈
The RTE shall provide to the serializer a buffer whose size is at least the size indicated for this buffer in the serializer's BswModuleEntry (SwDataDefProps for the buffer argument). ⌋*(SRS_Rte_00091)*

Note:
The serializer deduces this worst case size it needs by evaluating the sum of all worst case/maximum sizes of all contained IN-arguments.

Description: The parameters of the API shall have the following format:

- TransactionHandle
  according to [SWS_Rte_08732]
- buffer
  Buffer allocated by the RTE, where the serialized data has to be stored by the serializer. Size of the buffer shall equal the value published by the serializer in the correspnding BswModuleEntry (SwDataDefProps for the buffer argument).
- bufferLength
  Information about used length of the buffer. The length of the serialized data shall be calculated by the serialiazer during runtime and returned in this OUT-parameter. It may be smaller than the maximum buffer size used by the RTE for buffer allocation. This information shall be used by the RTE to set the value of the Length parameter in calls to Com_SendDynSignal. In the case of non-dynamic signals, these information can be ignored.
- <data_1>...<data_n>
  On client side: IN and IN/OUT arguments of the Client Server Call, in the same order as the arguments of the corresponding Rte_Call API.
  On server side: IN/OUT and OUT arguments of the Client Server Call, with the same types and order as IN/OUT and OUT arguments of the corresponding server runnable
- returnValue
  Serializer on the server side needs to serialize the return value. This argument is only available for server side serializer.

Return Value:

- **[constr_9068] Return value for successful serialization** ⌈
  E_OK – serialization passed successfully. ⌋
- **[constr_9069] Return value for a serialization error** ⌈
  RTE_E_SERIALIZATION_ERROR – A serialization error has been detected ⌋

Notes: There have to be two serializer functions for each Client-Server operation: one to serialize the call on the client side and one to serialize the return values on the server ECU.

5.10.3.2 Deserialization

The signature of the expected APIs is derived from the ClientServerInterface, which is referred by the SerializerBswModuleEntryRef-Mapping.

[SWS_Rte_08762] Serialization and deserialization in context ob Rte_ComCbk ⌈ The deserializing on client and server side shall be done on the same context as the Rte_COMCbk_<sn> function. ⌋*(SRS_Rte_00091)*

Purpose: The purpose of the deserialize function is to transfom a byte array into multiple arguments. How this is done is up to the implementation of the deserializer.

Signature: **[constr_9071] Signature of Deserializer** ⌈

```
Std_ReturnType
    <name>()(
    OUT Rte_Cs_TransactionHandleType *TransactionHandle,
    IN const uint8 *buffer,
    IN uint16 bufferLength,
    [OUT Std_ReturnType *returnValue,]
    [OUT <data_1>,]...
    [OUT <data_n>]
)
```

⌋

Where here <name> is the shortName of the BswModuleEntry referred as deserializer.

Existence: **[constr_9072] A BswModuleEntry representing a deserializer shall comply to a deserializer's signature** ⌈ If a BswModuleEntry is referred by a SerializerBswModuleEntryRef of a Server an API according to [constr_9071] has to be provided. ⌋

Description: The parameters of the API shall have the following format:

- TransactionHandle
  according to [SWS_Rte_08732]
- buffer
  Received buffer containing the data to be deserialized.
- bufferLength
  Length of the received buffer.

- <data_1>...<data_n>
  On client side: IN/OUT and IN/OUT arguments of the Client Server Call, with the same types and order as the arguments of the corresponding `Rte_Result` API.
  On server side: IN and IN/OUT arguments of the Client Server Call, with the same types and order as the IN and IN/OUT arguments of the corresponding server runnable.
- returnValue
  The returnValue of the call has to be deserialized on the client side. This argument exists on client side only.
  Client side deserializer: The return value equals the return value of the server runnable

Return Value:

- **[constr_9073] Return value for successful deserialization** ⌈ `E_OK` – deserialization passed successfully. ⌋
- **[constr_9074] Return value for a deserialization error** ⌈ `RTE_E_SERIALIZATION_ERROR` – A deserialization error has been detected ⌋

Notes: There have to be two deserializer functions for each Client-Server operation: one to deserialize the call on the server side and one to deserialize the return values on the client side.

5.11 VFB Tracing Reference

The RTE's "VFB Tracing" functionality permits the monitoring of AUTOSAR signals as they are sent and received across the VFB.

The RTE operates in at least two builds (some implementations may provide more than two builds). The first, production, does not enable VFB tracing whereas the second, debug, can be configured to trace some or all "interesting events".

[SWS_Rte_01327] ⌈ The RTE generator shall support a build where no VFB events are traced. ⌋*(SRS_Rte_00005)*

[SWS_Rte_01328] ⌈ The RTE generator shall support a build that traces (configured) VFB events. ⌋*(SRS_Rte_00005)*

The RTE generator's 'trace' build is enabled or disabled through definitions in the RTE Configuration Header File [SWS_Rte_01322] and [SWS_Rte_01323]. Note that this 'trace' build is intended to enable debugging of software components and not the RTE itself.

5.11.1 Principle of Operation

The "VFB Tracing" mechanism is designed to offer a lightweight means to monitor the interactions of AUTOSAR software-components with the VFB.

The VFB tracing in 'debug' build is implemented by a series of "hook" functions that are invoked automatically by the generated RTE when "interesting events" occur. Each hook function corresponds to a single event.

The supported trace events are defined in Section 5.11.5. A mechanism is described in Section 5.11.6 for configuring which of the many potential trace events are of interest.

5.11.2 Support for multiple clients

The "VFB Tracing" mechanism is designed to support multiple clients for each trace event.

[SWS_Rte_05093] ⌈ For each `RteVfbTraceClientPrefix` configured in the RTE Configuration input each Trace Event shall be generated using that *client prefix* in the optional `<client>` position of the API function name. ⌋*(SRS_Rte_00005, SRS_Rte_00008, SRS_Rte_00192)*

[SWS_Rte_05091] ⌈ The RTE Generator shall provide each Trace Event without a *client prefix*. ⌋*(SRS_Rte_00005, SRS_Rte_00008, SRS_Rte_00192)*

The generation of Trace Events without a *client prefix* ensures compatibility of the trace events with previous RTE releases.

[SWS_Rte_05092] ⌈ In case of multiple clients for one Trace Event the individual trace functions shall be called in the following order:

1. The trace function without *client prefix*.
2. The trace functions with *client prefix* in alphabetically ascending order of the `RteVfbTraceClientPrefix` (ASCII / ISO 8859-1).

⌋*(SRS_Rte_00005, SRS_Rte_00008, SRS_Rte_00192)*

The calling order specification ensures a deterministic execution of the multiple clients.

One example of the usage of *client prefix* is the parallel usage of Debugging [26] and Diagnostic Log and Trace [31]. In this example two `RteVfbTraceClientPrefix` would be specified:

- `Dbg`
- `Dlt`

This shall result in the declaration of three trace functions for the one Trace Event `Rte_[<client>_]Task_Activate(TaskType task)`:

- `Rte_Task_Activate(TaskType task)`

- `Rte_Dbg_Task_Activate(TaskType task)`
- `Rte_Dlt_Task_Activate(TaskType task)`

These trace functions (if all used in one project) will be called in the following order:

1. `Rte_Task_Activate(TaskType task)`
2. `Rte_Dbg_Task_Activate(TaskType task)`
3. `Rte_Dlt_Task_Activate(TaskType task)`

5.11.3 Support for Multiple Instantiation

[SWS_Rte_06031] ⌈ The Component Data Structure type for a multiply instantiatable SWC type shall be introduced as a forward reference when used within the VFB Tracing Header File. ⌋*(SRS_Rte_00005, SRS_Rte_00011)*

The use of a forward reference enables a pointer to the object to be taken (since the size of the data structure does not need to be known).

5.11.4 Contribution to the Basic Software Module Description

The RTE Generator in Generation Phase shall also update its Basic Software Module Description ([SWS_Rte_05086]) in order to document the possibly traceable functions and their signatures.

[SWS_Rte_05106] ⌈ For each generated hook function - including multiple trace clients ([SWS_Rte_05093]) - an entry in the Basic Software Module Description shall be entered describing the hook function and its signature. The `outgoingCallback` element of `BswModuleDescription` shall be used to capture the information. ⌋*(SRS_Rte_00005, SRS_Rte_00192)*

5.11.5 Trace Events

5.11.5.1 RTE API Trace Events

RTE API trace events occur when an AUTOSAR software-component interacts with the generated RTE API. For implicit S/R communication, however, tracing is not supported.

5.11.5.1.1 RTE API Start

Description: RTE API Start is invoked by the RTE when an API call is made by a component.

Signature: **[SWS_Rte_01238]** ⌈

```
void Rte_[<client>_]<api>Hook_<cts>_<ap>_Start
      ([const struct Rte_CDS_<cts>*, ]<param>)
```

Where `<api>` is the RTE API Name (Write, Call, etc.),

`<cts>` is the `component type symbol` of the `AtomicSwComponentType` and

`<ap>` the access point name (e.g. port and data element or operation name, exclusive area name, etc.).

The parameters of the API are the same as the corresponding RTE API. As with the API itself, the instance handle is included if and only if the software component's `supportsMultipleInstantiation` attribute is set to true. Note that `Rte_Instance` cannot be used directly, as there will be pointers to multiple components' structure types within the single VFB Tracing header file, and `Rte_Instance` would therefore be ambiguous. ⌋*(SRS_Rte_00045, SRS_Rte_00003, SRS_Rte_00004)*

5.11.5.1.2 RTE API Return

Description: RTE API Return is a trace event that is invoked by the RTE just before an API call returns control to a component.

Signature: **[SWS_Rte_01239]** ⌈

```
void Rte_[<client>_]<api>Hook_<cts>_<ap>_Return
      ([const struct Rte_CDS_<cts>*, ]<param>)
```

Where `<api>` is the RTE API Name (Write, Call, etc.),

`<cts>` is the `component type symbol` of the `AtomicSwComponentType` and

`<ap>` the access point name (e.g. port and data element or operation name, exclusive area name, etc.).

The parameters of the API are the same as the corresponding RTE API and contain the values of OUT and INOUT parameters on exit from the function. ⌋*(SRS_Rte_00045)*

As with the API itself, the instance handle is included if and only if the software component's `supportsMultipleInstantiation` attribute is set to true. Note that `Rte_Instance` cannot be used directly, as there will be pointers to multiple components' structure types within the single VFB Tracing header file, and `Rte_Instance` would therefore be ambiguous.

5.11.5.2 COM Trace Events

COM trace events occur when the generated RTE interacts with the AUTOSAR communication service.

5.11.5.2.1 Signal Transmission

Description: A trace event indicating a transmission request of an Inter-ECU signal (or signal in a signal group) by the RTE. Invoked by the RTE just before `Com_SendSignal`, `Com_SendDynSignal`, or `Com_UpdateShadowSignal` (deprecated) is invoked.

Signature: **[SWS_Rte_01240]** ⌈

```
void Rte_[<client>_]ComHook_<signalName>_SigTx
     (<data>[, <length>])
```

Where `<signalName>` is the COM signal name, `<data>` is a pointer to the signal data to be transmitted, and `<length>` is the length of the signal in case of a dynamic signal. ⌋*(SRS_Rte_00045, SRS_Rte_00003, SRS_Rte_00004)*

5.11.5.2.2 Signal Reception

Description: A trace event indicating a successful attempt to read an Inter-ECU signal (or signal in a signal group) by the RTE. Invoked by the RTE after return from `Com_ReceiveSignal`, `Com_ReceiveDynSignal`, or `Com_ReceiveShadowSignal` (deprecated).

Signature: **[SWS_Rte_01241]** ⌈

```
void Rte_[<client>_]ComHook_<signalName>_SigRx
     (<data>[, <length>])
```

Where `<signalName>` is the COM signal name, `<data>` is a pointer to the signal data received, and `<length>` is a pointer where the length of the dynamic signal is copied in case of a dynamic signal. ⌋*(SRS_Rte_00045, SRS_Rte_00003, SRS_Rte_00004)*

5.11.5.2.3 Signal Invalidation

Description: A trace event indicating a signal invalidation request of an Inter-ECU signal (or of a signal in a signal group) by the RTE. Invoked by the RTE just before `Com_InvalidateSignal` (if parameter `RteUseComShadowSignalApi` is `FALSE`), or `Com_InvalidateShadowSignal` (if parameter `RteUseComShadowSignalApi` is `TRUE`) is invoked.

Signature: **[SWS_Rte_03814]** ⌈

```
void Rte_[<client>_]ComHook_<signalName>_SigIv
     (void)
```

Where `<signalName>` is the COM signal or a signal group name.
⌋*(SRS_Rte_00045, SRS_Rte_00003, SRS_Rte_00004)*

5.11.5.2.4 Signal Group Invalidation

Description: A trace event indicating a signal group invalidation request of an Inter-ECU signal group by the RTE. Invoked by the RTE just before `Com_InvalidateSignalGroup` is invoked.

Signature: **[SWS_Rte_07639]** ⌈

```
void Rte_[<client>_]ComHook_<signalGroupName>_SigGroupIv
     (void)
```

Where `<signalGroupName>` is the name of the signal group.
⌋*(SRS_Rte_00045, SRS_Rte_00003, SRS_Rte_00004)*

5.11.5.2.5 COM Callback

Description: A trace event indicating the start of a COM call-back. Invoked by generated RTE code on entry to the COM call-back.

Signature: **[SWS_Rte_01242]** ⌈

```
void Rte_[<client>_]ComHook<Event>_<signalName>
     (void)
```

Where `<signalName>` is the name of the COM signal or signal group and `<Event>` indicates the callback type and can take the values

- “Rx” for a reception indication callback
- “Inv” for an invalidation callback
- “RxTOut” for a reception timeout callback
- “TxTOut” for a transmission timeout callback
- “TAck” for a transmission acknowledgement callback
- “TErr” for a transmission error callback

⌋*(SRS_Rte_00045, SRS_Rte_00003, SRS_Rte_00004)*

5.11.5.3 OS Trace Events

OS trace events occur when the generated RTE interacts with the AUTOSAR operating system.

5.11.5.3.1 Task Activate

Description: A trace event that is invoked by the RTE immediately prior to the activation of a task containing runnable entities.

Signature: **[SWS_Rte_01243]** ⌈

```
void Rte_[<client>_]Task_Activate(TaskType task)
```

Where `task` is the OS's handle for the task. ⌋*(SRS_Rte_00045)*

5.11.5.3.2 Task Dispatch

Description: A trace event that is invoked immediately an RTE generated task (containing runnable entities) has commenced execution.

Signature: **[SWS_Rte_01244]** ⌈

```
void Rte_[<client>_]Task_Dispatch(TaskType task)
```

Where `task` is the OS's handle for the task. ⌋*(SRS_Rte_00045)*

5.11.5.3.3 Task Termination

Description: A trace event invoked immediately prior to an RTE generated task (containing runnable entities) terminating execution. The same task termination VFB event is used whether the RTE generated task terminates by either a `TerminateTask` or a `ChainTask` OS Service call.

Signature: **[SWS_Rte_06032]** ⌈

```
void Rte_[<client>_]Task_Terminate(TaskType task)
```

Where `task` is the OS's handle for the task. ⌋*(SRS_Rte_00045)*

5.11.5.3.4 Set OS Event

Description: A trace event invoked immediately before generated RTE code attempts to set an OS Event.

Signature: **[SWS_Rte_01245]** ⌈

```
void Rte_[<client>_]Task_SetEvent(TaskType task,
                                  EventMaskType ev)
```

Where `task` is the OS's handle for the task for which the event is being set and `ev` the OS event mask. ⌋*(SRS_Rte_00045)*

5.11.5.3.5 Wait OS Event

Description: Invoked immediately before generated RTE code attempts to wait on an OS Event. This trace event does *not* indicate that the caller has suspended execution since the OS call may immediately return if the event was already set.

Signature: **[SWS_Rte_01246]** ⌈

```
void Rte_[<client>_]Task_WaitEvent(TaskType task,
                                   EventMaskType ev)
```

Where `task` is the OS's handle for the task (that is waiting for the event) and `ev` the OS event mask. ⌋*(SRS_Rte_00045)*

5.11.5.3.6 Received OS Event

Description: Invoked immediately after generated RTE code returns from waiting on an event.

Signature: **[SWS_Rte_01247]** ⌈

```
void Rte_[<client>_]Task_WaitEventRet(TaskType task,
                                      EventMaskType ev)
```

Where `task` is the OS's handle for the task (that was waiting for an event) and `ev` the event mask indicating the received event. ⌋*(SRS_Rte_00045)*

Note that not all of the trace events listed above may be available for a given input configuration. For example if a task is activated by a schedule table, it is activated by the OS rather than by the RTE, hence no trace hook function for task activation can be invoked by the RTE.

5.11.5.4 Runnable Entity Trace Events

Runnable entity trace events occur when a runnable entity is started.

5.11.5.4.1 Runnable Entity Invocation

Description: Event invoked by the RTE just before execution of runnable entry starts via its entry point. This trace event occurs after any copies of data elements are made to support the Rte_IRead API Call.

Signature: **[SWS_Rte_01248]** ⌈

```
void Rte_[<client>_]Runnable_<cts>_<reName>_Start
      ([const RTE_CDS_<cts>*])
```

Where `<cts>` is the component type symbol of the AtomicSwComponentType

and `reName` the runnable entity name.

The instance handle is included if and only if the software component's supportsMultipleInstantiation attribute is set to true. Note that `Rte_Instance` cannot be used directly, as there will be pointers to multiple components' structure types within the single VFB Tracing header file, and `Rte_Instance` would therefore be ambiguous. ⌋*(SRS_Rte_00045)*

5.11.5.4.2 Runnable Entity Termination

purpose: Event invoked by the RTE immediately execution returns to RTE code from a runnable entity. This trace event occurs before any write-back of data elements are made to support the Rte_IWrite API Call.

Signature: **[SWS_Rte_01249]** ⌈

```
void Rte_[<client>_]Runnable_<cts>_<reName>_Return
      ([const Rte_CDS_<cts>*])
```

Where `<cts>` is the component type symbol of the AtomicSwComponentType

and `reName` the runnable entity name.

The instance handle is included if and only if the software component's supportsMultipleInstantiation attribute is set to true. Note that `Rte_Instance` cannot be used directly, as there will be pointers to multiple components' structure types within the single VFB Tracing header file, and `Rte_Instance` would therefore be ambiguous. ⌋*(SRS_Rte_00045)*

5.11.6 Configuration

The VFB tracing mechanism works by the RTE invoking the tracepoint *hook* function whenever the tracing event occurs.

The support trace events and their hook function name and signature are defined in Section 5.11.5. There are many potential trace events and it is likely that only a few will be of interest at any one time. Therefore The RTE generator supports a mechanism to configure which trace events are of interest.

In order to minimize RTE Overheads, trace events that are not enabled should have no run-time effect on the generated system. This is achieved through generated code within the VFB Tracing Header File (see Section 5.3.7) and the user supplied definitions from the RTE Configuration Header file (see Section 5.3.8).

The definition of trace event hook functions is contained within user code. If a definition is encapsulated within a `#if` block, as follows, the definition will automatically be omitted when the trace event is disabled.

```
#if !defined(<trace event>)
void <trace event>(<params>)
{
  /* Function definition */
}
#endif
```

The configuration of which individual trace events are enabled is entirely under the control of the user via the definitions included in the RTE Configuration header file.

[SWS_Rte_08000] ⌈ When RteVfbTrace is set to "true", a user shall be able to enable any hook function in the RTE Configuration header file, regardless of whether it was not enabled in the RTE configuration with a `RteVfbTraceFunction` parameter. ⌋*(SRS_Rte_00005, SRS_Rte_00008)*

5.11.7 Interaction with Object-code Software-Components

VFB tracing is only available during the “RTE Generation” phase [SWS_Rte_01319] and therefore hook functions never appear in an application header file created during “RTE Contract” phase. However, object-code software-components are compiled against the “RTE Contract” phase header and can therefore only trace events that are inserted into the generated RTE. In particular they cannot trace events that require invocation of hook functions to be inserted into the API mapping such as the `Rte_Pim` API. However, many trace events are applicable to object-code software-components including trace events related to the explicit communication API, to task activity and for runnable entity start and stop.

This approach means that the external interactions of the object-code software-component can be monitored without requiring modification of the delivered object-code and without revealing the internal activity of the software-component. The approach is therefore considered to be consistent with the desire for IP protection that prompts delivery of a software-component as object-code. Finally, tracing can easily be disabled for a production build without invalidating tests of the object-code software-component.

6 Basic Software Scheduler Reference

6.1 Scope

This chapter presents the *Basic Software Scheduler* API from the perspective of *AUTOSAR Basic Software Module* – these API is not applicable for *AUTOSAR software-components*.

Section 6.2 presents basic principles of the API including naming conventions and supported programming languages. Section 6.3 describes the header files used by the *Basic Software Scheduler* and the files created by an RTE generator. The data types used by the API are described in Section 6.4 and Sections 6.5 and 6.6 provide a reference to the *Basic Software Scheduler* API itself including the definition of *Basic Software Module Entities*.

6.2 API Principles

6.2.1 Basic Software Scheduler Namespace

The *Basic Software Scheduler* is interleaved with the scheduling part of the *RTE*. Further on it is generated by the *RTE Generator* together with the *RTE* so *Basic Software Scheduler* and *RTE* can not be separated if both are generated. Therefore the *Basic Software Scheduler* uses the namespace of the *RTE* for internal symbols, variables and functions, see [SWS_Rte_01171].

The only exceptions are defines, data types and functions belonging to the interface of the *Basic Software Scheduler*. These are explicitly mentioned in the specification.

[SWS_Rte_07284] ⌈ All Basic Software Scheduler symbols (e.g. function names, data types, etc.) belonging to the *Basic Software Scheduler*s interfaces are required to use the `SchM_` prefix. ⌋*(SRS_BSW_00307, SRS_BSW_00300, SRS_Rte_00055)*

In case of *Basic Software Module*s supporting multiple instances of the same *Basic Software Module* the name space of the `BswSchedulableEntity`s and the *Basic Software Scheduler* API related to one instance of a *Basic Software Module* is extended by the `vendorId` and the `vendorApiInfix`. See document [12] [SRS_BSW_00347]. In the following chapters this optional part is denoted by usage of squared brackets `[_<vi>_<ai>]`.

[SWS_Rte_07528] ⌈ If the attribute `vendorApiInfix` exists for a *Basic Software Module*, the RTE generator shall insert the `vendorId` (`<vi>`) and the `vendorApiInfix` (`<ai>`) with leading underscores where it is denoted by `[_<vi>_<ai>]`. ⌋*(SRS_BSW_00347)*

6.2.2 BSW Scheduler Name Prefix and Section Name Prefix

Since the Basic Software Module Description supports the description of BSW Module Clusters one *Basic Software Module Description* can contain the content of several BSW Modules. In order to fulfill the Standardized Interfaces with the cluster interface different ICC3 *Module abbreviation*s [9] inside one cluster can occur. For the Basic Software Scheduler the *Module abbreviation* is used as *BSW Scheduler Name Prefix* in the SchM API. Nevertheless the `shortName` of the `BswModuleDescription` can as well describe the `BSW Scheduler Name Prefix` and `Section Name Prefix` in order to provide one common prefix in case of ICC3 modules.

In the Meta Model *Module abbreviation*s relevant for the Schedule Manager API are explicitly expressed with the meta class `BswSchedulerNamePrefix`. Further information can be found in document [9].

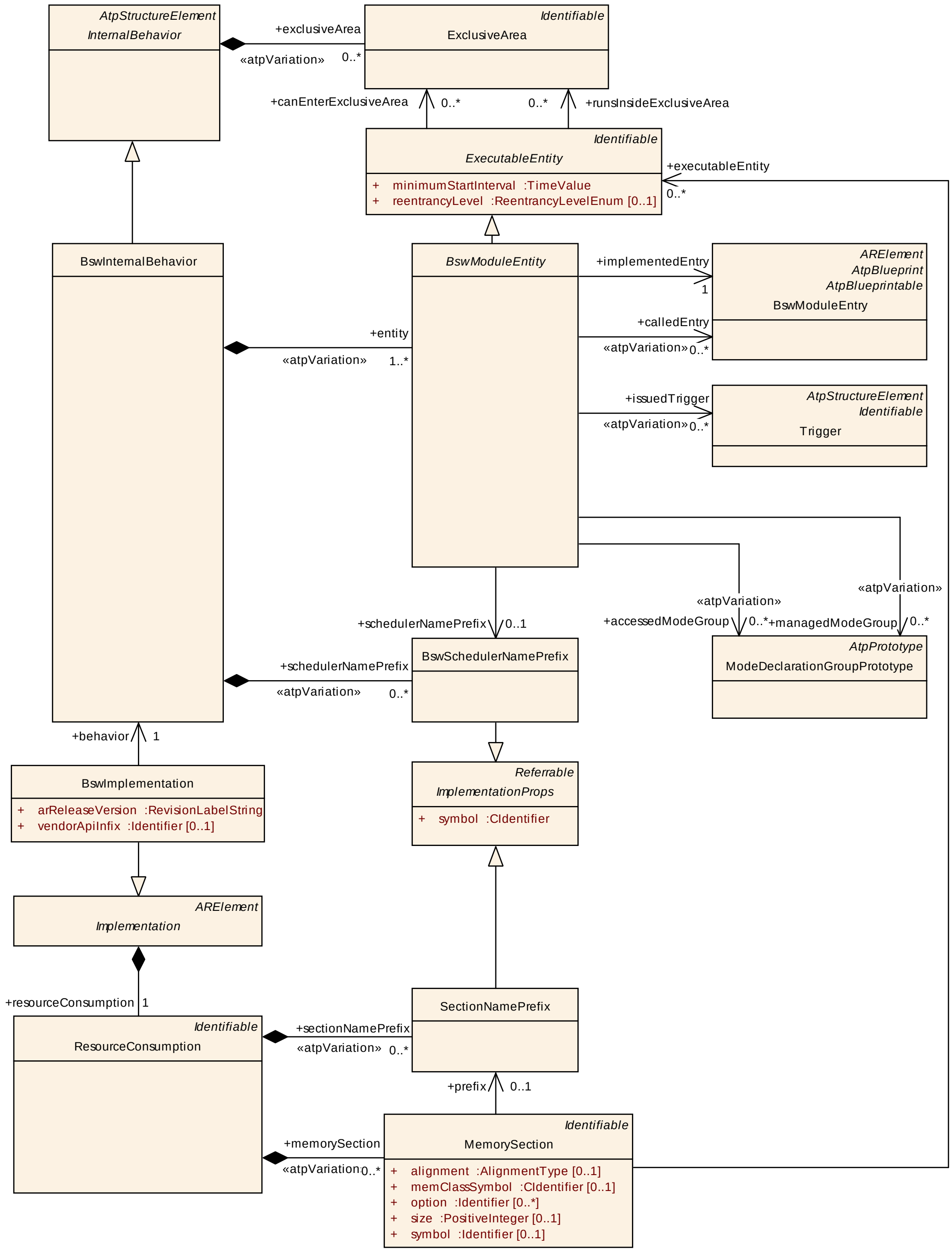


Figure 6.1: `BswSchedulerNamePrefix` and `SectionNamePrefix`

In several requirements of this specification the *Module Prefix* is required and determined as follows:

[SWS_Rte_07593] ⌈ The `BSW Scheduler Name Prefix` <bsnp> of the calling BSW module shall be derived from the `BswModuleDescription shortName` if no `BswSchedulerNamePrefix` is defined for the `BswModuleEntity` using the related Basic Software Scheduler API. ⌋*(SRS_Rte_00148, SRS_Rte_00149)*

[SWS_Rte_07594] ⌈ The BSW Scheduler Name Prefix <bsnp> shall be the value of the symbol attribute of the BswSchedulerNamePrefix of the BswModuleEntity if a BswSchedulerNamePrefix is defined for the BswModuleEntity using the related Basic Software Scheduler API. ⌋ *(SRS_Rte_00148, SRS_Rte_00149)*

Further on the *Memory Mapping* inside one cluster can either keep or abolish the ICC3 borders. For some cases (e.g. *Entry Point Prototype*) the RTE has to know the used prefixes for the *Memory Allocation Keyword*s as well.

In the Meta Model these prefixes are expressed with the meta class SectionNamePrefix. Further information can be found in document [9].

[SWS_Rte_07595] ⌈ The Section Name Prefix <snp> shall be the module abbreviation (in uppercase letters) of the BSW module derived from the BswModuleDescription's shortName if no SectionNamePrefix is defined for the BswModuleEntity implementing the related BswModuleEntry. ⌋ *(SRS_Rte_00148, SRS_Rte_00149)*

[SWS_Rte_07596] ⌈ The Section Name Prefix <snp> shall be the symbol of the SectionNamePrefix of the MemorySection associated to the BswModuleEntity implementing the related BswModuleEntry if a SectionNamePrefix is defined for the BswModuleEntity implementing the related BswModuleEntry. ⌋ *(SRS_Rte_00148, SRS_Rte_00149)*

For instance the following input configuration

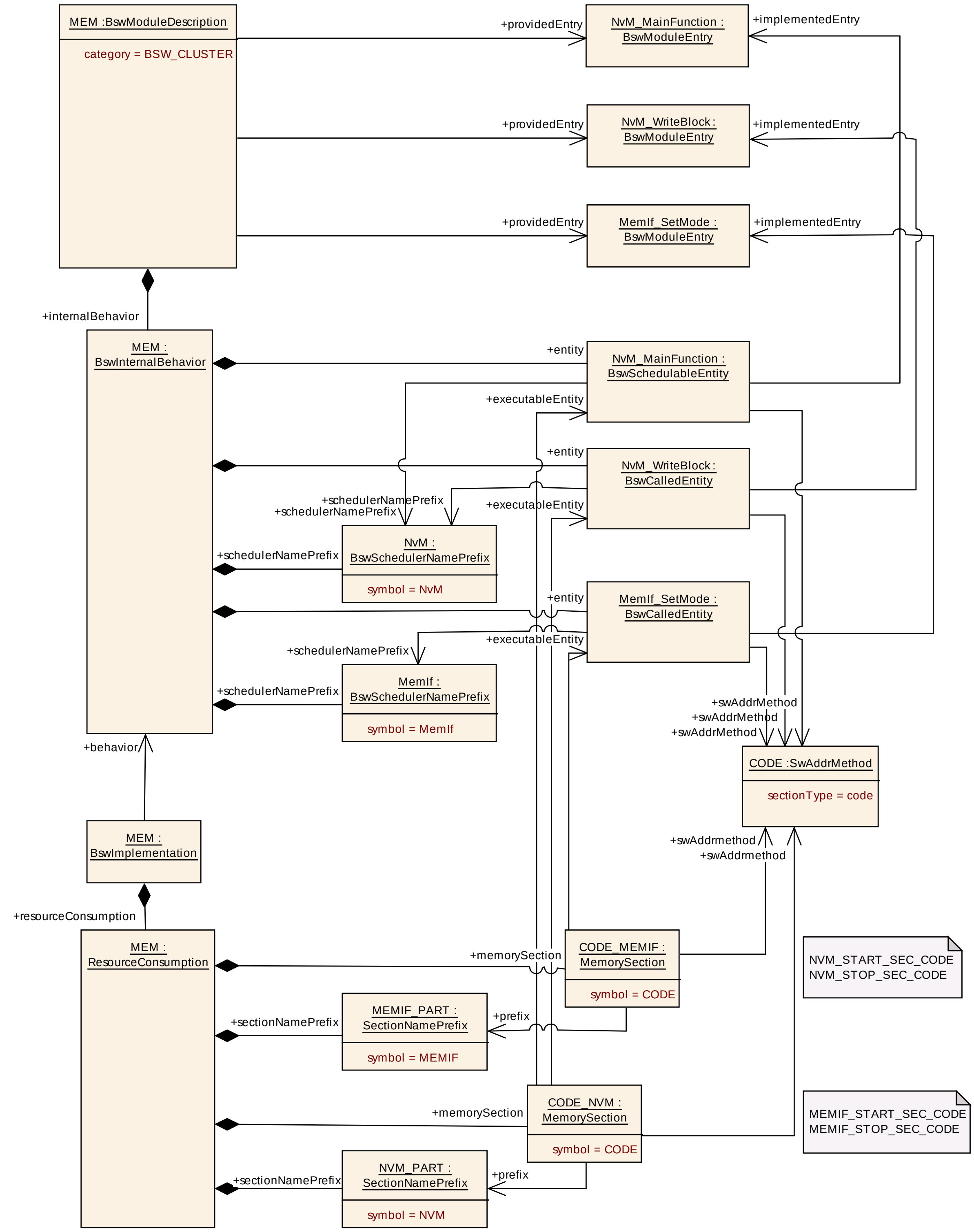


Figure 6.2: Example of ICC2 cluster

would result in the generation of the *Entry Point Prototype* according [SWS_Rte_07195] as:

```
#define NVM_START_SEC_CODE
#include "MEM_MemMap.h"

```

```
FUNC(void, NVM_CODE) NvM_MainFunction (void);

#define NVM_STOP_SEC_CODE
#include "MEM_MemMap.h"
```

6.3 Basic Software Scheduler modules

[SWS_Rte_07288] ⌈ Every file of the *Basic Software Scheduler* shall be named with the prefix SchM_. ⌋*(SRS_BSW_00300)*

6.3.1 Module Interlink Types Header

The *Module Interlink Types Header* defines specific types related to this basic software module derived either from the input configuration or from the RTE / Basic Software Scheduler implementation.

[SWS_Rte_07503] ⌈ The RTE generator shall create a *Module Interlink Types Header File* for each BswSchedulerNamePrefix in the BswInternalBehavior of each BswImplementation referencing such BswInternalBehavior defined in the input. ⌋*(SRS_BSW_00415)*

For instance a input configuration with two BswImplementations (typical with different API infix) referencing a BswInternalBehavior with three BswSchedulerNamePrefixes would result in the generation of six Module Interlink Types Header Files.

6.3.1.1 File Name

[SWS_Rte_07295] ⌈ The name of the *Module Interlink Types Header File* shall be formed in the following way:

SchM_<bsnp>_[<vi>_<ai>]Type.h

Where here

<bsnp> is the *BSW Scheduler Name Prefix* according [SWS_Rte_07593] and [SWS_Rte_07594],

<vi> is the vendorId of the BSW module and

<ai> is the vendorApiInfix of the BSW module.

The sub part in squared brackets [<vi>_<ai>] is omitted if no vendorApiInfix is defined for the *Basic Software Module*. See [SWS_Rte_07528]. ⌋*(SRS_BSW_00415, SRS_BSW_00300, SRS_BSW_00347)*

Example 6.1

The following declaration in the input XML:

```
<AR-PACKAGE>
  <SHORT-NAME>CanDriver</SHORT-NAME>
  <ELEMENTS>
    <BSW-MODULE-DESCRIPTION>
      <SHORT-NAME>Can</SHORT-NAME>
      <INTERNAL-BEHAVIORS>
        <BSW-INTERNAL-BEHAVIOR>
          <SHORT-NAME>YesWeCan</SHORT-NAME>
        </BSW-INTERNAL-BEHAVIOR>
      </INTERNAL-BEHAVIORS>
    </BSW-MODULE-DESCRIPTION>
    <BSW-IMPLEMENTATION>
      <SHORT-NAME>MyCanDrv</SHORT-NAME>
      <VENDOR-ID>25</VENDOR-ID>
      <BEHAVIOR-REF DEST="BSW-INTERNAL-BEHAVIOR">/CanDriver/Can/
          YesWeCan</BEHAVIOR-REF>
      <VENDOR-API-INFIX>Dev0815</VENDOR-API-INFIX>
    </BSW-IMPLEMENTATION>
  </ELEMENTS>
</AR-PACKAGE>
```

should result in the *Module Interlink Types Header* SchM_Can_25_Dev0815Type.h being generated.

The concatenation of the basic software module prefix (which has to be equally with the short name of the basic software module description) and the vendor API infix is required to support the separation of several basic software module instances. In difference to the multiple instantiation concept of software components, where the same component code is used for all component instances, basic software modules are multiple instantiated by creation of own code per instance in a different name space.

6.3.1.2 Scope

[SWS_Rte_07297] ⌈ The *Module Interlink Types Header* shall be valid for both C and C++ source. ⌋*(SRS_Rte_00126, SRS_Rte_00138)*

Requirement [SWS_Rte_07297] is met by ensuring that all definitions within the *Application Types Header File* are defined using C linkage if a C++ compiler is used.

[SWS_Rte_07298] ⌈ All definitions within in the *Module Interlink Types Header File* shall be preceded by the following fragment:

```
#ifdef __cplusplus
```

```
extern "C" {
#endif /* __cplusplus */
```

⌋(*SRS_Rte_00126, SRS_Rte_00138*)

[SWS_Rte_07299] ⌈ All definitions within the *Module Interlink Types Header* shall be suffixed by the following fragment:

```
#ifdef __cplusplus
} /* extern "C" */
#endif /* __cplusplus */
```

⌋(*SRS_Rte_00126, SRS_Rte_00138*)

6.3.1.3 File Contents

[SWS_Rte_07500] ⌈ The *Module Interlink Types Header* shall include the *RTE Types Header File*. ⌋(*SRS_BSW_00415*)

The name of the *RTE Types Header File* is defined in Section 5.3.4.

6.3.1.4 Basic Software Scheduler Modes

The *Module Interlink Types Header File* shall contain identifiers for the `ModeDeclarations` and type definitions for `ModeDeclarationGroups` as defined in Chapter 6.4.2

6.3.2 Module Interlink Header

The *Module Interlink Header* defines the *Basic Software Scheduler* API and any associated data structures that are required by the *Basic Software Scheduler* implementation. But the *Module Interlink Header* file is not allowed to create objects in memory.

[SWS_Rte_07501] ⌈ The RTE generator shall create a *Module Interlink Header File* for each `BswSchedulerNamePrefix` in the `BswInternalBehavior` of each `BswImplementation` referencing such `BswInternalBehavior` defined in the input.⌋(*SRS_BSW_00415*)

[constr_9059] Usage of *Basic Software Scheduler* API prerequisites the include of the *Module Interlink Header File* ⌈ Each BSW module implementation shall include its *Module Interlink Header File* if it uses *Basic Software Scheduler* API or if it implements `BswSchedulableEntitys`. ⌋

[SWS_Rte_07502] ⌈ The *Module Interlink Header File* shall not contain code that creates objects in memory. ⌋(*SRS_BSW_00308*)

6.3.2.1 File Name

[SWS_Rte_07504] ⌈

The name of the *Module Interlink Header File* shall be formed in the following way:

```
SchM_<bsnp>[_<vi>_<ai>].h
```

Where here

`<bsnp>` is the *BSW Scheduler Name Prefix* according [SWS_Rte_07593] and [SWS_Rte_07594],

`<vi>` is the `vendorId` of the BSW module and

`<ai>` is the `vendorApiInfix` of the BSW module.

The sub part in squared brackets `[_<vi>_<ai>]` is omitted if no `vendorApiInfix` is defined for the *Basic Software Module*. ⌋*(SRS_BSW_00415, SRS_BSW_00300, SRS_BSW_00347)*

Example 6.2

The following declaration in the input XML:

```xml
<AR-PACKAGE>
  <SHORT-NAME>CanDriver</SHORT-NAME>
  <ELEMENTS>
    <BSW-MODULE-DESCRIPTION>
      <SHORT-NAME>Can</SHORT-NAME>
      <INTERNAL-BEHAVIORS>
        <BSW-INTERNAL-BEHAVIOR>
          <SHORT-NAME>YesWeCan</SHORT-NAME>
        </BSW-INTERNAL-BEHAVIOR>
      </INTERNAL-BEHAVIORS>
    </BSW-MODULE-DESCRIPTION>
    <BSW-IMPLEMENTATION>
      <SHORT-NAME>MyCanDrv</SHORT-NAME>
      <VENDOR-ID>25</VENDOR-ID>
      <BEHAVIOR-REF DEST="BSW-INTERNAL-BEHAVIOR">/CanDriver/Can/
         YesWeCan</BEHAVIOR-REF>
      <VENDOR-API-INFIX>Dev0815</VENDOR-API-INFIX>
    </BSW-IMPLEMENTATION>
  </ELEMENTS>
</AR-PACKAGE>
```

should result in the *Module Interlink Header* `SchM_Can_25_Dev0815.h` being generated.

The concatenation of the basic software module prefix (which has to be equally with the short name of the basic software module description) and the `vendorApiInfix` is required to support the separation of several basic software module instances. In difference to the multiple instantiation concept of software components, where the same component code is used for all component instances, basic software modules are multiple instantiated by creation of own code per instance in a different name space.

6.3.2.2 Scope

[SWS_Rte_07505] ⌈ The *Module Interlink Header* for a component shall contain declarations relevant for that instance of a basic software module. ⌋*(SRS_BSW_00415)*

Requirement [SWS_Rte_07505] means that compile time checks ensure that a *Module Interlink Header File* that uses the *Module Interlink Header File* only accesses the generated data types to which it has been configured. The use of data types which are not used by the basic software module, will fail with a compiler error [SRS_Rte_00017].

6.3.2.3 File Contents

[SWS_Rte_07506] ⌈ The *Module Interlink Header File* shall include the *Module Interlink Types Header File*. ⌋*(SRS_BSW_00415)*

The name of the *Module Interlink Types Header File* is defined in Section 6.3.1.

[SWS_Rte_07507] ⌈ The *Module Interlink Header* shall be valid for both C and C++ source. ⌋*(SRS_Rte_00126, SRS_Rte_00138)*

Requirement [SWS_Rte_07507] is met by ensuring that all definitions within the *Application Types Header File* are defined using C linkage if a C++ compiler is used.

[SWS_Rte_07508] ⌈ All definitions within in the *Module Interlink Header File* shall be preceded by the following fragment:

```
#ifdef __cplusplus
extern "C" {
#endif /* __cplusplus */
```

⌋*(SRS_Rte_00126, SRS_Rte_00138)*

[SWS_Rte_07509] ⌈ All definitions within the *Module Interlink Header File* shall be suffixed by the following fragment:

```
#ifdef __cplusplus
} /* extern "C" */
#endif /* __cplusplus */
```

⌋*(SRS_Rte_00126, SRS_Rte_00138)*

6.3.2.3.1 Entry Point Prototype

The *Module Interlink Header File* also includes a prototype for each `BswSchedulableEntity`s entry point ([SWS_Rte_07283]).

6.3.2.3.2 Basic Software Scheduler - Basic Software Module Interface

The *Module Interlink Header File* defines the "interface" between a *Basic Software Module* and the *Basic Software Scheduler*. The interface consists of the *Basic Software Scheduler* API for the *Basic Software Module* and the prototypes for `BswSchedulableEntity`s entry point. The definition of the *Basic Software Scheduler* API requires in case of macro implementation that both relevant data structures and API calls are defined. In case of interfaces implemented as functions, the prototypes for the *Basic Software Scheduler* API of the particular *Basic Software Module* instance is sufficient. The data structures are dependent from the implementation and configuration of the *Basic Software Scheduler* and are not standardized. If data structures are required these shall be accessible via the *Module Interlink Header File* as well.

The RTE generator is required [SWS_Rte_07505] to limit the contents of the *Module Interlink Header* file to only that information that is relevant to that instance of a basic software module. This requirement includes the definition of the API.

[SWS_Rte_07510] ⌈ Only *Basic Software Scheduler* API calls that are valid for the particular instance of a basic software module shall be defined within the modules *Module Interlink Header File*. ⌋*(SRS_BSW_00415, SRS_Rte_00017)*

Requirement [SWS_Rte_07510] ensures that attempts to invoke invalid API calls will be rejected as a compile-time error [SRS_Rte_00017].

[SWS_Rte_06534] ⌈ The RTE Generator shall wrap each *Basic Software Scheduler* API definition of a variant existent API according table 4.26 if the variability shall be implemented.

```
#if (<condition> [||<condition>])

<Basic Software Scheduler API Definition>

#endif
```

where `condition` are the condition value macro(s) of the `VariationPoint`s relevant for the conditional existence of the RTE API (see table 4.26), `Basic Software Scheduler API Definition` is the code according an invariant *Basic Software Scheduler* API definition (see also [SWS_Rte_07510], [SWS_Rte_07250], [SWS_Rte_07253], [SWS_Rte_07255], [SWS_Rte_07260], [SWS_Rte_07556], [SWS_Rte_07263], [SWS_Rte_07266]) ⌋*(SRS_Rte_00229)*

The Basic Software Scheduler API for basic software modules is defined in 6.5

[SWS_Rte_07511] ⌈ The *Basic Software Scheduler* API of the particular *Basic Software Module* instance shall be implemented as functions if the basic software module is delivered as object code. ⌋*(SRS_BSW_00342)*

In case of basic software modules delivered as source code the definitions of the *Basic Software Scheduler* API contained in the *Module Interlink Header File* can be optimized during the "RTE Generation" phase when the mapping of the `BswSchedulableEntity`s to OS Tasks is known.

6.3.2.3.3 Provide activating Bsw event

The provide activating event feature is enabled if the executable entity has at least one activationReason defined.

[SWS_Rte_08056] ⌈ If the provide activating event feature is enabled, the RTE generator in contract phase shall generate the executable entity signature according to [SWS_Rte_07282] and [SWS_Rte_08071]. ⌋*(SRS_Rte_00238)*

[SWS_Rte_08057] ⌈ If the provide activating event feature is enabled, the RTE generator in contract phase shall generate the type SchM_ActivatingEvent_<name> (activation vector), where <name> is the symbol describing the executable entity's entry point, to store the activation bits. Based on the highest value of ExecutableEntityActivationReason.bitPosition for this executable entity the type shall be either uint8, uint16, or uint32 so that the highest value of bitPosition fits into the data type. ⌋*(SRS_Rte_00238)*

Note that it is considered an invalid configuration if ExecutableEntityActivationReason.bitPosition has a value higher than 31 (see [constr_1226] in software component template [2]).

[SWS_Rte_08058] ⌈ If the provide activating event feature is enabled, the RTE generator in contract phase shall generate for each ExecutableEntityActivationReason of one executable entity a definition to provide the specific bit position in the Rte_ActivatingEvent_<name> data type:

```
#define SchM_ActivatingEvent_<name>_<activation> xxU
```

The value of xx is defined by the bitPosition xx = 2^bitPosition. ⌋*(SRS_Rte_00238)*

For further details see section 4.2.3.3 Provide activating RTE event.

6.3.2.3.4 RunnableEntity mapped to BswModuleEntity

In the case that a RunnableEntity is mapped to a BswSchedulableEntity the RTE Generator emits a *Entry Point Prototype* for the RunnableEntity (5.7.2) as well as a *Entry Point Prototype* for the BswSchedulableEntity (6.3.2.3.1). This requires that both *Entry Point Prototypes* are compatible. Since RunnableEntity and BswModuleEntry define a overlapping set of attributes its technically possible to have redundancy in the AUTOSAR models between the BSW Module Description and the Software Component Description. In order to support a non redundant M1 model the RTE Generator has to determined common attributes from the BswModuleEntity and apply them to the mapped RunnableEntity.

[SWS_Rte_06731] ⌈ The RTE Generator shall determine the attribute values of

- RunnableEntity.symbol
- RunnableEntity.minimumStartInterval

- RunnableEntity.canBeInvokedConcurrently
- RunnableEntity.swAddrMethod

from the mapped BswModuleEntity and its referred BswModuleEntry if an applicable SwcBswRunnableMapping exists for the RunnableEntity. ⌋

Nevertheless if the attribute values are defined at both places for RunnableEntity and the mapped BswModuleEntity the values have to be consistent.

[SWS_Rte_06732] ⌈ The RTE generator shall reject configurations violating the [constr_4071]. ⌋*(SRS_Rte_00018)*

Within the scope of a SwcBswRunnableMapping both RTEEvents and BswEvents are applicable. Therefore the ExecutableEntityActivationReasons of the RunnableEntity and the mapped BswModuleEntity have to be overlayed.

[SWS_Rte_08071] ⌈ The signature of a RunnableEntity and a BswModuleEntity with a SwcBswRunnableMapping shall contain all ExecutableEntityActivationReasons that are defined for each entity. ⌋*(SRS_Rte_00238)*

Note: Multiple definition of identical activationReasons with respect to shortName and bitPosition yields to a valid configuration since both RunnableEntitys and BswModuleEntitys may provide separate activationReasons.

6.4 API Data Types

Besides the API functions for accessing *Basic Software Scheduler* services, the API also contains *Basic Software Scheduler* specific data types.

6.4.1 Predefined Error Codes for Std_ReturnType

The specification in [29] specifies a standard API return type Std_ReturnType. The Std_ReturnType defines the "'status"' and "'error values"' returned by API functions. It is defined as a uint8 type. The value “0” is reserved for “No error occurred”.

Symbolic name	Value	Comments
[SWS_Rte_07289] ⌈ SCHM_E_OK ⌋*(SRS_BSW_00327)*	0	No error occurred.
[SWS_Rte_07290] ⌈ SCHM_E_LIMIT ⌋*(SRS_BSW_00327)*	130	A internal *Basic Software Scheduler* limit has been exceeded. Request could not be handled. OUT buffers are not modified. Note: The value has to be identically with [SWS_Rte_01317]

Symbolic name	Value	Comments
[SWS_Rte_07562] ⌈ SCHM_E_NO_DATA ⌋*(SRS_BSW_00327)*	131	An explicit read API call returned no data. (This is no error.) Note: The value has to be identically with [SWS_Rte_01061]
[SWS_Rte_07563] ⌈ SCHM_E_TRANSMIT_ACK ⌋*(SRS_BSW_00327)*	132	Transmission acknowledgement received. Note: The value has to be identically with [SWS_Rte_01065]
[SWS_Rte_02747] ⌈ SCHM_E_IN_EXCLUSIVE_AREA ⌋*(SRS_BSW_00327)*	135	The error is returned by a blocking API and indicates that the schedulable entity could not enter a wait state, because one ExecutableEntity of the current task's call stack has entered an ExclusiveArea. Note: There are no blocking SchM APIs and therefore this value cannot be returned. It is defined here for future use and for consistency with [SWS_Rte_02739]. Both error values have to be identically.
[SWS_Rte_07054] ⌈ SCHM_E_TIMEOUT ⌋*(SRS_BSW_00327)*	129	The configured timeout exceeds before the intended result was ready. Note: The value has to be identically with [SWS_Rte_01064]

Table 6.1: Basic Software Scheduler Error and Status values

The underlying type for `Std_ReturnType` is defined as a `uint8` for reasons of compatibility. Consequently, `#define` is used to declare the error values:

```
typedef uint8 Std_ReturnType; /* defined in Std_Types.h */

#define SCHM_E_OK 0U
```

[SWS_Rte_07291] ⌈ The errors as defined in table 6.1 shall be defined in the *RTE Header File*. ⌋*(SRS_Rte_00051)*

An `Std_ReturnType` value can be directly compared (for equality) with the above pre-defined error identifiers.

6.4.2 Basic Software Modes

An `Rte_ModeType` is used to hold the identifiers for the `ModeDeclaration`s of a `ModeDeclarationGroup`.

[SWS_Rte_07292] ⌈ For each ModeDeclarationGroup, the *Module Interlink Types Header File* shall contain a type definition

```
#ifndef RTE_MODETYPE_<ModeDeclarationGroup>
#define RTE_MODETYPE_<ModeDeclarationGroup>
typedef <type> Rte_ModeType_<ModeDeclarationGroup>;
#endif
```

where `<ModeDeclarationGroup>` is the short name of the ModeDeclarationGroup and `<type>` is the shortName of the mapped ImplementationDataType.

⌋*(SRS_Rte_00213)*

Note: This requirement is deprecated to avoid incompatible or duplicate type definitions (see [SWS_Rte_07260]).

Within the Rte_ModeType_<ModeDeclarationGroup>, the (Rte_ModeType_<ModeDeclarationGroup>)<n> value, where <n> is the number of modes declared within the group, is reserved to express a transition between modes.

[SWS_Rte_07293] ⌈ For each ModeDeclarationGroup of category "ALPHABETIC_ORDER", the *Module Interlink Types Header File* shall contain a definition

```
#ifndef RTE_TRANSITION_<prefix><ModeDeclarationGroup>
#define RTE_TRANSITION_<prefix><ModeDeclarationGroup> \
  <n>U
#endif
```

where <ModeDeclarationGroup> is the short name of the ModeDeclarationGroup[1],

<prefix> is the optional prefix attribute defined by the IncludedModeDeclarationGroupSet referring the ModeDeclarationGroup and

<n> is the number of modes declared within the group. ⌋*(SRS_Rte_00213)*

[SWS_Rte_08600] ⌈ For each ModeDeclarationGroup of category "EXPLICIT_ORDER", the *Module Interlink Types Header File* shall contain a definition

```
#ifndef RTE_TRANSITION_<prefix><ModeDeclarationGroup>
#define RTE_TRANSITION_<prefix><ModeDeclarationGroup> \
      <onTransitionValue>U
#endif
```

where <ModeDeclarationGroup> is the short name of the ModeDeclarationGroup[2],

<prefix> is the optional prefix attribute defined by the IncludedModeDeclarationGroupSet referring the ModeDeclarationGroup and

[1] No additional capitalization is applied to the names.
[2] No additional capitalization is applied to the names.

<onTransitionValue> is the onTransitionValue of the *ModeDeclarationGroup*.
⌋*(SRS_Rte_00213)*

[SWS_Rte_07294] ⌈ For each mode of a ModeDeclarationGroup of category "ALPHABETIC_ORDER", the *Module Interlink Types Header File* shall contain a definition

```
#ifndef RTE_MODE_<prefix><ModeDeclarationGroup>_<ModeDeclaration>
#define RTE_MODE_<prefix><ModeDeclarationGroup>_<ModeDeclaration> \
        <index>U
#endif
```

where <ModeDeclarationGroup> is the short name of the ModeDeclarationGroup,

<prefix> is the optional prefix attribute defined by the IncludedModeDeclarationGroupSet referring the ModeDeclarationGroup

<ModeDeclaration> is the short name of a ModeDeclaration[3],

and <index> is the index of the ModeDeclarations in alphabetic ordering (ASCII / ISO 8859-1 code in ascending order) of the short names within the ModeDeclarationGroup.

The lowest index shall be '0' and therefore the range of assigned values is 0..<n> where <n> is the number of modes declared within the group ⌋*(SRS_Rte_00213)*

[SWS_Rte_08601] ⌈ For each mode of a ModeDeclarationGroup of category "EXPLICIT_ORDER", the *Module Interlink Types Header File* shall contain a definition

```
#ifndef RTE_MODE_<prefix><ModeDeclarationGroup>_<ModeDeclaration>
#define RTE_MODE_<prefix><ModeDeclarationGroup>_<ModeDeclaration> \
        <value>U
#endif
```

where <ModeDeclarationGroup> is the short name of the ModeDeclarationGroup,

<prefix> is the optional prefix attribute defined by the IncludedModeDeclarationGroupSet referring the ModeDeclarationGroup

<ModeDeclaration> is the short name of a ModeDeclaration[4],

and <value> is the *value* specified at the ModeDeclaration. ⌋*(SRS_Rte_00213)*

6.5 API Reference

This chapter defines the "interface" between a particular instance of a *Basic Software Module* and the *Basic Software Scheduler*. The wild-card <bsnp> is the *BSW Scheduler Name Prefix* according [SWS_Rte_07593] and [SWS_Rte_07594].

[3] No additional capitalization is applied to the names.
[4] No additional capitalization is applied to the names.

6.5.1 SchM_Enter

Purpose: SchM_Enter function enters an exclusive area of an *Basic Software Module*.

Signature: **[SWS_Rte_07250]** ⌈

```
void SchM_Enter_<bsnp>[_<vi>_<ai>]_<name>()
```

Where here

<bsnp> is the *BSW Scheduler Name Prefix* according [SWS_Rte_07593] and [SWS_Rte_07594],

<vi> is the vendorId of the calling BSW module,

<ai> vendorApiInfix of the calling BSW module and

<name> name is the exclusive area name.

The sub part in squared brackets [_<vi>_<ai>] is omitted if no vendorApiInfix is defined for the *Basic Software Module*. See [SWS_Rte_07528]. ⌋*(SRS_Rte_00222, SRS_BSW_00347, SRS_Rte_00046)*

Existence: **[SWS_Rte_07251]** ⌈ A SchM_Enter API shall be created for each ExclusiveArea that is declared in the BswInternalBehavior and which has an canEnterExclusiveArea association. ⌋*(SRS_Rte_00222, SRS_Rte_00046)*

Description: The SchM_Enter API call is invoked by an AUTOSAR BSW module to define the start of an exclusive area.

Return Value: None.

Notes: The *Basic Software Scheduler* is not required to support nested invocations of SchM_Enter for the same exclusive area.

[SWS_Rte_07252] ⌈ The *Basic Software Scheduler* shall permit calls to SchM_Enter and SchM_Exit to be nested as long as different exclusive areas are exited in the reverse order they were entered. ⌋*(SRS_Rte_00222, SRS_Rte_00046)*

[constr_9046] SchM_Enter and SchM_Exit API may only be used by BswModuleEntitys describing its usage ⌈ The SchM_Enter and SchM_Exit API may only be used by BswModuleEntitys that contain a corresponding canEnterExclusiveArea association ⌋

[constr_9047] Nested call of SchM_Enter and SchM_Exit API is restricted ⌈ The SchM_Enter and SchM_Exit API may only be called nested if different exclusive areas are invoked; in this case exclusive areas shall exited in the reverse order they were entered. ⌋

[SWS_Rte_07578] ⌈ The *Basic Software Scheduler* shall support calls of SchM_Enter and SchM_Exit after initialization of the OS but before the *Basic Software Scheduler* is initialized. ⌋*(SRS_Rte_00222, SRS_Rte_00046)*

[SWS_Rte_07579] ⌈ The *Basic Software Scheduler* shall support calls of SchM_Enter and SchM_Exit in the context of os tasks, category 1 and category 2 interrupts. ⌋*(SRS_Rte_00222, SRS_Rte_00046)*

Note: the possible implementation mechanism for such an exclusive area is limited in this case to mechanism available for the related kind of context. For instance SuspendAllInterrupts and ResumeAllInterrupts service of the OS are available for all kind of context but GetResource and ReleaseResource is only available for tasks and category 2 interrupts.

Within the *AUTOSAR OS* an attempt to lock a resource cannot fail because the lock is already held. The lock attempt can only fail due to configuration errors (e.g. caller not declared as accessing the resource) or invalid handle. Therefore the return type from this function is void.

Mutual exclusion of tasks requesting the same exclusive area shall be ensured across partition and core boundaries.

6.5.2 SchM_Exit

Purpose: SchM_Exit function leaves an exclusive area of an *Basic Software Module*.

Signature: **[SWS_Rte_07253]** ⌈

```
void
SchM_Exit_<bsnp>[_<vi>_<ai>]_<name>()
```

Where here

<bsnp> is the *BSW Scheduler Name Prefix* according [SWS_Rte_07593] and [SWS_Rte_07594],

<vi> is the vendorId of the calling BSW module,

<ai> vendorApiInfix of the calling BSW module and

<name> name is the exclusive area name.

The sub part in squared brackets [_<vi>_<ai>] is omitted if no vendorApiInfix is defined for the *Basic Software Module*. See [SWS_Rte_07528]. ⌋*(SRS_Rte_00222, SRS_BSW_00347, SRS_Rte_00046)*

Existence: **[SWS_Rte_07254]** ⌈ A SchM_Exit API shall be created for each ExclusiveArea that is declared in the BswInternalBehavior and which has an canEnterExclusiveArea association.. ⌋*(SRS_Rte_00222, SRS_Rte_00046)*

Description: The SchM_Exit API call is invoked by an AUTOSAR BSW module to define the end of an exclusive area.

Return Value: None.

Notes: The *Basic Software Scheduler* is not required to support nested invocations of SchM_Exit for the same exclusive area.

Requirement [SWS_Rte_07252] permits calls to SchM_Exit and SchM_Exit to be nested as long as different exclusive areas are exited in the reverse order they were entered.

[constr_9048] SchM_Exit API may only be used by BswModuleEntitys that describe its usage ⌈ The SchM_Exit API may only be used by BswModuleEntitys that contain a corresponding canEnterExclusiveArea association ⌋

6.5.3 SchM_Call

Purpose: Invokes a Client-Server operation between BSW modules, possibly crossing partition boundaries.

Signature: **[SWS_Rte_08733]** ⌈

```
Std_ReturnType SchM_Call_<bsnp>[_<vi>_<ai>]_<name>(
                    [OUT <typeOfReturnValue> returnValue]
                    [IN|IN/OUT|OUT]<data_1>...
                    [IN|IN/OUT|OUT] <data_n>)
```

where there is a BSW module providing a entry which is the base for a generated function

```
<typeOfReturnValue> <bsnp>[_<vi>_<ai>]_<name>(
    <data_1>...<data_n>)
```

with <typeOfReturnValue> is the returnType of the referenced BswModuleEntry. If the returnType of the referenced BswModuleEntry is of type void or execution is asynchronous, this part should be omitted.

<bsnp> is the BSW Scheduler Name Prefix of the BSW module providing the entry according to [SWS_Rte_07593] and [SWS_Rte_07594],

<vi> is the vendorId of the BSW module providing the entry,

<ai> is the vendorApiInfix of the BSW module providing the entry,

<name> is the API name of the provided server entry according to [SWS_BSW_00148].

The sub part in square brackets [_<vi>_<ai>] is omitted if no vendorApiInfix is defined for the Basic Software Module. See [SWS_Rte_07528]. ⌋*(SRS_Rte_00243)*

Existence:

[SWS_Rte_08734] ⌈ A synchronous SchM_Call API shall be generated if a callPoint association to a BswSynchronousServerCallPoint exists and the BswSynchronousServerCallPoint references a BswModuleClientServerEntry as calledEntry and this BswModuleClientServerEntry is referenced by the BswModuleDescription as a requiredClientServerEntry. ⌋*(SRS_Rte_00243)*

[SWS_Rte_08735] ⌈ An asynchronous SchM_Call API shall be generated if a callPoint association to a BswAsynchronousServerCallPoint exists and the BswAsynchronousServerCallPoint references a BswModuleClientServerEntry as calledEntry and this BswModuleClientServerEntry is referenced by the BswModuleDescription as a requiredClientServerEntry. ⌋*(SRS_Rte_00243)*

A configuration that includes both synchronous and asynchronous Call Points is invalid.

[constr_9079] SchM_Call API may only be used by the BswModuleEntity that describe its usage ⌈ The SchM_Call API may only be used within the BswModuleEntity that references the corresponding BswSynchronousServerCallPoint respectively BswAsynchronousServerCallPoint using a callPoint association. ⌋

Description:

Function to initiate Client-Server communication between BSW modules. The SchM_Call API is used for both synchronous and asynchronous calls.

When the BswModuleClientServerEntry is called the SchM shall invoke the referenced BswModuleEntry providing the C-function with the signature <bpns>[_<vi>_<ai>]_name(<data_1>...(<data_n>) on the partition of the task assigned to the respective BswOperationInvokedEvent, or on the local partition if the BswOperationInvokedEvent is not mapped to a task.

[SWS_Rte_08736] ⌈ The OUT parameter shall exist if the returnType of BswModuleEntry is different of void and the attribute isSynchronous of the BswModuleEntry is set to true. ⌋(SRS_Rte_00243)

[SWS_Rte_08737] ⌈ The datatype of the OUT parameter returnValue shall be equal to returnType of the called BswModuleEntry. ⌋(SRS_Rte_00243)

[SWS_Rte_08738] ⌈ The return value of the called BswModuleEntry shall be returned inside the OUT parameter returnValue. ⌋(SRS_Rte_00243)

[SWS_Rte_08739] ⌈ The SchM shall ensure that the BswModuleEntity implementing a server operation has completed the processing of a request before it begins processing the next request, if call serialization is required by the server operation, i.e the isReentrant attribute of the corresponding BswModuleClientServerEntry which is referenced as providedClientServerEntry is set to false and more than one BswModuleClientServerEntry in the role requiredClientServerEntry references this server. If the SchM_Call crosses partition borders, the call is mapped to IOCSend_<id>(). ⌋(SRS_Rte_00243)

The pointers to all parameters passed by reference must remain valid until the API call returns.

Return Value: **[SWS_Rte_08740]** ⌈ The return value shall be used to indicate infrastructure errors detected by the RTE during execution of the SchM_Call call and, for synchronous communication, infrastructure and application errors during execution of the BswModuleEntry. ⌋

- **[SWS_Rte_08741]** ⌈ SCHM_E_OK - The API call completed successfully. ⌋
- **[SWS_Rte_08742]** ⌈ SCHM_E_LIMIT - There are multiple outstanding asynchronous calls of the same BswModuleEntry. The invocation shall be discarded, the buffers of the return parameters shall not be modified. ⌋

6.5.4 SchM_Result

Purpose: Get the result of an asynchronous call of a BswModuleEntry.

Signature: **[SWS_Rte_08743]** ⌈

```
Std_ReturnType
SchM_Result_<bsnp>[_<vi>_<ai>]_<name>(
                 [IN|IN/OUT|OUT]<data_1> ...
                 [IN|IN/OUT|OUT] <data_n>)
```

where there is a BSW module providing a entry which is the base for a generated function `<bsnp>[_<vi>_<ai>]_name(<data_1>...<data_n>)`

with `<bsnp>` is the BSW Scheduler Name Prefix of the BSW module sending the callback according to [SWS_Rte_07593] and [SWS_Rte_07594],

`<vi>` is the `vendorId` of the BSW module sending the callback,

`<ai>` is the `vendorApiInfix` of the BSW module sending the callback,

`<name>` is the API name of the provided server entry according to [SWS_BSW_00148] (i.e. the same name as in the `SchM_Call` method causing this callback).

The sub part in squared brackets `[_<vi>_<ai>]` is omitted if no `vendorApiInfix` is defined for the *Basic Software Module*. See [SWS_Rte_07528]. ⌋*(SRS_Rte_00243)*

Existence: **[SWS_Rte_08744]** ⌈ A blocking `SchM_Result` API shall be generated if a `callPoint` association to a `BswAsynchronousServerCallResultPoint` exists. ⌋*(SRS_Rte_00243)*

[constr_9076] SchM_Result API may only be used by the BswModuleEntity that describe its usage ⌈ The `SchM_Result` API may only be used within the `BswModuleEntity` that references the corresponding `BswAsynchronousServerCallResultPoint` using a `callPoint` association. ⌋

Description: The `SchM_Result` is used to collect the result of an asynchronous call of a `BswModuleEntry` invoked by `SchM_Call_<bsnp>[_<vi>_<ai>]_name(<data_1>...<data_n>)`.

Using `SchM_Result` it is possible get back the result of call.

The `SchM_Result` API includes zero or more IN/OUT and OUT parameters to pass back results.

The pointers to all parameters passed by reference must remain valid until the API call returns.

If the `SchM_Result` crosses partition borders, the callback is mapped to `IOCSend_<id>()`.

Return Value: The return value is used to indicate errors from either the `SchM_Result` call itself or communication errors detected before the API call was made.

- **[SWS_Rte_08745]** ⌈ SCHM_E_OK - The API call completed successfully. ⌋
- **[SWS_Rte_08746]** ⌈ SCHM_E_NO_DATA - The BswModuleEntry's result is not available but no other error occurred within the API call or the BswModuleEntry was not called using SchM_Call. The buffers for the IN/OUT and OUT parameters shall not be modified. ⌋
- **[SWS_Rte_08759]** ⌈ SCHM_E_LIMIT - The BswModuleEntry's result could not be transfered because no buffer was available while transferring the data. The invocation shall be discarded, the buffers of the return parameters shall not be modified. ⌋

The SCHM_E_NO_DATA return value is not considered to be an error but rather indicate correct operation of the API call. When SCHM_E_NO_DATA occurs, a BSW module is free to invoke SchM_Result again and thus repeat the attempt to read the result.

6.5.5 SchM_Send

Purpose: Initiate an "explicit" sender-receiver transmission of data elements with "event" semantic (queued) between BSW modules.

Signature: **[SWS_Rte_08747]** ⌈

```
Std_ReturnType
SchM_Send_<bsnp>[_<vi>_<ai>]_<name>(IN <data>)
```

with <bsnp> is the BSW Scheduler Name Prefix of the BSW module providing the data according to [SWS_Rte_07593] and [SWS_Rte_07594],

<vi> is the vendorId of the BSW module providing the data,

<ai> is the vendorApiInfix of the BSW module providing the data,

<name> is the shortName of the VariableDataPrototype of this sender-receiver connection.

The sub part in square brackets [_<vi>_<ai>] is omitted if no vendorApiInfix is defined for the *Basic Software Module*. See [SWS_Rte_07528]. ⌋*(SRS_Rte_00243)*

Existence: **[SWS_Rte_08748]** ⌈ The existence of a dataSendPoint association to a providedData VariableDataPrototype shall result in the generation of a SchM_Send API for the provided VariableDataPrototype. ⌋*(SRS_Rte_00243)*

[constr_9077] SchM_Send API may only be used by the BswModuleEntity that describes its usage ⌈ The SchM_Send API may only be used within the BswModuleEntity that references the VariableDataPrototype using a dataSendPoint. ⌋

Description: When a BSW module writes data to a sender-receiver connection on a system with the BSW running on multiple partitions, it shall invoke SchM_Send_<bsnp>[_<vi>_<ai>]_<name>(<data>). The SchM_Send API call initiates a sender-receiver communication where the transmission occurs at the point the API call is made (cf. explicit transmission). The SchM_Send API call includes the IN parameter <data> to pass the data element to write. The IN parameter <data> is passed by value or reference according to the ImplementationDataType as described in the section 5.2.6.5. If the IN parameter <data> is passed by reference, the pointer must remain valid until the API call returns.

Return Value: The return value is used to indicate errors detected by the SchM during execution of the SchM_Send.

- **[SWS_Rte_08749]** ⌈ SCHM_E_OK - data passed to communication service successfully. ⌋
- **[SWS_Rte_08750]** ⌈ SCHM_E_LIMIT - an 'event' has been discarded due to a full queue by one of the partition local receivers. ⌋

Notes: The SchM_Send API is used to transmit data with "events" semantics which means that they are getting queued.

[SWS_Rte_08751] ⌈ In case of inter partition communication, the SchM_Send API call shall cause an immediate transmission request. ⌋*(SRS_Rte_00243)*

For inter-partition communication the IOC can be used for transmitting the data to the other partition.

[SWS_Rte_08752] ⌈ If the VariableDataPrototype in the providedData role is connected to multiple VariableDataPrototypes in the role requiredData, then the SchM shall ensure that writes to all receivers are independent. ⌋*(SRS_Rte_00243)*

This ensures that an error detected by the SchM when writing to one receiver does not prevent the transmission of this message to other BSW modules.

[SWS_Rte_08753] ⌈ In case of intra partition communication, the SchM_Send API call shall return after copying the data to RTE local memory or using IOC buffers. ⌋*(SRS_Rte_00243)*

6.5.6 SchM_Receive

Purpose: Perfoms an "explicit" sender-receiver reception of data elements with "event" semantic (queued) between BSW modules.

Signature: **[SWS_Rte_08754]** ⌈

```
Std_ReturnType
SchM_Receive_<bsnp>[_<vi>_<ai>]_<name>(OUT <data>)
```

with <bsnp> is the BSW Scheduler Name Prefix of the BSW module reading the data according to [SWS_Rte_07593] and [SWS_Rte_07594],

<vi> is the vendorId of the BSW module reading the data,

<ai> is the vendorApiInfix of the BSW module reading the data,

<name> is the shortName of the VariableDataPrototype of this sender-receiver connection.

The sub part in square brackets [_<vi>_<ai>] is omitted if no vendorApiInfix is defined for the *Basic Software Module*. See [SWS_Rte_07528]. ⌋*(SRS_Rte_00243)*

Existence: **[SWS_Rte_08755]** ⌈ The existence of a dataReceivePoint association to a requiredData VariableDataPrototype shall result in the generation of a SchM_Receive API for the required VariableDataPrototype. ⌋*(SRS_Rte_00243)*

[constr_9078] SchM_Receive API may only be used by the BswModuleEntity that describes its usage ⌈ The SchM_Receive API may only be used within the BswModuleEntity that references the VariableDataPrototype using a dataReceivePoint. ⌋

Description: When a BSW module handles a BswDataReceivedEvent on a system with the BSW running on multiple partitions, it shall invoke SchM_Receive_<bsnp>[_<vi>_<ai>]_<name>(<data>). For a sender-receiver connection crossing partition boundaries, the SchM shall then read the data from a shared buffer, where it has been put by SchM_Send.

The SchM_Receive API call includes the OUT parameter <data> to pass back the received data element.

The pointers to the OUT parameters must remain valid until the API call returns.

Return Value: The return value is used to indicate errors detected by the SchM during execution of the SchM_Receive or errors detected by the communication system.

- **[SWS_Rte_08757]** ⌈ SCHM_E_OK - data read successfully. ⌋

- **[SWS_Rte_08758]** ⌈ SCHM_E_NO_DATA - no "events" (means queued data) were received and no other error occurred when the read was attempted. ⌋

[SWS_Rte_08756] ⌈ In case return value is SCHM_E_NO_DATA the OUT parameters shall remain unchanged. ⌋*(SRS_Rte_00243)*

The SCHM_E_NO_DATA return value is not considered to be an error but rather indicates correct operation of the API call.

6.5.7 SchM_Switch

Purpose: Initiate a mode switch. The SchM_Switch API call is used for sending of a mode switch notification by a *Basic Software Module*.

Signature: **[SWS_Rte_07255]** ⌈

```
Std_ReturnType
SchM_Switch_<bsnp>[_<vi>_<ai>]_<name>(
                  IN <mode>)
```

Where here

<bsnp> is the *BSW Scheduler Name Prefix* according [SWS_Rte_07593] and [SWS_Rte_07594],

<vi> is the vendorId of the calling BSW module,

<ai> vendorApiInfix of the calling BSW module and

<name> is the provided (providedModeGroup) ModeDeclarationGroupPrototype name.

The sub part in squared brackets [_<vi>_<ai>] is omitted if no vendorApiInfix is defined for the *Basic Software Module*. See [SWS_Rte_07528]. ⌋*(SRS_Rte_00215, SRS_BSW_00347)*

Existence: **[SWS_Rte_07256]** ⌈ The existence of a managedModeGroup association to a providedModeGroup ModeDeclarationGroupPrototype shall result in the generation of a SchM_Switch API. ⌋*(SRS_Rte_00215)*

[constr_9049] SchM_Switch API may only be used by BswModuleEntitys that describe its usage ⌈ The SchM_Switch API may only be used by BswModuleEntitys that contain a corresponding managedModeGroup association ⌋

Description: The SchM_Switch triggers a mode switch for all connected required (requiredModeGroup) ModeDeclarationGroupPrototypes.

The SchM_Switch API call includes exactly one IN parameter for the next mode <mode>. The IN parameter <mode> is passed by value according to the ImplementationDataType on which the ModeDeclarationGroup is mapped. The type name shall be equal to the ImplementationDataType symbol.

Return Value: The return value is used to indicate errors detected by the *Basic Software Scheduler* during execution of the SchM_Switch call.

- **[SWS_Rte_07258]** ⌈ SCHM_E_OK – data passed to service successfully. ⌋*(SRS_Rte_00213, SRS_Rte_00214, SRS_Rte_00094)*
- **[SWS_Rte_07259]** ⌈ SCHM_E_LIMIT – a mode switch has been discarded due to a full queue. ⌋*(SRS_Rte_00213, SRS_Rte_00214, SRS_Rte_00143)*

Notes: SchM_Switch is restricted to ECU local communication.

If a mode instance is currently involved in a transition then the SchM_Switch API will attempt to queue the request and return [SWS_Rte_02667]. However if no transition is in progress for the mode instance, the mode disablings and the activations of *OnEntry*, *OnTransition*, and *OnExit* runnables for this mode instance are executed before the SchM_Switch API returns [SWS_Rte_02665].

Note that the mode switch might be discarded when the queue is full and a mode transition is in progress, see [SWS_Rte_02675].

[SWS_Rte_07286] ⌈ If the mode switched acknowledgment is enabled, the RTE shall notify the mode manager when the mode switch is completed. ⌋*(SRS_Rte_00213, SRS_Rte_00214, SRS_Rte_00122)*

6.5.8 SchM_Mode

There exist two versions of the SchM_Mode APIs. Depending on the attribute enhancedModeApi in the *basic software module description* there shall be provided different versions of this API (see also 6.5.9).

Purpose: Provides the currently active mode of a (requiredModeGroup or providedModeGroup) ModeDeclarationGroupPrototype.

Signature: **[SWS_Rte_07260]** ⌈

```
<return>
SchM_Mode_<bsnp>[_<vi>_<ai>]_<name>()
```

Where here

<bsnp> is the *BSW Scheduler Name Prefix* according [SWS_Rte_07593] and [SWS_Rte_07594],

<vi> is the vendorId of the calling BSW module,

<ai> vendorApiInfix of the calling BSW module and

<name> is the (requiredModeGroup or providedModeGroup) ModeDeclarationGroupPrototype name.

The sub part in squared brackets [_<vi>_<ai>] is omitted if no vendorApiInfix is defined for the *Basic Software Module*. See [SWS_Rte_07528]. ⌋*(SRS_Rte_00213, SRS_BSW_00347)*

Existence: **[SWS_Rte_07261]** ⌈ If a accessedModeGroup association to a providedModeGroup or requiredModeGroup ModeDeclarationGroupPrototype exists and if the attribute enhancedModeApi of the BswModeSenderPolicy resp. BswModeReceiverPolicy is set to *false* a SchM_Mode API according to [SWS_Rte_07260] shall be generated. ⌋*(SRS_Rte_00215)*

Note: This ensures the availability of the SchM_Mode API for the mode manager and mode user

[constr_9050] SchM_Mode API may only be used by BswModuleEntitys that describe its usage ⌈ The SchM_Mode API may only be used by BswModuleEntitys that contain a corresponding managedModeGroup association or accessedModeGroup association ⌋

Description: The SchM_Mode API tells the *Basic Software Module* which mode of a required or provided ModeDeclarationGroupPrototype is currently active. This is the information that the *RTE* uses for the ModeDisablingDependencys. A new mode will not be indicated immediately after the reception of a mode switch notification from a mode manager, see section 4.4.4.During mode transitions, i.e. during the execution of runnables that are triggered on exiting one mode or on entering the next mode, overlapping mode disablings of two modes are active. In this case, the SchM_Mode API will return RTE_TRANSITION_<ModeDeclarationGroup>.

The SchM_Mode will return the same mode for all required or provided ModeDeclarationGroupPrototypes that are connected. (see [SWS_Rte_02630]).

Return Value: The return type of SchM_Mode is dependent on the ImplementationDataType of the ModeDeclarationGroup. It shall return the value of the ModeDeclarationGroupPrototype. The type name shall be equal to the ImplementationDataType symbol.

[SWS_Rte_07262] ⌈ The SchM_Mode API shall return the following values:

- during mode transitions:
  RTE_TRANSITION_<ModeDeclarationGroup>,

  where <ModeDeclarationGroup> is the short name of the ModeDeclarationGroup.

- else:
  RTE_MODE_<ModeDeclarationGroup>_<ModeDeclaration>,

  where <ModeDeclarationGroup> is the short name of the ModeDeclarationGroup and <ModeDeclaration> is the short name of the currently active ModeDeclaration

⌋*(SRS_Rte_00144)*

Notes: None.

6.5.9 Enhanced SchM_Mode

Purpose: Provides the currently active mode of a (requiredModeGroup or providedModeGroup) ModeDeclarationGroupPrototype. If the corresponding mode machine instance is in transition additionally the values of the previous and the next mode are provided.

Signature: **[SWS_Rte_07694]** ⌈

```
<return>
SchM_Mode_<bsnp>[_<vi>_<ai>]_<name>(
                OUT <previousmode>,
                OUT <nextmode>)
)
```

Where here

<bsnp> is the *BSW Scheduler Name Prefix* according [SWS_Rte_07593] and [SWS_Rte_07594],

<vi> is the vendorId of the calling BSW module,

<ai> vendorApiInfix of the calling BSW module and

<name> is the (requiredModeGroup or providedModeGroup) ModeDeclarationGroupPrototype name.

The sub part in squared brackets [_<vi>_<ai>] is omitted if no vendorApiInfix is defined for the *Basic Software Module*. See [SWS_Rte_07528]. ⌋*(SRS_Rte_00213, SRS_BSW_00347)*

Existence: **[SWS_Rte_08507]** ⌈ The existence of a accessedModeGroup association to a providedModeGroup or requiredModeGroup ModeDeclarationGroupPrototype given that the attribute enhancedModeApi of the BswModeSenderPolicy resp. BswModeReceiverPolicy is set to *true* a SchM_Mode API according to [SWS_Rte_07694] shall be generated. ⌋*(SRS_Rte_00215)*

Note: This ensures the availability of the SchM_Mode API for the mode manager and mode user

[constr_9051] SchM_Mode API may only be used by BswModuleEntitys that describe its usage ⌈ The SchM_Mode API may only be used by BswModuleEntitys that contain a corresponding managedModeGroup association or accessedModeGroup association ⌋

Description: The SchM_Mode API tells the *Basic Software Module* which mode of a required or provided ModeDeclarationGroupPrototype is currently active. This is the information that the *RTE* uses for the ModeDisablingDependencys. A new mode will not be indicated immediately after the reception of a mode switch notification from a mode manager, see section 4.4.4.During mode transitions, i.e. during the execution of runnables that are triggered on exiting one mode or on entering the next mode, overlapping mode disablings of two modes are active. In this case, the SchM_Mode API will return RTE_TRANSITION_<ModeDeclarationGroup>. The parameter <previousmode> then contains the mode currently being left. The parameter <nextmode> contains the mode being entered.

The SchM_Mode will return the same mode for all required or provided ModeDeclarationGroupPrototypes that are connected. (see [SWS_Rte_02630]).

Return Value: The return type of SchM_Mode is dependent on the ImplementationDataType of the ModeDeclarationGroup. It shall return the value of the ModeDeclarationGroupPrototype. The type name shall be equal to the ImplementationDataType symbol.

[SWS_Rte_08509] ⌈ During transitions SchM_Mode API shall return the following values:

- the return value shall be RTE_TRANSITION_<ModeDeclarationGroup>
- <previousmode> shall contain the RTE_MODE_<ModeDeclarationGroup>_<ModeDeclaration> of the mode being left,

- <nextmode> shall contain the RTE_MODE_<ModeDeclarationGroup>_<ModeDeclaration> of the mode being entered,

where <ModeDeclarationGroup> is the short name of the ModeDeclarationGroup.

⌋(*SRS_Rte_00144*)

[SWS_Rte_08510] ⌈ If the mode machine instance is in a defined mode SchM_Mode shall return the follwing values:

- the return value shall contain the value of the RTE_MODE_<ModeDeclarationGroup>_<ModeDeclaration>,
- <previousmode> shall contain the value of the RTE_MODE_<ModeDeclarationGroup>_<ModeDeclaration>,
- <nextmode> shall contain the the value of RTE_MODE_<ModeDeclarationGroup>_<ModeDeclaration>,

where <ModeDeclarationGroup> is the short name of the ModeDeclarationGroup and <ModeDeclaration> is the short name of the currently active ModeDeclaration.

⌋(*SRS_Rte_00144*)

Notes: None.

6.5.10 SchM_SwitchAck

Purpose: Provide access to acknowledgment notifications for mode communication.

Signature: **[SWS_Rte_07556]** ⌈

```
Std_ReturnType
SchM_SwitchAck_<bsnp>[_<vi>_<ai>]_<name>()
```

Where here

<bsnp> is the *BSW Scheduler Name Prefix* according [SWS_Rte_07593] and [SWS_Rte_07594],

<vi> is the vendorId of the calling BSW module,

<ai> vendorApiInfix of the calling BSW module and

<name> is the required (requiredModeGroup) ModeDeclarationGroupPrototype name.

The sub part in squared brackets [_<vi>_<ai>] is omitted if no vendorApiInfix is defined for the *Basic Software Module*. See [SWS_Rte_07528]. ⌋*(SRS_BSW_00310, SRS_Rte_00213)*

Existence: **[SWS_Rte_07557]** ⌈ Acknowledgement is enabled for a provided (providedModeGroup) ModeDeclarationGroupPrototype by the presence of an ackRequest attribute of the BswModeSenderPolicy. ⌋*(SRS_Rte_00213, SRS_Rte_00122)*

[SWS_Rte_07558] ⌈ A non-blocking SchM_SwitchAck API shall be generated for a provided (providedModeGroup) ModeDeclarationGroupPrototype if acknowledgement is enabled and a managedModeGroup association references the providedModeGroup ModeDeclarationGroupPrototype. ⌋*(SRS_Rte_00213, SRS_Rte_00122)*

[constr_9052] SchM_SwitchAck API may only be used by BswModuleEntitys that describe its usage ⌈ The SchM_SwitchAck API may only be used by BswModuleEntitys that contain a corresponding managedModeGroup association ⌋

Description: The SchM_SwitchAck API takes no parameters – the return value is used to indicate the acknowledgement status to the caller.

Return Value: The return value is used to indicate the "status" status and errors detected by the *Basic Software Scheduler* during execution of the Rte_SwitchAck call.

- **[SWS_Rte_07560]** ⌈ SCHM_E_NO_DATA – (non-blocking read) no error is occurred when the SchM_SwitchAck read was attempted. ⌋*(SRS_Rte_00213, SRS_Rte_00122)*
- **[SWS_Rte_07561]** ⌈ SCHM_E_TRANSMIT_ACK – For communication of mode switches, this indicates, that the BswSchedulableEntitys on the transition have been executed and the mode disablings have been switched to the new mode (see [SWS_Rte_02587]). ⌋*(SRS_Rte_00213, SRS_Rte_00122)*
- **[SWS_Rte_07055]** ⌈ SCHM_E_TIMEOUT The configured timeout exceeds before the mode transition was completed.
  OR:
  The partition of the mode users is stopped or restarting or has been restarted while the mode switch was requested. ⌋*(SRS_Rte_00213, SRS_Rte_00122)*

The SCHM_E_TRANSMIT_ACK return value is not considered to be an error but rather indicates correct operation of the API call.

When SCHM_E_NO_DATA occurs, a *Basic Software Module* is free to reinvoke SchM_SwitchAck and thus repeat the attempt to read the mode switch acknowledgment status.

The SCHM_E_TIMEOUT return value can denote a stopped or restarting partition even for the SchM_SwitchAck API in case of a common mode machine instance.

Notes: If multiple transmissions on the same provided (providedModeGroup) ModeDeclarationGroupPrototype are outstanding it is not possible to determine which is acknowledged first. If this is important, transmissions should be serialized with the next occurring only when the previous transmission has been acknowledged or has timed out.

6.5.11 SchM_Trigger

Purpose: Triggers the activation of connected BswSchedulableEntitys of the same or other *Basic Software Modules*.

Signature: **[SWS_Rte_07263]** ⌈
signature without queuing support:

```
void
SchM_Trigger_<bsnp>[_<vi>_<ai>]_<name>()
```

signature with queuing support:

```
Std_ReturnType
SchM_Trigger_<bsnp>[_<vi>_<ai>]_<name>()
```

Where here

<bsnp> is the *BSW Scheduler Name Prefix* according [SWS_Rte_07593] and [SWS_Rte_07594],

<vi> is the vendorId of the calling BSW module,

<ai> vendorApiInfix of the calling BSW module and

<name> is the released (releasedTrigger) Trigger name.

The sub part in squared brackets [_<vi>_<ai>] is omitted if no vendorApiInfix is defined for the *Basic Software Module*. See [SWS_Rte_07528].

The signature for queuing support shall be generated by the RTE generator if the swImplPolicy of the Trigger is set to queued.
⌋*(SRS_Rte_00218, SRS_BSW_00347)*

Existence: **[SWS_Rte_07264]** ⌈ The existence of a issuedTrigger association to the released (releasedTrigger) Trigger shall result in the generation of a SchM_Trigger API. ⌋*(SRS_Rte_00218)*

[constr_9053] SchM_Trigger API may only be used by the BswModuleEntitys that describe its usage ⌈ The SchM_Trigger API may only be used by the BswModuleEntity that contains the corresponding issuedTrigger association.
⌋

Description: The SchM_Trigger triggers an execution for all BswSchedulableEntitys whose BswExternalTriggerOccurredEvent is associated to connected required Trigger.

Return Value: None in case of signature without queuing support.

[SWS_Rte_06722] ⌈ The SchM_Trigger API shall return the following values:

- SCHM_E_OK if the trigger was successfully queued or if no queue is configured
- SCHM_E_LIMIT if the trigger was not queued because the maximum queue size is already reached.

in the case of signature with queuing support. ⌋*(SRS_Rte_00235)*

Notes: SchM_Trigger is restricted to ECU local communication.

6.5.12 SchM_ActMainFunction

Purpose: Triggers the activation of the BswSchedulableEntity which is associated with an activationPoint of the same or *Basic Software Module*.

Signature: **[SWS_Rte_07266]** ⌈
signature without queuing support:

```
void
SchM_ActMainFunction_<bsnp>[_<vi>_<ai>]_<name>()
```

signature with queuing support:

```
Std_ReturnType
SchM_ActMainFunction_<bsnp>[_<vi>_<ai>]_<name>()
```

Where here

<bsnp> is the *BSW Scheduler Name Prefix* according [SWS_Rte_07593] and [SWS_Rte_07594],

<vi> is the vendorId of the calling BSW module,

<ai> vendorApiInfix of the calling BSW module and

<name> is the associated BswInternalTriggeringPoint short name.

The sub part in squared brackets [_<vi>_<ai>] is omitted if no vendorApiInfix is defined for the *Basic Software Module*. See [SWS_Rte_07528].

The signature for queuing support shall be generated by the RTE generator if the swImplPolicy of the BswInternalTriggeringPoint is set to queued. ⌋*(SRS_Rte_00218, SRS_BSW_00347)*

Existence: **[SWS_Rte_07267]** ⌈ The existence of an activationPoint shall result in the generation of a SchM_ActMainFunction API. ⌋*(SRS_Rte_00218)*

[constr_9054] SchM_ActMainFunction API may only be used by the BswModuleEntitys that describe its usage ⌈ The SchM_ActMainFunction API may only be used by the BswModuleEntity that contains the corresponding activationPoint association. ⌋

Description: The SchM_ActMainFunction triggers an execution for all BswSchedulableEntitys whose BswInternalTriggerOccurredEvent is associated by activationPoint.

Return Value: None in case of signature without queuing support.

[SWS_Rte_06723] ⌈ The SchM_ActMainFunction API shall return the following values:

- SCHM_E_OK if the trigger was successfully queued or if no queue is configured
- SCHM_E_LIMIT if the trigger was not queued because the maximum queue size is already reached.

in the case of signature with queuing support. ⌋*(SRS_Rte_00235)*

Notes: SchM_ActMainFunction is restricted to ECU local communication.

6.5.13 SchM_CData

Purpose: Provide access to the calibration parameter of a *Basic Software Module* defined internally. The ParameterDataPrototype in the role perInstanceParameter is used to define *Basic Software Module* internal calibration parameters. Internal because the ParameterDataPrototype cannot be reused outside the *Basic Software Module*. Access is read-only. Each instance has an own data value associated with it.

Signature: **[SWS_Rte_07093]** ⌈
`<return> SchM_CData_<bsnp>[_<vi>_<ai>]_<name>()`

Where here

`<bsnp>` is the *BSW Scheduler Name Prefix* according [SWS_Rte_07593] and [SWS_Rte_07594],

`<vi>` is the `vendorId` of the calling BSW module,

`<ai>` `vendorApiInfix` of the calling BSW module and

`<name>` is the `shortName` of the `ParameterDataPrototype`.

The sub part in squared brackets `[_<vi>_<ai>]` is omitted if no `vendorApiInfix` is defined for the *Basic Software Module*. See [SWS_Rte_07528]. ⌋*(SRS_BSW_00347, SRS_Rte_00155)*

Existence: **[SWS_Rte_07094]** ⌈ An `SchM_CData` API shall be created for each defined `ParameterDataPrototype` in the role `perInstanceParameter` ⌋*(SRS_Rte_00155)*

Description: The `SchM_CData` API provides access to the defined calibration parameter within a *Basic Software Module*. The actual data values for a *Basic Software Module* instance may be set after component compilation.

Return Value: The `SchM_CData` return value provide access to the data value of the `ParameterDataPrototype` in the role `perInstanceParameter`.

The return type of `SchM_CData` is dependent on the `ImplementationDataType` of the `ParameterDataPrototype` and can either be a value or a pointer to the location where the value can be accessed. Thus the component does not need to use type casting to convert access to the `ParameterDataPrototype` data.

For details of the `<return>` value definition see section 5.2.6.6.

[SWS_Rte_07095] ⌈ The return value of the corresponding `SchM_CData` API shall provide access to the calibration parameter value specific to the instance of the *Basic Software Module*. ⌋*(SRS_Rte_00155)*

Notes: None.

6.6 Bsw Module Entity Reference

An *AUTOSAR Basic Software Module* defines one or more *"BSW module entities"*. A BSW Module Entity is a piece of code with a single entry point and an associate set of attributes. In contrast to runnable entities which are exclusively scheduled by the RTE only a subset of the BSW module entities, the `BswSchedulableEntity`s and BswCalledEntitys are called by the *Basic Software Scheduler*. Others might implement 'C' function interfaces which are directly called by other BSW modules or interrupts which are called by OS / interrupt controller.

A *Basic Software Module Description* provides definitions for each BswModuleEntity within the BSW Module. The *Basic Software Scheduler* triggers the execution of BswSchedulableEntitys and BswCalledEntitys in response to different BswEvents.

The BswCalledEntitys are triggered by BswOperationInvokedEvents, the BswSchedulableEntitys by BswScheduleEvents.

For BSW modules implemented using C or C++ the entry point of a BswSchedulableEntity is implemented by a function with global scope defined within a BSW Modules source code. The following sections consider the function signature and prototype.

6.6.1 Signature

The definition of all BswSchedulableEntitys, whatever the BswScheduleEvent that triggers their execution, follows the same basic form.

Purpose: Trigger a BswSchedulableEntity if the related BswScheduleEvent defined within the BswModuleDescription is raised.

Signature: **[SWS_Rte_07282]** ⌈

```
FUNC(void, <memclass>) <bsnp>[_<vi>_<ai>]_<name>(
[IN SchM_ActivatingEvent_<name> <activation>])
```

⌋*(SRS_BSW_00347, SRS_Rte_00211, SRS_Rte_00213, SRS_Rte_00216, SRS_Rte_00238)*

The usage of SchM_ActivatingEvent is optional and defined in [SWS_Rte_08056].

For BswCalledEntitys the signature contains the parameters and return type. It can be seen in [SWS_Rte_08765].

Purpose: Trigger a BswCalledEntity if the related BswOperationInvokedEvent defined within the BswModuleDescription is raised.

Signature: **[SWS_Rte_08765]** ⌈

```
FUNC(<returnType>, <memclass>) <bsnp>[_<vi>_<ai>]_<name>(
[IN|IN/OUT|OUT] <parameter_1>...
[IN|IN/OUT|OUT] <parameter_n>)
```

⌋*(SRS_BSW_00347, SRS_Rte_00241, SRS_Rte_00243)*

There is currently no possibility to obtain the activating BswOperationInvokedEvent of a BswCalledEntity.

Where here for both of them

<bsnp> is the BSW Scheduler Name Prefix according [SWS_Rte_07593] and [SWS_Rte_07594],

<vi> is the vendorId of the BSW module,

<ai> is the vendorApiInfix of the BSW module

<name> is the substring after "<bsnp>_" of the BswModuleEntry shortName referred as implementedEntry. However if "<bsnp>_" is not the prefix of the related BswModuleEntry shortName then <name> shall be the BswModuleEntry shortName.

<memclass> is the Compiler Abstraction Memory Class according [SWS_Rte_06739] and [SWS_Rte_06740].

<returnType> is the return type defined in the SwServiceArg in the role returnType of the BswModuleEntry which is referenced by the BswModuleClientServerEntry in the role encapsulatedEntry. If no type is defined, the <returnType> is of type void.

<parameter_x> are the arguments defined in the SwServiceArgs in the role argument of the BswModuleEntry which is referenced by the BswModuleClientServerEntry in the role encapsulatedEntry. For each argument the type has to be give according to [SWS_Rte_08766].

The sub part in square brackets [_<vi>_<ai>] is omitted if no vendorApiInfix is defined for the *Basic Software Module*. See [SWS_Rte_07528].

[SWS_Rte_08766] ⌈ The datatype of the argument is depending on SwServiceArgs.

For category of SwServiceArg of type TYPE_REFERENCE:
If the ImplementationDataType in the role implementationDataType of the SwDataDefProps of the SwServiceArg resolves to a primitive and the direction of the SwServiceArg is IN, the datatype of the argument is defined by the ImplementationDataType (possibly referred over a chain of ImplementationDataTypes of category TYPE_REFERENCE) in the role implementationDataType of the SwDataDefProps of the SwServiceArg which represents the argument.

If the ImplementationDataType in the role implementationDataType of the SwDataDefProps of the SwServiceArg resolves to a pointer type where the final pointer target is a primitive or composite and the direction of the SwServiceArg is IN, INOUT or OUT, the datatype of the argument is defined by the SwPointerTargetProps element referred by the ImplementationDataType of category DATA_REFERENCE (possibly referred over a chain of ImplementationDataTypes of category TYPE_REFERENCE).

For category of SwServiceArg of type DATA_REFERENCE:
If the SwPointerTargetProps in the role swPointerTargetProps of the SwDataDefProps of the SwServiceArg resolves to a primitive or composite and the

direction of the SwServiceArg is IN, INOUT or OUT, the datatype of the argument is defined by the SwPointerTargetProps in the SwDataDefProps of the SwServiceArg which represents the argument (which may include resolving a chain of ImplementationDataTypes if the target category of the SwPointerTargetProps is TYPE_REFERENCE).

For category of SwServiceArg of type FUNCTION_REFERENCE:
This case is not supported.

⌋*(SRS_Rte_00243)*

[constr_9058] BswSchedulableEntity is not allowed to have service arguments or return value ⌈ The Basic Software Scheduler requires that the BswModuleEntry has no service arguments (unless SchM_ActivatingEvent is enabled) and no return value. ⌋

[SWS_Rte_06739] ⌈ <memclass> shall be defined as <snp>[_<vi>_<ai>]_<memClassSymbol> if a MemorySection.memClassSymbol and an associated MemorySection is defined and where

<snp> is the Section Name Prefix according [SWS_Rte_07595] and [SWS_Rte_07596],

<vi> is the vendorId of the BSW module,

<ai> is the vendorApiInfix of the BSW module, and

<memClassSymbol> is the value of the attribute memClassSymbol the of the MemorySection associated via executableEntity reference to the BswModuleEntity implementing the related BswModuleEntry. ⌋

[SWS_Rte_06740] ⌈ <memclass> shall be defined as <snp>[_<vi>_<ai>]_<sadm> if no MemorySection.memclassSymbol is applicable (see [SWS_Rte_06739]) and where

<snp> is the Section Name Prefix according [SWS_Rte_07595] and [SWS_Rte_07596],

<vi> is the vendorId of the BSW module,

<ai> is the vendorApiInfix of the BSW module, and

<sadm> is the shortName of the referred swAddrMethod. ⌋

6.6.2 Entry Point Prototype

The entry point defined in the Basic Software Modules source *must* be compatible with the called function when the BswSchedulableEntity or BswCalledEntity is triggered by the *Basic Software Scheduler* and therefore the RTE generator is required to emit a prototype for the function.

[SWS_Rte_07283] ⌈ The RTE generator shall emit an *Entry Point Prototype* for each BswSchedulableEntitys implementedEntry and each BswCalledEntitys implementedEntry in the *Module Interlink Header* file. See chapter 6.3.2 according [SWS_Rte_07282]. ⌋*(SRS_Rte_00211, SRS_Rte_00213, SRS_Rte_00216)*

[SWS_Rte_07195] ⌈ The RTE Generator shall wrap each BswSchedulableEntity's *Entry Point Prototype* in the *Module Interlink Header* with the *Memory Mapping* and *Compiler Abstraction* macros.

```
#define <snp>[_<vi>_<ai>]_START_SEC_<sadm>
#include "<MemMap_filename.h>"

FUNC(void, <memclass>) <bsnp>[_<vi>_<ai>]_<name>
    ([IN SchM_ActivatingEvent_<name> <activation>]);

#define <snp>[_<vi>_<ai>]_STOP_SEC_<sadm>
#include "<MemMap_filename.h>"
```

The RTE Generator shall wrap each BswCalledEntity's *Entry Point Prototype* in the *Module Interlink Header* with the *Memory Mapping* and *Compiler Abstraction* macros.

```
#define <snp>[_<vi>_<ai>]_START_SEC_<sadm>
#include "<MemMap_filename.h>"

FUNC(<returnType>, <memclass>) <bsnp>[_<vi>_<ai>]_<name>(
    [IN|IN/OUT|OUT] <parameter_1> ... [IN|IN/OUT|OUT] <parameter_n>);

#define <snp>[_<vi>_<ai>]_STOP_SEC_<sadm>
#include "<MemMap_filename.h>"
```

Where here for both of them

<bsnp> is the BSW Scheduler Name Prefix according [SWS_Rte_07593] and [SWS_Rte_07594],

<snp> is the Section Name Prefix according [SWS_Rte_07595] and [SWS_Rte_07596],

<vi> is the vendorId of the BSW module,

<ai> is the vendorApiInfix of the BSW module,

<name> is the substring after "<bsnp>_" of the BswModuleEntry shortName referred as implementedEntry. However if "<bsnp>_" is not the prefix of the related BswModuleEntry shortName then <name> shall be the BswModuleEntry shortName, and

<returnType> is the return type defined in the SwServiceArg in the role returnType of the BswModuleEntry which is referenced by the BswModuleClientServerEntry in the role encapsulatedEntry. If no type is defined, the <returnType> is of type void.

<parameter_x> are the arguments defined in the SwServiceArgs in the role argument of the BswModuleEntry which is referenced by the BswModule-

ClientServerEntry in the role encapsulatedEntry. For each argument the type has to be give according to [SWS_Rte_08766].

<sadm> is the shortName of the referred swAddrMethod.

<memclass> is the Compiler Abstraction Memory Class according [SWS_Rte_06739] and [SWS_Rte_06740]

<MemMap_filename.h> is the *Applicable Memory Mapping Header File Name* according [SWS_Rte_07830], [SWS_Rte_07831] and [SWS_Rte_07832].

The sub part in square brackets [_<vi>_<ai>] is omitted if no vendorApiInfix is defined for the *Basic Software Module*. See [SWS_Rte_07528].

The usage of SchM_ActivatingEvent is optional for BswSchedulableEntity and defined in [SWS_Rte_08056]. It does currently not exist for BswCalledEntitys.

The Memory Mapping macros could wrap several *Entry Point Prototype* if these referring the same swAddrMethod. If the BswSchedulableEntity or the BswCalledEntity does not refer a swAddrMethod the <sadm> is set to CODE. ⌋*(SRS_Rte_00148, SRS_Rte_00149, SRS_Rte_00238)*

[SWS_Rte_07830] ⌈ The RTE Generator shall emit the *Applicable Memory Mapping Header File Name* <MemMap_filename.h> as <Msn>[_<vi>_<ai>]_MemMap.h if the BswImplementation does not contain a DependencyOnArtifact in the role requiredArtifact where the DependencyOnArtifact.category is set to MEMMAP. <Msn> is the shortName (case sensitive) of the BswModuleDescription. ⌋*(SRS_Rte_00148)*

[SWS_Rte_07831] ⌈ The RTE generator shall emit the *Applicable Memory Mapping Header File Name* <MemMap_filename.h> identical to the attribute value requiredArtifact.artifactDescriptor.shortLabel if the BswImplementation does contain exactly one DependencyOnArtifact in the role requiredArtifact where the DependencyOnArtifact.category is set to MEMMAP. ⌋*(SRS_Rte_00148)*

[SWS_Rte_07832] ⌈ The RTE Generator shall emit the *Applicable Memory Mapping Header File Name* <MemMap_filename.h> identical to the attribute value requiredArtifact.artifactDescriptor.shortLabel of the DependencyOnArtifact in the role requiredArtifact where the DependencyOnArtifact.category is set to MEMMAP and which is associated with the SectionNamePrefix implementedIn of the MemorySection associated to the BswModuleEntity. ⌋*(SRS_Rte_00148)*

Please note the example 6.2 of *Entry Point Prototype*.

[SWS_Rte_06533] ⌈ The RTE Generator shall wrap each *Entry Point Prototype* in the *Module Interlink Header* file of a variant existent BswSchedulableEntity or BswCalledEntity if the variability shall be implemented.

```
#if (<condition>)
```

```

<Entry Point Prototype>

#endif
```

where condition is the *Condition Value Macro* of the VariationPoint relevant for the variant existence of the BswSchedulableEntity or BswCalledEntity (see table 4.28), Entry Point Prototype is the code according an invariant *Entry Point Prototype* (see also [SWS_Rte_07282], [SWS_Rte_07283]). ⌋*(SRS_Rte_00229)*

6.6.3 Reentrancy

The BswSchedulableEntitys and BswCalledEntitys are declared within a BSW Module. The *Basic Software Module Scheduler* ensures that concurrent activation of the same BswSchedulableEntity or BswCalledEntity is only allowed if the implemented entry points attribute "isReentrant" is set to "true" (see Section 4.2.6).

Consistency rule:

[SWS_Rte_07588] ⌈ The RTE Generator shall reject configurations where a BswSchedulableEntity whose referenced BswModuleEntry in the role implementedEntry has its isReentrant attribute set to false, and this BswSchedulableEntity is mapped to different tasks which can pre-empt each other. ⌋*(SRS_Rte_00018)*

6.6.4 Provide activating Bsw event

[SWS_Rte_08059] ⌈ If the provide activating Bsw event feature is enabled, the RTE shall collect the activating Bsw events, which have the activationReasonRepresentation reference defined, in the context of the OS task the executable entity is mapped to in an activation vector at the corresponding bit position as defined in [SWS_Rte_08058]. ⌋*(SRS_Rte_00238)*

[SWS_Rte_08060] ⌈ If the provide activating Bsw event feature is enabled, the RTE shall provide the collected activating Bsw events (activation vector) to the executable entity API when the executable entity is "started". The activation vector shall be reset immediately after it has been provided. ⌋*(SRS_Rte_00238)*

Provision of the activating Bsw event is curerntly not availbale for BswCalledEntitys.

Since it is possible that there is a time gap between the activation and the execution (start) of a executable entity the subsequent activations are summed up and provided with the start of the executable entity.

Activations during the execution of a executable entity are collected for the next start of that runnable entity.

6.7 Basic Software Scheduler Lifecycle API Reference

6.7.1 SchM_Init

Purpose: Initialize the *Basic Software Scheduler* part of the RTE.

Signature: **[SWS_Rte_07270]** ⌈

```
void SchM_Init([const SchM_ConfigType * ConfigPtr])
```

⌋*(SRS_BSW_00101, SRS_Rte_00116)*

Existence: **[SWS_Rte_07271]** ⌈ The SchM_Init API is always created. ⌋*(SRS_BSW_00101)*

Description: SchM_Init is intended to allocate and initialize system resources used by the *Basic Software Scheduler* part of the RTE for the core on which it is called. After initialization the scheduling of BswSchedulableEntitys is enabled.

[constr_9055] SchM_Init shall be called only once ⌈ SchM_Init shall be called only once by the *EcuStateManager* on each core after the basic software modules required by the *Basic Software Scheduler* part of the RTE are initialized. ⌋

These modules include:

- OS

[SWS_Rte_06544] ⌈ The optional parameter configPtr shall be a pointer to a post build data set which is used to resolve the PostBuild Variability of the *Basic Software Scheduler* and RTE. ⌋*(SRS_BSW_00405, SRS_Rte_00229, SRS_Rte_00204, SRS_Rte_00206, SRS_Rte_00207)*

[SWS_Rte_06545] ⌈ The parameter configPtr shall only be provided if the input configuration of the RTE and *Basic Software Scheduler* contains PostBuild Variability which has to be implemented by the RTE Generator. ⌋*(SRS_BSW_00405, SRS_Rte_00229, SRS_Rte_00204, SRS_Rte_00206, SRS_Rte_00207)*

[constr_9056] SchM_Deinit API may only be used after the was RTE finalized ⌈ The SchM_Deinit API may only be used after the RTE finalized (after termination of the Rte_Stop) ⌋

[SWS_Rte_07273] ⌈ SchM_Init shall return within finite execution time – it must not enter an infinite loop. ⌋*(SRS_BSW_00101)*

SchM_Init may be implemented as a function or a macro.

Return Value: None

Notes: SchM_Init is declared in the lifecycle header file Rte_Main.h.

6.7.2 SchM_Deinit

Purpose: Finalize the *Basic Software Scheduler* part of the RTE on the core it is called.

Signature: **[SWS_Rte_07274]** ⌈

```
void SchM_Deinit(void)
```

⌋(*SRS_BSW_00336*)

Existence: **[SWS_Rte_07275]** ⌈ The SchM_Deinit API is always created. ⌋(*SRS_BSW_00336*)

Description: SchM_Deinit is used to finalize *Basic Software Scheduler* part of the RTE of the core on which it is called. This service releases all system resources allocated by the *Basic Software Scheduler* part on that core.

[constr_9057] SchM_Deinit shall be called before shut down of BSW ⌈ SchM_Deinit shall be called by the *EcuStateManager* before the basic software modules required by *Basic Software Scheduler* part are shut down. ⌋

These modules include:

- OS

[SWS_Rte_07277] ⌈ SchM_Deinit shall return within finite execution time. ⌋(*SRS_BSW_00336*)

SchM_Deinit may be implemented as a function or a macro.

Return Value: None

Notes: SchM_Deinit is declared in the lifecycle header file Rte_Main.h.

6.7.3 SchM_GetVersionInfo

Purpose: Returns the version information of the *Basic Software Scheduler*.

Signature: **[SWS_Rte_07278]** ⌈

```
void SchM_GetVersionInfo(Std_VersionInfoType * versioninfo)
```

⌋(*SRS_BSW_00407*)

Existence: **[SWS_Rte_07279]** ⌈ The SchM_GetVersionInfo API is only created if RteSchMVersionInfoApi is set to true. ⌋(*SRS_BSW_00407*)

Description: **[SWS_Rte_07280]** ⌈ SchM_GetVersionInfo shall return the version information of the RTE module which includes the *Basic Software Scheduler*. The version information includes:

- Module Id
- Vendor Id
- Vendor specific version numbers

⌋(*SRS_BSW_00407*)

[SWS_Rte_07281] ⌈ The parameter versioninfo of the SchM_GetVersionInfo shall point to the memory location holding the version information of the *Basic Software Scheduler*. ⌋(*SRS_BSW_00407*)

SchM_GetVersionInfo may be implemented as a function or a macro.

Return Value: None

Notes: SchM_GetVersionInfo is declared in the lifecycle header file Rte_Main.h.

The existence of the API SchM_GetVersionInfo depends on the parameter RteSchMVersionInfoApi.

Vendor specific version numbers shall represent build version which depends from the RTE generator version and the input configuration. It is not in the scope if this specification to standardize the way how the version numbers are created in detail because these are the vendor specific version numbers.

7 RTE ECU Configuration

The RTE provides the glue layer between the AUTOSAR software-components and the Basic Software thus enabling several AUTOSAR software-components to be integrated on one ECU. The RTE layer is shown in figure 7.1.

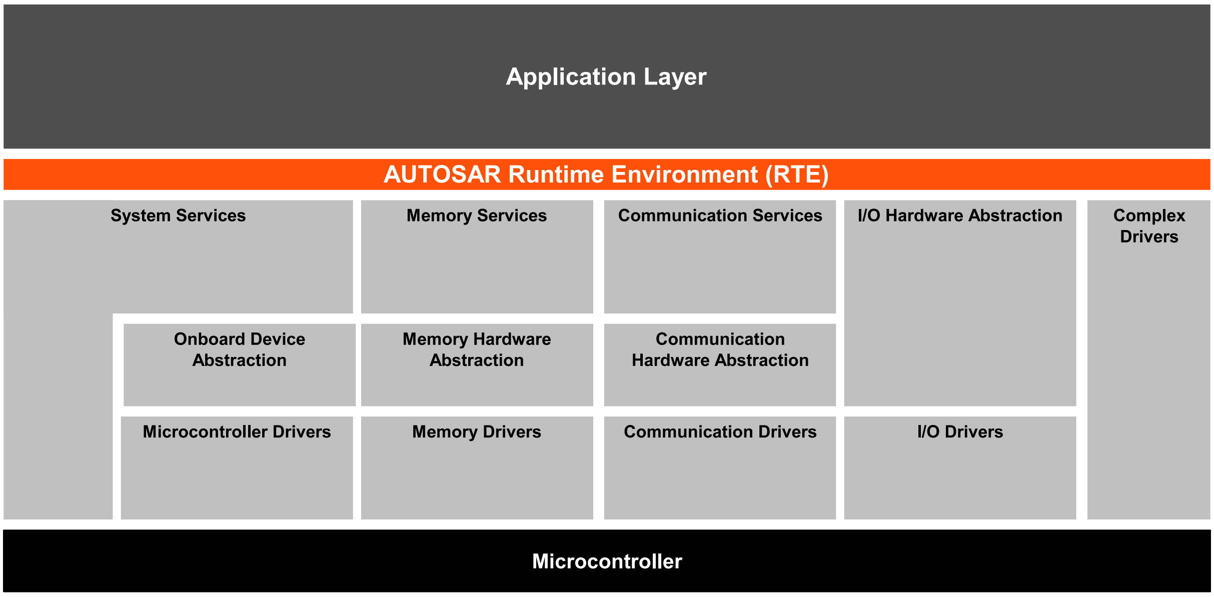


Figure 7.1: ECU Architecture RTE

The overall structure of the RTE configuration parameters is shown in figure 7.2. It has to be distinguished between the configuration parameters for the RTE generator and the configuration parameters for the generated RTE itself.

Most of the information needed to generate an RTE is already available in the ECU Extract of the System Description [8]. From this extract also the links to the AUTOSAR software-component descriptions and ECU Resource description are available. So only additional information not covered by the three aforementioned formats needs to be provided by the ECU Configuration description.

To additionally allow the most flexibility and freedom in the implementations of the RTE, only configuration parameters which are common to all implementations are standardized in the ECU Configuration Parameter definition. Any additional configuration parameters which might be needed to configure a full functional RTE have to be specified using the vendor specific parameter definition mechanism described in the ECU Configuration specification document [5].

7.1 Ecu Configuration Variants

The RTE shall supports two Ecu Configuration Variants:

[SWS_Rte_05103] ⌈ `VARIANT-PRE-COMPILE` Only parameters with "Pre-compile time" configuration are allowed in this variant. ⌋*(SRS_BSW_00345, SRS_BSW_00397)*

[SWS_Rte_05104] ⌈ `VARIANT-POST-BUILD` Parameters with "Pre-compile time", "Link time" and "Post-build time" are allowed in this variant. ⌋*(SRS_BSW_00399, SRS_BSW_00400, SRS_Rte_00201, SRS_Rte_00204, SRS_Rte_00206, SRS_Rte_00207, SRS_Rte_00229)*

For details on the ECU Configuration approach please refer to the *Specification of ECU Configuration* [5].

7.2 RTE Module Configuration

Figure 7.2 shows the module configuration of the Rte and its sub-containers.

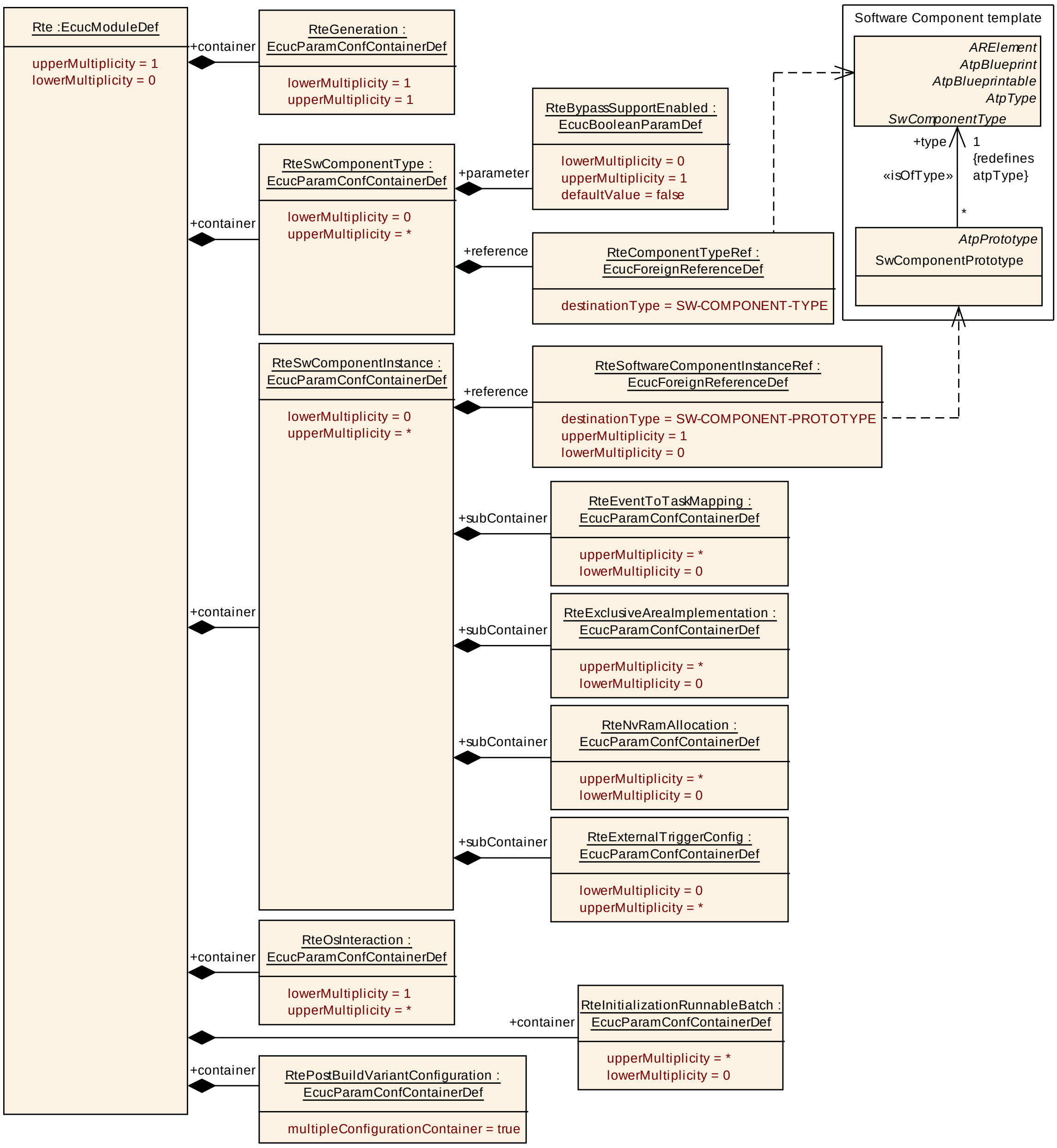


Figure 7.2: RTE configuration overview

Module Name	Rte	
Module Description	Configuration of the Rte (Runtime Environment) module.	
Included Containers		
Container Name	**Multiplicity**	**Scope / Dependency**
RteBswGeneral	1	General configuration parameters of the Bsw Scheduler section.
RteBswModuleInstance	0..*	Represents one instance of a Bsw-Module configured on one ECU.
RteGeneration	1	This container holds the parameters for the configuration of the RTE Generation.
RteImplicitCommunication	0..*	Configuration of the Implicit Communication behavior to be generated.

Container Name	Multiplicity	Scope / Dependency
RteInitializationBehavior	1..*	Specifies the initialization strategy for variables allocated by RTE with the purpose to implement VariableDataPrototypes. The container defines a set of RteSectionInitializationPolicys and one RteInitializationStrategy which is applicable for this set.
RteInitializationRunnable Batch	0..*	This container corresponds to an Rte_Init_<shortName of this container> function invoking the mapped RunnableEntities.
RteOsInteraction	1..*	Interaction of the Rte with the Os.
RtePostBuildVariant Configuration	1	Specifies the PostbuildVariantSets for each of the PostBuild configurations of the RTE. The shortName of this container defines the name of the RtePostBuildVariant.
RteSerialized Communication	0..1	This container collects all serialization functions that are used by the RTE.
RteSwComponentInstance	0..*	Representation of one SwComponentPrototype located on the to be configured ECU. All subcontainer configuration aspects are in relation to this SwComponentPrototype. The RteSwComponentInstance can be associated with either a AtomicSwComponentType or ParameterSwComponentType.
RteSwComponentType	0..*	Representation of one SwComponentType for the base of all configuration parameter which are affecting the whole type and not a specific instance.

7.2.1 RTE Configuration Version Information

In order to identify the RTE Configuration version a dedicated RTE code has been generated from the RTE Configuration information may contain one or more `DOC-REVISION` elements in the `ECUC-MODULE-CONFIGURATION-VALUES` element of the RTE Configuration (see example 7.1).

[SWS_Rte_05184] ⌈ The `REVISION-LABEL` shall be parsed according to the rules defined in the Generic Structure Template [10] for `RevisionLabelString` allowing to parse the three version informations for AUTOSAR:

- major version: first part of the `REVISION-LABEL`
- minor version: second part of the `REVISION-LABEL`
- patch version: third part of the `REVISION-LABEL`
- optional fourth part shall be used for documentation purposes in the Basic Software Module Description (see section 3.4.3)

If the parsing fails all three version numbers shall be set to zero. ⌋*(SRS_Rte_00233)*

[SWS_Rte_05185] ⌈ If there are several DOC-REVISION elements in the input ECUC-MODULE-CONFIGURATION-VALUES the newest according to the DATE shall be taken into account.

If the search for the newest DOC-REVISION fails three version numbers shall be set to zero. ⌋*(SRS_Rte_00233)*

Example 7.1

```
<AUTOSAR xmlns="http://autosar.org/4.0.0" xmlns:xsi="http://www.w3.org
    /2001/XMLSchema-instance" xsi:schemaLocation="http://autosar.org/4.0.0␣
    AUTOSAR.xsd">
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>Rte_Example</SHORT-NAME>
      <ELEMENTS>
        <ECUC-MODULE-CONFIGURATION-VALUES>
          <SHORT-NAME>Rte_Configuration</SHORT-NAME>
          <ADMIN-DATA>
            <DOC-REVISIONS>
              <DOC-REVISION>
                <REVISION-LABEL>2.1.34</REVISION-LABEL>
                <DATE>2009-05-09T00:00:00.0Z</DATE>
              </DOC-REVISION>
              <DOC-REVISION>
                <REVISION-LABEL>2.1.35</REVISION-LABEL>
                <DATE>2009-06-21T09:30:00.0Z</DATE>
              </DOC-REVISION>
            </DOC-REVISIONS>
          </ADMIN-DATA>
          <DEFINITION-REF DEST="ECUC-MODULE-DEF">/AUTOSAR/Rte</DEFINITION-
              REF>
          <CONTAINERS>
            <!-- ... -->
          </CONTAINERS>
        </ECUC-MODULE-CONFIGURATION-VALUES>
      </ELEMENTS>
    </AR-PACKAGE>
  </AR-PACKAGES>
</AUTOSAR>
```

7.3 RTE Generation Parameters

The parameters in the container RteGeneration are used to configure the RTE generator. They all need to be defined during pre-compile time.

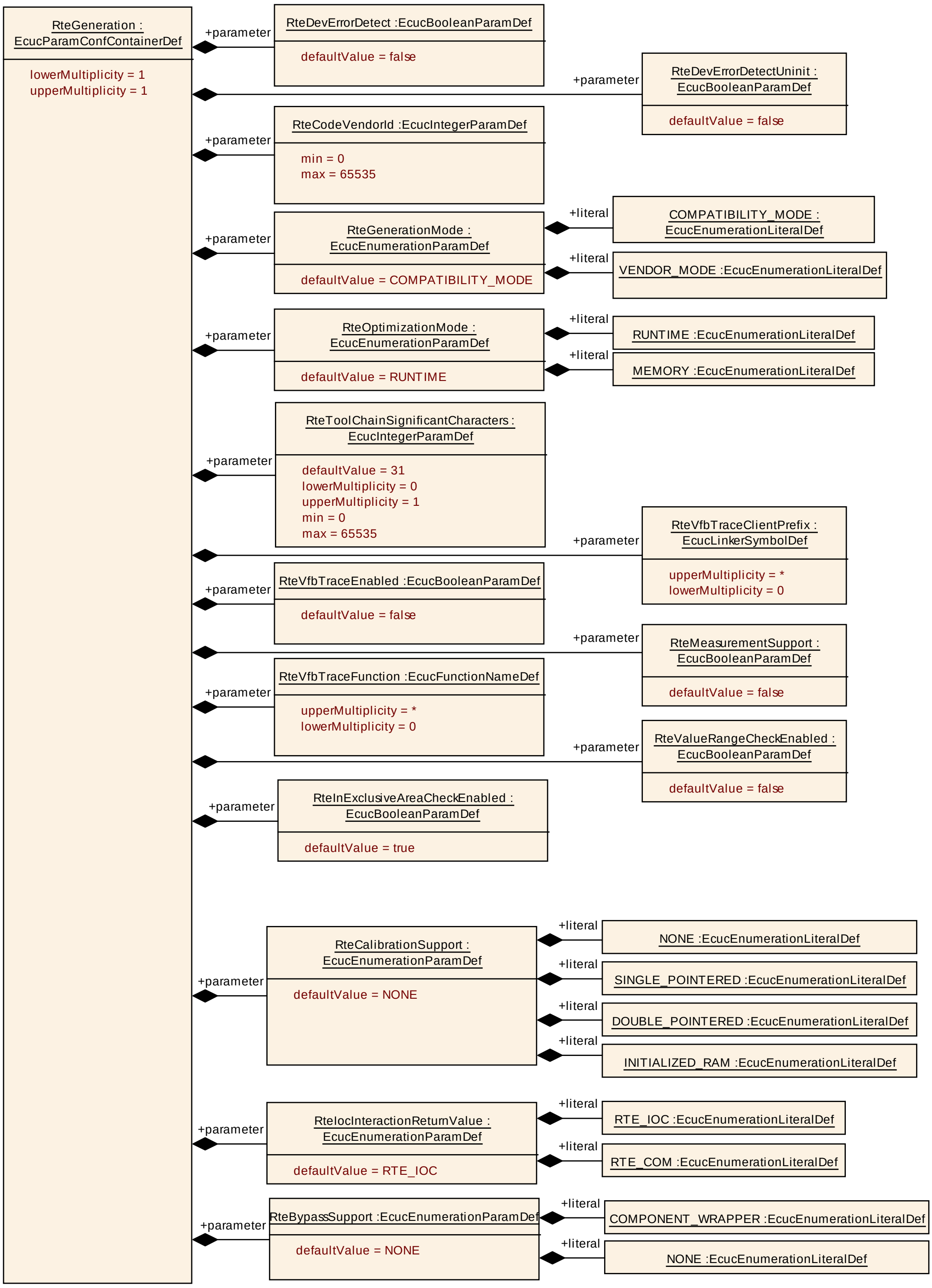


Figure 7.3: RTE generation parameters

RteGeneration

SWS Item	[ECUC_Rte_09009]		
Container Name	RteGeneration		
Description	This container holds the parameters for the configuration of the RTE Generation.		
Configuration Parameters			

Name	RteBypassSupport {RTE_BYPASS_SUPPORT} [ECUC_Rte_09113]		
Description	General switch to enable and select the bypass support method.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	COMPONENT_WRAPPER		
	NONE	(default)	
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteCalibrationSupport {RTE_CALIBRATION_SUPPORT} [ECUC_Rte_09007]		
Description	The RTE generator shall have the option to switch off support for calibration for generated RTE code. This option shall influence complete RTE code at once.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	DOUBLE_POINTERED		
	INITIALIZED_RAM		
	NONE	(default)	
	SINGLE_POINTERED		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteCodeVendorId {RTE_CODE_VENDOR_ID} [ECUC_Rte_09086]		
Description	Holds the vendor ID of the generated Rte code.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default Value			
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteDevErrorDetect {RTE_DEV_ERROR_DETECT} [ECUC_Rte_09008]		
Description	The Rte shall log development errors to the Det module.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value	false		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteDevErrorDetectUninit {RTE_DEV_ERROR_DETECT_UNINIT} [ECUC_Rte_09085]		
Description	The Rte shall detect if it is started when its APIs are called, and the BSW Scheduler shall check if it is initialized when its APIs are called.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value	false		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local dependency: Shall only be used when RteDevErrorDetect equals true.		

Name	RteGenerationMode {RTE_GENERATION_MODE} [ECUC_Rte_09010]		
Description	Switch between the two available generation modes of the RTE generator.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	COMPATIBILITY_MODE	(default)	
	VENDOR_MODE		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteInExclusiveAreaCheckEnabled [ECUC_Rte_09126]		
Description	Enables the check for RTE_E_IN_EXCLUSIVE_AREA (for blocking APIs).		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value	true		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteIocInteractionReturnValue {RTE_IOC_INTERACTION_RETURN_VALUE} [ECUC_Rte_09094]		
Description	Defines whether the return value of RTE APIs is based on RTE-IOC interaction or RTE-COM interaction.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	RTE_COM		
	RTE_IOC	(default)	
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteMeasurementSupport {RTE_MEASUREMENT_SUPPORT} [ECUC_Rte_09011]		
Description	The RTE generator shall have the option to switch off support for measurement for generated RTE code. This option shall influence complete RTE code at once.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value	false		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteOptimizationMode {RTE_OPTIMIZATION_MODE} [ECUC_Rte_09012]		
Description	Switch between the two available optimization modes of the RTE generator.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	MEMORY		
	RUNTIME	(default)	
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteToolChainSignificantCharacters {RTE_TOOL_CHAIN_SIGNIFICANT_CHARACTERS} [ECUC_Rte_09013]	
Description	If present, the RTE generator shall provide the list of C RTE identifiers whose name is not unique when only the first RteToolChainSignificantCharacters characters are considered.	
Multiplicity	0..1	
Type	EcucIntegerParamDef	
Range	0 .. 65535	
Default Value	31	

Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteValueRangeCheckEnabled {RTE_VALUE_RANGE_CHECK_ENABLED} [ECUC_Rte_09014]		
Description	If set to true the RTE generator shall enable the value range checking for the specified VariableDataPrototypes.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value	false		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteVfbTraceClientPrefix [ECUC_Rte_09016]		
Description	Defines an additional prefix for all VFB trace functions to be generated. With this approach it is possible to have debugging and DLT trace functions at the same time.		
Multiplicity	0..*		
Type	EcucLinkerSymbolDef		
Default Value			
Regular Expression			
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteVfbTraceEnabled {RTE_VFB_TRACE_ENABLED} [ECUC_Rte_09015]		
Description	The RTE generator shall globally enable VFB tracing when RteVfbTrace is set to "true".		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value	false		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteVfbTraceFunction [ECUC_Rte_09017]		
Description	The RTE generator shall enable VFB tracing for a given hook function when there is a #define in the RTE configuration header file for the hook function name and tracing is globally enabled. Example: #define Rte_WriteHook_i1_p1_a_Start This also applies to VFB trace functions with a RteVfbTraceClientPrefix, e.g. Rte_Dbg_WriteHook_I1_P1_a_Start.		
Multiplicity	0..*		
Type	EcucFunctionNameDef		
Default Value			
Regular Expression			
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

No Included Containers

7.4 RTE PreBuild configuration

In order to support PreBuild configuration variation of the Rte input (see also section 4.7) the container `EcucVariationResolver` is providing a set of references to `PredefinedVariant`. These define values for `SwSystemconst`.

Note that the information for the `EcucVariationResolver` is provided in the `EcuC` part of the ECU Configuration, since it does not only influence the Rte but also many other BSW Modules.

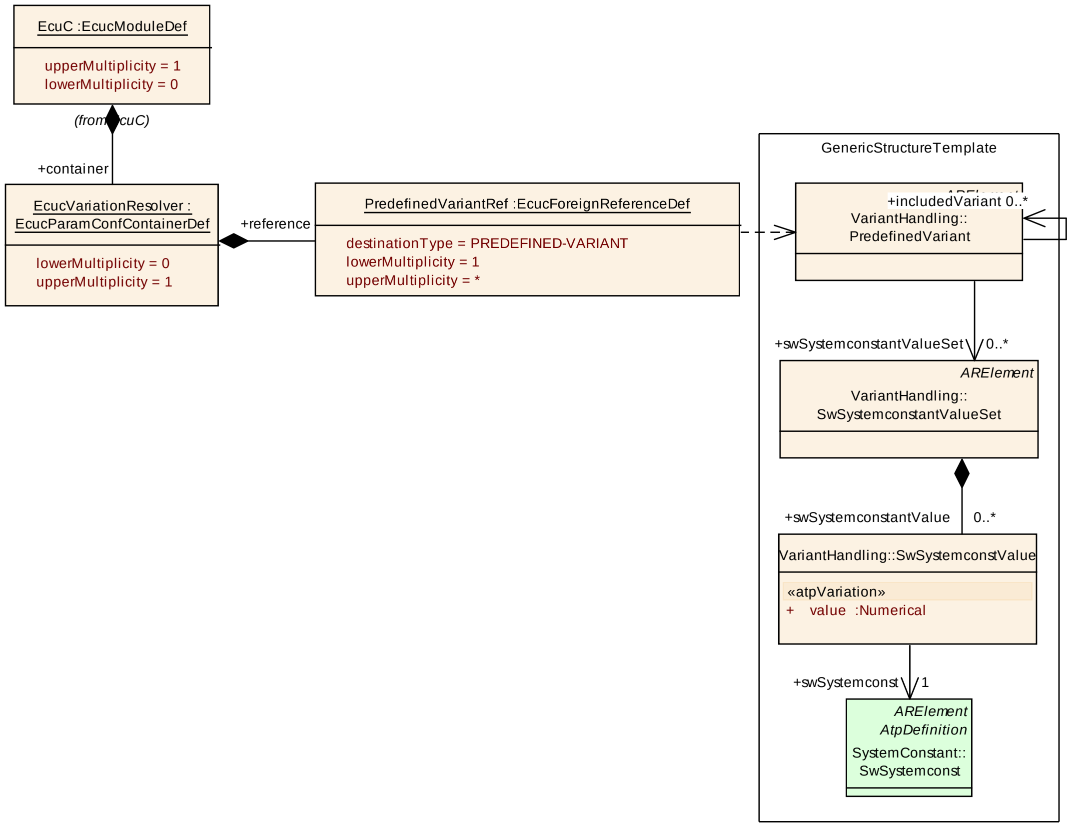


Figure 7.4: RTE PreBuild configuration

EcucVariationResolver

SWS Item	[ECUC_EcuC_00009]
Container Name	EcucVariationResolver
Description	Collection of PredefinedVariant elements containing definition of values for SwSystemconst which shall be applied when resolving the variability during ECU Configuration.
Configuration Parameters	

Name	PredefinedVariantRef [ECUC_EcuC_00010]		
Description			
Multiplicity	1..*		
Type	Foreign reference to PREDEFINED-VARIANT		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency			

No Included Containers

7.5 RTE PostBuild configuration

In order to support PostBuild configuration variation of the generated Rte (see also section 4.7) the container RtePostBuildVariantConfiguration is used. Each instance of this container specifies *one* PostBuild variant of the generated Rte. The shortName of the container RtePostBuildVariantConfiguration specifies the variant name.

The actual values for the PostBuildVariantCriterion are defined in a two step approach:

1. The reference RtePostBuildUsedPredefinedVariant collects the PredefinedVariant elements.
2. Each PredefinedVariant element collects a set of PostBuildVariantCriterionValueSet.
3. Each PostBuildVariantCriterionValueSet defines the PostBuildVariantCriterionValues for a set of PostBuildVariantCriterion.

The basic idea is that

- the PostBuildVariantCriterionValueSet can be provided by sub-system engineer,
- the PredefinedVariant can be designed by the Ecu integrator.

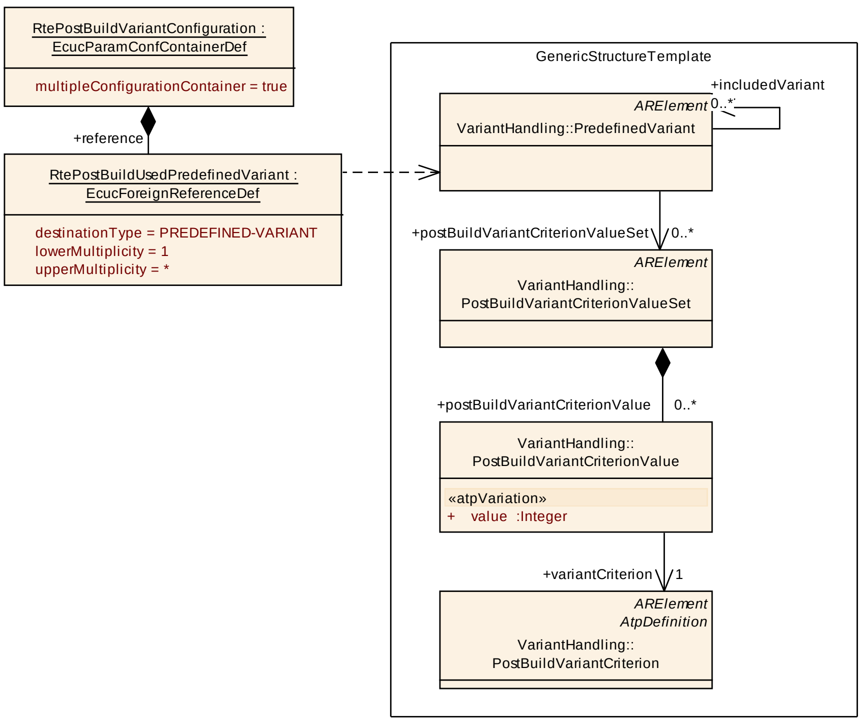


Figure 7.5: RTE PostBuild configuration

RtePostBuildVariantConfiguration

SWS Item	[ECUC_Rte_09084]
Container Name	RtePostBuildVariantConfiguration[Multi Config Container]
Description	Specifies the PostbuildVariantSets for each of the PostBuild configurations of the RTE. The shortName of this container defines the name of the RtePostBuildVariant.
Configuration Parameters	

Name	RtePostBuildUsedPredefinedVariant [ECUC_Rte_09083]		
Description	Reference to the PredefinedVariant element which defines the values for PostBuildVariationCriterion elements.		
Multiplicity	1..*		
Type	Foreign reference to PREDEFINED-VARIANT		
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

No Included Containers

7.6 Handling of Software Component instances

When entities of Software-Components are to be configured there is the need to actually address the instances of the AtomicSwComponentType. Since the Ecu Extract of System Description contains a flat view on the Ecu's Software-Components [8] the SwComponentPrototypes in the Ecu Extract already represent the instances of the Software Components.

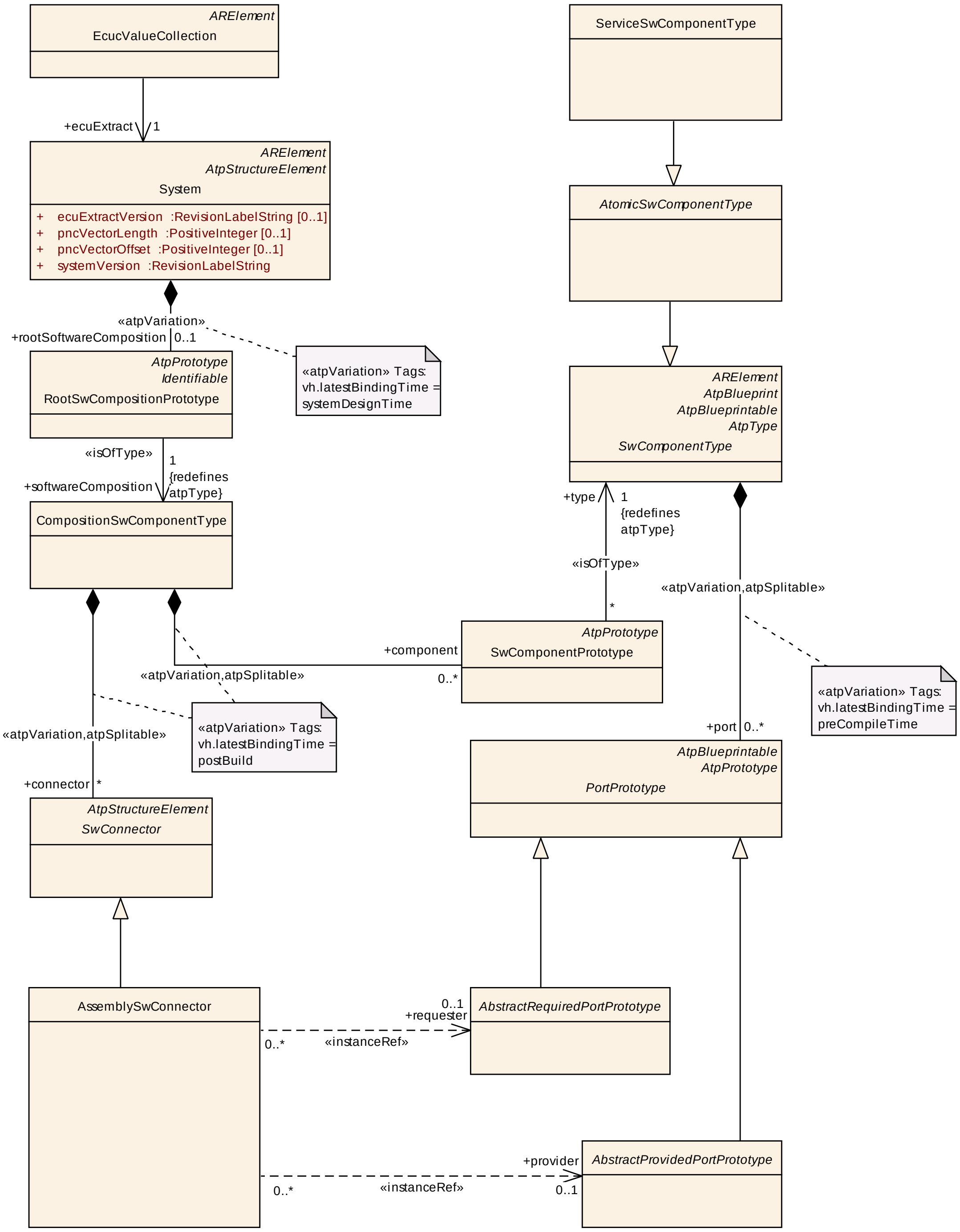


Figure 7.6: Services in the ECU Configuration

RteSwComponentInstance

SWS Item	[ECUC_Rte_09005]
Container Name	RteSwComponentInstance
Description	Representation of one SwComponentPrototype located on the to be configured ECU. All subcontainer configuration aspects are in relation to this SwComponentPrototype. The RteSwComponentInstance can be associated with either a AtomicSwComponentType or ParameterSwComponentType.
Configuration Parameters	

Name	RteSoftwareComponentInstanceRef [ECUC_Rte_09004]		
Description	Reference to a SwComponentPrototype.		
Multiplicity	0..1		
Type	Foreign reference to SW-COMPONENT-PROTOTYPE		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Included Containers		
Container Name	**Multiplicity**	**Scope / Depedency**
RteEventToTaskMapping	0..*	Maps an instance of a RunnableEntity onto one OsTask based on the activating RTEEvent. In the case of a RunnableEntity executed via a direct function call this RteEventToTaskMapping is still specified but no RteMappedToTask element is included. The RtePositionInTask parameter is necessary to provide an ordering of events invoked by the same RTE API.
RteExclusiveArea Implementation	0..*	Specifies the implementation to be used for the data consistency of this ExclusiveArea.
RteExternalTriggerConfig	0..*	Defines the configuration of External Trigger Event Communication for Software Components
RteInternalTriggerConfig	0..*	Defines the configuration of Inter Runnable Triggering for Software Components
RteNvRamAllocation	0..*	Specifies the relationship between the AtomicSwComponentType's NVRAMMapping / NVRAM needs and the NvM module configuration.

The container `RteSwComponentInstance` collects all the configuration information related to one specific instance of a `AtomicSwComponentType`. The individual aspects will be described in the next sections.

7.6.1 RTE Event to task mapping

One of the major fragments of the RTE configuration is the mapping of AUTOSAR Software-Components' `RunnableEntity`s to OS Tasks. The parameters defined to achieve this are shown in figure 7.7.

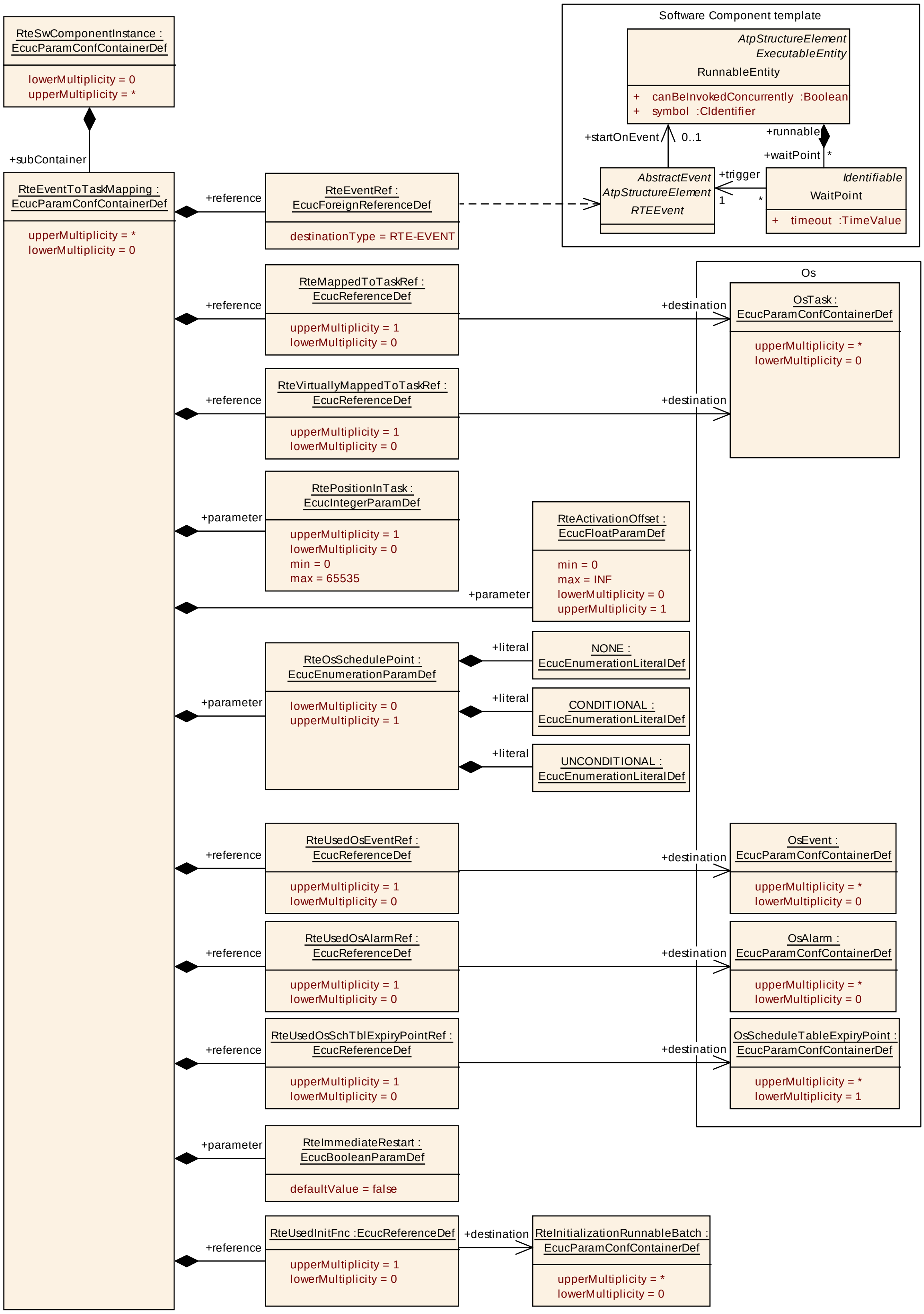


Figure 7.7: RTE Event to task mapping

The mapping is based on the RTEEvent because it is the source of the activation. For each RunnableEntity which belongs to an AUTOSAR Software-Component instance mapped on the ECU there needs to be a mapping container specifying how this RunnableEntity activation shall be handled.

[SWS_Rte_07843] ⌈ The RTE Generator shall reject configurations where the same RTEEvent instance which can start a RunnableEntity is referenced by multiple task mappings. ⌋

One major constraint is posed by the canBeInvokedConcurrently attribute of each RunnableEntity because data consistency issues have to be considered.

7.6.1.1 Evaluation and execution order

Another important parameter is the RtePositionInTask which provides an order of RunnableEntitys within the associated OsTask. When the task is executed periodically the RtePositionInTask parameter defines the order of execution within the test. When the task is used to define a context for event activated RunnableEntitys the RtePositionInTask parameter defines the order of evaluation which actual RunnableEntity shall be executed. Thus providing means to define a deterministic delay between the beginning of execution of the task and the actual execution of the RunnableEntity's code.

In case of triggered runnables, OnEntry ExecutableEntitys, OnTransition ExecutableEntitys, OnExit ExecutableEntitys, and mode switch acknowledge ExecutableEntitys the RtePositionInTask parameter defines the order of evaluation which actual RunnableEntity shall be executed. All other parameters or references are not required.

7.6.1.2 Direct function call

[SWS_Rte_06798] ⌈ If the ExecutableEntity is a server ExecutableEntity, triggered ExecutableEntity, OnEntry ExecutableEntity, OnTransition ExecutableEntity, OnExit ExecutableEntity, or a mode switch acknowledge ExecutableEntity and shall be executed in the context of the caller (i.e. using a direct function call) then the element RteEventToTaskMapping or RteBswEventToTaskMapping still shall be provided to indicate that this RTEEvent / BswEvent has been considered in the mapping. ⌋

In case of server ExecutableEntitys its not possible that several servers get invoked by the same API call. Therefore no further parameters in the RteEventToTaskMapping or RteBswEventToTaskMapping associated to the RTEEvent / BswEvent are required to configure the direct function call for server ExecutableEntitys.

[SWS_Rte_06799] ⌈ For directly invoked server ExecutableEntitys no further parameters or references are required (e.g. RteMappedToTaskRef can be left out).
⌋

In case of ExecutableEntitys which are not server ExecutableEntitys it is possible that several ExecutableEntitys get invoked by the same API call when direct function call configuration is used. Thereby the RteMappedToTaskRef / RteBswMappedToTaskRef is omitted. However the order of invocation needs to be configured with the RtePositionInTask and RteBswPositionInTask parameters.

[SWS_Rte_06800] ⌈ For directly invoked triggered ExecutableEntity, OnEntry ExecutableEntity, OnTransition ExecutableEntity, OnExit ExecutableEntity, or a mode switch acknowledge ExecutableEntity the RtePositionInTask and RteBswPositionInTask parameter respectively is required to indicate the order of invocation. ⌋

The invocation context for an ExecutableEntity can be either a task or a function call. For ExecutableEntitys invoked from an OsTasks then [constr_9082] means that all mapped ExecutableEntities must have unique values for the task to ensure predictable generation of the task body. In the case of RTEEvents or BswEvents invoked by direct invocation from an RTE-generated API function then [constr_9082] means that all events invoked by the calling function must have unique values to ensure predictable generation of the calling API.

[constr_9082] RtePositionInTask and RteBswPositionInTask values shall be unique in a particular context ⌈ RtePositionInTask and RteBswPositionInTask shall have unique values for any particular task in the case RTEEvents and BswEvents are mapped to OsTasks and shall have unique values for any particular scope of direct invocation in the case that the a direct function call is configured. The only exception are RtePositionInTask values for RteEventToTaskMappings mapping the OperationInvokedEvents for several operations to the same server runnables. ⌋

Concerning the mapping of several operations to the same server runnables see [SWS_Rte_08001].

Example 7.2

BSW module BswA defines BswModuleEntity BswA_ProcessBigBang triggered by BswExternalTriggerOccurredEvent Ev_BswA_ProcessBigBang

Software component SwcA defines RunnableEntity SwcA_Run_BigBang triggered by ExternalTriggerOccurredEvent Ev_SwcA_Run_BigBang

Software component SwcB defines RunnableEntity SwcB_Run_BigBang triggered by ExternalTriggerOccurredEvent Ev_SwcB_Run_BigBang

All required Triggers are connected to one common synchronized Trigger.

Scenario A

A configuration:

Ev_BswA_ProcessBigBang is mapped to OsTask T_BIG_BANG with RtePositionInTask = 1

Ev_SwcA_Run_BigBang is mapped to OsTask T_BIG_BANG with RtePositionInTask = 2

Ev_SwcB_Run_BigBang is mapped to OsTask T_BIG_BANG with RtePositionInTask = 3

results in Rte code where the ExecutableEntitys are called in the context of the OsTask T_BIG_BANG in the order:

1. Ev_BswA_ProcessBigBang
2. Ev_SwcA_Run_BigBang
3. Ev_SwcB_Run_BigBang

In addition [constr_9082] is fulfilled even if the RtePositionInTask values 1, 2, 3 are used for other RteEventToTaskMappings mapping to other OsTask or configuring a direct function call.

Scenario B

A configuration:

Ev_BswA_ProcessBigBang is not mapped to any OsTask and RtePositionInTask = 1

Ev_SwcA_Run_BigBang is not mapped to any OsTask and RtePositionInTask = 2

Ev_SwcB_Run_BigBang is not mapped to any OsTask and RtePositionInTask =

results in Rte code where the ExecutableEntitys are called in the context of the issuing Trigger API, e.g SchM_Trigger which invokes the ExecutableEntitys in the order:

1. Ev_BswA_ProcessBigBang
2. Ev_SwcA_Run_BigBang
3. Ev_SwcB_Run_BigBang

7.6.1.3 Schedule Points

In order to allow explicit calls to the Os scheduler in an non-preemptive scheduling setup, the configuration element RteOsSchedulePoint shall be used.

[SWS_Rte_05113] ⌈ The RTE Generator shall create an unconditional call to the Os API *Schedule* after the execution call of the RunnableEntity if the RteOsSchedulePoint configuration parameter is set to UNCONDITIONAL. In the generated code the call to the Os API *Schedule* shall always be performed, even when the RunnableEntity itself has not been executed (called). ⌋

Since the execution of a RunnableEntity may be performed (e.g. due to mode dependent scheduling) the call of the Os API *Schedule* without any RunnableEntity execution in between might occur. in order to prohibit such a call chain the CONDITIONAL schedule point is available.

[SWS_Rte_05114] ⌈ The RTE Generator shall create a conditional call to the Os API *Schedule* after the execution call of the RunnableEntity if the RteOsSchedulePoint configuration parameter is set to CONDITIONAL. In the generated code the call to the Os API *Schedule* shall be omitted when there was already a call to the Os API *Schedule* before without any RunnableEntity execution in between. ⌋

[SWS_Rte_07042] ⌈ The Os API *Schedule* according [SWS_Rte_05113] and [SWS_Rte_05114] shall be called after the data written with implicit write access by the RunnableEntity are propagated to other RunnableEntitys as specified in [SWS_Rte_07021], [SWS_Rte_03957], [SWS_Rte_07041] and [SWS_Rte_03584] ⌋

[SWS_Rte_07043] ⌈ The Os API *Schedule* according [SWS_Rte_05113] and [SWS_Rte_05114] shall be called before the Preemption Area specific buffer used for a implicit read access of the successor RunnableEntity are filled with actual data by a copy action according [SWS_Rte_07020]. ⌋

[SWS_Rte_05115] ⌈ The RTE Generator shall create no call to the Os API *Schedule* after the execution of the RunnableEntity if the RteOsSchedulePoint configuration parameter is not present or is set to NONE. ⌋

[SWS_Rte_01373] ⌈ The RTE Generator shall support the independent setting of RteOsSchedulePoint for RteEventToTaskMappings that map the same RunnableEntity. ⌋*(SRS_Rte_00018)*

7.6.1.4 Timeprotection support

[SWS_Rte_07801] ⌈ If RteMappedToTaskRef is configured but RteVirtuallyMappedToTaskRef is not configured, the RTE shall implement/evaluate the RTEEvent that activates the RunnableEntity and execute the RunnableEntity in the OsTask referenced by RteMappedToTaskRef. ⌋

[SWS_Rte_07802] ⌈ If both RteMappedToTaskRef and RteVirtuallyMappedToTaskRef are configured, the RTE shall implement/evaluate the RTEEvent that activates the RunnableEntity in the OsTask referenced by RteVirtuallyMappedToTaskRef but execute the RunnableEntity in the OsTask referenced by RteMappedToTaskRef. The RTE shall implement this by an activation of the OsTask referenced by RteMappedToTaskRef when the RTEEvent is evaluated as "TRUE" in the OsTask referenced by RteVirtuallyMappedToTaskRef. ⌋*(SRS_Rte_00193)*

[SWS_Rte_07803] ⌈ The RTE shall reject the configuration if RteMappedToTaskRef is not configured but RteVirtuallyMappedToTaskRef is configured. ⌋*(SRS_Rte_00018)*

7.6.1.5 Os Interaction

When an OsEvent is used to activate the OsTask the reference RteUsedOsEventRef specifies which OsEvent is used.

When an OsAlarm is used to implement a TimingEvent or a BackgroundEvent the reference RteUsedOsAlarmRef specifies which OsAlarm is used.

[SWS_Rte_07806] ⌈ If RteUsedOsAlarmRef is configured and RteEventRef references a TimingEvent the RTE shall implement the TimingEvent with the OsAlarm referenced by RteUsedOsAlarmRef. ⌋*(SRS_Rte_00232)*

[SWS_Rte_07179] ⌈ If RteUsedOsAlarmRef is configured and RteEventRef references a BackgroundEvent the RTE shall implement the BackgroundEvent with the OsAlarm referenced by RteUsedOsAlarmRef. ⌋

When an OsScheduleTableExpiryPoint is used to implement a TimingEvent or a BackgroundEvent the reference RteUsedOsSchTblExpiryPointRef specifies which OsScheduleTableExpiryPoint is used.

[SWS_Rte_07807] ⌈ If RteUsedOsSchTblExpiryPointRef is configured and RteEventRef references a TimingEvent the RTE shall implement the TimingEvent with the OsScheduleTableExpiryPoint referenced by RteUsedOsSchTblExpiryPointRef. ⌋*(SRS_Rte_00232)*

[SWS_Rte_07180] ⌈ If RteUsedOsSchTblExpiryPointRef is configured and RteEventRef references a BackgroundEvent the RTE shall implement the BackgroundEvent with the OsScheduleTableExpiryPoint referenced by RteUsedOsSchTblExpiryPointRef. ⌋

If neither RteUsedOsSchTblExpiryPointRef nor RteUsedOsAlarmRef are configured and RteEventRef references a TimingEvent the RTE is free to implement the TimingEvent with the OsAlarm or OsScheduleTableExpiryPoint of its choice.

[SWS_Rte_07808] ⌈ The RTE shall reject the configuration if both RteUsedOsAlarmRef and RteUsedOsSchTblExpiryPointRef are configured. ⌋*(SRS_Rte_00018)*

[SWS_Rte_07809] ⌈ The RTE shall reject the configuration if RteUsedOsAlarmRef or RteUsedOsSchTblExpiryPointRef is configured and RteEventRef doesn't reference a TimingEvent or a BackgroundEvent. ⌋*(SRS_Rte_00018)*

7.6.1.6 Background activation

If neither RteUsedOsSchTblExpiryPointRef nor RteUsedOsAlarmRef is configured and RteEventRef references a BackgroundEvent the RteMappedToTaskRef has to reference the OsTask used for *Background* activation of *RunnableEntities* and *Basic Software Schedulable Entities* on the related CPU core where the partition of the software component is mapped.

The OsTask used for BackgroundEvent triggering has to have the lowest priority on the core. There can only be one 'Background' OsTask per CPU core.

[SWS_Rte_07181] ⌈ The RTE shall reject the configuration if

- RteEventRef references a BackgroundEvent and
- neither RteUsedOsAlarmRef nor RteUsedOsSchTblExpiryPointRef are configured and
- if RteMappedToTaskRef reference an OsTask which has not the lowest priority of the core.

⌋*(SRS_Rte_00018)*

7.6.1.7 Constraints

There are some constraints which do apply when actually mapping the RunnableEntity to an OsTask:

[SWS_Rte_05082] ⌈ The following restrictions apply to RTEEvents which are used to activate RunnableEntity. OsEvents that are used to wakeUpFromWaitPoint shall not be included in the mapping. ⌋

When a wakeUpFromWaitPoint is occurring the RunnableEntity resumes its execution in the context of the originally activated OsTask.

[SWS_Rte_05083] ⌈ The RTE Generator shall reject configurations where a RunnableEntity has its canBeInvokedConcurrently attribute set to *false*, and this RunnableEntity is mapped to different tasks which can preempt each other. ⌋

[SWS_Rte_07229] ⌈ To evaluate [SWS_Rte_05083] in case of triggered runnables which are activated by a direct function call ([SWS_Rte_07214], [SWS_Rte_07224] and [SWS_Rte_07554]) the OsTask (context of the caller) is defined by the RunnableEntity's containing the activating InternalTriggeringPoint or ExternalTriggeringPoint. ⌋*(SRS_Rte_00162, SRS_Rte_00163, SRS_Rte_00230)*

[SWS_Rte_07155] ⌈ To evaluate [SWS_Rte_05083] in case of OnEntry ExecutableEntitys, OnTransition ExecutableEntitys, OnExit ExecutableEntitys, and mode switch acknowledge ExecutableEntitys which are activated by a direct function call the OsTask (context of the caller) is

defined by the RunnableEntity's containing the activating ModeSwitchPoint.
⌋(SRS_Rte_00143, SRS_Rte_00144)

RteEventToTaskMapping

SWS Item	[ECUC_Rte_09020]
Container Name	RteEventToTaskMapping
Description	Maps an instance of a RunnableEntity onto one OsTask based on the activating RTEEvent. In the case of a RunnableEntity executed via a direct function call this RteEventToTaskMapping is still specified but no RteMappedToTask element is included. The RtePositionInTask parameter is necessary to provide an ordering of events invoked by the same RTE API.
Configuration Parameters	

Name	RteActivationOffset [ECUC_Rte_09018]		
Description	Activation offset in seconds.		
Multiplicity	0..1		
Type	EcucFloatParamDef		
Range	0 .. INF		
Default Value			
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteEventRef [ECUC_Rte_09019]		
Description	Reference to the description of the RTEEvent which is pointing to the RunnableEntity being mapped. This allows a fine grained mapping of RunnableEntites based on the activating RTEEvent.		
Multiplicity	1		
Type	Foreign reference to RTE-EVENT		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteImmediateRestart [ECUC_Rte_09092]
Description	When RteImmediateRestart is set to true the RunnableEntitiy shall be immediately re-started after termination if it was activated by this RTEEvent while it was already started. This parameter shall not be set to true when the mapped RTEEvent refers to a RunnableEntity which minimumStartInterval attribute is > 0.
Multiplicity	1
Type	EcucBooleanParamDef
Default Value	false

Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteMappedToTaskRef [ECUC_Rte_09021]		
Description	Reference to the OsTask the RunnableEntity activated by the RteEventRef is mapped to. If no reference to the OsTask is specified the RunnableEntity shall be executed via a direct function call. The fact that no reference to an OsTask is specified for a RunnableEntity does not necessarily imply that every RTE generator has to support the implementation of this RunnableEntity as a direct function call. The standard set of use cases for direct function calls that has to be supported by every RTE generator is explicitly stated as requirements in this document. For further optimization RTE vendors are free to support additional scenarios of direct function call implementations that are not explicitly required in this document.		
Multiplicity	0..1		
Type	Reference to OsTask		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteOsSchedulePoint [ECUC_Rte_09022]	
Description	Introduce a schedule point by explicitly calling Os Schedule service after the execution of the ExecutableEntity. The Rte generator is allowed to optimize several consecutive calls to Os schedule into one single call if the ExecutableEntity executions in between have been skipped. The absence of this parameter is interpreted as "NONE". It shall be considered an invalid configuration if the task is preemptable and the value of this parameter is not set to "NONE" or the parameter is absent.	
Multiplicity	0..1	
Type	EcucEnumerationParamDef	
Range	CONDITIONAL	A Schedule Point shall be introduced at the end of the execution of this ExecutableEntity. The Schedule Point can be skipped if several Schedule Points would be called without any ExecutableEntity execution in between.
	NONE	No Schedule Point shall be introduced at the end of the execution of this ExecutableEntity.

	UNCONDITIONAL	A Schedule Point shall always be introduced at the end of the execution of this ExecutableEntity.	
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RtePositionInTask [ECUC_Rte_09023]		
Description	Each RunnableEntity mapped to an OsTask has a specific position within the task execution. For periodic activation this is the order of execution. For event driver activation this is the order of evaluation which actual RunnableEntity has to be executed. In case of direct function calls this parameter is necessary to provide an ordering of events when several ExecutableEntities are invoked by the same RTE API.		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default Value			
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteUsedInitFnc [ECUC_Rte_09116]		
Description	The RunnableEntity is executed during initialization in the context of the Rte_Init_<InitContainer> function.		
Multiplicity	0..1		
Type	Reference to RteInitializationRunnableBatch		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteUsedOsAlarmRef [ECUC_Rte_09024]		
Description	If an OsAlarm is used to activate the OsTask this RteEvent is mapped to it shall be referenced here.		
Multiplicity	0..1		
Type	Reference to OsAlarm		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteUsedOsEventRef [ECUC_Rte_09025]		
Description	If an OsEvent is used to activate the OsTask this RteEvent is mapped to it shall be referenced here.		
Multiplicity	0..1		
Type	Reference to OsEvent		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteUsedOsSchTblExpiryPointRef [ECUC_Rte_09026]		
Description	If an OsScheduleTableExpiryPoint is used to activate the OsTask this RteEvent is mapped to it shall be referenced here.		
Multiplicity	0..1		
Type	Reference to OsScheduleTableExpiryPoint		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteVirtuallyMappedToTaskRef [ECUC_Rte_09027]		
Description	Optional reference to an OsTask where the activation of this RteEvent shall be evaluated. The actual execution of the Runnable Entity shall happen in the OsTask referenced by RteMappedToTaskRef.		
Multiplicity	0..1		
Type	Reference to OsTask		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

No Included Containers

7.6.2 Rte Os Interaction

This section contains configuration items which are closely related to the interaction of the Rte with the Os.

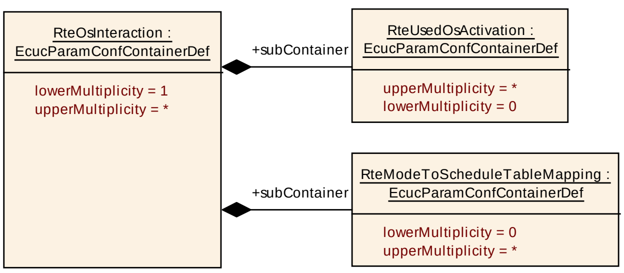


Figure 7.8: Specification of the Rte/Os Interaction

7.6.2.1 Activation using Os features

This is a collection of possible ways how the Rte might utilize Os to achieve various activation scenarios. The used Os objects are referenced in these configuration entities.

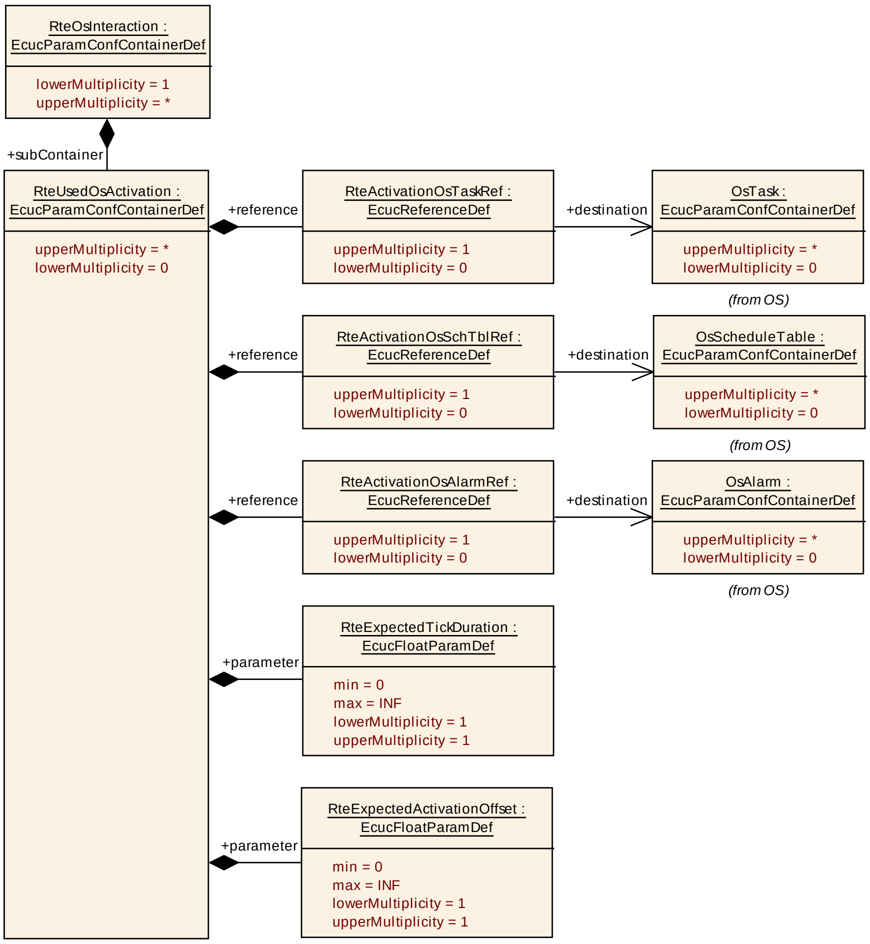


Figure 7.9: Configuration how activation is implemented

RteUsedOsActivation

SWS Item	[ECUC_Rte_09060]
Container Name	RteUsedOsActivation
Description	Attributes used in the activation of OsTasks and Runnable Entities.
Configuration Parameters	

Name	RteActivationOsAlarmRef [ECUC_Rte_09045]		
Description	Reference to an OsAlarm.		
Multiplicity	0..1		
Type	Reference to OsAlarm		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteActivationOsSchTblRef [ECUC_Rte_09046]		
Description	Reference to an OsScheduleTable.		
Multiplicity	0..1		
Type	Reference to OsScheduleTable		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteActivationOsTaskRef [ECUC_Rte_09047]		
Description	Reference to an OsTask.		
Multiplicity	0..1		
Type	Reference to OsTask		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteExpectedActivationOffset [ECUC_Rte_09048]		
Description	Activation offset in seconds. Important: This is a requirement from the Rte towards the Os/Mcu setup. The Rte Generator shall assume this activation offset to be fulfilled.		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	0 .. INF		
Default Value			
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteExpectedTickDuration [ECUC_Rte_09049]		
Description	The expected tick duration in seconds which shall be configured to drive the OsScheduleTables or OsAlarm. Important: This is a requirement from the Rte towards the Os/Mcu setup. The Rte Generator shall assume this tick duration to be fulfilled.		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	0 .. INF		
Default Value			
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

No Included Containers

7.6.2.2 Modes and Schedule Tables

Optional configuration of the Rte to support the mapping of modes and Os' schedule tables.

[SWS_Rte_05146] ⌈ The referenced schedule table of `RteModeScheduleTableRef` shall be activated if one of the modes referenced in `RteModeSchtblMapModeDeclarationRef` is active in the `mode machine instances` from the references of

- `RteModeSchtblMapSwc` or
- `RteModeSchtblMapBsw`.

⌋

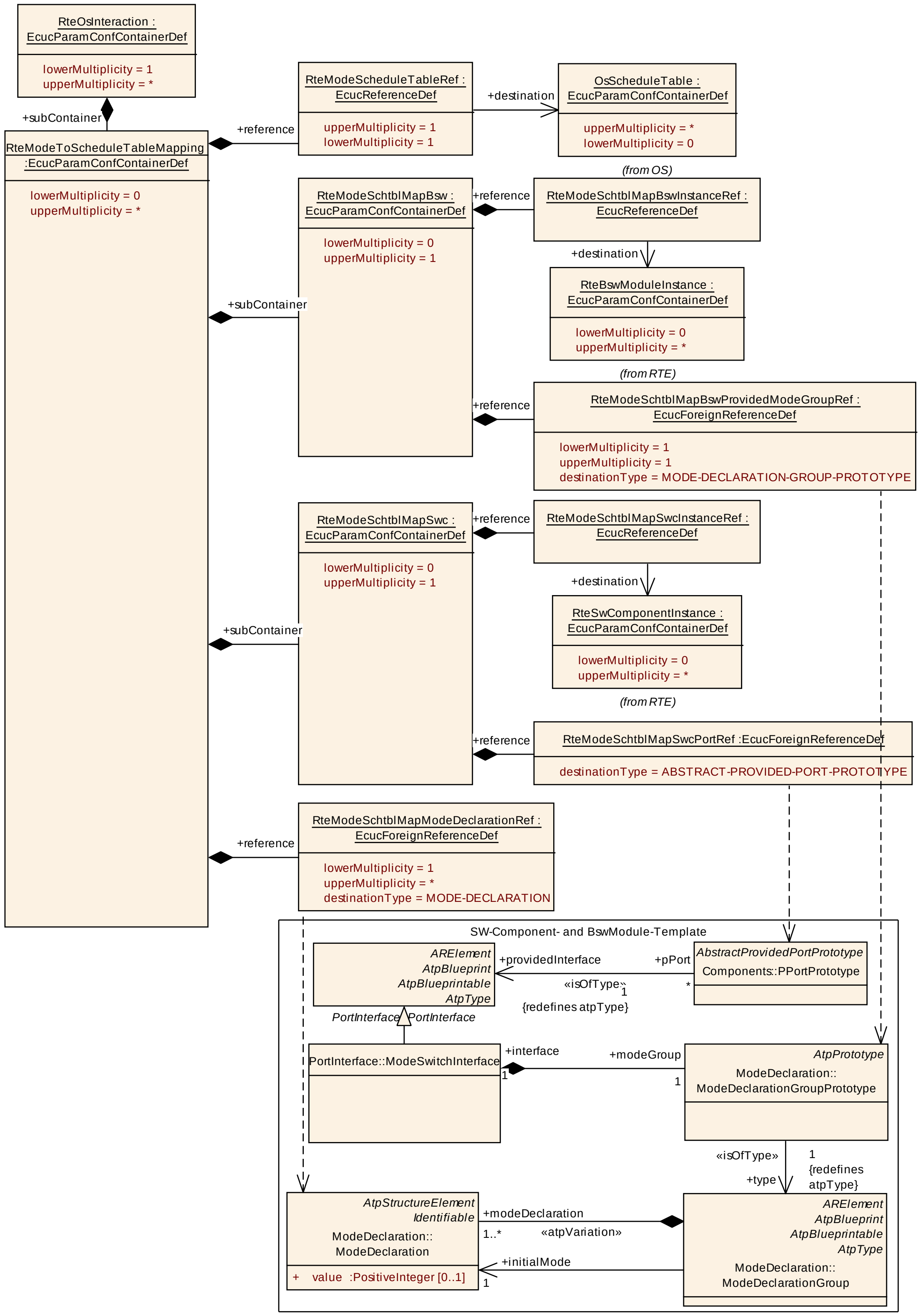


Figure 7.10: Configuration how modes are interacting with schedule tables

[SWS_Rte_02759] ⌈ RTE shall reject a configuration, if the RteModeSchtblMapSwcPortRef : EcucForeignReferenceDef does not reference a PPortPrototype or PRPortPrototype of the type of an ModeSwitchInterface. ⌋

[SWS_Rte_02760] ⌈ RTE shall reject a configuration, if the ModeDeclarationGroupPrototype referenced by a RteModeSchtblMapBswProvidedModeGroupRef:EcucForeignReferenceDef is not in the role of a providedModeGroup. ⌋

RteModeToScheduleTableMapping

SWS Item	[ECUC_Rte_09058]
Container Name	RteModeToScheduleTableMapping
Description	Provides configuration input in which Modes of a ModeDeclarionGroupPrototype of a Mode Manager a OsScheudleTable shall be active. The Mode Manager is either specified as a SwComponentPrototype (RteModeSchtblMapSwc) or as a BSW-Module (RteModeSchtblMapBsw).
Configuration Parameters	

Name	RteModeScheduleTableRef [ECUC_Rte_09050]		
Description	Reference to the OsScheduleTable which shall be active in the specified RteModeSchblMapModeDeclarationRefs.		
Multiplicity	1		
Type	Reference to OsScheduleTable		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteModeSchtblMapModeDeclarationRef [ECUC_Rte_09054]		
Description	Reference to the ModeDeclarations.		
Multiplicity	1..*		
Type	Foreign reference to MODE-DECLARATION		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Included Containers		
Container Name	**Multiplicity**	**Scope / Depedency**
RteModeSchtblMapBsw	0..1	Specifies an instance of a ModeDeclarationGroupPrototype of a Bsw-Module.
RteModeSchtblMapSwc	0..1	Specifies an instance of a ModeDeclarationGroupPrototype of a SwComponentPrototype.

RteModeSchtblMapSwc

SWS Item	[ECUC_Rte_09055]		
Container Name	RteModeSchtblMapSwc		
Description	Specifies an instance of a ModeDeclarationGroupPrototype of a SwComponentPrototype.		
Configuration Parameters			

Name	RteModeSchtblMapSwcInstanceRef [ECUC_Rte_09056]		
Description	Reference to an instance specification of a SwComponentPrototype.		
Multiplicity	1		
Type	Reference to RteSwComponentInstance		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteModeSchtblMapSwcPortRef [ECUC_Rte_09057]		
Description	Reference to the PPortPrototype of a SwComponentPrototype.		
Multiplicity	1		
Type	Foreign reference to ABSTRACT-PROVIDED-PORT-PROTOTYPE		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

No Included Containers

RteModeSchtblMapBsw

SWS Item	[ECUC_Rte_09051]		
Container Name	RteModeSchtblMapBsw		
Description	Specifies an instance of a ModeDeclarationGroupPrototype of a Bsw-Module.		
Configuration Parameters			

Name	RteModeSchtblMapBswInstanceRef [ECUC_Rte_09052]		
Description	Reference to an instance specification of a Bsw-Module.		
Multiplicity	1		
Type	Reference to RteBswModuleInstance		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteModeSchtblMapBswProvidedModeGroupRef [ECUC_Rte_09053]		
Description	Reference to an instance of a ModeDeclarationGroupPrototype of a Bsw-Module.		
Multiplicity	1		
Type	Foreign reference to MODE-DECLARATION-GROUP-PROTOTYPE		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

No Included Containers

7.6.3 Exclusive Area implementation

The RTE Generator can be configured to implement a different data consistency mechanism for each ExclusiveArea defined for an AUTOSAR software-component.

In figure 7.11 the configuration of the actually selected data consistency mechanism is shown.

[constr_3510] Exclude usage of OS_SPINLOCK in RteExclusiveAreaImplementation ⌈ The usage of the enumeration literal OS_SPINLOCK for the parameter RteExclusiveAreaImplMechanism shall be excluded if the parameter RteExclusiveAreaImplMechanism is used in the context of the container RteExclusiveAreaImplementation. ⌋

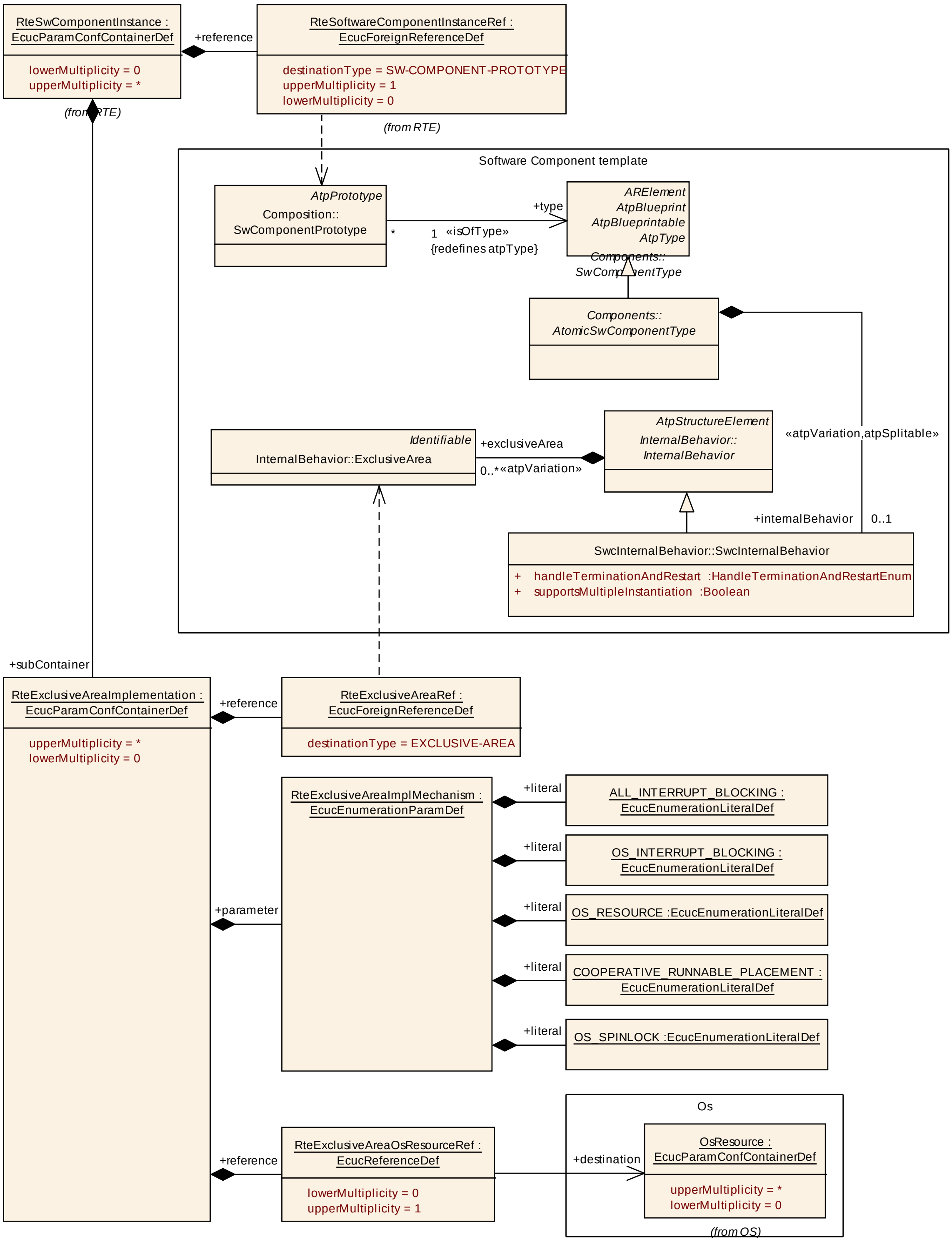


Figure 7.11: Configuration of the `ExclusiveArea` implementation

RteExclusiveAreaImplementation

SWS Item	[ECUC_Rte_09030]
Container Name	RteExclusiveAreaImplementation
Description	Specifies the implementation to be used for the data consistency of this ExclusiveArea.
Configuration Parameters	

Name	RteExclusiveAreaImplMechanism [ECUC_Rte_09029]		
Description	To be used implementation mechanism for the specified ExclusiveArea.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	ALL_INTERRUPT_BLOCKING		
	COOPERATIVE_RUNNABLE_PLACEMENT		
	OS_INTERRUPT_BLOCKING		
	OS_RESOURCE		
	OS_SPINLOCK		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteExclusiveAreaOsResourceRef [ECUC_Rte_09031]		
Description	Optional reference to an OsResource in case RteExclusiveAreaImplMechanism is configured to OS_RESOURCE for this ExclusiveArea.		
Multiplicity	0..1		
Type	Reference to OsResource		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteExclusiveAreaRef [ECUC_Rte_09032]		
Description	Reference to the ExclusiveArea.		
Multiplicity	1		
Type	Foreign reference to EXCLUSIVE-AREA		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

No Included Containers

7.6.4 NVRam Allocation

The configuration of the NVRam access does involve several templates, because it closes the gap between the AUTOSAR software-components, the NVRAM Manager Services and the BSW Modules.

In figure 7.12 the related information from the AUTOSAR Software Component Template is shown.

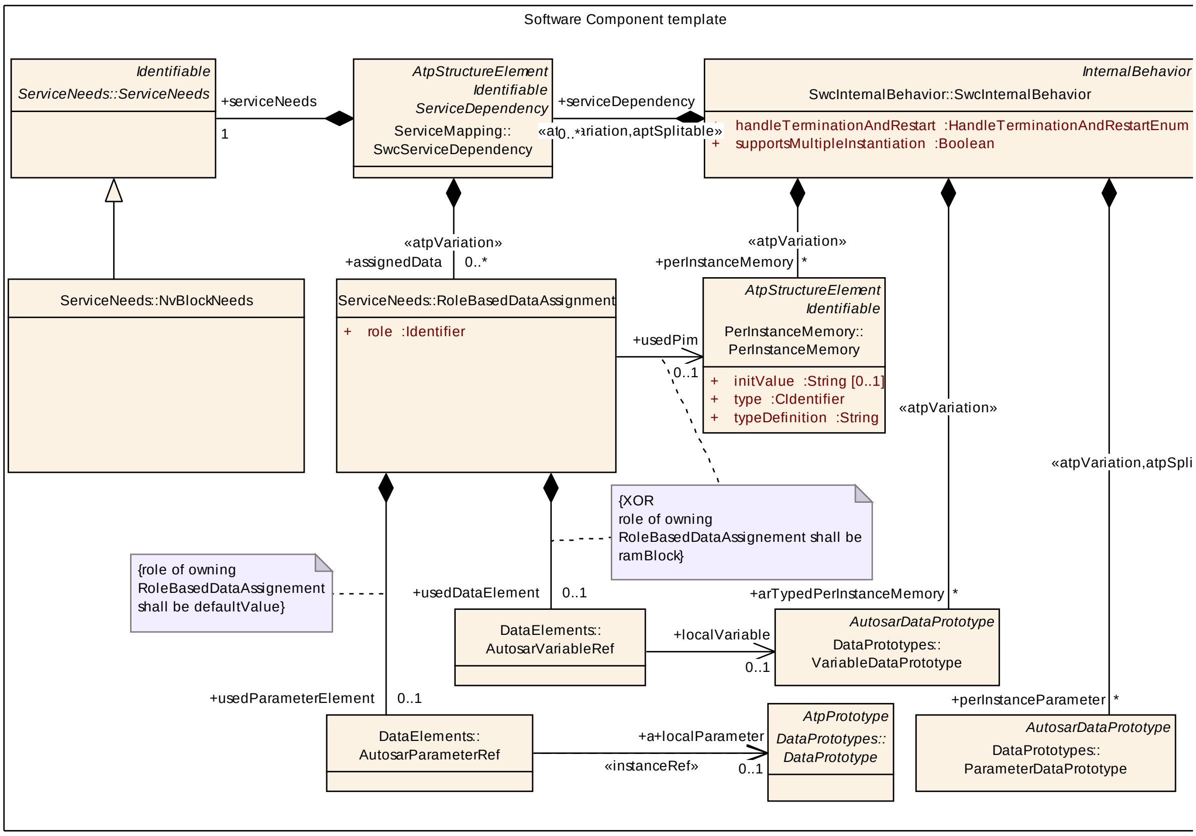


Figure 7.12: software-component information of NVRam Service needs

In figure 7.13 the ECU Configuration part of the NVRam allocation is shown. It relates the software-components' `SwcServiceDependency` and `NvBlockNeeds` information with the NVRam Managers `NvMBlockDescriptor` and the linker symbols of the RAM and ROM sections to be used.

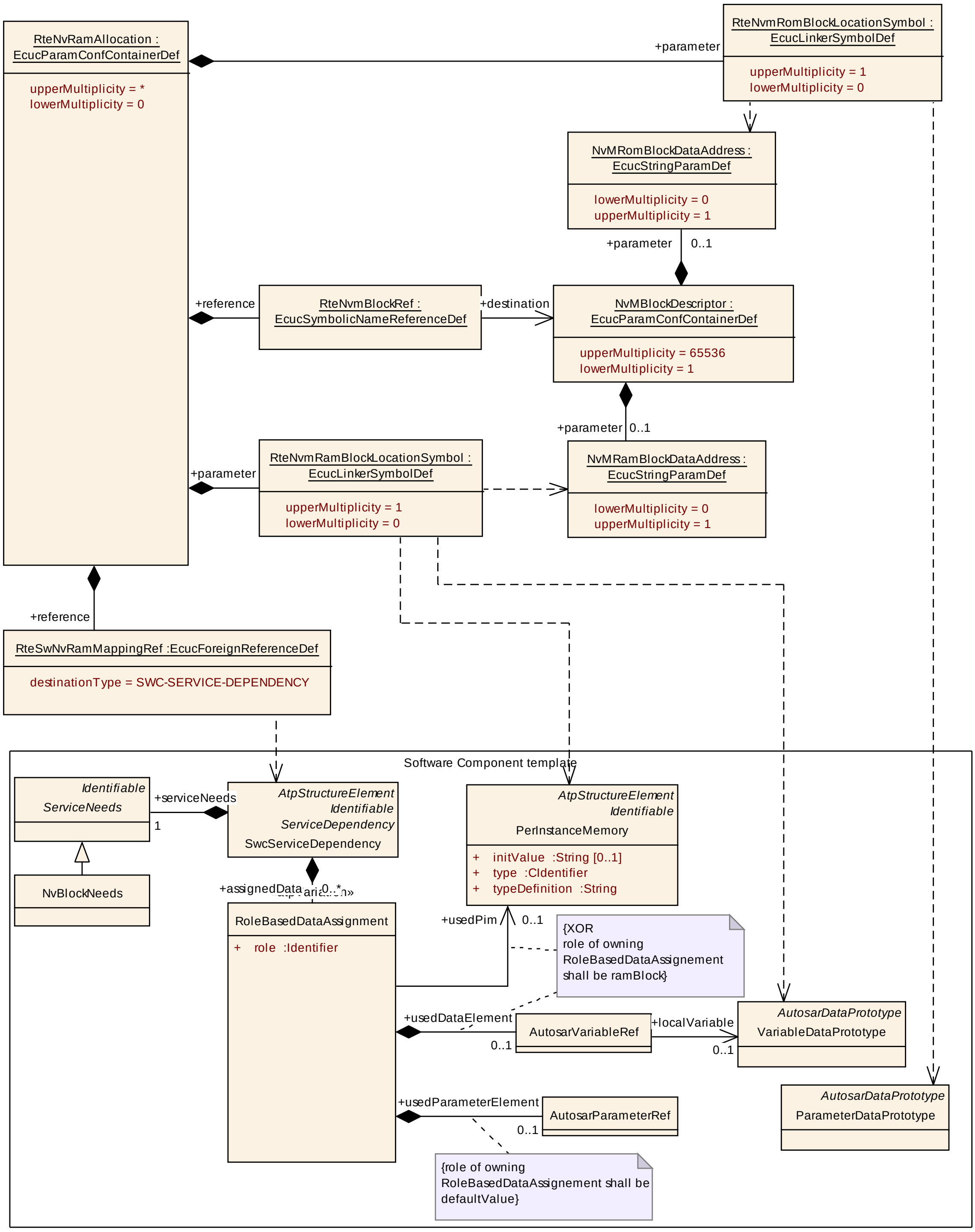


Figure 7.13: ECU Configuration of the NVRam Service

RteNvRamAllocation

SWS Item	[ECUC_Rte_09040]
Container Name	RteNvRamAllocation
Description	Specifies the relationship between the AtomicSwComponentType's NVRAMMapping / NVRAM needs and the NvM module configuration.
Configuration Parameters	

Name	RteNvmBlockRef [ECUC_Rte_09041]		
Description	Reference to the used NvM block for storage of the NVRAMMapping information.		
Multiplicity	1		
Type	Symbolic name reference to NvMBlockDescriptor		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteNvmRamBlockLocationSymbol [ECUC_Rte_09042]		
Description	This is the name of the linker object name where the NVRam Block will be mirrored by the Nvm. This symbol will be resolved into the parameter "NvmRamBlockDataAddress" from the "NvmBlockDescriptor".		
Multiplicity	0..1		
Type	EcucLinkerSymbolDef		
Default Value			
Regular Expression			
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteNvmRomBlockLocationSymbol [ECUC_Rte_09043]		
Description	This is the name of the linker object name where the NVRom Block will be accessed by the Nvm. This symbol will be resolved into the parameter "NvmRomBlockDataAddress" from the "NvmBlockDescriptor".		
Multiplicity	0..1		
Type	EcucLinkerSymbolDef		
Default Value			
Regular Expression			
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteSwNvRamMappingRef [ECUC_Rte_09044]		
Description	Reference to the SwSeriveDependency which is used to specify the NvBlockNeeds.		
Multiplicity	1		
Type	Foreign reference to SWC-SERVICE-DEPENDENCY		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

No Included Containers

7.6.5 SWC Trigger queuing

This configuration determine the size of the queue queuing the issued triggers.

The RteExternalTriggerConfig container and RteInternalTriggerConfig container is defined in the context of the RteSwComponentInstance which already predefines the context of the Trigger / InternalTriggeringPoint.

[constr_9005] The references RteSwcTriggerSourceRef has to be consistent with the RteSoftwareComponentInstanceRef ⌈ The references RteSwcTriggerSourceRef has to be consistent with the RteSoftwareComponentInstanceRef. This means the referenced Trigger / InternalTriggeringPoint has to belong to the AtomicSwComponentType which is referenced by the related SwComponentPrototype. ⌋

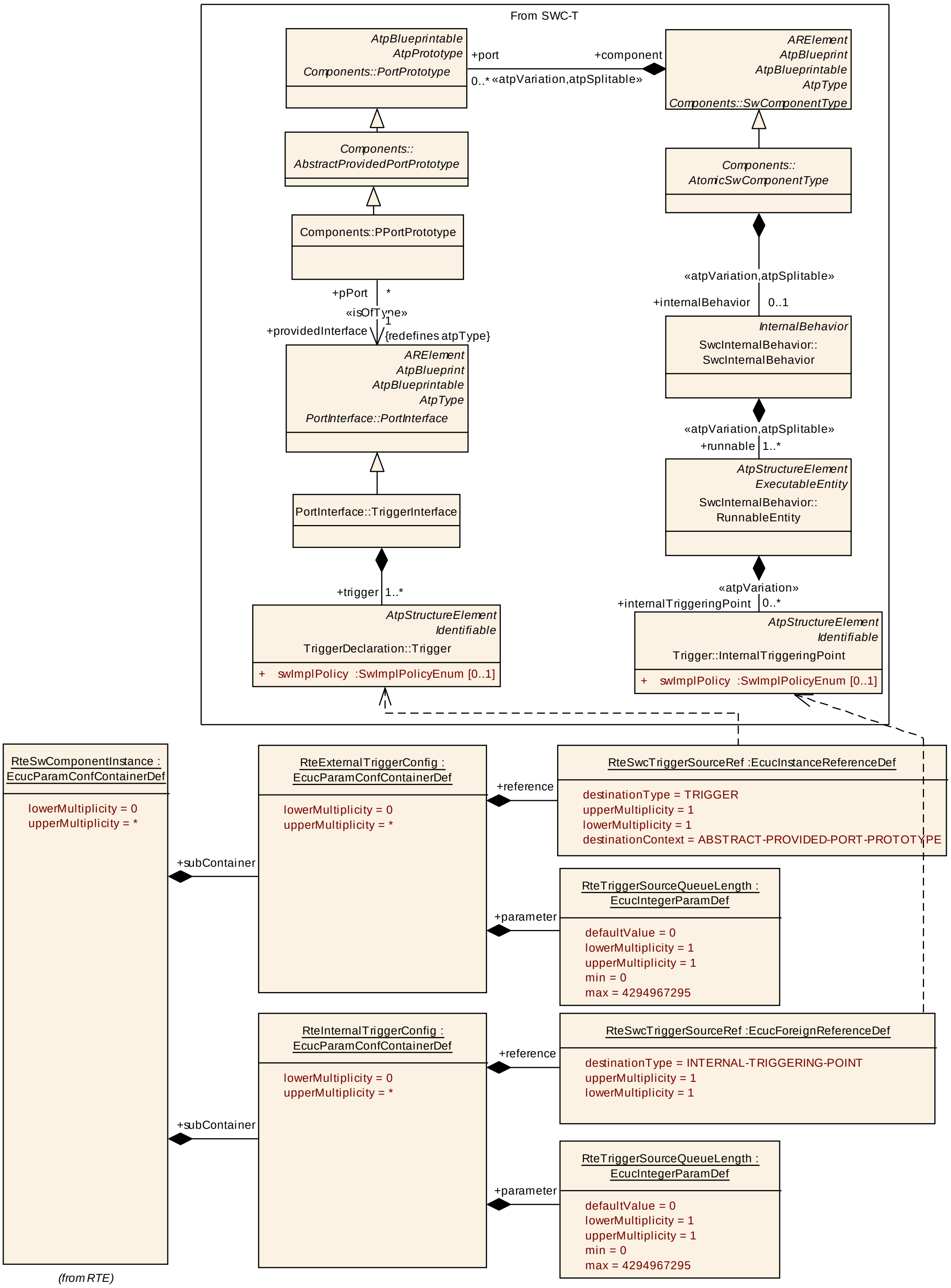


Figure 7.14: Configuration of SWC Trigger queuing

RteExternalTriggerConfig

SWS Item	[ECUC_Rte_09105]		
Container Name	RteExternalTriggerConfig		
Description	Defines the configuration of External Trigger Event Communication for Software Components		
Configuration Parameters			

Name	RteSwcTriggerSourceRef [ECUC_Rte_09106]		
Description	Reference to a Trigger instance in the pPortPrototype of the related component instance. The referenced Trigger instance has to belong to the same software component instance as the RteSwComponentInstance owning this parameter configures.		
Multiplicity	1		
Type	Instance reference to TRIGGER context: ABSTRACT-PROVIDED-PORT-PROTOTYPE		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteTriggerSourceQueueLength [ECUC_Rte_09095]		
Description	Length of trigger queue on the trigger source side. The queue is implemented by the RTE. A value greater or equal to 1 requests an queued behavior. Setting the value of RteTriggerSourceQueueLength to 0 requests an none queued implementation of the trigger communication. If there is no RteTriggerSourceQueueLength configured for a Trigger Emitter the default value of 0 applies as well.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default Value	0		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

No Included Containers

RteInternalTriggerConfig

SWS Item	[ECUC_Rte_09096]		
Container Name	RteInternalTriggerConfig		
Description	Defines the configuration of Inter Runnable Triggering for Software Components		
Configuration Parameters			

Name	RteSwcTriggerSourceRef [ECUC_Rte_09097]		
Description	Reference to an InternalTriggeringPoint of the related component instance. The referenced InternalTriggeringPoint has to belong to the same software component instance as the RteSwComponentInstance owning this parameter configures.		
Multiplicity	1		
Type	Foreign reference to INTERNAL-TRIGGERING-POINT		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteTriggerSourceQueueLength [ECUC_Rte_09098]		
Description	Length of trigger queue on the trigger source side. The queue is implemented by the RTE. A value greater or equal to 1 requests an queued behavior. Setting the value of RteTriggerSourceQueueLength to 0 requests an none queued implementation of the trigger communication. If there is no RteTriggerSourceQueueLength configured for a Trigger Emitter the default value of 0 applies as well.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default Value	0		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

No Included Containers

7.7 Handling of Software Component types

7.7.1 Selection of Software-Component Implementation

During the system development there is no need to select the actual implementation which will be later integrated on one ECU. Therefore the *ECU Extract of System Description* may not specify the SwcImplementation information yet.

For RTE Generation the information about the to be used SwcImplementation for each SwComponentType needs be provided to the RTE Generator (regardless whether the information is from the Ecu Extract or the Ecu Configuration.

The mapping of SwcImplementation to SwComponentType is done in the Ecu Configuration of the Rte using the two references RteComponentTypeRef and RteImplementationRef (see figure 7.15). For the mapping in the Ecu Extract please refer to the Specification of the System Template [8].

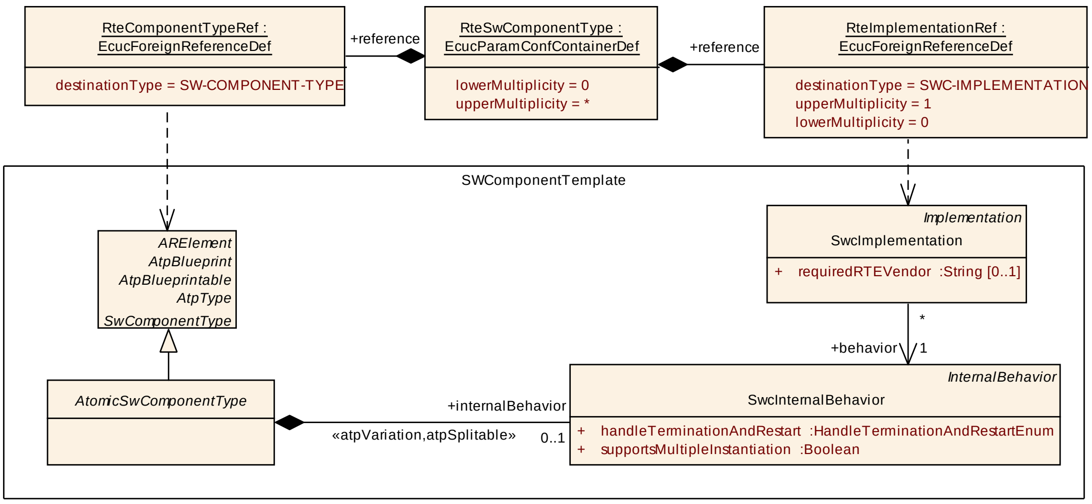


Figure 7.15: Selection of the Implementation for an AtomicSwComponentType

RteSwComponentType

SWS Item	[ECUC_Rte_09006]
Container Name	RteSwComponentType
Description	Representation of one SwComponentType for the base of all configuration parameter which are affecting the whole type and not a specific instance.
Configuration Parameters	

Name	RteBypassSupportEnabled [ECUC_Rte_09114]
Description	Individual switch to enable the bypass support for this software component type.
Multiplicity	0..1
Type	EcucBooleanParamDef
Default Value	false

Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteComponentTypeRef [ECUC_Rte_09003]		
Description	Reference to either AtomicSwComponentType or ParameterSwComponentType.		
Multiplicity	1		
Type	Foreign reference to SW-COMPONENT-TYPE		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteImplementationRef [ECUC_Rte_09028]		
Description	The Implementation which shall be assigned to the SwComponentType.		
Multiplicity	0..1		
Type	Foreign reference to SWC-IMPLEMENTATION		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Included Containers		
Container Name	**Multiplicity**	**Scope / Depedency**
RteComponentType Calibration	0..1	Specifies for each ParameterSwComponentType or AtomicSwComponentType whether calibration is enabled. If references to SwAddrMethod are provided in RteCalibrationSwAddrMethodRef only ParameterDataPrototypes with the referenced SwAddrMethod shall have software calibration support enabled.

7.7.2 Component Type Calibration

In the AUTOSAR Software Component Template two places may provide calibration data: the `ParameterSwComponentType` and the `AtomicSwComponentType` (or more precisely the subclasses of `AtomicSwComponentType`). Whether the calibration is enabled for a specific `SwComponentType` can be configured as shown in figure 7.16.

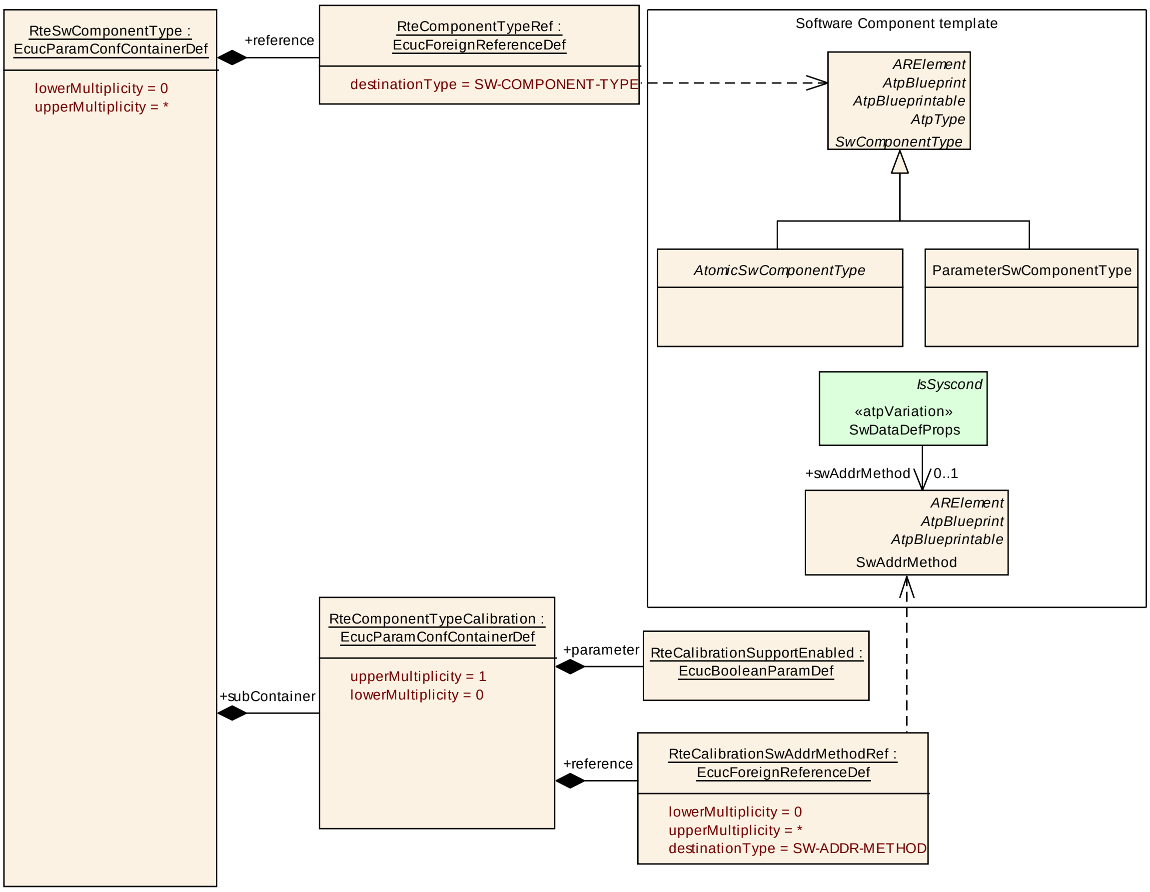


Figure 7.16: Configuration of the calibration for the ParameterSwComponentType

The foreign reference RteComponentTypeRef identifies the SwComponentType (which is limited to ParameterSwComponentType and AtomicSwComponentType). The boolean parameter RteCalibrationSupportEnabled specifies whether calibration shall be enabled for the specified SwComponentType.

[SWS_Rte_05145] ⌈ For a ParameterDataPrototype of the referenced SwComponentType software calibration support shall be enabled if the parameter RteCalibrationSupportEnabled is set to *true* and in the corresponding container RteComponentTypeCalibration

- not a single RteCalibrationSwAddrMethodRef exists or
- a reference RteCalibrationSwAddrMethodRef to the SwAddrMethod of the ParameterDataPrototype exists.

⌋*(SRS_Rte_00154, SRS_Rte_00156, SRS_Rte_00158)*

RteComponentTypeCalibration

SWS Item	[ECUC_Rte_09039]
Container Name	RteComponentTypeCalibration
Description	Specifies for each ParameterSwComponentType or AtomicSwComponentType whether calibration is enabled. If references to SwAddrMethod are provided in RteCalibrationSwAddrMethodRef only ParameterDataPrototypes with the referenced SwAddrMethod shall have software calibration support enabled.
Configuration Parameters	

Name	RteCalibrationSupportEnabled [ECUC_Rte_09037]		
Description	Enables calibration support for the specified ParameterSwComponentType or AtomicSwComponentType.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value			
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteCalibrationSwAddrMethodRef [ECUC_Rte_09038]		
Description	Reference to the SwAddrMethod for which software calibration support shall be enabled.		
Multiplicity	0..*		
Type	Foreign reference to SW-ADDR-METHOD		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

No Included Containers

7.8 Implicit communication configuration

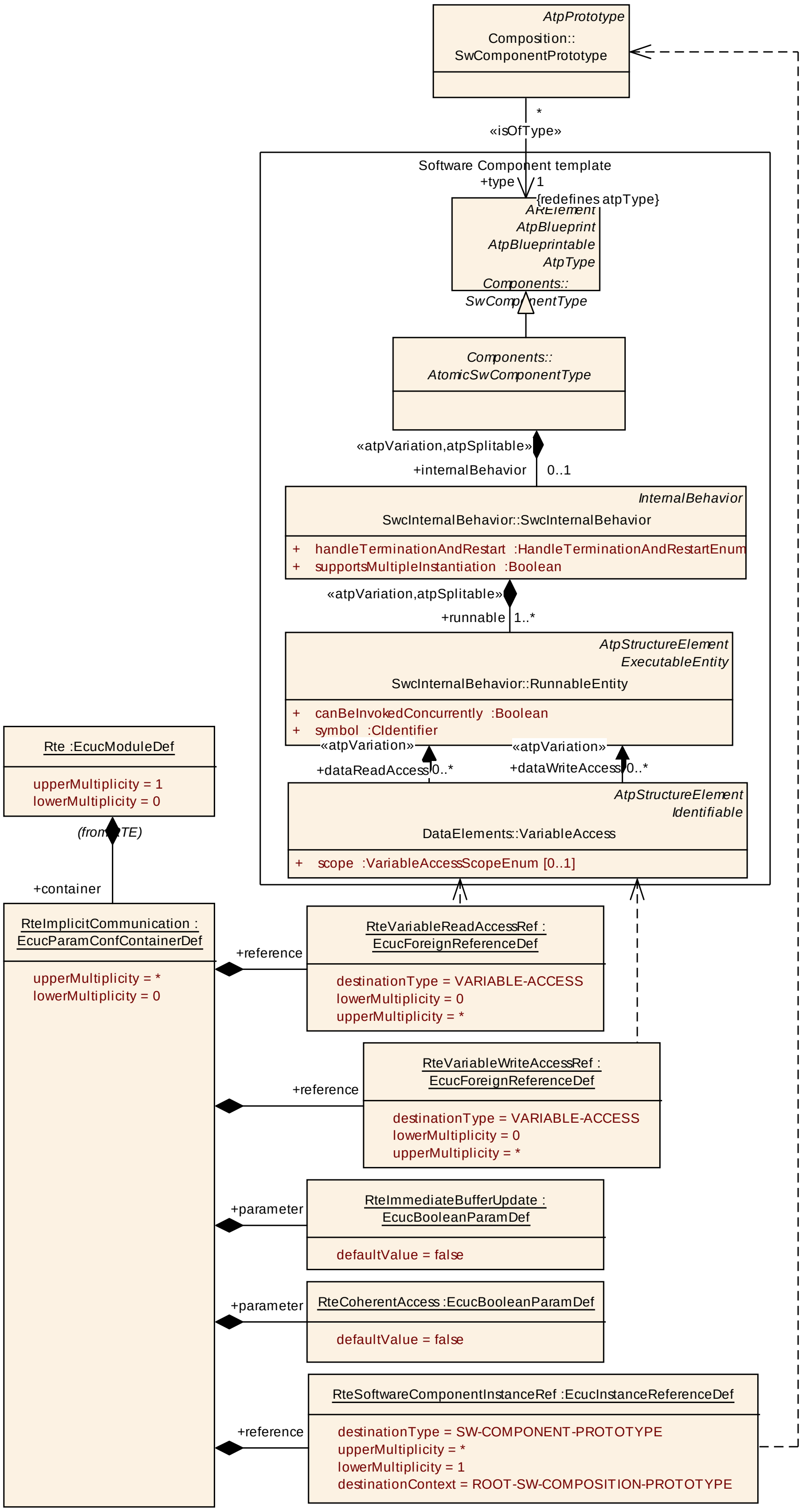


Figure 7.17: Configuration of the implicit communication

RteImplicitCommunication

SWS Item	[ECUC_Rte_09034]
Container Name	RteImplicitCommunication
Description	Configuration of the Implicit Communication behavior to be generated.
Configuration Parameters	

Name	RteCoherentAccess [ECUC_Rte_09091]		
Description	If set to true the referenced VariableAccess'es of this RteImplicitCommunication container are in one CoherencyGroup. Data values for Coherent Implicit Read Access'es are read before the first reading RunnbaleEntity starts and are stable during the execution of all the reading RunnableEntitys; except Coherent Implicit Write Access'es belongs to the same Coherency Group. Data values written by Coherent Implicit Write Access'es are available for readers not belonging to the Coherency Group after the last writing RunnableEntity has terminated. Please note that a Coherent Implicit Data Access can be defined for VariableAccess'es to same and different VariableDataElements. Nevertheless all Coherent Implicit Data Access'es of one Coherency Group have to be executed in the same task.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value	false		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteImmediateBufferUpdate [ECUC_Rte_09033]		
Description	If set to true the RTE will perform preemption area specific buffer update immediately before (for VariableAccess in the role dataReadAccess) resp. after (for VariableAccess in the role dataWriteAccess) Runnable execution.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value	false		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteSoftwareComponentInstanceRef [ECUC_Rte_09090]		
Description	Reference to a SwComponentPrototype. This denotes the instances of the VariableAccess belonging to the RteImplicitCommunication.		
Multiplicity	1..*		
Type	Instance reference to SW-COMPONENT-PROTOTYPE context: ROOT-SW-COMPOSITION-PROTOTYPE		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteVariableReadAccessRef [ECUC_Rte_09035]		
Description	Reference to the VariableAccess in the dataReadAccess role.		
Multiplicity	0..*		
Type	Foreign reference to VARIABLE-ACCESS		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteVariableWriteAccessRef [ECUC_Rte_09036]		
Description	Reference to the VariableAccess in the dataWriteAccess role.		
Multiplicity	0..*		
Type	Foreign reference to VARIABLE-ACCESS		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

No Included Containers

Please note, that `RteImplicitCommunication` is defined as a container of `RteEcucModuleDef` to support the creation of the ECU Configuration Parameter Values related to `RteImplicitCommunication` independent from the other ECU Configuration Parameter Values. Typically the need for `Coherent Implicit Data Accesses` is known by the vendor of a set of software components. As long as `shortName`s of the `RootSwCompositionPrototype` and the referenced `CompositionSwComponentType` - describing the software of a flat ECU Extract - are known the ECU Configuration Parameter Values related to `RteImplicitCommunication` can be prescribed. In this case it is preferable to use relative references to the Vendor Specific Module Definition (VSMD), to `RootSwCompositionPrototype` and `CompositionSwComponentType` describing the software of a flat ECU Extract. With this relative references the ECU Configuration Parameter Values are independent from `ARPackage` structure only known by the ECU integrator. Nevertheless the `shortName` and location of of the `EcucModuleConfigurationValues` must be defined upfront.

7.9 Communication infrastructure

The configuration of the communication infrastructure (interaction of the RTE with the Com-Stack) is entirely predetermined by the ECU Extract provided as an input. The required input can be found in the AUTOSAR System Template [8] sections "Data Mapping" and "Communication".

In case the RTE does utilize the Com module for intra-ECU communication it is up to the vendor-specific configuration of the RTE to ensure configuration consistency.

7.10 Configuration of Client-Server Serialization

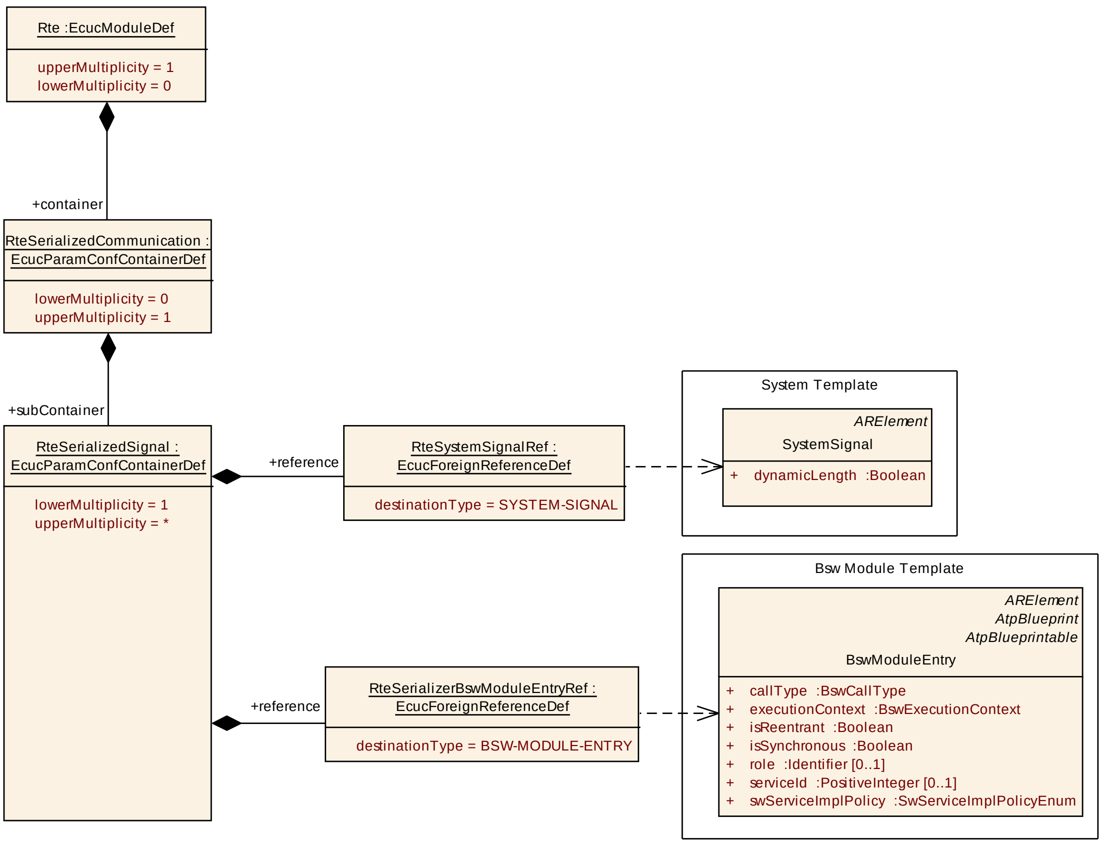


Figure 7.18: Configuration of Client-Server Serialization

RteSerializedCommunication

SWS Item	[ECUC_Rte_09108]
Container Name	RteSerializedCommunication
Description	This container collects all serialization functions that are used by the RTE.
Configuration Parameters	

Included Containers		
Container Name	**Multiplicity**	**Scope / Depedency**
RteSerializedSignal	1..*	The system description defines which SystemSignal is used to transport the serialized data and which kind of serializer has to be used. This configuration container defines which API (BswModuleEntry) provided by a Basic Software Module is called for the serialization/deserialization. The (de-)Serialization Function has to implement the serialization method given in the system description.

RteSerializedSignal

SWS Item	[ECUC_Rte_09109]
Container Name	RteSerializedSignal
Description	The system description defines which SystemSignal is used to transport the serialized data and which kind of serializer has to be used. This configuration container defines which API (BswModuleEntry) provided by a Basic Software Module is called for the serialization/deserialization. The (de-)Serialization Function has to implement the serialization method given in the system description.
Configuration Parameters	

Name	RteSerializerBswModuleEntryRef [ECUC_Rte_09111]		
Description	Reference to the BswModuleEntry used for serialization.		
Multiplicity	1		
Type	Foreign reference to BSW-MODULE-ENTRY		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteSystemSignalRef [ECUC_Rte_09110]		
Description	Reference to a SystemSignal in the system description that is used to transport the serialized data.		
Multiplicity	1		
Type	Foreign reference to SYSTEM-SIGNAL		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

No Included Containers

7.11 Configuration of the BSW Scheduler

The configuration of the BSW Scheduler part of the RTE is shown in the overview in figure 7.19.

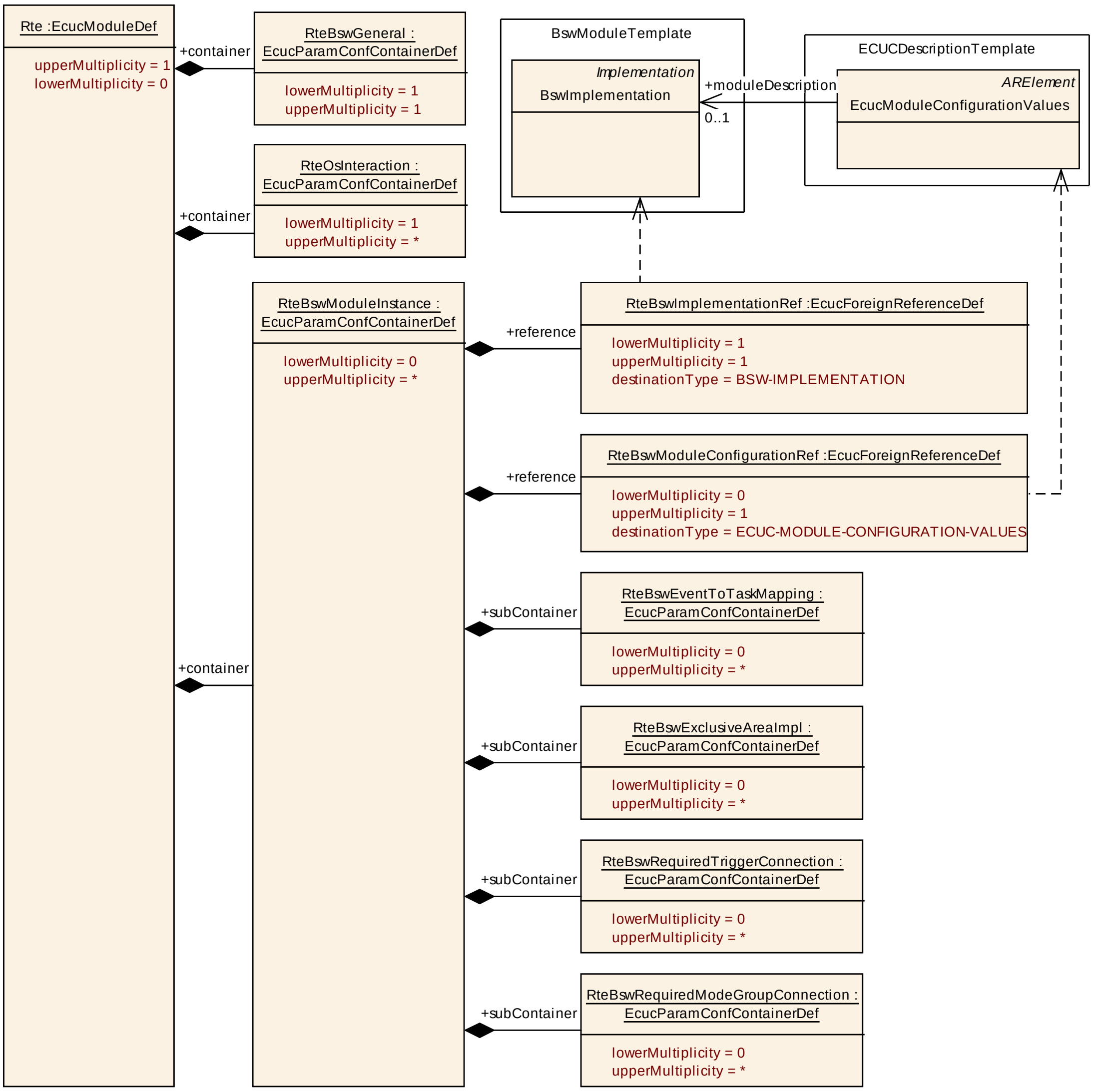


Figure 7.19: Configuration of BSW Scheduler overview

7.11.1 BSW Scheduler General configuration

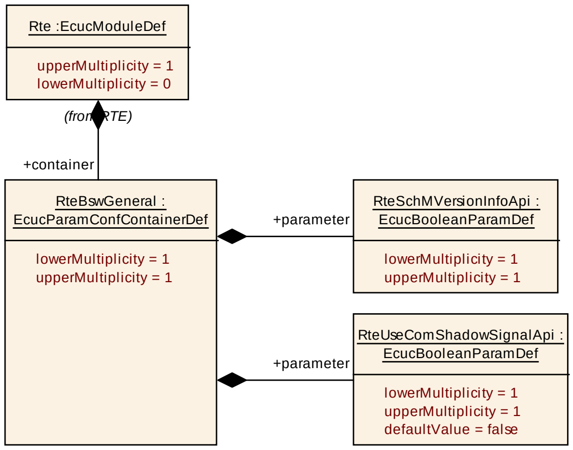


Figure 7.20: General configuration of BSW Scheduler

RteBswGeneral

SWS Item	[ECUC_Rte_09061]
Container Name	RteBswGeneral
Description	General configuration parameters of the Bsw Scheduler section.
Configuration Parameters	

Name	RteSchMVersionInfoApi [ECUC_Rte_09062]		
Description	Enables the generation of the SchM_GetVersionInfo() API.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value			
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteUseComShadowSignalApi [ECUC_Rte_09107]
Description	This parameter defines whether the ComShadowSignalAPIs ((Com_UpdateShadowSignal, Com_InvalidateShadowSignal, Com_ReceiveShadowSignal) are used or not. If this parameter is set to true the ShadowSignal APIs and Signal APIs (Com_SendSignal, Com_InvalidateSignal, Com_ReceiveSignal) are used. If this parameter is set to false only the Signal APIs (Com_SendSignal, Com_InvalidateSignal, Com_ReceiveSignal) are used.
Multiplicity	1
Type	EcucBooleanParamDef
Default Value	false

Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

No Included Containers

7.11.2 BSW Module Instance configuration

RteBswModuleInstance

SWS Item	[ECUC_Rte_09002]
Container Name	RteBswModuleInstance
Description	Represents one instance of a Bsw-Module configured on one ECU.
Configuration Parameters	

Name	RteBswImplementationRef [ECUC_Rte_09066]		
Description	Reference to the BswImplementation for which the Rte /SchM is configured.		
Multiplicity	1		
Type	Foreign reference to BSW-IMPLEMENTATION		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteBswModuleConfigurationRef [ECUC_Rte_09001]		
Description	Reference to the ECU Configuration Values provided for this BswImplementation.		
Multiplicity	0..1		
Type	Foreign reference to ECUC-MODULE-CONFIGURATION-VALUES		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Included Containers		
Container Name	**Multiplicity**	**Scope / Depedency**

RteBswEventToTask Mapping	0..*	Maps a BswModuleEntity onto an OsTask based on the activating BswEvent. A BswModuleEntity can be activated by more than one BswEvent and thus be mapped to more than one OsTask. In the case of a BswSchedulableEntity executed via a direct function call this RteBswEventToTaskMapping is still specified but no RteBswMappedToTaskRef element is included. The RteBswPositionInTask parameter is necessary to provide an ordering of events invoked by the same RTE API.
RteBswExclusiveArea Impl	0..*	Represents one ExclusiveArea of one BswImplementation. Used to specify the implementation means of this ExclusiveArea.
RteBswExternalTrigger Config	0..*	Defines the configuration of Inter Basic Software Module Entity Triggering
RteBswInternalTrigger Config	0..*	Defines the configuration of internal Basic Software Module Entity Triggering
RteBswRequiredClient ServerConnection	0..*	Defines the connection between one requiredClientServerEntry and one providedClientServerEntry of a BswModuleDescription. This container shall be provided on the client side of the connection.
RteBswRequiredMode GroupConnection	0..*	Defines the connection between one requiredModeGroup of this BSW Module instance and one providedModeGroup instance.
RteBswRequiredSender ReceiverConnection	0..*	Defines the connection between one requiredData and one providedData of a BswModuleDescription. This container shall be provided on the receiver side of the connection.
RteBswRequiredTrigger Connection	0..*	Defines the connection between one requiredTrigger of this BSW Module instance and one releasedTrigger instance.

7.11.2.1 BSW ExclusiveArea configuration

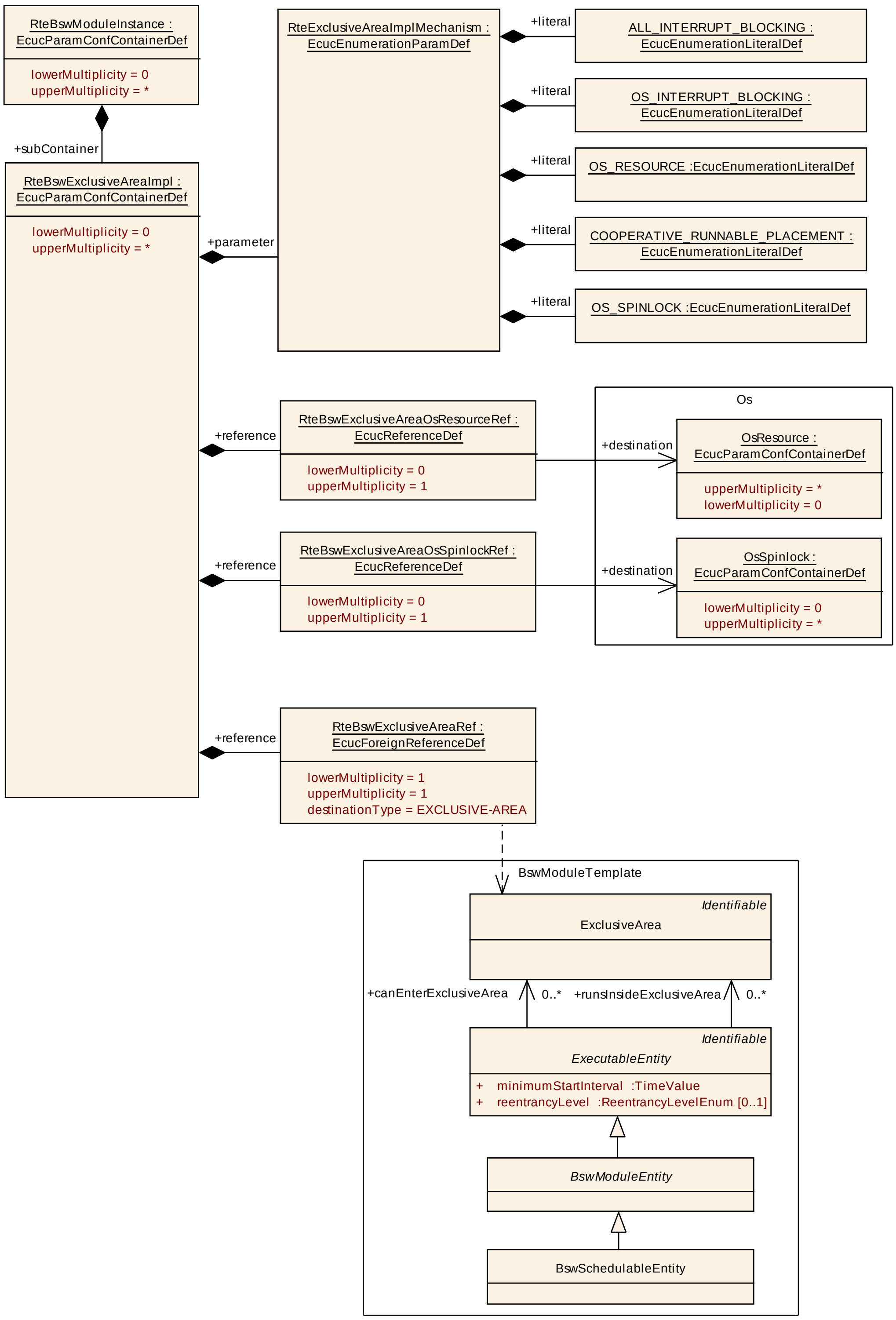


Figure 7.21: Configuration of BSW ExclusiveArea

RteBswExclusiveAreaImpl

SWS Item	[ECUC_Rte_09072]
Container Name	RteBswExclusiveAreaImpl
Description	Represents one ExclusiveArea of one BswImplementation. Used to specify the implementation means of this ExclusiveArea.
Configuration Parameters	

Name	RteBswExclusiveAreaOsResourceRef [ECUC_Rte_09073]		
Description	Optional reference to an OsResource in case RteExclusiveAreaImplMechanism is configured to OS_RESOURCE for this ExclusiveArea.		
Multiplicity	0..1		
Type	Reference to OsResource		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteBswExclusiveAreaOsSpinlockRef [ECUC_Rte_09112]		
Description	Optional reference to an OsSpinlock in case RteExclusiveAreaImplMechanism is configured to OS_SPINLOCK for this ExclusiveArea.		
Multiplicity	0..1		
Type	Reference to OsSpinlock		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteBswExclusiveAreaRef [ECUC_Rte_09074]		
Description	Reference to the ExclusiveArea for which the implementation mechanism shall be specified.		
Multiplicity	1		
Type	Foreign reference to EXCLUSIVE-AREA		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteExclusiveAreaImplMechanism [ECUC_Rte_09029]	
Description	To be used implementation mechanism for the specified ExclusiveArea.	
Multiplicity	1	
Type	EcucEnumerationParamDef	
Range	ALL_INTERRUPT_BLOCKING	
	COOPERATIVE_RUNNABLE_PLACEMENT	

	OS_INTERRUPT_BLOCKING		
	OS_RESOURCE		
	OS_SPINLOCK		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

No Included Containers

7.11.2.2 BswEvent to task mapping

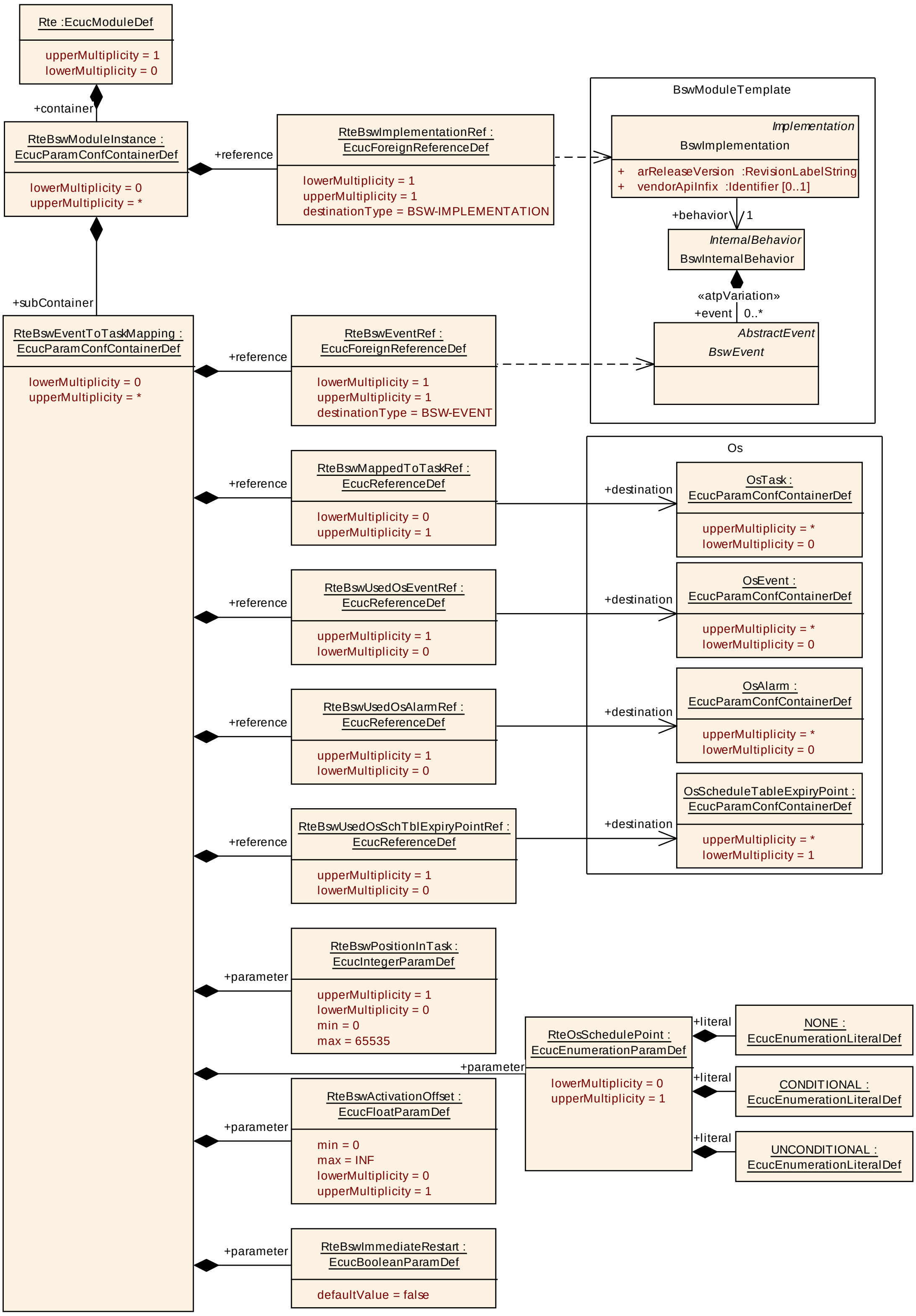


Figure 7.22: Configuration of BSW Event to Task Mapping

RteBswEventToTaskMapping

SWS Item	[ECUC_Rte_09065]		
Container Name	RteBswEventToTaskMapping		
Description	Maps a BswModuleEntity onto an OsTask based on the activating BswEvent. A BswModuleEntity can be activated by more than one BswEvent and thus be mapped to more than one OsTask. In the case of a BswSchedulableEntity executed via a direct function call this RteBswEventToTaskMapping is still specified but no RteBswMappedToTaskRef element is included. The RteBswPositionInTask parameter is necessary to provide an ordering of events invoked by the same RTE API.		
Configuration Parameters			

Name	RteBswActivationOffset [ECUC_Rte_09063]		
Description	Activation offset in seconds.		
Multiplicity	0..1		
Type	EcucFloatParamDef		
Range	0 .. INF		
Default Value			
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteBswEventRef [ECUC_Rte_09064]		
Description	Reference to the BswEvent.		
Multiplicity	1		
Type	Foreign reference to BSW-EVENT		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteBswImmediateRestart [ECUC_Rte_09093]		
Description	When RteBswImmediateRestart is set to true the BswSchedulableEntitiy shall be immediately re-started after termination if it was activated by this BswEvent while it was already started. This parameter shall not be set to true when the mapped BswEvent refers to a BswSchedulableEntitiy which minimumStartInterval attribute is > 0.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value	false		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteBswMappedToTaskRef [ECUC_Rte_09067]		
Description	Reference to the OsTask the BswSchedulableEntity activated by the RteBswEventRef is mapped to. If no reference to the OsTask is specified the BswSchedulableEntity activated by this BswEvent is executed in the context of the caller.		
Multiplicity	0..1		
Type	Reference to OsTask		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteBswPositionInTask [ECUC_Rte_09068]		
Description	Each BswSchedulableEntity activation mapped to an OsTask has a specific position within the task execution. For periodic activation this is the order of execution. For event driver activation this is the order of evaluation which actual BswSchedulableEntity has to be executed. In case of direct function calls this parameter is necessary to provide an ordering of events when several ExecutableEntities are invoked by the same RTE API.		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default Value			
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteBswUsedOsAlarmRef [ECUC_Rte_09069]		
Description	If an OsAlarm is used to activate the OsTask this BswEvent is mapped to it shall be referenced here.		
Multiplicity	0..1		
Type	Reference to OsAlarm		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteBswUsedOsEventRef [ECUC_Rte_09070]		
Description	If an OsEvent is used to activate the OsTask this BswEvent is mapped to it shall be referenced here.		
Multiplicity	0..1		
Type	Reference to OsEvent		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteBswUsedOsSchTblExpiryPointRef [ECUC_Rte_09071]		
Description	If an OsScheduleTableExpiryPoint is used to activate the OsTask this BswEvent is mapped to it shall be referenced here.		
Multiplicity	0..1		
Type	Reference to OsScheduleTableExpiryPoint		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteOsSchedulePoint [ECUC_Rte_09022]		
Description	Introduce a schedule point by explicitly calling Os Schedule service after the execution of the ExecutableEntity. The Rte generator is allowed to optimize several consecutive calls to Os schedule into one single call if the ExecutableEntity executions in between have been skipped. The absence of this parameter is interpreted as "NONE". It shall be considered an invalid configuration if the task is preemptable and the value of this parameter is not set to "NONE" or the parameter is absent.		
Multiplicity	0..1		
Type	EcucEnumerationParamDef		
Range	CONDITIONAL	A Schedule Point shall be introduced at the end of the execution of this ExecutableEntity. The Schedule Point can be skipped if several Schedule Points would be called without any ExecutableEntity execution in between.	
	NONE	No Schedule Point shall be introduced at the end of the execution of this ExecutableEntity.	
	UNCONDITIONAL	A Schedule Point shall always be introduced at the end of the execution of this ExecutableEntity.	
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

No Included Containers

7.11.2.3 BSW Trigger configuration

7.11.2.3.1 BSW Trigger connection

The RteBswRequiredTriggerConnection container is defined in the context of the RteBswModuleInstance which is the required trigger context. So the reference to the RteBswRequiredTriggerRef is sufficient to define the required trigger. For the released trigger the tuple of RteBswReleasedTriggerModInstRef and RteBswReleasedTriggerRef is specified.

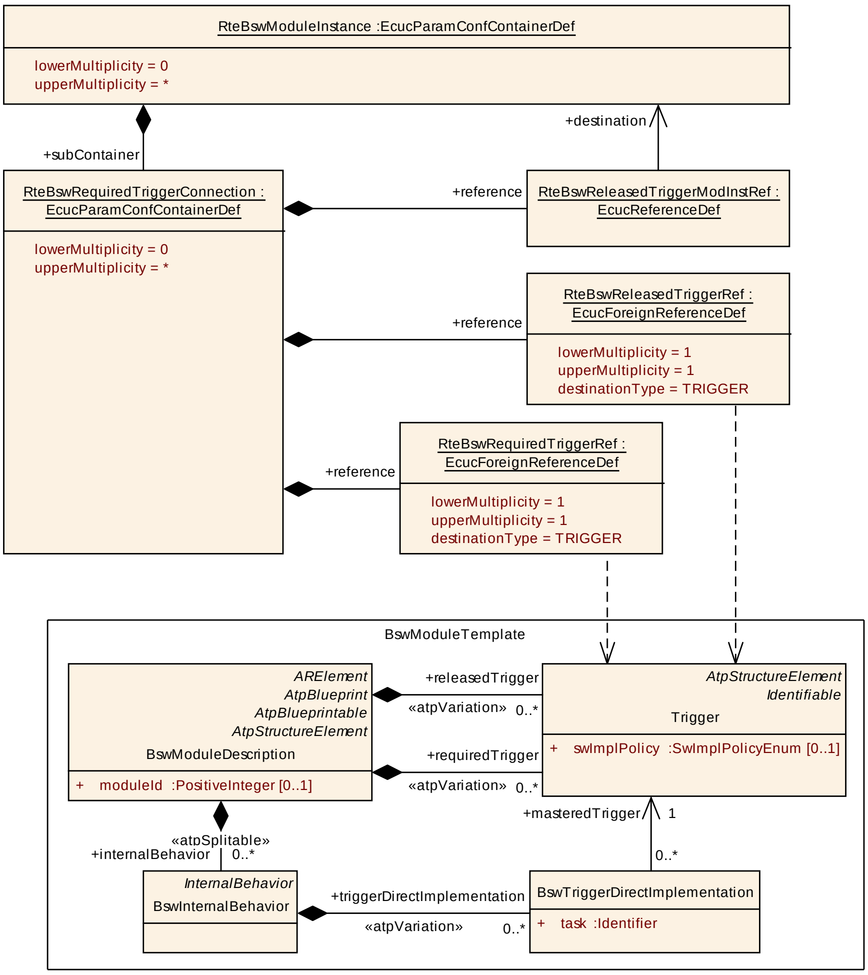


Figure 7.23: Configuration of BSW Trigger connection

RteBswRequiredTriggerConnection

SWS Item	[ECUC_Rte_09077]
Container Name	RteBswRequiredTriggerConnection
Description	Defines the connection between one requiredTrigger of this BSW Module instance and one releasedTrigger instance.
Configuration Parameters	

Name	RteBswReleasedTriggerModInstRef [ECUC_Rte_09075]		
Description	Reference to the RteBswModuleInstance configuration container which identifies the instance of the BSW Module. Used with the RteBswReleasedTriggerRef to unambiguously identify the Trigger instance.		
Multiplicity	1		
Type	Reference to RteBswModuleInstance		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteBswReleasedTriggerRef [ECUC_Rte_09076]		
Description	References the releasedTrigger to which this requiredTrigger shall be connected.		
Multiplicity	1		
Type	Foreign reference to TRIGGER		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteBswRequiredTriggerRef [ECUC_Rte_09078]		
Description	References one requiredTrigger which shall be connected to the releasedTrigger.		
Multiplicity	1		
Type	Foreign reference to TRIGGER		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

No Included Containers

7.11.2.3.2 BSW Trigger queuing

This configuration determine the size of the queue queuing the issued triggers.

The `RteBswExternalTriggerConfig` container and `RteBswInternalTriggerConfig` container is defined in the context of the `RteBswModuleInstance` which already predefines the context of the provided `Trigger` / `BswInternalTriggeringPoint`.

[constr_9006] The references `RteBswTriggerSourceRef` has to be consistent with the `RteBswImplementationRef` ⌈ The references `RteBswTriggerSourceRef` has to be consistent with the `RteBswImplementationRef`. This means

the referenced Trigger / BswInternalTriggeringPoint has to belong to the BswModuleDescription which is referenced by the related BswImplementation.
⌋

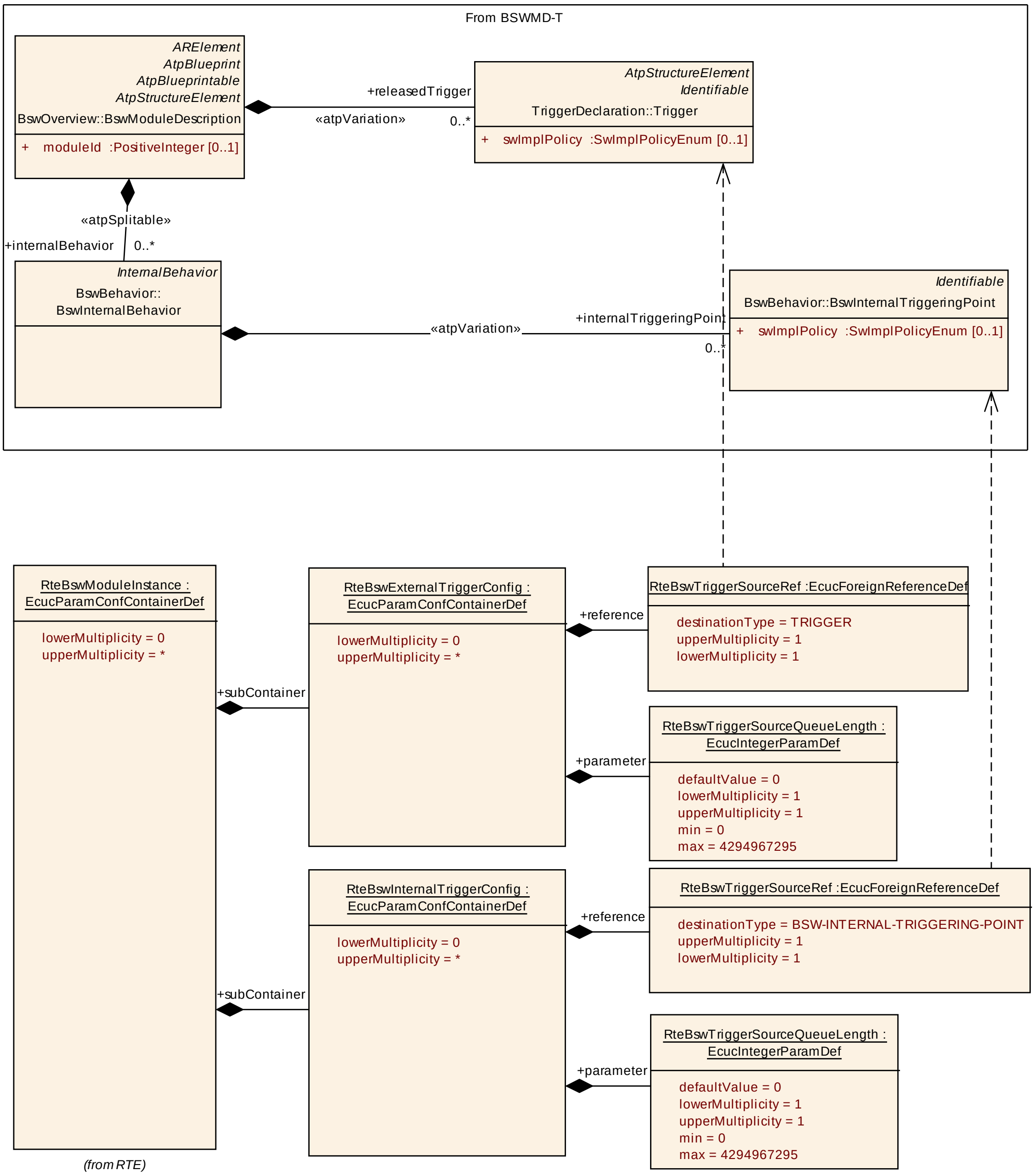


Figure 7.24: Configuration of BSW Trigger queuing

RteBswExternalTriggerConfig

SWS Item	[ECUC_Rte_09099]
Container Name	RteBswExternalTriggerConfig
Description	Defines the configuration of Inter Basic Software Module Entity Triggering
Configuration Parameters	

Name	RteBswTriggerSourceQueueLength [ECUC_Rte_09101]		
Description	Length of trigger queue on the trigger source side. The queue is implemented by the RTE. A value greater or equal to 1 requests an queued behavior. Setting the value of RteTriggerSourceQueueLength to 0 requests an none queued implementation of the trigger communication. If there is no RteBswTriggerSourceQueueLength configured for a Trigger Emitter the default value of 0 applies as well.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default Value	0		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteBswTriggerSourceRef [ECUC_Rte_09100]		
Description	Reference to a Trigger instance in the role releasedTrigger of the related BSW Module instance. The referenced Trigger has to belong to the same BSW Module instance as the RteBswModuleInstance owning this parameter configures.		
Multiplicity	1		
Type	Foreign reference to TRIGGER		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

No Included Containers

RteBswInternalTriggerConfig

SWS Item	[ECUC_Rte_09102]
Container Name	RteBswInternalTriggerConfig
Description	Defines the configuration of internal Basic Software Module Entity Triggering
Configuration Parameters	

Name	RteBswTriggerSourceQueueLength [ECUC_Rte_09104]		
Description	Length of trigger queue on the trigger source side. The queue is implemented by the RTE. A value greater or equal to 1 requests an queued behavior. Setting the value of RteTriggerSourceQueueLength to 0 requests an none queued implementation of the trigger communication. If there is no RteBswTriggerSourceQueueLength configured for a Trigger Emitter the default value of 0 applies as well.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default Value	0		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteBswTriggerSourceRef [ECUC_Rte_09103]		
Description	Reference to a BswInternalTriggeringPoint of the related BSW Module instance. The referenced BswInternalTriggeringPoint has to belong to the same BSW Module instance as the RteBswModuleInstance owning this parameter configures.		
Multiplicity	1		
Type	Foreign reference to BSW-INTERNAL-TRIGGERING-POINT		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

No Included Containers

7.11.2.4 BSW ModeDeclarationGroup configuration

The `RteBswRequiredModeGroupConnection` container is defined in the context of the `RteBswModuleInstance` which is the required mode group context. So the reference to the `RteBswRequiredModeGroupRef` is sufficient to define the required mode group. For the provided mode group the tuple of RteBswProvidedModeGrpModInstRef and RteBswProvidedModeGroupRef is specified.

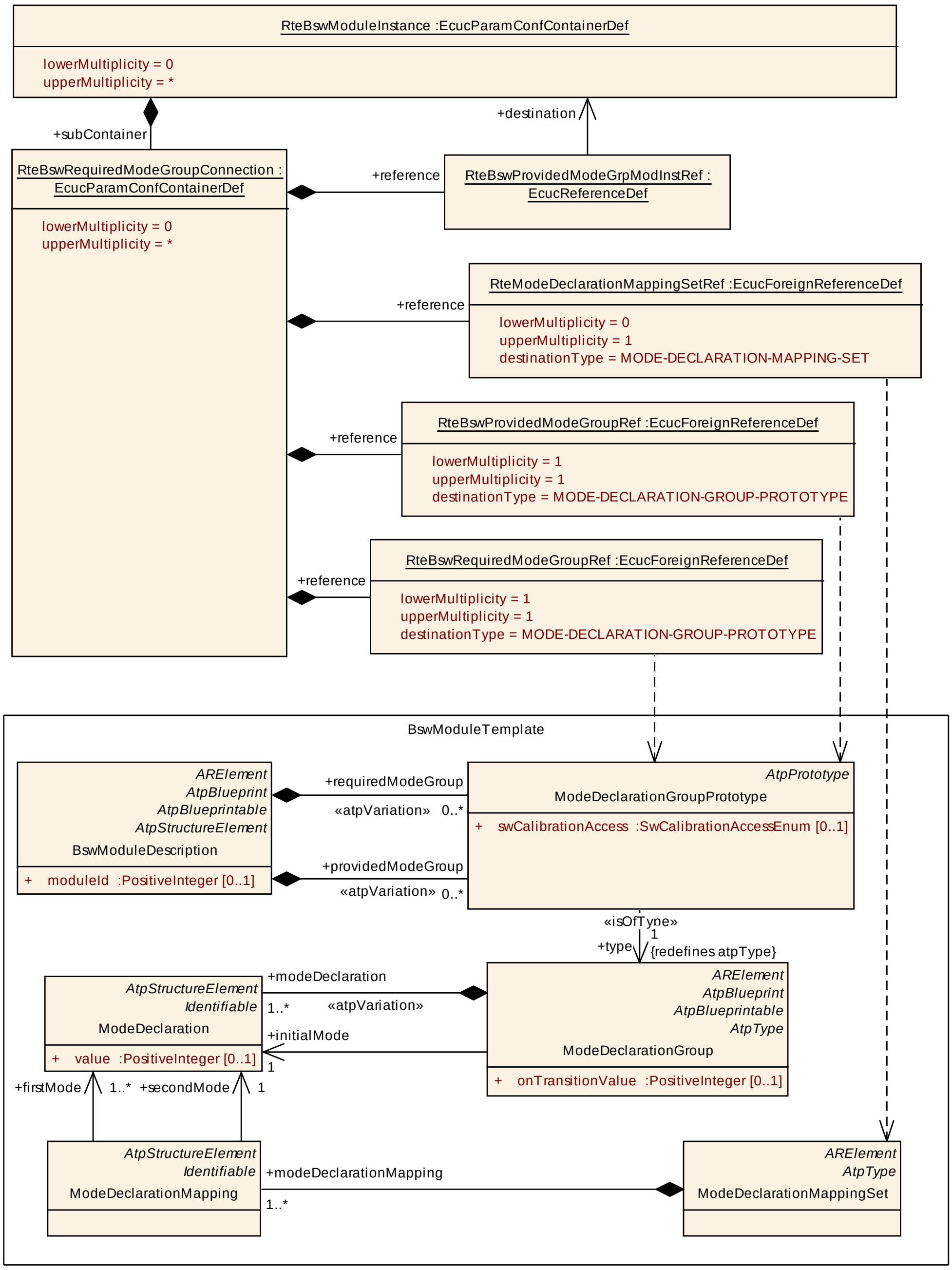


Figure 7.25: Configuration of BSW Scheduler overview

RteBswRequiredModeGroupConnection

SWS Item	[ECUC_Rte_09081]		
Container Name	RteBswRequiredModeGroupConnection		
Description	Defines the connection between one requiredModeGroup of this BSW Module instance and one providedModeGroup instance.		
Configuration Parameters			

Name	RteBswProvidedModeGroupRef [ECUC_Rte_09079]		
Description	References the providedModeGroupPrototype to which this requiredModeGroup shall be connected.		
Multiplicity	1		
Type	Foreign reference to MODE-DECLARATION-GROUP-PROTOTYPE		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteBswProvidedModeGrpModInstRef [ECUC_Rte_09080]		
Description	Reference to the RteBswModuleInstance configuration container which identifies the instance of the BSW Module. Used with the RteBswProvidedModeGroupRef to unambiguously identify the ModeDeclarationGroupPrototype instance.		
Multiplicity	1		
Type	Reference to RteBswModuleInstance		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteBswRequiredModeGroupRef [ECUC_Rte_09082]		
Description	References requiredModeGroupPrototype which shall be connected to the providedModeGroupPrototype.		
Multiplicity	1		
Type	Foreign reference to MODE-DECLARATION-GROUP-PROTOTYPE		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteModeDeclarationMappingSetRef [ECUC_Rte_09125]		
Description	This defines the effective ModeDeclarationMappingSet in the case that the provided ModeDeclarationGroupPrototype and the required ModeDeclarationGroupPrototype are not compatible.		
Multiplicity	0..1		
Type	Foreign reference to MODE-DECLARATION-MAPPING-SET		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

No Included Containers

7.11.2.5 BSW Client Server configuration

The `RteBswRequiredClientServerConnection` container is defined in the context of the `RteBswModuleInstance`. So the reference to the `RteBswRequiredClientServerEntryRef` is sufficient to define the required `BswModuleClientServerEntry`. For the provided `BswModuleClientServerEntry` the `RteBswProvidedClientServerEntryRef` is specified.

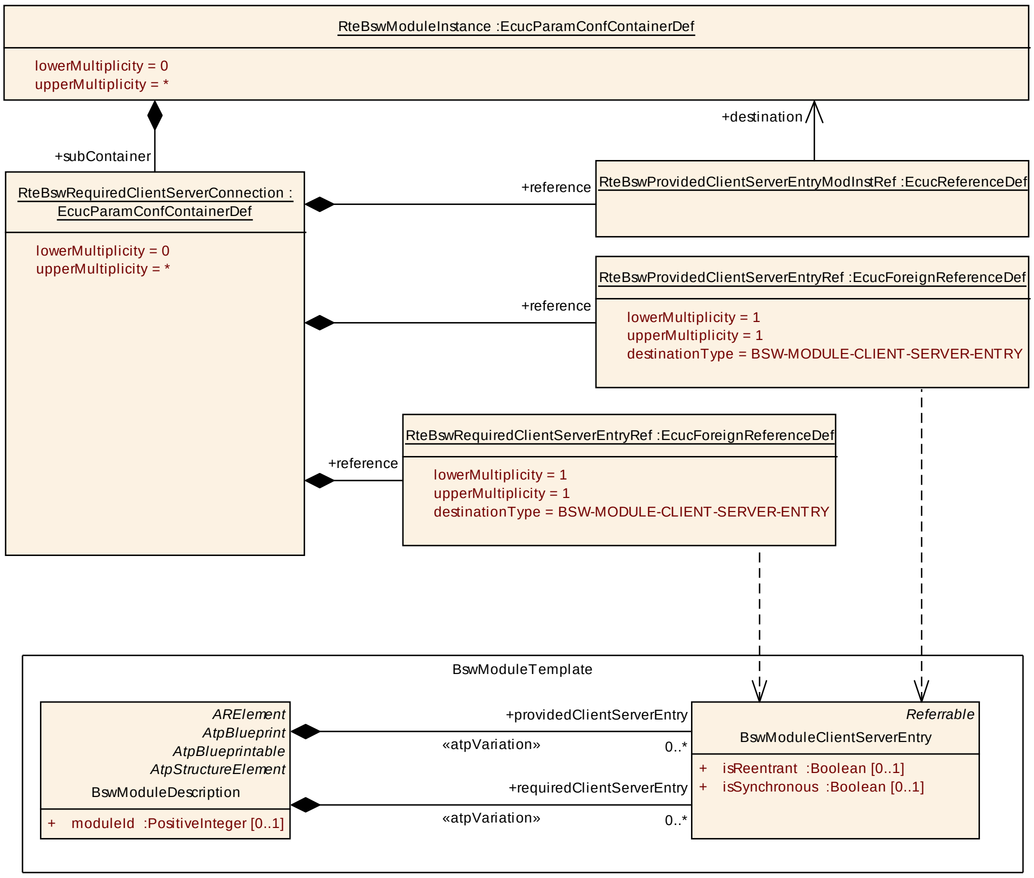


Figure 7.26: Configuration of BSW Client Server Communication

RteBswRequiredClientServerConnection

SWS Item	[ECUC_Rte_09117]		
Container Name	RteBswRequiredClientServerConnection		
Description	Defines the connection between one requiredClientServerEntry and one providedClientServerEntry of a BswModuleDescription. This container shall be provided on the client side of the connection.		
Configuration Parameters			

Name	RteBswProvidedClientServerEntryModInstRef [ECUC_Rte_09124]		
Description	Reference to the RteBswModuleInstance configuration container which identifies the instance of the BSW Module. Used with the RteBswProvidedClientServerEntryRef to unambiguously identify the BswModuleClientServerEntry instance.		
Multiplicity	1		
Type	Reference to RteBswModuleInstance		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteBswProvidedClientServerEntryRef [ECUC_Rte_09119]		
Description	Reference the providedClientServerEntry for this connection.		
Multiplicity	1		
Type	Foreign reference to BSW-MODULE-CLIENT-SERVER-ENTRY		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteBswRequiredClientServerEntryRef [ECUC_Rte_09118]		
Description	Reference the requiredClientServerEntry for this connection.		
Multiplicity	1		
Type	Foreign reference to BSW-MODULE-CLIENT-SERVER-ENTRY		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

No Included Containers

7.11.2.6 BSW Sender Receiver configuration

The `RteBswRequiredSenderReceiverConnection` container is defined in the context of the `RteBswModuleInstance`. So the reference to the `RteBswRequiredVariableDataPrototypeRef` is sufficient to define the required `VariableDat-`

aPrototype. For the provided VariableDataPrototype the RteBswProvided-VariableDataPrototypeRef is specified.

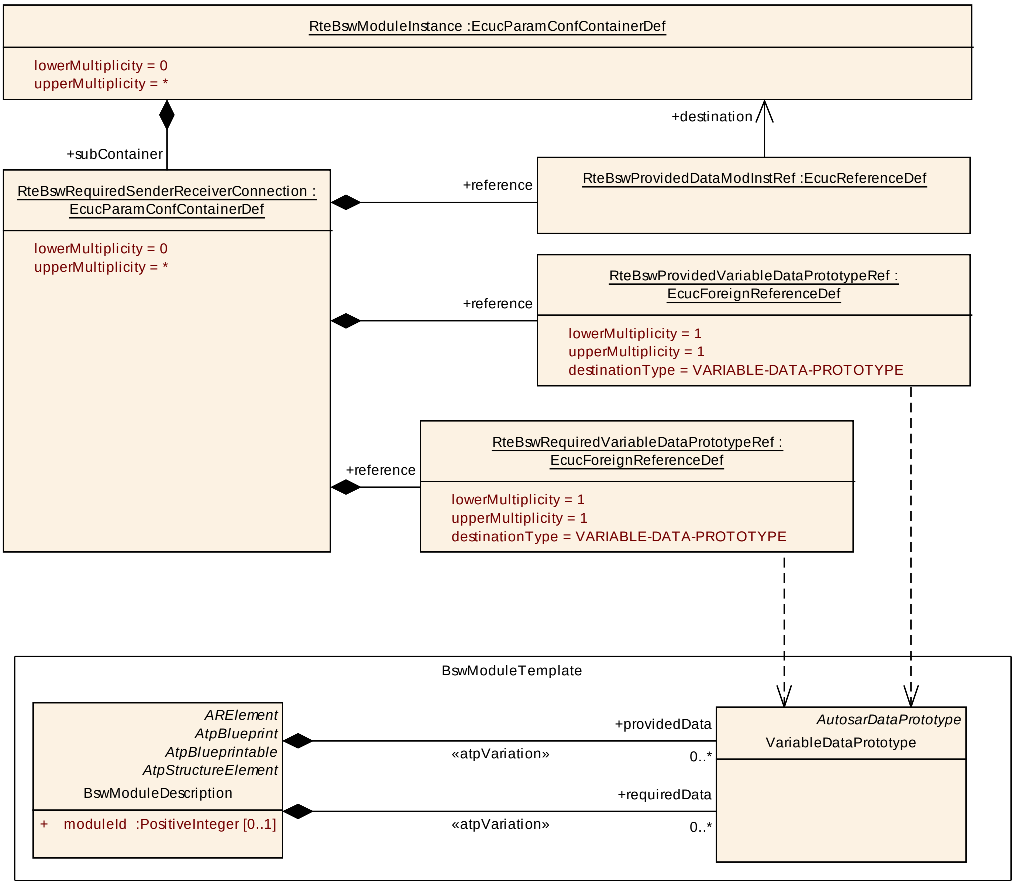


Figure 7.27: Configuration of BSW Sender Receiver Communication

RteBswRequiredSenderReceiverConnection

SWS Item	[ECUC_Rte_09120]
Container Name	RteBswRequiredSenderReceiverConnection
Description	Defines the connection between one requiredData and one providedData of a BswModuleDescription. This container shall be provided on the receiver side of the connection.
Configuration Parameters	

Name	RteBswProvidedDataModInstRef [ECUC_Rte_09123]		
Description	Reference to the RteBswModuleInstance configuration container which identifies the instance of the BSW Module. Used with the RteBswProvidedVariableDataPrototypeRef to unambiguously identify the VariableDataPrototype instance.		
Multiplicity	1		
Type	Reference to RteBswModuleInstance		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteBswProvidedVariableDataPrototypeRef [ECUC_Rte_09122]		
Description	Reference the providedData for this connection.		
Multiplicity	1		
Type	Foreign reference to VARIABLE-DATA-PROTOTYPE		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteBswRequiredVariableDataPrototypeRef [ECUC_Rte_09121]		
Description	Reference the requiredData for this connection.		
Multiplicity	1		
Type	Foreign reference to VARIABLE-DATA-PROTOTYPE		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

No Included Containers

7.12 Configuration of Initialization

In order to support different interactions with the start up code of the ECU the RTE supports different initialization strategies for variables implementing VariableDataPrototypes. Basically the initialization can be done either by start-up code or by the Rte_Start function. Further on it is possible to avoid any initialization for data which has to be reset safe or is explicitly initialized by other SW, e.g. the NVRAM Blocks might be initialized by NVRAM Manager.

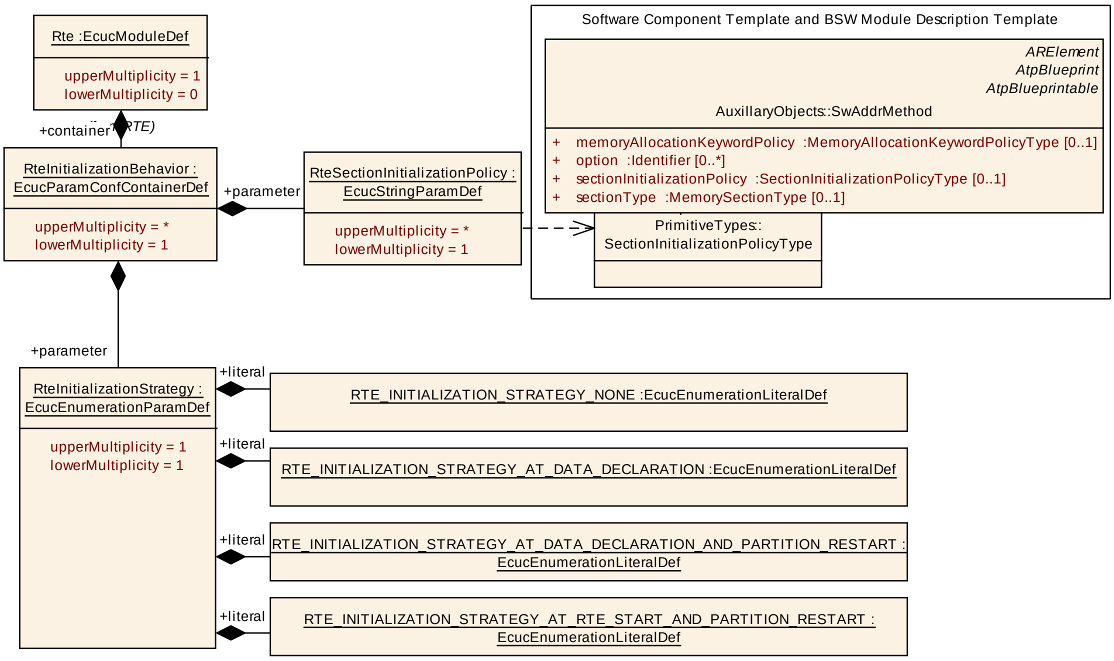


Figure 7.28: Configuration of initialization strategy

RteInitializationBehavior

SWS Item	[ECUC_Rte_09087]
Container Name	RteInitializationBehavior
Description	Specifies the initialization strategy for variables allocated by RTE with the purpose to implement VariableDataPrototypes. The container defines a set of RteSectionInitializationPolicys and one RteInitializationStrategy which is applicable for this set.
Configuration Parameters	

Name	RteInitializationStrategy [ECUC_Rte_09089]	
Description	Definition of the initialization strategy applicable for the SectionInitializationPolicys selected by RteSectionInitializationPolicy.	
Multiplicity	1	
Type	EcucEnumerationParamDef	
Range	RTE_INITIALIZATION_STRATEGY_AT_DATA_DECLARATION	Variables shall be initialized at its declaration to the value defined by the related initValue attribute.
	RTE_INITIALIZATION_STRATEGY_AT_DATA_DECLARATION_AND_PARTITION_RESTART	Variables shall be initialized at its declaration to the value defined by the related initValue attribute and during execution of Rte_RestartPartition to the value defined by the related initValue attribute.

	RTE_INITIALIZATION_STRATEGY_AT_RTE_START_AND_PARTITION_RESTART	Variables shall be initialized during execution of Rte_Start and Rte_RestartPartition to the value defined by the related initValue attribute.	
	RTE_INITIALIZATION_STRATEGY_NONE	Variables shall not be initialized at all.	
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	RteSectionInitializationPolicy [ECUC_Rte_09088]		
Description	This parameter describes the SectionInitializationPolicys for which a particular RTE initialization strategy applies. The SectionInitializationPolicy describes the intended initialization of MemorySections. The following values are standardized in AUTOSAR Methodology: • **NO-INIT**: No initialization and no clearing is performed. Such data elements must not be read before one has written a value into it. • **INIT**: To be used for data that are initialized by every reset to the specified value (initValue). • **POWER-ON-INIT**: To be used for data that are initialized by "Power On" to the specified value (initValue). Note: there might be several resets between power on resets. • **CLEARED**: To be used for data that are initialized by every reset to zero. • **POWER-ON-CLEARED**: To be used for data that are initialized by "Power On" to zero. Note: there might be several resets between power on resets.		
Multiplicity	1..*		
Type	EcucStringParamDef		
Default Value			
Regular Expression			
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

No Included Containers

[SWS_Rte_07075] ⌈ The RTE generator shall reject configurations where not all occurring `sectionInitializationPolicy` attribute values are configured to an `RteInitializationStrategy`. ⌋*(SRS_Rte_00018)*

The call of Rte_Start may trigger RunnableEntitys for initialization purpose. Those RunnableEntitys are either triggered by SwcModeSwitchEvents or InitEvents. To support the scheduling of such RunnableEntitys in the start up code of the ECU (e.g. by BswM or EcuM) its possible to map such RTEEvents to RteInitializationRunnableBatch containers which results in the existence of Rte_Init APIs.

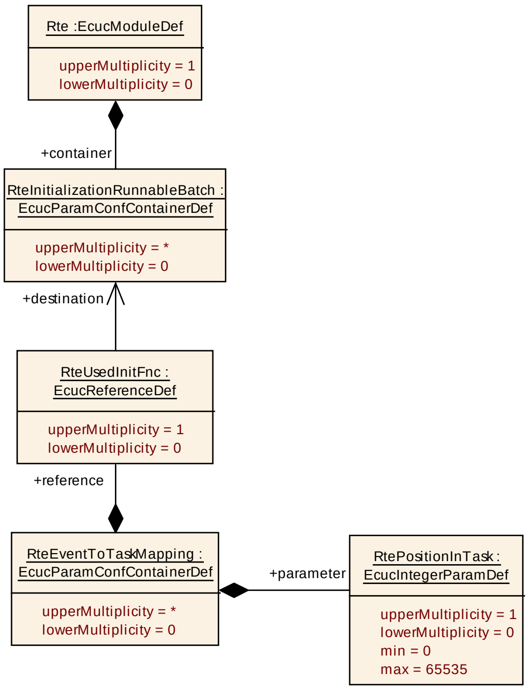


Figure 7.29: Configuration of Rte_Init functions

RteInitializationRunnableBatch

SWS Item	[ECUC_Rte_09115]
Container Name	RteInitializationRunnableBatch
Description	This container corresponds to an Rte_Init_<shortName of this container> function invoking the mapped RunnableEntities.
Configuration Parameters	
No Included Containers	

Rte_Init API may only schedule RunnableEntitys for initialization purpose ore which are OnEntry Runnable Entities.

[constr_9063] Restricted kinds of RTEEvents which may mapped to RteInitializationRunnableBatch containers ⌈ Only SwcModeSwitchEvents with activation = onEntry and referring to the initialMode or InitEvents may be mapped to RteInitializationRunnableBatch containers with the means of a RteUsedInitFnc reference. ⌋

[SWS_Rte_06769] ⌈ The RTE Generator shall reject configurations violating [constr_9063]. ⌋*(SRS_Rte_00143, SRS_Rte_00240, SRS_Rte_00018)*

[constr_9064] A single `RteInitializationRunnableBatch` container may not handle `RTEEvent`s of different partitions ⌈ All `RTEEvent`s mapped to a `RteInitializationRunnableBatch` container may only trigger `RunnableEntity`s belonging to the same partition. ⌋

[SWS_Rte_06770] ⌈ The RTE Generator shall reject configurations violating [constr_9064]. ⌋*(SRS_Rte_00143, SRS_Rte_00240, SRS_Rte_00018)*

A Metamodel Restrictions

This chapter lists all the restrictions to the AUTOSAR meta-model this version of the AUTOSAR RTE specification document relies on. The RTE generator shall reject configuration where any of the specified restrictions are violated.

A.1 Restrictions concerning WaitPoint

1. **[SWS_Rte_01358]** ⌈ The RTE shall raise an error if [constr_1091] is violated, so if RunnableEntity has WaitPoint connected to any of the following RTE-Events:

    - OperationInvokedEvent
    - SwcModeSwitchEvent
    - TimingEvent
    - BackgroundEvent
    - DataReceiveErrorEvent
    - ExternalTriggerOccurredEvent
    - InternalTriggerOccurredEvent
    - DataWriteCompletedEvent

    These events can only start a runnable. ⌋*(SRS_Rte_00092, SRS_Rte_00018)*
    Note: The only events that can unblock a WaitPoint are those listed in [constr_1091].

    Rationale: For OperationInvokedEvents, SwcModeSwitchEvents, TimingEvents, BackgroundEvents DataReceiveErrorEvent, ExternalTriggerOccurredEvent, InternalTriggerOccurredEvent, and DataWriteCompletedEvent it suffices to allow the activation of a RunnableEntity.

2. **[SWS_Rte_07402]** ⌈ The RTE generator shall reject a model where two (or more) different RunnableEntitys in the same internal behavior each have a WaitPoint referencing the same DataReceivedEvent, and the runnables are mapped to different tasks. ⌋*(SRS_Rte_00092, SRS_Rte_00018)*

    Rationale: In the same software components, the two runnables will attempt to read from the same queue, and only the one that accesses the queue first will actually receive the data.

A.2 Restrictions concerning RTEEvent

1. **[SWS_Rte_03526]** ⌈ The RTE generator shall reject configurations in which a RunnableEntity is triggered by multiple OperationInvokedEvents but violating the constraint [constr_2000] *Compatibility of ClientServerOperations triggering the same RunnableEntity* as defined in document [2] ⌋*(SRS_Rte_00072, SRS_Rte_00018)*

   Rationale: The signature of the RunnableEntity is dependent on its connected RTEEvent. Multiple OperationInvokedEvents are only supported if all referred ClientServerOperations would result in the same RunnableEntity prototype for the server runnable (see 5.7.5.6).

2. **[SWS_Rte_03010]** ⌈ One runnable entity shall only be resumed by one single RTEEvent on its WaitPoint. The RTE doesn't support the WaitPoint of one runnable entity connected to several RTEEvents. ⌋*(SRS_Rte_00092, SRS_Rte_00018)*

   Rationale: The WaitPoint of the runnable entity is caused by calling of the RTE API. One runnable entity can only call one RTE API at a time, and so it can only wait for one RTEEvent.

3. **[SWS_Rte_07007]** ⌈ The RTE generator shall reject configurations where different execution instances of a runnable entity, which use implicit data access, are mapped to different Preemption Areas. ⌋*(SRS_Rte_00018, SRS_Rte_00128, SRS_Rte_00129, SRS_Rte_00133, SRS_Rte_00142)*

   Rationale: Buffers used for implicit communication shall be consistent during the whole task execution. If it is guaranteed that one task does not preempt the other, direct accesses to the same copy buffer from different tasks are possible.

4. **[SWS_Rte_07403]** ⌈ The RTE generator shall reject a model where in the same SwcInternalBehavior two (or more) different DataReceivedEvents, that reference the same VariableDataPrototype with event semantics, trigger different runnable entities mapped to different tasks. ⌋*(SRS_Rte_00072, SRS_Rte_00018)*

   Rationale: In the same software components, the two runnables will attempt to read from the same queue, and only the one that accesses the queue first will actually receive the data.

A.3 Restrictions concerning queued implementation policy

1. **[SWS_Rte_03018]** ⌈ RTE does not support receiving with WaitPoint for VariableDataPrototypes with their swImplPolicy attribute is not set to queued. ⌋*(SRS_Rte_00109, SRS_Rte_00092, SRS_Rte_00018)*

Rationale: unqueued implementation policy indicates that the receiver shall not wait for the VariableDataPrototype.

2. All the VariableAccesses in the dataSendPoint role referring to one VariableDataPrototype through one PPortPrototype are considered to have the same behavior by sending and acknowledgment reception. All DataSendCompletedEvents that reference VariableAccesses in the dataSendPoint role referring to the same VariableDataPrototype are considered equivalent.

   Rationale: The API Rte_Send/Rte_Write is dependent on the port name and the VariableDataPrototype name, not on the VariableAccesses. For each combination of one VariableDataPrototype and one port only one API will be generated and implemented for sending or acknowledgement reception.

A.4 Restrictions concerning ServerCallPoint

1. **[SWS_Rte_03014]** ⌈ All the ServerCallPoints referring to one ClientServerOperation through one RPortPrototype are considered to have the same behavior by calling service. The RTE generator shall reject configuration where this is violated. ⌋*(SRS_Rte_00051, SRS_Rte_00018)*

   Rationale: The API Rte_Call is dependent on the port name and the operation name, not on the ServerCallPoints. For each combination of one operation and one port only one API will be generated and implemented for calling a service. It is e.g. not possible to have different timeout values specified for different ServerCallPoints of the same ClientServerOperation. It is also not allowed to specify both, a synchronous and an asynchronous server call point for the same ClientServerOperation instance.

2. **[SWS_Rte_03605]** ⌈ If several require ports of a software component are categorized by the same client/server interface, all invocations of the same operation of this client/server interface have to be either synchronous, or all invocations of the same operation have to be asynchronous. This restriction applies under the following conditions:

   - the usage of the indirect API is specified for at least one of the respective port prototypes **and/or**
   - the software component supports multiple instantiation, **and** the RTE generation shall be performed in compatibility mode.

   ⌋*(SRS_Rte_00051, SRS_Rte_00018)*

   Rationale: The signature of Rte_Call and the existence of Rte_Result depend on the kind of invocation.

3. **[SWS_Rte_07170]** ⌈ The RTE generator shall reject the configuration where [constr_2006] is violated, so where an AsynchronousServerCallPoint shall

be referenced by exactly one AsynchronousServerCallResultPoint only. ⌋(*SRS_Rte_00051*, *SRS_Rte_00018*)

Rationale: The support of several AsynchronousServerCallResultPoints per AsynchronousServerCallPoint would potentially support multiple AsynchronousServerCallReturnsEvents as well as multiple WaitPoints for the same AsynchronousServerCallPoint.

A.5 Restriction concerning multiple instantiation of software components

1. **[SWS_Rte_07101]** ⌈ The RTE generator shall reject configurations where [constr_2024] is violated, so in which a PortAPIOption with enableTakeAddress = *TRUE* is defined by a software-component supporting multiple instantiation. ⌋(*SRS_Rte_00018*)

   Rationale: The main focus of the feature is support for configuration of AUTOSAR Services which are limited to single instances.

A.6 Restrictions concerning runnable entity

1. **[SWS_Rte_03527]** ⌈ The RTE does NOT support multiple Runnable Entities that share the same entry point. ⌋(*SRS_Rte_00072*, *SRS_Rte_00018*)

   Rationale: The name of the runnable entity entry point is formed by a combination of SWC symbol prefix and symbol attribute of RunnableEntity. This means that two runnables in different SWCs can have the same symbol attribute as long as different SWC prefixes are used.

2. **[SWS_Rte_02733]** ⌈ The RTE Generator shall reject a configuration where a runnable has the attribute canBeInvokedConcurrently set to true and the attribute minimumStartInterval set to greater zero. ⌋(*SRS_Rte_00018*)

   Rationale: If a runnable should run concurrently (i.e., have several ExecutableEntity execution-instances), this implies that the minimum interval between the start of the runnables is zero. The configuration to be rejected is inconsistent.

A.7 Restrictions concerning runnables with dependencies on modes

1. Operations may not be disabled by a ModeDisablingDependency.

[SWS_Rte_02706] ⌈ RTE shall reject configurations that contain OperationInvokedEvents with a ModeDisablingDependency. ⌋*(SRS_Rte_00143, SRS_Rte_00018)*

Rationale: It is a preferable implementation, if the server responds with an explicit application error, when the server operation is not supported in a mode. To implement the disabling of operations would require a high amount of book keeping even for internal client server communication to prevent that the unique request response mapping gets lost.

2. Only a category 1 runnable may be triggered by

   - a SwcModeSwitchEvent
   - an RTEEvent with a mode disabling dependency

   [SWS_Rte_02500] ⌈ The RTE generator shall reject configurations with category 2 runnables connected to SwcModeSwitchEvents and RTEEvents / BswEvents with ModeDisablingDependencys if the mode machine instance is synchronous. The rejection may be reduced to a warning when the RTE generator is explicitly set to a non strict mode. ⌋*(SRS_Rte_00143, SRS_Rte_00213, SRS_Rte_00018)*

   Rationale: The above runnables are executed or terminated on the transitions between different modes. To execute the mode switch withing finite time, also these runnables have to be executed within finite execution time.

3. All OnEntry ExecutableEntitys, OnTransition ExecutableEntitys, and OnExit ExecutableEntitys of the same mode machine instance should be mapped to the same task in case of synchronous mode switching procedure.

   [SWS_Rte_02662] ⌈ The RTE generator shall reject configurations with OnEntry, OnTransition, or OnExit ExecutableEntity's of the same mode machine instance that are mapped to different tasks in case of synchronous mode switching procedure. ⌋*(SRS_Rte_00143, SRS_Rte_00213, SRS_Rte_00018)*

   In case of asynchronous mode switching procedure, a mapping of all affected runnables to no task is also possible.

   Rationale: This restriction simplifies the implementation of the semantics of a synchronous mode switch.

4. To guarantee that all mode disabling dependent ExecutableEntitys of a mode machine instance have terminated before the start of the OnExit ExecutableEntitys of the transition, the mode disabling dependent ExecutableEntitys should run with higher or equal priority.

   [SWS_Rte_02663] ⌈ The RTE generator shall reject configurations with mode disabling dependent ExecutableEntitys that are mapped to a task with lower priority than the task that contains the OnEntry ExecutableEntitys and OnExit ExecutableEntitys of that mode machine instance supporting a synchronous mode switching procedure. ⌋(*SRS_Rte_00143*, *SRS_Rte_00213*, *SRS_Rte_00018*)

5. **[SWS_Rte_02664]** ⌈ The RTE generator shall reject configurations of a task with

   - `OnExit ExecutableEntitys` mapped after `OnEntry ExecutableEntitys` or
   - `OnTransition ExecutableEntitys` mapped after `OnEntry ExecutableEntitys` or
   - `OnExit ExecutableEntitys` mapped after `OnTransition ExecutableEntitys`

   of the same `mode machine instance` supporting a synchronous mode switching procedure. ⌋(*SRS_Rte_00143*, *SRS_Rte_00213*, *SRS_Rte_00018*)

   Rationale: This restriction simplifies the implementation of the semantics of a synchronous mode switch.

6. **[SWS_Rte_07157]** ⌈ The RTE generator shall reject configurations with

   - `OnExit ExecutableEntitys` mapped after `OnEntry ExecutableEntitys` or
   - `OnTransition ExecutableEntitys` mapped after `OnEntry ExecutableEntitys` or
   - `OnExit ExecutableEntitys` mapped after `OnTransition ExecutableEntitys`

   of the same software component or Basic Software Module for a `mode machine instance` supporting an asynchronous mode switching procedure. ⌋(*SRS_Rte_00143*, *SRS_Rte_00213*, *SRS_Rte_00018*)

   Rationale: This restriction simplifies the implementation of the semantics of an asynchronous mode switch.

7. If a mode is used to trigger a runnable for entering or leaving the mode, but this runnable has a `ModeDisablingDependency` on the same mode, the `ModeDisablingDependency` inhibits the activation of the runnable on the transition (see section 4.4.4).

   To prevent such a misleading configuration, it is strongly recommended not to configure a `ModeDisablingDependency` for an `OnEntry ExecutableEntity` or `OnExit ExecutableEntity`, using the same mode.

8. In case that the mode machine instance is initialized by `Rte_Init` API the related `OnEntry Runnable Entities` for the `initialMode` have to be executed in the context of the `Rte_Init` API. In order to enable the complete transition to the `initialMode` it is required that all `OnEntry Runnable Entities` are mapped to `RteInitializationRunnableBatch` containers otherwise a part

of the OnEntry Runnable Entities wouldn't be scheduled during the transition to the initialMode.

[constr_9062] Entire mapping of OnEntry Runnable Entities for initialMode to RteInitializationRunnableBatch containers ⌈ Either all or none of the OnEntry Runnable Entities of a particular mode machine instance for the initialMode shall be mapped to RteInitializationRunnableBatch containers. ⌋

[SWS_Rte_06768] ⌈ The RTE Generator shall reject configurations violating [constr_9062]. ⌋*(SRS_Rte_00143, SRS_Rte_00240, SRS_Rte_00018)*

Please note as well [constr_9063] which limits the applicability of the mapping to RteInitializationRunnableBatch containers.

A.8 Restriction concerning SwcInternalBehavior

1. **[SWS_Rte_07686]** ⌈ The RTE Generator shall reject configurations where an AtomicSwComponentType does not contain a SwcInternalBehavior. ⌋*(SRS_Rte_00018)*

A.9 Restrictions concerning Initial Value

1. **[SWS_Rte_07642]** ⌈ When the *external configuration switch* strictInitialValuesCheck is enabled, the RTE Generator shall reject configurations where a SwAddrMethod has a sectionInitializationPolicy set to init but no initValues are specified on the sender or receiver side. ⌋*(SRS_Rte_00068, SRS_Rte_00108, SRS_Rte_00018)*

   Rationale: The initValue is used to guarantee that the RTE won't deliver undefined values.

2. **[SWS_Rte_08311]** ⌈ When the *external configuration switch* strictInitialValuesCheck is enabled, the RTE Generator shall reject configurations where a SwAddrMethod has a sectionInitializationPolicy set to init but no initValue is specified on the inter runnable variable. ⌋*(SRS_Rte_00068, SRS_Rte_00108, SRS_Rte_00018)*

   Rationale: The initValue is used to guarantee that the RTE won't deliver undefined values.

3. **[SWS_Rte_07681]** ⌈ If strict checking of initial values is enabled (see [SWS_Rte_07680]), the RTE Generator shall reject configurations where a ParameterDataPrototype has no initValues. ⌋*(SRS_Rte_00108, SRS_Rte_00018)*

Rationale: This allows to provide the values with a calibration without any involvements from the RTE Generator, and still permits to enable a stricter check on projects where it is required.

A.10 Restriction concerning PerInstanceMemory

1. **[SWS_Rte_07045]** ⌈ The RTE generator shall reject configurations where the `type` attribute of a `'C' typed PerInstanceMemory` is equal to the name of a `ImplementationDataType` contained in the input configuration. ⌋*(SRS_Rte_00013, SRS_Rte_00077)*

   Rationale: This would lead to equally named C type definitions.

A.11 Restrictions concerning unconnected r-port

1. **[SWS_Rte_03019]** ⌈ If strict checking has been enabled (see [SWS_Rte_05099]) there shall not be unconnected r-port. The RTE generator shall in this case reject the configuration with unconnected r-port. ⌋*(SRS_Rte_00139, SRS_Rte_00018)*

   Rationale: Unconnected r-port is considered as wrong configuration of the system.

2. **[SWS_Rte_02750]** ⌈ The RTE Generator shall reject configurations where an r-port typed with a `ParameterInterface` is not connected and an `initValue` of a `ParameterRequireComSpec` is not provided for each `ParameterDataPrototype`s of this `ParameterInterface`. ⌋*(SRS_Rte_00139, SRS_Rte_00159, SRS_Rte_00018)*

A.12 Restrictions regarding communication of mode switch notifications

1. **[SWS_Rte_02670]** ⌈RTE shall not support connections with multiple senders (n:1 communication) of `mode switch notifications` connected to the same receiver. The RTE generator shall reject configurations with multiple senders of `mode switch notifications` connected to the same receiver. ⌋*(SRS_Rte_00131, SRS_Rte_00018)*

   Rationale: No use case is known to justify the required complexity.

2. **[SWS_Rte_02724]** ⌈RTE shall reject configurations where one `ModeDeclarationGroupPrototype` of a provide port is connected to `ModeDeclarationGroupPrototype`s of require ports from more than one partition. ⌋*(SRS_Rte_00131, SRS_Rte_00018)*

RTE does not support a configuration in which the mode user of one mode machine instance are distributed over several partitions except the mode user which is part of the mode manager.

3. For each ModeDeclarationGroup, used in the SW-C's ports, RTE needs a unique mapping to an ImplementationDataType.

   [SWS_Rte_02738] ⌈ RTE shall reject a configuration, in which there is not exactly one ModeRequestTypeMap referencing the ModeDeclarationGroup used in a ModeDeclarationGroupPrototype of the SW-C's ports. ⌋*(SRS_Rte_00144, SRS_Rte_00018)*

A.13 Restrictions regarding Measurement and Calibration

1. **[SWS_Rte_03951]** ⌈ RTE does not support measurement of queued communication. ⌋*(SRS_Rte_00153, SRS_Rte_00018)*

   Rationale: Measurement of queued communication is not supported yet. Reasons are:

   - A queue can be empty. What's to measure then? Data interpretation is ambiguous.
   - Which of the queue entries the measurement data has to be taken from (first pending entry, last entry, an intermediate one, mean value, min. or max. value)? Needs might differ out of user view? Data interpretation is ambiguous.
   - Compared e.g. to sender-receiver last-is-best approach only inefficient solutions are possible because implementation of queues entails storage of information dynamically at different memory locations. So always additional copies are required.

2. **[SWS_Rte_03970]** ⌈ The RTE generator shall reject configurations violating [constr_1092] so containing require ports attached to ParameterSwComponentTypes. ⌋*(SRS_Rte_00154, SRS_Rte_00156, SRS_Rte_00018)*

   Rationale: Require ports on ParameterSwComponentTypes don't make sense. ParameterSwComponentTypes only have to provide calibration parameters to other SwComponentTypes.

A.14 Restriction concerning ExclusiveAreaImplMechanism

1. **Usage of WaitPoints is restricted depending on *ExclusiveAreaImplMechanism***
   If an exclusive area's configuration value for *ExclusiveAreaImplMechanism* is *In-*

terruptBlocking or *OsResource*, no runnable entity shall contain any WaitPoint inside this exclusive area.

Please note that a wait point can either be a modelling WaitPoint e. g. a WaitPoint in the SW-C description caused by the usage of a blocking API (e. g. Rte_Receive) or an implementation wait point caused by a special implementation to fulfill the requirements of the ECU configuration, e. g. the runnable-to-task mapping.

Rationale: The operating system has the limitation that a WaitEvent call is not allowed with disabled interrupts. Therefore the implementation mechanism *InterruptBlocking* cannot be used if the exclusive area contains a WaitPoint.

Further the operating system has the limitation that an OS WaitPoint cannot be entered with occupied OS Resources. This implies that the implementation mechanism *OsResource* cannot be used if the exclusive area contains a WaitPoint.

A.15 Restrictions concerning AtomicSwComponentTypes

1. **[SWS_Rte_07190]** ⌈ The RTE generator shall reject configurations where multiple SwComponentTypes have the same component type symbol regardless of the ARPackage hierarchy. ⌋*(SRS_Rte_00018)*

   Rational: This is required to generated unique names for the *Application Header Files* and component data structures.

2. **[SWS_Rte_07191]** ⌈ The RTE generator shall reject configurations where a SwComponentType has PortPrototypes typed by different PortInterfaces with equal short name but conflicting ApplicationErrors. ApplicationErrors are conflicting if ApplicationErrors with same name do have different errorCodes. ⌋*(SRS_Rte_00018)*

   Rational: This is required to generated unique symbolic names for ApplicationErrors. (see also [SWS_Rte_02576])

A.16 Restriction concerning the enableUpdate attribute of NonqueuedReceiverComSpecs

1. **[SWS_Rte_07654]** ⌈ The RTE Generator shall reject configurations violating [constr_1103] so where a VariableDataPrototype is referenced by a NonqueuedReceiverComSpec with the enableUpdate attribute enabled, when this VariableDataPrototype is referenced by a VariableAccess in the dataReadAccess role. ⌋*(SRS_Rte_00179, SRS_Rte_00018)*

   Rational: the update flag is restricted to explicit communication currently.

A.17 Restrictions concerning the large and dynamic data type

1. **[SWS_Rte_07810]** ⌈ The RTE shall reject the configuration if a `dataElement` with an `ImplementationDataType` with `subElements` with `arraySizeSemantics` equal to `variableSize` resolves to another type than uint8[n]. ⌋*(SRS_Rte_00018)*

   Rationale: COM limits the dynamic signals to the `ComSignalType` UINT_8DYN (see the requirement COM569). COM doesn't support dynamic signals included into signal groups. See more explanations in chapter 4.3.1.14.

2. **[SWS_Rte_07811]** ⌈ The RTE shall reject the configuration if a `dataElement` mapped to a PDU with `ComIPduType` equal to `TP` has a `swImplPolicy` different from `queued`. ⌋*(SRS_Rte_00018)*

   Rationale: Otherwise COM might return `COM_BUSY`. See more explanations in chapter 4.3.1.15.

3. **[SWS_Rte_07812]** ⌈ The RTE shall reject the configuration if a `dataElement` with an `ImplementationDataType` with `subElements` with `arraySizeSemantics` equal to `variableSize` has a `swImplPolicy` different from `queued`. ⌋*(SRS_Rte_00018)*

   Rationale: Otherwise COM might return `COM_BUSY`. See more explanations in chapter 4.3.1.15.

A.18 Restriction concerning REFERENCE types

1. **[SWS_Rte_07670]** ⌈ The RTE shall reject the configuration if an `ImplementationDataType` with category `DATA_REFERENCE` is used in a `PortInterface` and neither sender nor receiver component is a service, complex device driver or ECU abstraction. ⌋*(SRS_Rte_00018)*

   Rationale: Only for AUTOSAR services, complex device drivers or ECU abstraction, the use of references is allowed to prevent the misuse of references for communication via the referenced memory (intra-partition scope). For example, such a misuse could occur with application software components communicating together and mapped to different partitions or ECUs.

A.19 Restriction concerning ModeDeclarationGroup categories and value attributes

1. **[SWS_Rte_06801]** ⌈ The RTE generator shall reject a configuration if constraint [constr_1298] is violated. ⌋*(SRS_Rte_00018)*

[SWS_Rte_06802] ⌈The RTE generator shall reject a configuration if constraint [constr_1299] is violated. ⌋*(SRS_Rte_00018)*

[SWS_Rte_06803] ⌈The RTE generator shall reject a configuration if constraint [constr_1181] is violated. ⌋*(SRS_Rte_00018)*

Rationale: In case of category EXPLICIT_ORDER the onTransitionValue and value attributes are required to generate the according definitions (see 5.5.3 and 6.4.2). Thereby unique numbers are required. In case of ALPHABETIC_ORDER the definition of those values are meaningless and causing the risk of inconsistency to the numbering according the alphabetical sorting.

A.20 Restrictions concerning C/S Interfaces

1. **[SWS_Rte_07845]** ⌈The Rte Generator shall reject configurations where a ClientServerOperation in a PPortPrototype is defined but no RunnableEntity is triggered by an OperationInvokedEvent that references the ClientServerOperation. ⌋*(SRS_Rte_00029, SRS_Rte_00018)*

   Rationale: Otherwise the implementation by a server runnable of the operation in the C/S interface does not exist.

B External Requirements

A summary on model constraints is provided in document [32].

C MISRA C Compliance

In general, all RTE code, whether generated or not, shall conform to the HIS subset of the MISRA C standard [SWS_Rte_01168] [24]. This chapter lists all the MISRA C rules of the HIS subset that may be violated by the generated RTE.

The MISRA C standard was defined with having mainly hand-written code in mind. Part of the MISRA C rules only apply to hand-written code, they do not make much sense in the context of automatic code generation. Additonally, there are some rules that are violated because of technical reasons, mainly to reduce RTE overhead.

The rules listed in this chapter are expected to be violated by RTE code. Violations to the rules listed here do not need to be documented as non-compliant to MISRA C in the generated code itself.

MISRA rule	11
Description	Identifiers (internal and external) shall not rely on significance of more than 31 characters. Furthermore the compiler/linker shall be checked to ensure that 31 character significance and case sensitivity are supported for external identifiers.
Violations	The defined RTE naming convention may result in identifiers with more than 31 characters. The compliance to this rule is under user's control.

MISRA rule	23
Description	All declarations at file scope should be static where possible.
Violations	E.g. for the purpose of monitoring during calibration or debugging it may be necessary to use non-static declarations at file scope.

MISRA rule	42
Description	The comma operator shall not be used, except in the control expression of a *for* loop.
Violations	Function-like macros may have to use the comma operator. Function-like macros are required for efficiency reasons [SRS_BSW_00330].

MISRA rule	45
Description	Type casting from any type to or from pointers shall not be used.
Violations	Casting to/from pointer type may be needed for the interface with COM. Casting from a pointer to a `Data Element with Status` to a pointer to a `Data Element without Status`.

MISRA rule	54

Description	A null statement shall only occur on a line by itself, and shall not have any other text on the same line.
Violations	In an optimized RTE, API calls may result in a null statement. Therefore the compliance to this rule cannot be guaranteed.

D Referenced Meta Classes

Class	**ARElement (abstract)**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::ARPackage			
Note	An element that can be defined stand-alone, i.e. without being part of another element (except for packages of course).			
Base	ARObject,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
–	–	–	–	–

Table D.1: ARElement

Class	**ARPackage**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::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,AtpBlueprintable,CollectableElement,Identifiable,MultilanguageReferrable,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
arPackage	ARPackage	*	aggr	This represents a sub package within an ARPackage, thus allowing for an unlimited package hierarchy. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=blueprintDerivationTime xml.sequenceOffset=30
element	PackageableElement	*	aggr	Elements that are part of this package **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=systemDesignTime xml.sequenceOffset=20
referenceBase	ReferenceBase	*	aggr	This denotes the reference bases for the package. This is the basis for all relative references within the package. The base needs to be selected according to the base attribute within the references. **Stereotypes:** atpSplitable **Tags:** atp.splitkey=shortLabel xml.sequenceOffset=10

Table D.2: ARPackage

Class	ApplicationArrayDataType			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::Datatypes			
Note	An application data type which is an array, each element is of the same application data type. **Tags:** atp.recommendedPackage=ApplicationDataTypes			
Base	ARElement,ARObject,ApplicationCompositeDataType,ApplicationDataType,AtpBlueprint,AtpBlueprintable,AtpClassifier,AtpType,AutosarDataType,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
element	ApplicationArrayElement	1	aggr	This association implements the concept of an array element. That is, in some cases it is necessary to be able to identify single array elements, e.g. as input values for an interpolation routine.

Table D.3: ApplicationArrayDataType

Class	ApplicationArrayElement			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::DataPrototypes			
Note	Describes the properties of the elements of an application array data type.			
Base	ARObject,ApplicationCompositeElementDataPrototype,AtpFeature,AtpPrototype,DataPrototype,Identifiable,MultilanguageReferrable,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
arraySizeSemantics	ArraySizeSemanticsEnum	0..1	attr	This attribute controls how the information about the array size shall be interpreted.
maxNumberOfElements	PositiveInteger	1	attr	The maximum number of elements that the array can contain. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime

Table D.4: ApplicationArrayElement

Class	ApplicationCompositeDataType (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::Datatypes			
Note	Abstract base class for all application data types composed of other data types.			
Base	ARElement,ARObject,ApplicationDataType,AtpBlueprint,AtpBlueprintable,AtpClassifier,AtpType,AutosarDataType,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
–	–	–	–	–

Table D.5: ApplicationCompositeDataType

Class	**ApplicationCompositeElementDataPrototype (abstract)**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::DataPrototypes			
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 instanceRefs.			
Base	ARObject,AtpFeature,AtpPrototype,DataPrototype,Identifiable,Multilanguage Referrable,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
type	ApplicationData Type	1	tref	This represents the corresponding data type. **Stereotypes:** isOfType

Table D.6: ApplicationCompositeElementDataPrototype

Class	**ApplicationDataType (abstract)**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::Datatypes			
Note	ApplicationDataType defines a data type from the application point of view. Especially it should be used whenever something "physical" is at stake. 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 ApplicationDataTypes only.			
Base	ARElement,ARObject,AtpBlueprint,AtpBlueprintable,AtpClassifier,AtpType,Autosar DataType,CollectableElement,Identifiable,MultilanguageReferrable,Packageable Element,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
–	–	–	–	–

Table D.7: ApplicationDataType

Class	**ApplicationError**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	This is a user-defined error that is associated with an element of an AUTOSAR interface. It is specific for the particular functionality or service provided by the AUTOSAR software component.			
Base	ARObject,Identifiable,MultilanguageReferrable,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
errorCode	Integer	1	attr	The RTE generator is forced to assign this value to the corresponding error symbol. Note that for error codes certain ranges are predefined (see RTE specification).

Table D.8: ApplicationError

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

Table D.9: ApplicationPrimitiveDataType

Class	ApplicationRecordDataType			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::Datatypes			
Note	An application data type which can be decomposed into prototypes of other application data types. **Tags:** atp.recommendedPackage=ApplicationDataTypes			
Base	ARElement,ARObject,ApplicationCompositeDataType,ApplicationDataType,AtpBlueprint,AtpBlueprintable,AtpClassifier,AtpType,AutosarDataType,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	Datatype	Mul.	Kind	Note
element (ordered)	ApplicationRecordElement	1..*	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 ApplicationrecordDataType. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime

Table D.10: ApplicationRecordDataType

Class	ApplicationRecordElement			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::DataPrototypes			
Note	Describes the properties of one particular element of an application record data type.			
Base	ARObject,ApplicationCompositeElementDataPrototype,AtpFeature,AtpPrototype,DataPrototype,Identifiable,MultilanguageReferrable,Referrable			
Attribute	Datatype	Mul.	Kind	Note
–	–	–	–	–

Table D.11: ApplicationRecordElement

Class	ApplicationRuleBasedValueSpecification			
Package	M2::AUTOSARTemplates::CommonStructure::Constants			
Note	This meta-class represents rule based values for DataPrototypes typed by ApplicationDataTypes (ApplicationArrayDataType or a compound ApplicationPrimitiveDataType which also boils down to an array-nature).			
Base	ARObject,AbstractRuleBasedValueSpecification,ValueSpecification			
Attribute	Datatype	Mul.	Kind	Note
category	Identifier	1	ref	This represents the category of the RuleBasedValueSpecification **Tags:** xml.sequenceOffset=-20
swAxisCont (ordered)	RuleBasedAxisCont	*	aggr	This represents the axis values of a Compound Primitive Data Type (curve or map). The first swAxisCont describes the x-axis, the second swAxisCont describes the y-axis, the third swAxisCont describes the z-axis. In addition to this, the axis can be denoted in swAxisIndex.
swValueCont	RuleBasedValueCont	0..1	aggr	This represents the values of an array or Compound Primitive Data Type.

Table D.12: ApplicationRuleBasedValueSpecification

Class	ApplicationSwComponentType			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	The ApplicationSwComponentType is used to represent the application software. **Tags:** atp.recommendedPackage=SwComponentTypes			
Base	ARElement,ARObject,AtomicSwComponentType,AtpBlueprint,AtpBlueprintable,AtpClassifier,AtpType,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable,SwComponentType			
Attribute	Datatype	Mul.	Kind	Note
–	–	–	–	–

Table D.13: ApplicationSwComponentType

Class	ApplicationValueSpecification			
Package	M2::AUTOSARTemplates::CommonStructure::Constants			
Note	This meta-class represents values for DataPrototypes typed by ApplicationDataTypes (this includes in particular compound primitives). For further details refer to ASAM CDF 2.0. This meta-class corresponds to some extent with SW-INSTANCE in ASAM CDF 2.0.			
Base	ARObject,ValueSpecification			
Attribute	Datatype	Mul.	Kind	Note

Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
category	Identifier	1	ref	Specifies to which category of ApplicationDataType this ApplicationValueSpecification can be applied (e.g. as an initial value), thus imposing constraints on the structure and semantics of the contained values, see [constr_1006] and [constr_2051].
swAxisCont (ordered)	SwAxisCont	*	aggr	This represents the axis values of a Compound Primitive Data Type (curve or map). The first swAxisCont describes the x-axis, the second swAxisCont describes the y-axis, the third swAxisCont describes the z-axis. In addition to this, the axis can be denoted in swAxisIndex.
swValueCont	SwValueCont	0..1	aggr	This represents the values of a Compound Primitive Data Type.

Table D.14: ApplicationValueSpecification

Class	**ArgumentDataPrototype**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	An argument of an operation, much like a data element, but also carries direction information and is owned by a particular ClientServerOperation.			
Base	ARObject,AtpFeature,AtpPrototype,AutosarDataPrototype,DataPrototype,Identifiable,MultilanguageReferrable,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
direction	ArgumentDirectionEnum	1	attr	This attribute specifies the direction of the argument prototype.
serverArgumentImplPolicy	ServerArgumentImplPolicyEnum	0..1	attr	This defines how the argument type of the servers RunnableEntity is implemented. If the attribute is not defined this has the same semantic as if the attribute is set to useArgumentType
typeBlueprint	AutosarDataType	0..1	ref	This allows to denote the intended type within blueprints. It shall be replaced by a proper type when deriving Interfaces from the Blueprint. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=blueprintDerivationTime

Table D.15: ArgumentDataPrototype

Enumeration	**ArraySizeSemanticsEnum**
Package	M2::AUTOSARTemplates::CommonStructure::ImplementationDataTypes
Note	This type controls how the information about the number of elements in an ApplicationArrayDataType is to be interpreted.
Literal	**Description**
fixedSize	This means that the ApplicationArrayDataType will always have a fixed number of elements.

variableSize	This implies that the actual number of elements in the ApplicationArrayDataType might vary at run-time. The value of arraySize represents the maximum number of elements in the array.

Table D.16: ArraySizeSemanticsEnum

Class	**ArrayValueSpecification**			
Package	M2::AUTOSARTemplates::CommonStructure::Constants			
Note	Specifies the values for an array.			
Base	ARObject,ValueSpecification			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
element (ordered)	ValueSpecification	1..*	aggr	The value for a single array element. All ValueSpecifications aggregated by ArrayValueSpecification shall have the same structure. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime

Table D.17: ArrayValueSpecification

Class	**AssemblySwConnector**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Composition			
Note	AssemblySwConnectors are exclusively used to connect SwComponentPrototypes in the context of a CompositionSwComponentType.			
Base	ARObject,AtpClassifier,AtpFeature,AtpStructureElement,Identifiable,Multilanguage Referrable,Referrable,SwConnector			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
provider	AbstractProvidedPortPrototype	0..1	iref	Instance of providing port.
requester	AbstractRequiredPortPrototype	0..1	iref	Instance of requiring port.

Table D.18: AssemblySwConnector

Class	**AsynchronousServerCallPoint**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::ServerCall			
Note	An AsynchronousServerCallPoint is used for asynchronous invocation of a ClientServerOperation. IMPORTANT: a ServerCallPoint cannot be used concurrently. Once the client RunnableEntity has made the invocation, the ServerCallPoint cannot be used until the call returns (or an error occurs!) at which point the ServerCallPoint becomes available again.			
Base	ARObject,AtpClassifier,AtpFeature,AtpStructureElement,Identifiable,Multilanguage Referrable,Referrable,ServerCallPoint			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table D.19: AsynchronousServerCallPoint

Class	**AsynchronousServerCallResultPoint**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::ServerCall			
Note	If a RunnableEntity owns a AsynchronousServerCallResultPoint it is entitled to get the result of the referenced AsynchronousServerCallPoint. If it is associated with AsynchronousServerCallReturnsEvent, this RTEEvent notifies the completion of the required ClientServerOperation or a timeout. The occurrence of this event can either unblock a WaitPoint or can lead to the invocation of a RunnableEntity.			
Base	ARObject,AtpClassifier,AtpFeature,AtpStructureElement,Identifiable,Multilanguage Referrable,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
asynchronousServerCallPoint	AsynchronousServerCallPoint	1	ref	The referenced Asynchronous Server Call Point defines the asynchronous server call from which the results are returned.

Table D.20: AsynchronousServerCallResultPoint

Class	**AsynchronousServerCallReturnsEvent**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::RTE Events			
Note	This event is raised when an asynchronous server call is finished.			
Base	ARObject,AbstractEvent,AtpClassifier,AtpFeature,AtpStructure Element,Identifiable,MultilanguageReferrable,RTEEvent,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
eventSource	AsynchronousServerCallResultPoint	1	ref	The referenced AsynchronousServerCallResultPoint which is raises the RTEEvent in case of returning asynchronous server call.

Table D.21: AsynchronousServerCallReturnsEvent

Class	**AtomicSwComponentType (abstract)**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	An atomic software component is atomic in the sense that it cannot be further decomposed and distributed across multiple ECUs.			
Base	ARElement,ARObject,AtpBlueprint,AtpBlueprintable,AtpClassifier,Atp Type,CollectableElement,Identifiable,MultilanguageReferrable,Packageable Element,Referrable,SwComponentType			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
internalBehavior	SwcInternalBehavior	0..1	aggr	The SwcInternalBehaviors owned by an AtomicSwComponentType can be located in a different physical file. Therefore the aggregation is «atpSplitable». **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=internalBehavior, variation Point.shortLabel vh.latestBindingTime=preCompileTime

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
symbolProps	SymbolProps	0..1	aggr	This represents the SymbolProps for the AtomicSwComponentType. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=shortName

Table D.22: AtomicSwComponentType

Class	≪atpMixedString≫ **AttributeValueVariationPoint (abstract)**			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling::AttributeValueVariationPoints			
Note	This class represents the ability to derive the value of the Attribute from a system constant (by SwSystemconstDependentFormula). It also provides a bindingTime.			
Base	ARObject,FormulaExpression,SwSystemconstDependentFormula			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
bindingTime	BindingTimeEnum	0..1	attr	This is the binding time in which the attribute value needs to be bound. If this attribute is missing, the attribute is not a variation point. In particular this means that It needs to be a single value according to the type specified in the pure model. It is an error if it is still a formula. **Tags:** xml.attribute=true
blueprintValue	String	0..1	attr	This represents a description that documents how the value shall be defined when deriving objects from the blueprint. **Tags:** xml.attribute=true
sd	String	0..1	attr	This special data is provided to allow synchronization of Attribute value variation points with variant management systems. The usage is subject of agreement between the involved parties. **Tags:** xml.attribute=true
shortLabel	PrimitiveIdentifier	0..1	attr	This allows to identify the variation point. It is also intended to allow RTE support for CompileTime Variation points. **Tags:** xml.attribute=true

Table D.23: AttributeValueVariationPoint

Class	AutosarDataPrototype (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::DataPrototypes			
Note	Base class for prototypical roles of an AutosarDataType.			
Base	ARObject,AtpFeature,AtpPrototype,DataPrototype,Identifiable,MultilanguageReferrable,Referrable			
Attribute	Datatype	Mul.	Kind	Note
type	AutosarDataType	1	tref	This represents the corresponding data type. **Stereotypes:** isOfType

Table D.24: AutosarDataPrototype

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

Table D.25: AutosarDataType

Class	BackgroundEvent			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::RTEEvents			
Note	This event is used to trigger RunnableEntities that are supposed to be executed in the background.			
Base	ARObject,AbstractEvent,AtpClassifier,AtpFeature,AtpStructureElement,Identifiable,MultilanguageReferrable,RTEEvent,Referrable			
Attribute	Datatype	Mul.	Kind	Note
–	–	–	–	–

Table D.26: BackgroundEvent

Class	BaseType (abstract)			
Package	M2::AUTOSARTemplates::CommonStructure::BaseTypes			
Note	This abstract meta-class represents the ability to specify a platform dependant base type.			
Base	ARElement,ARObject,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	Datatype	Mul.	Kind	Note
baseTypeDefinition	BaseTypeDefinition	1	aggr	This is the actual definition of the base type. **Tags:** xml.roleElement=false; xml.roleWrapperElement=false; xml.sequenceOffset=20; xml.typeElement=false; xml.typeWrapperElement=false

Table D.27: BaseType

Class	**BaseTypeDirectDefinition**			
Package	M2::AUTOSARTemplates::CommonStructure::BaseTypes			
Note	This BaseType is defined directly (as opposite to a derived BaseType)			
Base	ARObject,BaseTypeDefinition			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
baseTypeEncoding	BaseTypeEncodingString	1	attr	This specifies, how an object of the current BaseType is encoded, e.g. in an ECU within a message sequence. **Tags:** xml.sequenceOffset=90
baseTypeSize	PositiveInteger	0..1	attr	Describes the length of the data type specified in the container in bits. **Tags:** xml.sequenceOffset=70
byteOrder	ByteOrderEnum	0..1	attr	This attribute specifies the byte order of the base type. **Tags:** xml.sequenceOffset=110
maxBaseTypeSize	PositiveInteger	0..1	attr	Describes the maximum length of the BaseType in bits. **Tags:** xml.sequenceOffset=80
memAlignment	PositiveInteger	0..1	attr	This attribute describes the alignment of the memory object in bits. E.g. "8" specifies, that the object in question is aligned to a byte while "32" specifies that it is aligned four byte. If the value is set to "0" the meaning shall be interpreted as "unspecified". **Tags:** xml.sequenceOffset=100

Attribute	***Datatype***	***Mul.***	***Kind***	***Note***

nativeDeclaration	NativeDeclarationString	0..1	attr	This attribute describes the declaration of such a base type in the native programming language, primarily in the Programming language C. This can then be used by a code generator to include the necessary declarations into a header file. For example
				
				BaseType with
				`shortName: "MyUnsignedInt"`
				`nativeDeclaration: "unsigned short"`
				
				Results in
				`typedef unsigned short MyUnsignedInt;`
				
				If the attribute is not defined the referring ImplementationDataTypes will not be generated as a typedef by RTE.
				
				If a nativeDeclaration type is given it shall fulfill the characteristic given by basetypeEncoding and baseTypeSize.
				
				This is required to ensure the consistent handling and interpretation by software components, RTE, COM and MCM systems.
				
				Tags: xml.sequenceOffset=120

Table D.28: BaseTypeDirectDefinition

Enumeration	**BindingTimeEnum**
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling
Note	This enumerator specifies the applicable binding times for the pre build variation points.
Literal	***Description***
codeGenerationTime	• Coding by hand, based on requirements document. • Tool based code generation, e.g. from a model. • The model may contain variants. • Only code for the selected variant(s) is actually generated.
linkTime	Configure what is included in object code, and what is omitted Based on which variant(s) are selected E.g. for modules that are delivered as object code (as opposed to those that are delivered as source code)

preCompile Time	This is typically the C-Preprocessor. Exclude parts of the code from the compilation process, e.g., because they are not required for the selected variant, because they are incompatible with the selected variant, because they require resources that are not present in the selected variant. Object code is only generated for the selected variant(s). The code that is excluded at this stage code will not be available at later stages.
systemDesignTime	• Designing the VFB. • Software Component types (PortInterfaces). • SWC Prototypes and the Connections between SWCprototypes. • Designing the Topology • ECUs and interconnecting Networks • Designing the Communication Matrix and Data Mapping

Table D.29: BindingTimeEnum

Class	≪`atpMixedString`≫ **BooleanValueVariationPoint**			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling::AttributeValueVariationPoints			
Note	This class represents an attribute value variation point for Boolean attributes. Note that this class might be used in the extended meta-model on			
Base	ARObject,AttributeValueVariationPoint,FormulaExpression,SwSystemconstDependentFormula			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table D.30: BooleanValueVariationPoint

Class	**BswAsynchronousServerCallPoint**			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	Represents an asynchronous procedure call point via the BSW Scheduler.			
Base	ARObject,BswModuleCallPoint,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
calledEntry	BswModuleClientServerEntry	1	ref	The entry to be called.

Table D.31: BswAsynchronousServerCallPoint

Class	BswAsynchronousServerCallResultPoint			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	The callback point for an BswAsynchronousServerCallPoint i.e. the point at which the result can be retrieved from the BSW Scheduler.			
Base	ARObject,BswModuleCallPoint,Referrable			
Attribute	Datatype	Mul.	Kind	Note
asynchronousServerCallPoint	BswAsynchronousServerCallPoint	1	ref	The call point invoking the call to which the result belongs.

Table D.32: BswAsynchronousServerCallResultPoint

Class	BswBackgroundEvent			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	A recurring BswEvent which is used to perform background activities. It is similar to a BswTimingEvent but has no fixed time period and is activated only with low priority.			
Base	ARObject,AbstractEvent,BswEvent,BswScheduleEvent,Identifiable,MultilanguageReferrable,Referrable			
Attribute	Datatype	Mul.	Kind	Note
–	–	–	–	–

Table D.33: BswBackgroundEvent

Class	BswCalledEntity			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	BSW module entity which is designed to be called from another BSW module or cluster.			
Base	ARObject,BswModuleEntity,ExecutableEntity,Identifiable,MultilanguageReferrable,Referrable			
Attribute	Datatype	Mul.	Kind	Note
–	–	–	–	–

Table D.34: BswCalledEntity

Class	BswDataReceivedEvent			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	This event is thrown on reception of the referenced data via Sender-Receiver-Communication over the BSW Scheduler.			
Base	ARObject,AbstractEvent,BswEvent,BswScheduleEvent,Identifiable,MultilanguageReferrable,Referrable			
Attribute	Datatype	Mul.	Kind	Note
data	VariableDataPrototype	1	ref	The received data.

Table D.35: BswDataReceivedEvent

Class	BswDataReceptionPolicy (abstract)			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	Specifies the reception policy for the referred data in sender-receiver communication over the BSW Scheduler. To be used for inter-partition and/or inter-core communication.			
Base	ARObject			
Attribute	Datatype	Mul.	Kind	Note
receivedData	VariableDataPrototype	1	ref	The data received over the BSW Scheduler using this policy.

Table D.36: BswDataReceptionPolicy

Class	BswDebugInfo			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswImplementation			
Note	Collects the information on the data provided to the AUTOSAR debug module.			
Base	ARObject,Identifiable,MultilanguageReferrable,Referrable			
Attribute	Datatype	Mul.	Kind	Note
localDebugData	ImplementationDataTypeElement	*	aggr	A data element declared locally to this module, cluster or library. It shall be used (within AUTOSAR) only for debugging purposes.
parameterAccessedForDebug	ParameterDataPrototype	*	ref	Indicates a parameter as to be debugged.
variableAccessedForDebug	VariableDataPrototype	*	ref	Indicates a variable as to be debugged.

Table D.37: BswDebugInfo

Class	BswEvent (abstract)			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	Base class of various kinds of events which are used to trigger a BswModuleEntity of this BSW module or cluster. The event is local to the BSW module or cluster. The short name of the meta-class instance is intended as an input to configure the required API of the BSW Scheduler.			
Base	ARObject,AbstractEvent,Identifiable,MultilanguageReferrable,Referrable			
Attribute	Datatype	Mul.	Kind	Note
contextLimitation	BswDistinguishedPartition	*	ref	The existence of this reference indicates that the usage of the event is limited to the context of the referred BswDistinguishedPartitions.
disabledInMode	ModeDeclaration	*	iref	The modes, in which this event is disabled. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=disabledInMode
startsOnEvent	BswModuleEntity	1	ref	The entity which is started by the event.

Table D.38: BswEvent

Enumeration	**BswExecutionContext**
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswInterfaces
Note	Specifies the execution context required or guaranteed for the call associated with this service.
Literal	*Description*
hook	Context of an OS "hook" routine always
interruptCat1	CAT1 interrupt context always
interruptCat2	CAT2 interrupt context always
task	Task context always
unspecified	The execution context is not specified by the API

Table D.39: BswExecutionContext

Class	**BswExternalTriggerOccurredEvent**			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	A BswEvent resulting from a trigger released by another module or cluster.			
Base	ARObject,AbstractEvent,BswEvent,BswScheduleEvent,Identifiable,MultilanguageReferrable,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
trigger	Trigger	1	ref	The trigger associated with this event. The trigger is external to this module.

Table D.40: BswExternalTriggerOccurredEvent

Class	**BswImplementation**			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswImplementation			
Note	Contains the implementation specific information in addition to the generic specification (BswModuleDescription and BswBehavior). It is possible to have several different BswImplementations referring to the same BswBehavior. **Tags:** atp.recommendedPackage=BswImplementations			
Base	ARElement,ARObject,CollectableElement,Identifiable,Implementation,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
arReleaseVersion	RevisionLabelString	1	attr	Version of the AUTOSAR Release on which this implementation is based. The numbering contains three levels (major, minor, revision) which are defined by AUTOSAR.
behavior	BswInternalBehavior	1	ref	The behavior of this implementation.
debugInfo	BswDebugInfo	0..1	aggr	Collects the debug info for this implementation. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime

Attribute	Datatype	Mul.	Kind	Note
preconfiguredConfiguration	EcucModuleConfigurationValues	*	ref	Reference to the set of preconfigured (i.e. fixed) configuration values for this BswImplementation. If the BswImplementation represents a cluster of several modules, more than one EcucModuleConfigurationValues element can be referred (at most one per module), otherwise at most one such element can be referred. **Tags:** xml.roleWrapperElement=true
recommendedConfiguration	EcucModuleConfigurationValues	*	ref	Reference to one or more sets of recommended configuration values for this module or module cluster.
vendorApiInfix	Identifier	0..1	ref	In driver modules which can be instantiated several times on a single ECU, SRS_BSW_00347 requires that the names of files, APIs, published parameters and memory allocation keywords are extended by the vendorId and a vendor specific name. This parameter is used to specify the vendor specific name. In total, the implementation specific API name is generated as follows: <ModuleName>_<vendorId>_ <vendorApiInfix>_<API name from SWS>. E.g. assuming that the vendorId of the implementer is 123 and the implementer chose a vendorApiInfix of "v11r456" an API name Can_Write defined in the SWS will translate to Can_123_v11r456_Write. This attribute is mandatory for all modules with upper multiplicity > 1. It shall not be used for modules with upper multiplicity =1. See also SWS_BSW_00102.
vendorSpecificModuleDef	EcucModuleDef	*	ref	Reference to • the vendor specific EcucModuleDef used in this BswImplementation if it represents a single module • several EcucModuleDefs used in this BswImplementation if it represents a cluster of modules • one or no EcucModuleDefs used in this BswImplementation if it represents a library **Tags:** xml.roleWrapperElement=true

Table D.41: BswImplementation

Class	BswInternalBehavior			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	Specifies the behavior of a BSW module or a BSW cluster w.r.t. the code entities visible by the BSW Scheduler. It is possible to have several different BswInternalBehaviors referring to the same BswModuleDescription.			
Base	ARObject,AtpClassifier,AtpFeature,AtpStructureElement,Identifiable,Internal Behavior,MultilanguageReferrable,Referrable			
Attribute	Datatype	Mul.	Kind	Note
distinguishedPartition	BswDistinguishedPartition	*	aggr	Indicates an abstract partition context in which the enclosing BswModuleEntity can be executed. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=60
entity	BswModuleEntity	1..*	aggr	A code entity for which the behavior is described **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=5
event	BswEvent	*	aggr	An event required by this module behavior. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=10
internalTriggeringPoint	BswInternalTriggeringPoint	*	aggr	An internal triggering point. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=2
modeReceiverPolicy	BswModeReceiverPolicy	*	aggr	Implementation policy for the reception of mode switches. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=25
modeSenderPolicy	BswModeSenderPolicy	*	aggr	Implementation policy for providing a mode group. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=20

Attribute	Datatype	Mul.	Kind	Note
perInstanceParameter	ParameterDataPrototype	*	aggr	Describes a read only memory object containing characteristic value(s) needed by this BswInternalBehavior. The role name perInstanceParameter is chosen in analogy to the similar role in the context of SwcInternalBehavior. In contrast to constantMemory, this object is not allocated locally by the module's code, but by the BSW Scheduler and it is accessed from the BSW module via the BSW Scheduler API. The main use case is the support of software emulation of calibration data. The aggregation is subject to variability with the purpose to support implementation variants. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=45
receptionPolicy	BswDataReceptionPolicy	*	aggr	Data reception policy for inter-partition and/or inter-core communication. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=55
schedulerNamePrefix	BswSchedulerNamePrefix	*	aggr	Optional definition of one or more prefixes to be used for the BswScheduler. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=50
serviceDependency	BswServiceDependency	*	aggr	Defines the requirements on AUTOSAR Services for a particular item. The aggregation is subject to variability with the purpose to support the conditional existence of ServiceNeeds. The aggregation is splitable in order to support that ServiceNeeds might be provided in later development steps. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=serviceDependency, variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=40
triggerDirectImplementation	BswTriggerDirectImplementation	*	aggr	Specifies a trigger to be directly implemented via OS calls. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=15

Attribute	Datatype	Mul.	Kind	Note

Table D.42: BswInternalBehavior

Class	BswInternalTriggerOccurredEvent			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	A BswEvent, which can happen sporadically. The event is activated by explicit calls from the module to the BSW Scheduler. The main purpose for such an event is to cause a context switch, e.g. from an ISR context into a task context. Activation and switching are handled within the same module or cluster only.			
Base	ARObject,AbstractEvent,BswEvent,BswScheduleEvent,Identifiable,Multilanguage Referrable,Referrable			
Attribute	Datatype	Mul.	Kind	Note
eventSource	BswInternalTriggeringPoint	1	ref	The activation point is the source of this event.

Table D.43: BswInternalTriggerOccurredEvent

Class	BswInternalTriggeringPoint			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	Represents the activation point for one or more BswInternalTriggerOccurredEvents.			
Base	ARObject,Identifiable,MultilanguageReferrable,Referrable			
Attribute	Datatype	Mul.	Kind	Note
swImplPolicy	SwImplPolicyEnum	0..1	attr	This attribute, when set to value queued, specifies a queued processing of the internal trigger event.

Table D.44: BswInternalTriggeringPoint

Class	BswInterruptEntity			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	BSW module entity, which is designed to be triggered by an interrupt.			
Base	ARObject,BswModuleEntity,ExecutableEntity,Identifiable,Multilanguage Referrable,Referrable			
Attribute	Datatype	Mul.	Kind	Note
interruptCategory	BswInterruptCategory	1	attr	Category of the interrupt
interruptSource	String	1	attr	Allows a textual documentation of the intended interrupt source.

Table D.45: BswInterruptEntity

Class	BswModeReceiverPolicy			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	Specifies the details for the reception of a mode switch for the referred mode group.			
Base	ARObject			
Attribute	Datatype	Mul.	Kind	Note

Attribute	Datatype	Mul.	Kind	Note
enhancedModeApi	Boolean	0..1	attr	This controls the creation of the enhanced mode API that returns information about the previous mode and the next mode. If set to TRUE the enhanced mode API is supposed to be generated. For more details please refer to the SWS_RTE.
requiredModeGroup	ModeDeclarationGroupPrototype	1	ref	The required mode group for which the policy is specified.
supportsAsynchronousModeSwitch	Boolean	1	attr	Specifies whether the module can handle the reception of an asynchronous mode switch (true) or not (false).

Table D.46: BswModeReceiverPolicy

Class	BswModeSenderPolicy			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	Specifies the details for the sending of a mode switch for the referred mode group.			
Base	ARObject			
Attribute	Datatype	Mul.	Kind	Note
ackRequest	BswModeSwitchAckRequest	0..1	aggr	Request for acknowledgement
enhancedModeApi	Boolean	0..1	attr	
providedModeGroup	ModeDeclarationGroupPrototype	1	ref	The provided mode group for which the policy is specified.
queueLength	PositiveInteger	1	attr	Length of call queue on the sender side. The queue is implemented by the RTE resp.BswScheduler. The value must be greater or equal to 0. Setting the value of queueLength to 0 implies non-queued communication.

Table D.47: BswModeSenderPolicy

Class	BswModeSwitchAckRequest			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	Requests acknowledgements that a mode switch has been processed successfully			
Base	ARObject			
Attribute	Datatype	Mul.	Kind	Note
timeout	TimeValue	1	attr	Number of seconds before an error is reported.

Table D.48: BswModeSwitchAckRequest

Class	**BswModeSwitchEvent**			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	A BswEvent resulting from a mode switch.			
Base	ARObject,AbstractEvent,BswEvent,BswScheduleEvent,Identifiable,MultilanguageReferrable,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
activation	ModeActivationKind	1	attr	Kind of activation w.r.t. to the referred mode.
mode (ordered)	ModeDeclaration	1..2	iref	Reference to one or two Modes that initiate the Mode Switch Event.

Table D.49: BswModeSwitchEvent

Class	**BswModeSwitchedAckEvent**			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	The event is raised after a switch of the referenced mode group has been acknowledged or an error occurs. The referenced mode group must be provided by this module.			
Base	ARObject,AbstractEvent,BswEvent,BswScheduleEvent,Identifiable,MultilanguageReferrable,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
modeGroup	ModeDeclarationGroupPrototype	1	ref	A mode group provided by this module. The acknowledgement of a switch of this group raises this event.

Table D.50: BswModeSwitchedAckEvent

Class	**BswModuleCallPoint (abstract)**			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	Represents a point at which a BswModuleEntity handles a procedure call into a BswModuleEntry, either directly or via the BSW Scheduler.			
Base	ARObject,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
contextLimitation	BswDistinguishedPartition	*	ref	The existence of this reference indicates that the call point is used only in the context of the referred BswDistinguishedPartitions.

Table D.51: BswModuleCallPoint

Class	**BswModuleClientServerEntry**			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswInterfaces			
Note	This meta-class represents a single API entry into the BSW module or cluster that has the ability to be called in client-server fashion via the BSW Scheduler. In this regard it is more special than BswModuleEntry and can be seen as a wrapper around the BswModuleEntry to which it refers (property encapsulatedEntry). **Tags:** atp.recommendedPackage=BswModuleEntrys			
Base	ARObject,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
encapsulatedEntry	BswModuleEntry	1	ref	The underlying BswModuleEntry. **Tags:** xml.sequenceOffset=5
isReentrant	Boolean	0..1	attr	Reentrancy from the viewpoint of clients invoking the service via the BSW Scheduler: • True: Enables the service to be invoked again, before the service has finished. • False: It is prohibited to invoke the service again before is has finished. **Tags:** xml.sequenceOffset=10
isSynchronous	Boolean	0..1	attr	Synchronicity from the viewpoint of clients invoking the service via the BSW Scheduler: • True: This calls a synchronous service, i.e. the service is completed when the call returns. • False: The service (on semantical level) may not be complete when the call returns. **Tags:** xml.sequenceOffset=15

Table D.52: BswModuleClientServerEntry

Class	**BswModuleDependency**			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswInterfaces			
Note	This class collects the dependencies of a BSW module or cluster on a certain other BSW module.			
Base	ARObject,Identifiable,MultilanguageReferrable,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
expectedCallback	BswModuleEntry	*	ref	Indicates a callback expected to be called from another module and implemented by this module. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=15

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
requiredEntry	BswModuleEntry	*	ref	Indicates an entry into another modules which is required by this module. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=10
serviceItem	ServiceNeeds	*	aggr	A single item (example: Nv block) for which the quality of a service is defined. The aggregation is marked as «atpSplitable» to allow for extension during the ECU configuration process. This association is deprecated since R4.0.3, since ServiceNeeds shall be associated with the new element BswServiceDependency within the BswInternalBehavior. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=shortName; atp.Status=obsolete xml.sequenceOffset=20
targetModuleId	PositiveInteger	0..1	attr	AUTOSAR identifier of the target module of which the dependencies are defined. This information is optional, because the target module may also be identified by targetModuleRef. **Tags:** xml.sequenceOffset=5
targetModuleRef	BswModuleDescription	0..1	ref	Reference to the target module. It is an «atpUriDef» because the reference shall be used to identify the target module without actually needing the description of that target module. **Stereotypes:** atpUriDef; atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=7

Table D.53: BswModuleDependency

Class	**BswModuleDescription**			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswOverview			
Note	Root element for the description of a single BSW module or BSW cluster. In case it describes a BSW module, the short name of this element equals the name of the BSW module. **Tags:** atp.recommendedPackage=BswModuleDescriptions			
Base	ARElement,ARObject,AtpBlueprint,AtpBlueprintable,AtpClassifier,AtpFeature,AtpStructureElement,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
bswModuleDependency	BswModuleDependency	*	aggr	Describes the dependency to another BSW module. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=20
bswModuleDocumentation	SwComponentDocumentation	0..1	aggr	This adds a documentation to the BSW module. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=bswModuleDocumentation, variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=6
internalBehavior	BswInternalBehavior	*	aggr	The various BswInternalBehaviors associated with a BswModuleDescription can be distributed over several physical files. Therefore the aggregation is «atpSplitable». **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=shortName xml.sequenceOffset=65
moduleId	PositiveInteger	0..1	attr	Refers to the BSW Module Identifier defined by the AUTOSAR standard. For non-standardized modules, a proprietary identifier can be optionally chosen. **Tags:** xml.sequenceOffset=5
outgoingCallback	BswModuleEntry	*	ref	Specifies a callback, which will be called from this module if required by another module. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=15
providedClientServerEntry	BswModuleClientServerEntry	*	aggr	Specifies that this module provides a client server entry which can be called from another parition or core.This entry is declared locally to this context and will be connected to the requiredClientServerEntry of another or the same module via the configuration of the BSW Scheduler. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=45

Attribute	Datatype	Mul.	Kind	Note
providedData	VariableDataPrototype	*	aggr	Specifies a data prototype provided by this module in order to be read from another partition or core.The providedData is declared locally to this context and will be connected to the requiredData of another or the same module via the configuration of the BSW Scheduler. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=55
providedEntry	BswModuleEntry	*	ref	Specifies an entry provided by this module which can be called by other modules. This includes "main" functions and interrupt routines, but not callbacks (because the signature of a callback is defined by the caller). **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=10
providedModeGroup	ModeDeclarationGroupPrototype	*	aggr	A set of modes which is owned and provided by this module or cluster. It can be connected to the requiredModeGroups of other modules or clusters via the configuration of the BswScheduler. It can also be synchronized with modes provided via ports by an associated ServiceSwComponentType, EcuAbstractionSwComponentType or ComplexDeviceDriverSwComponentType. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=25
releasedTrigger	Trigger	*	aggr	A Trigger released by this module or cluster. It can be connected to the requiredTriggers of other modules or clusters via the configuration of the BswScheduler. It can also be synchronized with Triggers provided via ports by an associated ServiceSwComponentType, EcuAbstractionSwComponentType or ComplexDeviceDriverSwComponentType. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=35

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
requiredClientServerEntry	BswModuleClientServerEntry	*	aggr	Specifies that this module requires a client server entry which can be implemented on another parition or core.This entry is declared locally to this context and will be connected to the providedClientServerEntry of another or the same module via the configuration of the BSW Scheduler. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=50
requiredData	VariableDataPrototype	*	aggr	Specifies a data prototype required by this module in oder to be provided from another partition or core.The requiredData is declared locally to this context and will be connected to the providedData of another or the same module via the configuration of the BswScheduler. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=60
requiredModeGroup	ModeDeclarationGroupPrototype	*	aggr	Specifies that this module or cluster depends on a certain mode group. The requiredModeGroup is local to this context and will be connected to the providedModeGroup of another module or cluster via the configuration of the BswScheduler. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=30
requiredTrigger	Trigger	*	aggr	Specifies that this module or cluster reacts upon an external trigger.This requiredTrigger is declared locally to this context and will be connected to the providedTrigger of another module or cluster via the configuration of the BswScheduler. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=40

Table D.54: BswModuleDescription

Class	**BswModuleEntity (abstract)**			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	Specifies the smallest code fragment which can be described for a BSW module or cluster within AUTOSAR.			
Base	ARObject,ExecutableEntity,Identifiable,MultilanguageReferrable,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*

Attribute	Datatype	Mul.	Kind	Note
accessedModeGroup	ModeDeclarationGroupPrototype	*	ref	A mode group which is accessed via API call by this entity. It must be a ModeDeclarationGroupPrototype required by this module or cluster. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
activationPoint	BswInternalTriggeringPoint	*	ref	Activation point used by the module entity to activate one or more internal triggers. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
callPoint	BswModuleCallPoint	*	aggr	A call point used in the code of this entitiy. The variablity of this association is especially targeted at debug scenarios: It is possible to have one variant calling into the AUTOSAR debug module and another one which doesn't. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
calledEntry	BswModuleEntry	*	ref	The entry of another (or the same) BSW module which is called by this entry (usually via C function call). This information allows to set up a model of call chains. The variablity of this association is especially targeted at debug scenarios: It is possible to have one variant calling into the AUTOSAR debug module and another one which doesn't. Note that this relation has been merked as obsolete, since the more powerful definition of a callPoint should be used. **Stereotypes:** atpVariation **Tags:** atp.Status=obsolete vh.latestBindingTime=preCompileTime
dataReceivePoint	BswVariableAccess	*	aggr	The data is received via the BSW Scheduler. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
dataSendPoint	BswVariableAccess	*	aggr	The data is sent via the BSW Scheduler. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
implementedEntry	BswModuleEntry	1	ref	The entry which is implemented by this module entity.

Attribute	Datatype	Mul.	Kind	Note
issuedTrigger	Trigger	*	ref	A trigger issued by this entity via BSW Scheduler API call. It must be a BswTrigger released (i.e. owned) by this module or cluster. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
managedModeGroup	ModeDeclarationGroupPrototype	*	ref	A mode group which is managed by this entity. It must be a ModeDeclarationGroupPrototype provided by this module or cluster. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
schedulerNamePrefix	BswSchedulerNamePrefix	0..1	ref	A prefix to be used in generated names for the BswModuleScheduler in the context of this BswModuleEntity, for example entry point prototypes, macros for dealing with exclusive areas, header file names. Details are defined in the SWS RTE. The prefix supersedes default rules for the prefix of those names.

Table D.55: BswModuleEntity

<table>
<tr><td>Class</td><td colspan="4">BswModuleEntry</td></tr>
<tr><td>Package</td><td colspan="4">M2::AUTOSARTemplates::BswModuleTemplate::BswInterfaces</td></tr>
<tr><td>Note</td><td colspan="4">This class represents a single API entry (C-function prototype) into the BSW module or cluster.

The name of the C-function is equal to the short name of this element with one exception: In case of multiple instances of a module on the same CPU, special rules for "infixes" apply, see description of class BswImplementation.

Tags: atp.recommendedPackage=BswModuleEntrys</td></tr>
<tr><td>Base</td><td colspan="4">ARElement,ARObject,AtpBlueprint,AtpBlueprintable,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable</td></tr>
<tr><td>Attribute</td><td>Datatype</td><td>Mul.</td><td>Kind</td><td>Note</td></tr>
<tr><td>argument (ordered)</td><td>SwServiceArg</td><td>*</td><td>aggr</td><td>An argument belonging to this BswModuleEntry.

Stereotypes: atpVariation
Tags: vh.latestBindingTime=blueprintDerivationTime
xml.sequenceOffset=45</td></tr>
<tr><td>callType</td><td>BswCallType</td><td>1</td><td>attr</td><td>The type of call associated with this service.

Tags: xml.sequenceOffset=25</td></tr>
</table>

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
executionContext	BswExecutionContext	1	attr	Specifies the execution context which is required (in case of entries into this module) or guaranteed (in case of entries called from this module) for this service. **Tags:** xml.sequenceOffset=30
isReentrant	Boolean	1	attr	Reentrancy from the viewpoint of function callers: • True: Enables the service to be invoked again, before the service has finished. • False: It is prohibited to invoke the service again before is has finished. **Tags:** xml.sequenceOffset=15
isSynchronous	Boolean	1	attr	Synchronicity from the viewpoint of function callers: • True: This calls a synchronous service, i.e. the service is completed when the call returns. • False: The service (on semantical level) may not be complete when the call returns. **Tags:** xml.sequenceOffset=20
returnType	SwServiceArg	0..1	aggr	The return type belonging to this bswModuleEntry. **Tags:** xml.sequenceOffset=40
role	Identifier	0..1	ref	Specifies the role of the entry in the given context. It shall be equal to the standardized name of the service call, especially in cases where no ServiceIdentifier is specified, e.g. for callbacks. Note that the ShortName is not always sufficient because it maybe vendor specific (e.g. for callbacks which can have more than one instance). **Tags:** xml.sequenceOffset=10
serviceId	PositiveInteger	0..1	attr	Refers to the service identifier of the Standardized Interfaces of AUTOSAR basic software. For non-standardized interfaces, it can optionally be used for proprietary identification. **Tags:** xml.sequenceOffset=5
swServiceImplPolicy	SwServiceImplPolicyEnum	1	attr	Denotes the implementation policy as a standard function call, inline function or macro. This has to be specified on interface level because it determines the signature of the call. **Tags:** xml.sequenceOffset=35

Table D.56: BswModuleEntry

Class	BswOperationInvokedEvent			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	This event is thrown on operation invocation in Client-Server-Communication via the BSW Scheduler. Its "entry" reference provides the BswClientServerEntry that is called subsequently. Note this event is not needed in case of direct function calls.			
Base	ARObject,AbstractEvent,BswEvent,Identifiable,MultilanguageReferrable,Referrable			
Attribute	Datatype	Mul.	Kind	Note
entry	BswModuleClientServerEntry	1	ref	The providedClientServerEntry invoked by this event.

Table D.57: BswOperationInvokedEvent

Class	BswQueuedDataReceptionPolicy			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	Reception policy attributes specific for queued receiving.			
Base	ARObject,BswDataReceptionPolicy			
Attribute	Datatype	Mul.	Kind	Note
queueLength	PositiveInteger	1	attr	Length of queue for received events.

Table D.58: BswQueuedDataReceptionPolicy

Class	BswSchedulableEntity			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	BSW module entity, which is designed for control by the BSW Scheduler. It may for example implement a so-called "main" function.			
Base	ARObject,BswModuleEntity,ExecutableEntity,Identifiable,MultilanguageReferrable,Referrable			
Attribute	Datatype	Mul.	Kind	Note
–	–	–	–	–

Table D.59: BswSchedulableEntity

Class	BswScheduleEvent (abstract)			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	BswEvent that is able to start a BswSchedulabeEntity.			
Base	ARObject,AbstractEvent,BswEvent,Identifiable,MultilanguageReferrable,Referrable			
Attribute	Datatype	Mul.	Kind	Note
–	–	–	–	–

Table D.60: BswScheduleEvent

Class	**BswSchedulerNamePrefix**			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	A prefix to be used in names of generated code artifacts which make up the interface of a BSW module to the BswScheduler.			
Base	ARObject,ImplementationProps,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
–	–	–	–	–

Table D.61: BswSchedulerNamePrefix

Class	**BswSynchronousServerCallPoint**			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	Represents a synchronous procedure call point via the BSW Scheduler.			
Base	ARObject,BswModuleCallPoint,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
calledEntry	BswModuleClientServerEntry	1	ref	The entry to be called.
calledFromWithinExclusiveArea	ExclusiveAreaNestingOrder	0..1	ref	This indicates that the call point is located at the deepest level inside one or more ExclusiveAreas that are nested in the given order.

Table D.62: BswSynchronousServerCallPoint

Class	**BswTimingEvent**			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	A recurring BswEvent driven by a time period.			
Base	ARObject,AbstractEvent,BswEvent,BswScheduleEvent,Identifiable,MultilanguageReferrable,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
period	TimeValue	1	attr	Requirement for the time period (in seconds) by which this event is triggered.

Table D.63: BswTimingEvent

Class	**BswTriggerDirectImplementation**			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	Specifies a released trigger to be directly implemented via OS calls, for example in a Complex Driver module.			
Base	ARObject			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
masteredTrigger	Trigger	1	ref	The trigger which is directly mastered by this module. There may be several different BswTriggerDirectImplementations mastering the same Trigger. This may be required e.g. due to memory partitioning.

Attribute	Datatype	Mul.	Kind	Note
task	Identifier	1	ref	The name of the OS task, which is controlled by the referred trigger. This means, that the module uses the trigger condition to directly activate an OS task instead of calling an API of the BswScheduler. The task name is required by the RTE generator resp. BswScheduler to raise the appropriate events in components or modules receiving the trigger.

Table D.64: BswTriggerDirectImplementation

Class	**CalibrationParameterValue**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::MeasurementAndCalibration::CalibrationParameterValues			
Note	Specifies instance specific calibration parameter values used to initialize the memory objects implementing calibration parameters in the generated RTE code. RTE generator will use the implInitValue to override the initial values specified for the DataPrototypes of a component type. The applInitValue is used to exchange init values with the component vendor not publishing the transformation algorithm between ApplicationDataTypes and ImplementationDataTypes or defining a instance specific initialization of components which are only defined with ApplicationDataTypes. Note: If both representations of init values are available these need to represent the same content. Note further that in this case an explicit mapping of ValueSpecification is not implemented because calibration parameters are delivered back after the calibration phase.			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
applInitValue	ValueSpecification	0..1	aggr	This is the initial value specification structured according to the ApplicationDataType
implInitValue	ValueSpecification	0..1	aggr	This is the initial value specification structured according to the ImplementationDataType
initializedParameter	FlatInstanceDescriptor	1	ref	This represents the parameter that is initialized by the CalibrationParameterValue.

Table D.65: CalibrationParameterValue

Class	**ClientServerApplicationErrorMapping**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	This meta-class represents the ability to map ApplicationErrors onto each other.			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
firstApplicationError	ApplicationError	1	ref	This represents the first ApplicationError in the context of the ClientServerApplicationErrorMapping.

Attribute	Datatype	Mul.	Kind	Note
secondApplicationError	ApplicationError	1	ref	This represents the second ApplicationError in the context of the ClientServerApplicationErrorMapping.

Table D.66: ClientServerApplicationErrorMapping

Class	ClientServerInterface			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	A client/server interface declares a number of operations that can be invoked on a server by a client. **Tags:** atp.recommendedPackage=PortInterfaces			
Base	ARElement,ARObject,AtpBlueprint,AtpBlueprintable,AtpClassifier,AtpType,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,PortInterface,Referrable			
Attribute	Datatype	Mul.	Kind	Note
operation	ClientServerOperation	1..*	aggr	ClientServerOperation(s) of this ClientServerInterface. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=blueprintDerivationTime
possibleError	ApplicationError	*	aggr	Application errors that are defined as part of this interface.

Table D.67: ClientServerInterface

Class	ClientServerInterfaceMapping			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	Defines the mapping of ClientServerOperations in context of two different ClientServerInterfaces.			
Base	ARObject,AtpBlueprint,AtpBlueprintable,Identifiable,MultilanguageReferrable,PortInterfaceMapping,Referrable			
Attribute	Datatype	Mul.	Kind	Note
errorMapping	ClientServerApplicationErrorMapping	*	aggr	Map two different ApplicationErrors defined in the context of two different ClientServerInterfaces.
operationMapping	ClientServerOperationMapping	1..*	aggr	Mapping of two ClientServerOperations in two different ClientServerInterfaces

Table D.68: ClientServerInterfaceMapping

Class	**ClientServerOperation**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	An operation declared within the scope of a client/server interface.			
Base	ARObject,AtpClassifier,AtpFeature,AtpStructureElement,Identifiable,Multilanguage Referrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
argument (ordered)	ArgumentDataPrototype	*	aggr	An argument of this ClientServerOperation **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=blueprintDerivationTime
possibleError	ApplicationError	*	ref	Possible errors that may by raised by the referring operation.

Table D.69: ClientServerOperation

Class	**ClientServerToSignalMapping**			
Package	M2::AUTOSARTemplates::SystemTemplate::DataMapping			
Note	This element maps the ClientServerOperation to call- and return-SystemSignals. The serialization is defined by the referenced SerializationTechnology. **Tags:** atp.Status=draft			
Base	ARObject,DataMapping			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
callSignal	SystemSignal	1	ref	Reference to the callSignal to which the IN and INOUT ArgumentDataPrototypes are mapped.
clientServerOperation	ClientServerOperation	1	iref	Reference to a ClientServerOperation, which is mapped to a call SystemSignal and a return SystemSignal.
lengthClientId	PositiveInteger	0..1	attr	This attribute defines the length of the used client identifier in bits. If the attribute does not exist or its value is set to 0 this means that the client identifier is not used.
lengthSequenceCounter	PositiveInteger	0..1	attr	The purpose of a sequence counter is to map a response to the correct request of a known client. This attribute describes the length of the used sequence counter in bits. If the attribute does not exist or its value is set to 0 this means that the sequence counter is not used.
returnSignal	SystemSignal	0..1	ref	Reference to the returnSignal to which the OUT and INOUT ArgumentDataPrototypes are mapped.
serializer	SerializationTechnology	1	ref	The referenced SerializationTechnology element contains the necessary information how data shall be serialized.

Table D.70: ClientServerToSignalMapping

Class	ComplexDeviceDriverSwComponentType			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	The ComplexDeviceDriverSwComponentType is a special AtomicSwComponentType that has direct access to hardware on an ECU and which is therefore linked to a specific ECU or specific hardware. The ComplexDeviceDriverSwComponentType introduces the possibility to link from the software representation to its hardware description provided by the ECU Resource Template. **Tags:** atp.recommendedPackage=SwComponentTypes			
Base	ARElement,ARObject,AtomicSwComponentType,AtpBlueprint,AtpBlueprintable,AtpClassifier,AtpType,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable,SwComponentType			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
hardwareElement	HwDescriptionEntity	*	ref	Reference from the ComplexDeviceDriverSwComponentType to the description of the used HwElements.

Table D.71: ComplexDeviceDriverSwComponentType

Class	CompositionSwComponentType			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Composition			
Note	A CompositionSwComponentType aggregates SwComponentPrototypes (that in turn are typed by SwComponentTypes) as well as SwConnectors for primarily connecting SwComponentPrototypes among each others and towards the surface of the CompositionSwComponentType. By this means hierarchical structures of software-components can be created. **Tags:** atp.recommendedPackage=SwComponentTypes			
Base	ARElement,ARObject,AtpBlueprint,AtpBlueprintable,AtpClassifier,AtpType,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable,SwComponentType			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**

Attribute	Datatype	Mul.	Kind	Note
component	SwComponentPrototype	*	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 in that they are not scheduled by the RTE. The aggregation is marked as atpSplitable in order to allow the addition of service components to the ECU extract during the ECU integration. The use case for having 0 components owned by the CompositionSwComponentType could be to deliver an empty CompositionSwComponentType to e.g. a supplier for filling the internal structure. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, 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. The aggregation is marked as atpSplitable in order to allow the extension of the ECU extract with AssemblySwConnectors between ApplicationSwComponentTypes and ServiceSwComponentTypes during the ECU integration. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=postBuild
constantValueMapping	ConstantSpecificationMappingSet	*	ref	Reference to the ConstantSpecificationMapping to be applied for initValues of PPortComSpecs and RPortComSpec.

Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
dataTypeMapping	DataTypeMappingSet	*	ref	Reference to the DataTypeMapping to be applied for the used ApplicationDataTypes in PortInterfaces. Background: when developing subsystems it may happen that ApplicationDataTypes are used on the surface of CompositionSwComponentTypes. In this case it would be reasonable to be able to also provide the intended mapping to the ImplementationDataTypes. However, this mapping shall be informal and not technically binding for the implementers 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.
instantiationRTEEventProps	InstantiationRTEEventProps	*	aggr	This allows to define instantiation specific properties for RTE Events, in particular for instance specific scheduling. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortLabel, variationPoint.shortLabel vh.latestBindingTime=codeGenerationTime

Table D.72: CompositionSwComponentType

Class	**CompuConst**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::ComputationMethod			
Note	This meta-class represents the fact that the value of a computation method scale is constant.			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
compuConstContentType	CompuConstContent	1	aggr	This is the actual content of the constant compu method scale. **Tags:** xml.roleElement=false; xml.roleWrapperElement=false; xml.sequenceOffset=10; xml.typeElement=false; xml.typeWrapperElement=false

Table D.73: CompuConst

Class	**CompuMethod**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::ComputationMethod			
Note	This meta-class represents the ability to express the relationship between a physical value and the mathematical representation. Note that this is still independent of the technical implementation in data types. It only specifies the formula how the internal value corresponds to its physical pendant. **Tags:** atp.recommendedPackage=CompuMethods			
Base	ARElement,ARObject,AtpBlueprint,AtpBlueprintable,Collectable Element,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
compuInter nalToPhys	Compu	0..1	aggr	This specifies the computation from internal values to physical values. **Tags:** xml.sequenceOffset=80
compuPhy sToInternal	Compu	0..1	aggr	This represents the computation from physical values to the internal values. **Tags:** xml.sequenceOffset=90
displayFor mat	DisplayFormatS tring	0..1	attr	This property specifies, how the physical value shall be displayed e.g. in documents or measurement and calibration tools. **Tags:** xml.sequenceOffset=20
unit	Unit	0..1	ref	This is the physical unit of the Physical values for which the CompuMethod applies. **Tags:** xml.sequenceOffset=30

Table D.74: CompuMethod

Class	**CompuNominatorDenominator**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::ComputationMethod			
Note	This class represents the ability to express a polynomial either as Nominator or as Denominator.			
Base	ARObject			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
v (ordered)	Numerical	*	attr	this is the list of polynomial factors. Note that the first vf represents the power=0. The polynomial is v[0] * xˆ 0 + v[1] * xˆ 1 ... **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.roleElement=true; xml.roleWrapper Element=false; xml.sequenceOffset=20; xml.type Element=false; xml.typeWrapperElement=false

Table D.75: CompuNominatorDenominator

Class	**CompuRationalCoeffs**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::ComputationMethod			
Note	This meta-class represents the ability to express a rational function by specifying the coefficients of nominator and denominator.			
Base	ARObject			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
compuDenominator	CompuNominatorDenominator	1	aggr	This is the denominator of the expression. **Tags:** xml.sequenceOffset=30
compuNumerator	CompuNominatorDenominator	1	aggr	This is the numerator of the rational expression. **Tags:** xml.sequenceOffset=20

Table D.76: CompuRationalCoeffs

Class	**CompuScale**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::ComputationMethod			
Note	This meta-class represents the ability to specify one segment of a segmented computation method.			
Base	ARObject			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
desc	MultiLanguageOverviewParagraph	0..1	aggr	<desc> represents a general but brief description of the object in question. **Tags:** xml.sequenceOffset=30
compuInverseValue	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:** xml.sequenceOffset=60
compuScaleContents	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
lowerLimit	Limit	0..1	ref	This specifies the lower limit of the scale. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=40

Attribute	Datatype	Mul.	Kind	Note
mask	PositiveInteger	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	ref	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	ref	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	ref	This specifies the upper limit of a of the scale. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=50

Table D.77: CompuScale

Class	**CompuScaleConstantContents**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::ComputationMethod			
Note	This meta-class represents the fact that a particular scale of the computation method is constant.			
Base	ARObject,CompuScaleContents			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
compuConst	CompuConst	1	aggr	This represents the fact that the scale is a constant. The use case is mainly a non interplolated scale. It is a simplification of the fact that a constant scale can also be expressed as Rational Function of oder 0. **Tags:** xml.sequenceOffset=90

Table D.78: CompuScaleConstantContents

Class	**CompuScales**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::ComputationMethod			
Note	This meta-class represents the ability to stepwise express a computation method.			
Base	ARObject,CompuContent			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
compuScale (ordered)	CompuScale	*	aggr	This represents one scale within the compu method. Note that it contains a Variationpoint in order to support blueprints of enumerations. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=blueprintDerivationTime xml.roleElement=true; xml.roleWrapperElement=true; xml.sequenceOffset=40; xml.typeElement=false; xml.typeWrapperElement=false

Table D.79: CompuScales

Class	≪atpMixedString≫ **ConditionByFormula**			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling			
Note	This class represents a condition which is computed based on system constants according to the specified expression. The expected result is considered as boolean value. The result of the expression is interpreted as a condition. • "0" represents "false"; • a value other than zero is considered "true"			
Base	ARObject,FormulaExpression,SwSystemconstDependentFormula			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
bindingTime	BindingTimeEnum	1	attr	This attribute specifies the point in time when condition may be evaluated at earliest. At this point in time all referenced system constants shall have a value. **Tags:** xml.attribute=true

Table D.80: ConditionByFormula

Class	**ConsistencyNeeds**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::ImplicitCommunicationBehavior			
Note	This meta-class represents the ability to define requirements on the implicit communication behavior.			
Base	ARObject,AtpBlueprint,AtpBlueprintable,Identifiable,MultilanguageReferrable,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**

Attribute	Datatype	Mul.	Kind	Note
dpgDoesNotRequireCoherency	DataPrototypeGroup	*	aggr	This group of VariableDataPrototypes does not require coherency with respect to the implicit communication behavior. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=preCompileTime
dpgRequiresCoherency	DataPrototypeGroup	*	aggr	This group of VariableDataPrototypes requires coherency with respect to the implicit communication behavior, i.e. all read and write access to VariableDataPrototypes in the DataPrototypeGroup by the RunnableEntitys of the RunnableEntityGroup need to be handled in a coherent manner. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=preCompileTime
regDoesNotRequireStability	RunnableEntityGroup	*	aggr	This group of RunnableEntities does not require stability with respect to the implicit communication behavior. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=preCompileTime
regRequiresStability	RunnableEntityGroup	*	aggr	This group of RunnableEntities requires stability with respect to the implicit communication behavior, i.e. all read and write access to VariableDataPrototypes in the DataPrototypeGroup by the RunnableEntitys of the RunnableEntityGroup need to be handled in a stable manner. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=preCompileTime

Table D.81: ConsistencyNeeds

Class	ConstantSpecificationMapping			
Package	M2::AUTOSARTemplates::CommonStructure::Constants			
Note	This meta-class is used to create an association of two ConstantSpecifications. One ConstantSpecification is supposed to be defined in the application domain while the other should be defined in the implementation domain. Hence the ConstantSpecificationMapping needs to be used where a ConstantSpecification defined in one domain needs to be associated to a ConstantSpecification in the other domain. This information is crucial for the RTE generator.			
Base	ARObject			
Attribute	Datatype	Mul.	Kind	Note
applConstant	ConstantSpecification	1	ref	A ConstantSpecification defined in the application domain.
implConstant	ConstantSpecification	1	ref	A ConstantSpecification defined in the implementation domain.

Table D.82: ConstantSpecificationMapping

Class	DataConstr			
Package	M2::AUTOSARTemplates::CommonStructure::GlobalConstraints			
Note	This meta-class represents the ability to specify constraints on data. **Tags:** atp.recommendedPackage=DataConstrs			
Base	ARElement,ARObject,AtpBlueprint,AtpBlueprintable,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	Datatype	Mul.	Kind	Note
dataConstrRule	DataConstrRule	*	aggr	This is one particular rule within the data constraints. **Tags:** xml.roleElement=true; xml.roleWrapperElement=true; xml.sequenceOffset=30; xml.typeElement=false; xml.typeWrapperElement=false

Table D.83: DataConstr

Class	DataMapping (abstract)			
Package	M2::AUTOSARTemplates::SystemTemplate::DataMapping			
Note	Mapping of port elements (data elements and parameters) to frames and signals.			
Base	ARObject			
Attribute	Datatype	Mul.	Kind	Note
communicationDirection	CommunicationDirectionType	0..1	attr	This attribute controls the direction into which the mapped SystemSignal is communicated with respect to the kind of PortPrototype used as the context element of the DataMapping.
eventGroup	ConsumedEventGroup	*	ref	Via this reference a connection between the VFB View and the Ethernet EventGroups can be created.

Attribute	Datatype	Mul.	Kind	Note
eventHandler	EventHandler	*	ref	Via this reference a connection between the VFB View and the Ethernet EventHandlers can be created.
introduction	DocumentationBlock	0..1	aggr	This represents introductory documentation about the data mapping.
serviceInstance	AbstractServiceInstance	*	ref	Via this reference a connection between the VFB View and the Ethernet Services can be created.

Table D.84: DataMapping

Class	**DataPrototype (abstract)**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::DataPrototypes			
Note	Base class for prototypical roles of any data type.			
Base	ARObject,AtpFeature,AtpPrototype,Identifiable,MultilanguageReferrable,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
swDataDefProps	SwDataDefProps	0..1	aggr	This property allows to specify data definition properties which apply on data prototype level.

Table D.85: DataPrototype

Class	**DataPrototypeGroup**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::ImplicitCommunicationBehavior			
Note	This meta-class represents the ability to define a collection of DataPrototypes that are subject to the formal definition of implicit communication behavior. The definition of the collection can be nested.			
Base	ARObject,AtpClassifier,AtpFeature,AtpStructureElement,Identifiable,MultilanguageReferrable,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
dataPrototypeGroup	DataPrototypeGroup	*	iref	This represents the ability to define nested groups of VariableDataPrototypes. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
implicitDataAccess	VariableDataPrototype	*	iref	This represents a collection of VariableDataPrototypes that belong to the enclosing DataPrototypeGroup **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime

Table D.86: DataPrototypeGroup

Class	DataReceiveErrorEvent			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::RTE Events			
Note	This event is raised by the RTE when the Com layer detects and notifies an error concerning the reception of the referenced data element.			
Base	ARObject,AbstractEvent,AtpClassifier,AtpFeature,AtpStructure Element,Identifiable,MultilanguageReferrable,RTEEvent,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
data	VariableDataPrototype	0..1	iref	Data element referenced by event

Table D.87: DataReceiveErrorEvent

Class	DataReceivedEvent			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::RTE Events			
Note	The event is raised when the referenced data elements are received.			
Base	ARObject,AbstractEvent,AtpClassifier,AtpFeature,AtpStructure Element,Identifiable,MultilanguageReferrable,RTEEvent,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
data	VariableDataPrototype	0..1	iref	Data element referenced by event

Table D.88: DataReceivedEvent

Class	DataSendCompletedEvent			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::RTE Events			
Note	The event is raised when the referenced data elements have been sent or an error occurs.			
Base	ARObject,AbstractEvent,AtpClassifier,AtpFeature,AtpStructure Element,Identifiable,MultilanguageReferrable,RTEEvent,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
eventSource	VariableAccess	1	ref	The variable access that triggers the event.

Table D.89: DataSendCompletedEvent

Class	DataTypeMap			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::Datatypes			
Note	This class represents the relationship between ApplicationDataType and its implementing ImplementationDataType.			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
applicationDataType	ApplicationDataType	1	ref	This is the corresponding ApplicationDataType

Attribute	Datatype	Mul.	Kind	Note
implementationDataType	ImplementationDataType	1	ref	This is the corresponding ImplementationDataType.

Table D.90: DataTypeMap

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

Table D.91: DataTypeMappingSet

Enumeration	DataTypePolicyEnum
Package	M2::AUTOSARTemplates::SystemTemplate::DataMapping
Note	This class lists the supported DataTypePolicies.
Literal	**Description**
legacy	In case the System Description doesn't use a complete Software Component Description (VFB View) this value can be chosen. This supports the inclusion of legacy signals. The aggregation of SwDataDefProps shall be used to configure the "ComSignalDataInvalidValue" and the Data Semantics.
networkRepresentationFromComSpec	Ignore any networkRepresentationProps of this ISignal and use the networkRepresentation from the ComSpec. Please note that the usage does not imply the existence of the SwDataDefProps in the role networkRepresentation aggregated by the SenderComSpec or ReceiverComSpec if an ImplementationDataType is defined.
override	If this value is chosen the requirements specified in the ComSpec (networkRepresentationFromComSpec) are not fullfilled by the aggregated SwDataDefProps. In this case the networkRepresentation is specified by the aggregated swDataDefProps.

portInterface Definition	This enumeration literal is deprecated and will be removed in future. Old description: Ignore any networkRepresentationProps of this ISignal and use the networkRepresentation specified in the VariableDataPrototypes owned by PortInterface (portInterfaceDefinition). **Tags:** atp.Status=obsolete

Table D.92: DataTypePolicyEnum

Class	**DataWriteCompletedEvent**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::RTE Events			
Note	This event is raised if an implicit write access was successful or an error occurred.			
Base	ARObject,AbstractEvent,AtpClassifier,AtpFeature,AtpStructure Element,Identifiable,MultilanguageReferrable,RTEEvent,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
eventSource	VariableAccess	1	ref	The variable access that triggers the event.

Table D.93: DataWriteCompletedEvent

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

Table D.94: DelegationSwConnector

Class	**DependencyOnArtifact**			
Package	M2::AUTOSARTemplates::CommonStructure::Implementation			
Note	Dependency on the existence of another artifact, e.g. a library.			
Base	ARObject,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
artifactDescriptor	AutosarEngineeringObject	1	aggr	The specified artifact needs to exist.
usage	DependencyUsageEnum	1..*	attr	Specification for which process step(s) this dependency is required.

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*

Table D.95: DependencyOnArtifact

Class	**EcuAbstractionSwComponentType**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	The ECUAbstraction is a special AtomicSwComponentType that resides between a software-component that wants to access ECU periphery and the Microcontroller Abstraction. The EcuAbstractionSwComponentType introduces the possibility to link from the software representation to its hardware description provided by the ECU Resource Template. **Tags:** atp.recommendedPackage=SwComponentTypes			
Base	ARElement,ARObject,AtomicSwComponentType,AtpBlueprint,AtpBlueprintable,AtpClassifier,AtpType,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable,SwComponentType			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
hardwareElement	HwDescriptionEntity	*	ref	Reference from the EcuAbstractionComponentType to the description of the used HwElements.

Table D.96: EcuAbstractionSwComponentType

Enumeration	**EcucConfigurationClassEnum**
Package	M2::AUTOSARTemplates::ECUCParameterDefTemplate
Note	Possible configuration classes for the AUTOSAR configuration parameters.
Literal	*Description*
Link	Link Time: parts of configuration are delivered from another object code file
PostBuild	PostBuildTime: the configuration parameter has to be stored at a known memory location.
PreCompile	PreCompile Time: after compilation a configuration parameter can not be changed any more.
Published Information	PublishedInformation is used to specify the fact that certain information is fixed even before the pre-compile stage.

Table D.97: EcucConfigurationClassEnum

Class	**EcucForeignReferenceDef**			
Package	M2::AUTOSARTemplates::ECUCParameterDefTemplate			
Note	Specify a reference to an XML description of an entity described in another AUTOSAR template.			
Base	ARObject,AtpDefinition,EcucAbstractReferenceDef,EcucCommonAttributes,EcucDefinitionElement,Identifiable,MultilanguageReferrable,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
destination Type	String	1	attr	The type in the AUTOSAR Metamodel to which instance this reference is allowed to point to.

Table D.98: EcucForeignReferenceDef

Class	**EcucModuleConfigurationValues**			
Package	M2::AUTOSARTemplates::ECUCDescriptionTemplate			
Note	Head of the configuration of one Module. A Module can be a BSW module as well as the RTE and ECU Infrastructure. As part of the BSW module description, the EcucModuleConfigurationValues element has two different roles: The recommendedConfiguration contains parameter values recommended by the BSW module vendor. The preconfiguredConfiguration contains values for those parameters which are fixed by the implementation and cannot be changed. These two EcucModuleConfigurationValues are used when the base EcucModuleConfigurationValues (as part of the base ECU configuration) is created to fill parameters with initial values. **Tags:** atp.recommendedPackage=EcucModuleConfigurationValuess			
Base	ARElement,ARObject,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
container	EcucContainerValue	1..*	aggr	Aggregates all containers that belong to this module configuration. atpVariation: [RS_ECUC_00078] **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=definition, shortName, variationPoint.shortLabel vh.latestBindingTime=postBuild xml.sequenceOffset=10
definition	EcucModuleDef	1	ref	Reference to the definition of this EcucModuleConfigurationValues element. Typically, this is a vendor specific module configuration. **Tags:** xml.sequenceOffset=-10
ecucDefEdition	RevisionLabelString	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.
implementationConfigVariant	EcucConfigurationVariantEnum	1	attr	Specifies the kind of deliverable this EcucModuleConfigurationValues element provides. If this element is not used in a particular role (e.g. preconfiguredConfiguration or recommendedConfiguration) then the value must be one of VariantPreCompile, VariantLinkTime, VariantPostBuild.

Attribute	Datatype	Mul.	Kind	Note
moduleDescription	BswImplementation	0..1	ref	Referencing the BSW module description, which this EcucModuleConfigurationValues element is configuring. This is optional because the EcucModuleConfigurationValues 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.

Table D.99: EcucModuleConfigurationValues

Class	EcucModuleDef			
Package	M2::AUTOSARTemplates::ECUCParameterDefTemplate			
Note	Used as the top-level element for configuration definition for Software Modules, including BSW and RTE as well as ECU Infrastructure. **Tags:** atp.recommendedPackage=EcucModuleDefs			
Base	ARElement,ARObject,AtpBlueprint,AtpBlueprintable,AtpDefinition,CollectableElement,EcucDefinitionElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
apiServicePrefix	CIdentifier	0..1	ref	For CDD modules this attribute holds the apiServicePrefix. The shortName of the module definition of a Complex Driver is always "CDD". Therefore for CDD modules the module apiServicePrefix is described with this attribute.
container	EcucContainerDef	1..*	aggr	Aggregates the top-level container definitions of this specific module definition. **Tags:** xml.sequenceOffset=11
refinedModuleDef	EcucModuleDef	0..1	ref	Optional reference from the Vendor Specific Module Definition to the Standardized Module Definition it refines. In case this EcucModuleDef has the category STANDARDIZED_MODULE_DEFINITION this reference shall not be provided. In case this EcucModuleDef has the category VENDOR_SPECIFIC_MODULE_DEFINITION this reference is mandatory. **Stereotypes:** atpUriDef
supportedConfigVariant	EcucConfigurationVariantEnum	*	attr	Specifies which ConfigurationVariants are supported by this software module. This attribute is optional if the EcucModuleDef has the category STANDARDIZED_MODULE_DEFINITION. If the category attribute of the EcucModuleDef is set to VENDOR_SPECIFIC_MODULE_DEFINITION then this attribute is mandatory.

Attribute	Datatype	Mul.	Kind	Note

Table D.100: EcucModuleDef

Class	**EcucParamConfContainerDef**			
Package	M2::AUTOSARTemplates::ECUCParameterDefTemplate			
Note	Used to define configuration containers that can hierarchically contain other containers and/or parameter definitions.			
Base	ARObject,AtpDefinition,EcucContainerDef,EcucDefinitionElement,Identifiable,MultilanguageReferrable,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
multipleConfigurationContainer	Boolean	1	attr	Specifies whether this container is used to define multiple configuration sets. Only one container in the whole EcucModuleDef shall have this enabled.
parameter	EcucParameterDef	*	aggr	The parameters defined within the EcucParamConfContainerDef.
reference	EcucAbstractReferenceDef	*	aggr	The references defined within the EcucParamConfContainerDef.
subContainer	EcucContainerDef	*	aggr	The containers defined within the EcucParamConfContainerDef.

Table D.101: EcucParamConfContainerDef

Class	**EngineeringObject (abstract)**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::EngineeringObject			
Note	This class specifies an engineering object. Usually such an object is represented by a file artifact. The properties of engineering object are such that the artifact can be found by querying an ASAM catalog file. The engineering object is uniquely identified by domain+category+shortLabel+revisionLabel.			
Base	ARObject			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
category	NameToken	1	attr	This denotes the role of the engineering object in the development cycle. Categories are such as • SWSRC for source code • SWOBJ for object code • SWHDR for a C-header file Further roles need to be defined via Methodology. **Tags:** xml.sequenceOffset=20

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
domain	NameToken	0..1	attr	This denotes the domain in which the engineering object is stored. This allows to indicate various segments in the repository keeping the engineering objects. The domain may segregate companies, as well as automotive domains. Details need to be defined by the Methodology. Attribute is optional to support a default domain. **Tags:** xml.sequenceOffset=40
revisionLabel	RevisionLabelString	*	attr	This is a revision label denoting a particular version of the engineering object. **Tags:** xml.sequenceOffset=30
shortLabel	NameToken	1	attr	This is the short name of the engineering object. Note that it is modeled as NameToken and not as Identifier since in ASAM-CC it is also a NameToken. **Tags:** xml.sequenceOffset=10

Table D.102: EngineeringObject

Class	**ExclusiveArea**			
Package	M2::AUTOSARTemplates::CommonStructure::InternalBehavior			
Note	Prevents an executable entity running in the area from being preempted.			
Base	ARObject,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table D.103: ExclusiveArea

Class	**ExecutableEntity (abstract)**			
Package	M2::AUTOSARTemplates::CommonStructure::InternalBehavior			
Note	Abstraction of executable code.			
Base	ARObject,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
activationReason	ExecutableEntityActivationReason	*	aggr	If the ExecutableEntity provides at least one activationReason element the RTE resp. BSW Scheduler shall provide means to read the activation vector of this executable entity execution. If no activationReason element is provided the feature of being able to determine the activating RTEEvent is disabled for this ExecutableEntity.
canEnterExclusiveArea	ExclusiveArea	*	ref	This means that the executable entity can enter/leave the referenced exclusive area through explicit API calls.

Attribute	Datatype	Mul.	Kind	Note
exclusiveAreaNestingOrder	ExclusiveAreaNestingOrder	*	ref	This represents the set of ExclusiveAreaNestingOrders recognized by this ExecutableEntity.
minimumStartInterval	TimeValue	1	attr	Specifies the time in seconds by which two consecutive starts of an ExecutableEntity are guaranteed to be separated.
reentrancyLevel	ReentrancyLevelEnum	0..1	attr	The reentrancy level of this ExecutableEntity. See the documentation of the enumeration type ReentrancyLevelEnum for details. Please note that nonReentrant interfaces can have also reentrant or multicoreReentrant implementations, and reentrant interfaces can also have multicoreReentrant implementations.
runsInsideExclusiveArea	ExclusiveArea	*	ref	The executable entity runs completely inside the referenced exclusive area.
swAddrMethod	SwAddrMethod	0..1	ref	Addressing method related to this code entity. Via an association to the same SwAddrMethod, it can be specified that several code entities (even of different modules or components) shall be located in the same memory without already specifying the memory section itself.

Table D.104: ExecutableEntity

Class	ExecutableEntityActivationReason			
Package	M2::AUTOSARTemplates::CommonStructure::InternalBehavior			
Note	This meta-class represents the ability to define the reason for the activation of the enclosing ExecutableEntity.			
Base	ARObject,ImplementationProps,Referrable			
Attribute	Datatype	Mul.	Kind	Note
bitPosition	PositiveInteger	1	attr	This attribute allows for defining the position of the enclosing ExecutableEntityActivationReason in the activation vector.

Table D.105: ExecutableEntityActivationReason

Class	ExternalTriggerOccurredEvent			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::RTEEvents			
Note	The event is raised when the referenced trigger have been occurred.			
Base	ARObject,AbstractEvent,AtpClassifier,AtpFeature,AtpStructureElement,Identifiable,MultilanguageReferrable,RTEEvent,Referrable			
Attribute	Datatype	Mul.	Kind	Note
trigger	Trigger	0..1	iref	Reference to the applicable Trigger.

Table D.106: ExternalTriggerOccurredEvent

Class	ExternalTriggeringPoint			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::Trigger			
Note	If a RunnableEntity owns an ExternalTriggeringPoint it is entitled to raise an ExternalTriggerOccurredEvent.			
Base	ARObject			
Attribute	Datatype	Mul.	Kind	Note
ident	ExternalTriggeringPointIdent	0..1	aggr	The aggregation in the role ident provides the ability to make the ExternalTriggeringPoint identifiable. From the semantical point of view, the ExternalTriggeringPoint is considered a first-class Identifiable and therefore the aggregation in the role ident shall always exist (until it may be possible to let ModeAccessPoint directly inherit from Identifiable). **Tags:** atp.Status=shallBecomeMandatory xml.sequenceOffset=-100
trigger	Trigger	0..1	iref	The trigger taken for the ExternalTriggeringPoint. **Tags:** xml.namePlural=TRIGGER-IREF; xml.roleElement=false; xml.roleWrapperElement=true; xml.typeElement=true; xml.typeWrapperElement=false

Table D.107: ExternalTriggeringPoint

Class	FlatInstanceDescriptor			
Package	M2::AUTOSARTemplates::CommonStructure::FlatMap			
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.			
Base	ARObject,Identifiable,MultilanguageReferrable,Referrable			
Attribute	Datatype	Mul.	Kind	Note

Attribute	Datatype	Mul.	Kind	Note
ecuExtractReference	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 AtomicSoftwareComponentType, which is refered by the particular SwcInternalBehavior. **Tags:** xml.sequenceOffset=40
role	Identifier	0..1	ref	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. ModeDeclarationGroupPrototypes which are measurable. In this case the RTE will provide locations for current mode, previous mode and next mode.
swDataDefProps	SwDataDefProps	0..1	aggr	The properties of this FlatInstanceDescriptor.
upstreamReference	AtpFeature	0..1	iref	Refers to the instance in the context of an "upstream" descriptions, wich could be the system or system extract description, the basic software module description 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 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 the instance of the component prototype that contains the particular instance of SwcInternalBehavior. **Tags:** xml.sequenceOffset=20

Table D.108: FlatInstanceDescriptor

Class	**FlatMap**			
Package	M2::AUTOSARTemplates::CommonStructure::FlatMap			
Note	Contains a flat list of references to software objects. This list is used to identify instances and to resolve name conflicts. The scope is given by the RootSwCompositionPrototype for which it is used, i.e. it can be applied to a system, system extract or ECU-extract. An instance of FlatMap may also be used in a preliminary context, e.g. in the scope of a software component before integration into a system. In this case it is not referred by a RootSwCompositionPrototype. **Tags:** atp.recommendedPackage=FlatMaps			
Base	ARElement,ARObject,AtpBlueprint,AtpBlueprintable,Collectable Element,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
instance	FlatInstanceDes criptor	1..*	aggr	A descriptor instance aggregated in the flat map. The variation point accounts for the fact, that the system in scope can be subject to variability, and thus the existence of some instances is variable. The aggregation has been made splitable because the content might be contributed by different stakeholders at different times in the workflow. Plus, the overall size might be so big that eventually it becomes more manageable if it is distributed over several files. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variation Point.shortLabel vh.latestBindingTime=postBuild

Table D.109: FlatMap

Enumeration	**HandleInvalidEnum**
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication
Note	Strategies of handling the reception of invalidValue.
Literal	**Description**
dontInvali- date	Invalidation is switched off.
keep	The application software is supposed to handle signal invalidation on RTE API level either by DataReceiveErrorEvent or check of error code on read access.
replace	Replace a received invalidValue. The replacement value is specified by the initValue.

Table D.110: HandleInvalidEnum

Enumeration	**HandleOutOfRangeEnum**
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication

Note	A value of this type is taken for controlling the range checking behavior of the AUTOSAR RTE.
Literal	***Description***
default	The RTE will use the initValue if the actual value is out of the specified bounds.
external Replacement	This indicates that the value replacement is sourced from the externalReplacement.
ignore	The RTE will ignore any attempt to send or receive the corresponding dataElement if the value is out of the specified range.
invalid	The RTE will use the invalidValue if the value is out of the specified bounds.
none	A range check is not required.
saturate	The RTE will saturate the value of the dataElement such that it is limited to the applicable upper bound if it is greater than the upper bound. Consequently, it is limited to the applicable lower bound if the value is less than the lower bound.

Table D.111: HandleOutOfRangeEnum

Enumeration	**HandleOutOfRangeStatusEnum**
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication
Note	This enumeration defines how the RTE handles values that are out of range.
Literal	***Description***
indicate	The RTE sets the return status to RTE_E_OUT_OF_RANGE if the received value is out of range and the attribute handleOutOfRange is not set to "none" or "invalid".
silent	The RTE sets the return status to RTE_E_OK

Table D.112: HandleOutOfRangeStatusEnum

Class	**HwElement**			
Package	M2::AUTOSARTemplates::EcuResourceTemplate			
Note	This represents the ability to describe Hardware Elements on an instance level. The particular types of hardware are distinguished by the category. This category determines the applicable attributes. The possible categories and attributes are defined in HwCategory. **Tags:** atp.recommendedPackage=HwElements			
Base	ARElement,ARObject,CollectableElement,HwDescription Entity,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
hwElement Connection	HwElementConnector	*	aggr	This represents one particular connection between two hardware elements. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=systemDesignTime xml.sequenceOffset=110

Attribute	Datatype	Mul.	Kind	Note
hwPinGroup	HwPinGroup	*	aggr	This aggregation is used to describe the connection facilities of a hardware element. Note that hardware element has no pins but only pingroups. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=systemDesignTime xml.sequenceOffset=90
nestedElement	HwElement	*	ref	This association is used to establish hierarchies of hw elements. Note that one particular HwElement can be target of this association only once. I.e. multiple instantiation of the same HwElement is not supported (at any hierarchy level). **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=systemDesignTime xml.sequenceOffset=70

Table D.113: HwElement

Class	ISignal			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::FibexCore::CoreCommunication			
Note	Signal of the Interaction Layer. The RTE supports a "signal fan-out" where the same System Signal is sent in different SignalIPdus to multiple receivers. To support the RTE "signal fan-out" each SignalIPdu contains ISignals. If the same System Signal is to be mapped into several SignalIPdus there is one ISignal needed for each ISignalToIPduMapping. ISignals describe the Interface between the Precompile configured RTE and the potentially Postbuild configured Com Stack (see ECUC Parameter Mapping). In case of the SystemSignalGroup an ISignal must be created for each SystemSignal contained in the SystemSignalGroup. **Tags:** atp.recommendedPackage=ISignals			
Base	ARObject,CollectableElement,FibexElement,Identifiable,Multilanguage Referrable,PackageableElement,Referrable			
Attribute	Datatype	Mul.	Kind	Note

Attribute	Datatype	Mul.	Kind	Note
dataTypePolicy	DataTypePolicyEnum	1	attr	With the aggregation of SwDataDefProps an ISignal specifies how it is represented on the network. This representation follows a particular policy. Note that this causes some redundancy which is intended and can be used to support flexible development methodology as well as subsequent integrity checks. If the policy "networkRepresentationFromComSpec" is chosen the network representation from the ComSpec that is aggregated by the PortPrototype shall be used. If the "override" policy is chosen the requirements specified in the PortInterface and in the ComSpec are not fulfilled by the networkRepresentationProps. In case the System Description doesn't use a complete Software Component Description (VFB View) the "legacy" policy can be chosen.
iSignalProps	ISignalProps	0..1	aggr	Additional optional ISignal properties that may be stored in different files. **Stereotypes:** atpSplitable
initValue	ValueSpecification	0..1	aggr	Optional definition of a ISignal's initValue in case the System Description doesn't use a complete Software Component Description (VFB View). This supports the inclusion of legacy system signals. This value can be used to configure the Signal's "InitValue". If a full DataMapping exist for the SystemSignal this information may be available from a configured SenderComSpec and ReceiverComSpec. In this case the initvalues in SenderComSpec and/or ReceiverComSpec override this optional value specification. Further restrictions apply from the RTE specification.
length	Integer	1	attr	Size of the signal in bits. The size needs to be derived from the mapped VariableDataPrototype according to the mapping of primitive DataTypes to BaseTypes as used in the RTE. Indicates maximum size for dynamic length signals. The ISignal length of zero bits is allowed.

Attribute	Datatype	Mul.	Kind	Note
networkRepresentationProps	SwDataDefProps	0..1	aggr	Specification of the actual network representation. The usage of SwDataDefProps for this purpose is restricted to the attributes compuMethod and baseType. The optional baseType attributes "memAllignment" and "byteOrder" shall not be used. The attribute "dataTypePolicy" in the SystemTemplate element defines whether this network representation shall be ignored and the information shall be taken over from the network representation of the ComSpec. If "override" is chosen by the system integrator the network representation can violate against the requirements defined in the PortInterface and in the network representation of the ComSpec. In case that the System Description doesn't use a complete Software Component Description (VFB View) this element is used to configure "ComSignalDataInvalidValue" and the Data Semantics.
systemSignal	SystemSignal	1	ref	Reference to the System Signal that is supposed to be transmitted in the ISignal.

Table D.114: ISignal

Class	**ISignalGroup**			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::FibexCore::CoreCommunication			
Note	SignalGroup of the Interaction Layer. The RTE supports a "signal fan-out" where the same System Signal Group is sent in different SignalIPdus to multiple receivers. An ISignalGroup refers to a set of ISignals that shall always be kept together. A ISignalGroup represents a COM Signal Group. Therefore it is recommended to put the ISignalGroup in the same Package as ISignals (see atp.recommendedPackage) **Tags:** atp.recommendedPackage=ISignalGroup			
Base	ARObject,CollectableElement,FibexElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
iSignal	ISignal	*	ref	Reference to a set of ISignals that shall always be kept together.
systemSignalGroup	SystemSignalGroup	1	ref	Reference to the SystemSignalGroup that is defined on VFB level and that is supposed to be transmitted in the ISignalGroup.

Table D.115: ISignalGroup

Class	ISignalProps			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::FibexCore::CoreCommunication			
Note	Additional ISignal properties that may be stored in different files.			
Base	ARObject			
Attribute	Datatype	Mul.	Kind	Note
handleOutOfRange	HandleOutOfRangeEnum	1	attr	This attribute defines the outOfRangeHandling for received and sent signals.

Table D.116: ISignalProps

Class	Identifiable (abstract)			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Identifiable			
Note	Instances of this class can be referred to by their identifier (within the namespace borders). In addition to this, Identifiables are objects which contribute significantly to the overall structure of an AUTOSAR description. In particular, Identifiables might contain Identifiables.			
Base	ARObject,MultilanguageReferrable,Referrable			
Attribute	Datatype	Mul.	Kind	Note
desc	MultiLanguageOverviewParagraph	0..1	aggr	This represents a general but brief (one paragraph) description what the object in question is about. It is only one paragraph! Desc is intended to be collected into overview tables. This property helps a human reader to identify the object in question. More elaborate documentation, (in particular how the object is built or used) should go to "introduction". **Tags:** xml.sequenceOffset=-60
category	CategoryString	0..1	attr	This element assigns a category to the parent element. The category is intended to specialize the usage and/or the content identifiable object. Such a specialization may also impose particular semantic constraints on the entire substructure (not only the identifiable itself). **Tags:** xml.sequenceOffset=-50
adminData	AdminData	0..1	aggr	This represents the administrative data for the identifiable object. **Tags:** xml.sequenceOffset=-40
annotation	Annotation	*	aggr	Possibility to provide additional notes while defining a model element (e.g. the ECU Configuration Parameter Values). These are not intended as documentation but are mere design notes. **Tags:** xml.sequenceOffset=-25

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
introduction	DocumentationBlock	0..1	aggr	This represents more information about how the object in question is built or is used. Therefore it is a DocumentationBlock. **Tags:** xml.sequenceOffset=-30
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". **Tags:** xml.attribute=true

Table D.117: Identifiable

Class	**Implementation (abstract)**			
Package	M2::AUTOSARTemplates::CommonStructure::Implementation			
Note	Description of an implementation a single software component or module.			
Base	ARElement,ARObject,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
buildActionManifest	BuildActionManifest	0..1	ref	A manifest specifying the intended build actions for the software delivered with this implementation. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=codeGenerationTime
codeDescriptor	Code	1..*	aggr	Specifies the provided implementation code.
compiler	Compiler	*	aggr	Specifies the compiler for which this implementation has been released

Attribute	Datatype	Mul.	Kind	Note
generatedArtifact	DependencyOnArtifact	*	aggr	Relates to an artifact that will be generated during the integration of this Implementation by an associated generator tool. Note that this is an optional information since it might not always be in the scope of a single module or component to provide this information. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
hwElement	HwElement	*	ref	The hardware elements (e.g. the processor) required for this implementation.
linker	Linker	*	aggr	Specifies the linker for which this implementation has been released.
mcSupport	McSupportData	0..1	aggr	The measurement & calibration support data belonging to this implementation. The aggregtion is «atpSplitable» because in case of an already exisiting BSW Implementation model, this description will be added later in the process, namely at code generation time. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=mcSupport
programmingLanguage	ProgramminglanguageEnum	1	attr	Programming language the implementation was created in.
requiredArtifact	DependencyOnArtifact	*	aggr	Specifies that this Implementation depends on the existance of another artifact (e.g. a library). This aggregation of DependencyOnArtifact is subject to variability with the purpose to support variability in the implementations. Different algorithms in the implementation might cause different dependencies, e.g. the number of used libraries. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
requiredGeneratorTool	DependencyOnArtifact	*	aggr	Relates this Implementation to a generator tool in order to generate additional artifacts during integration. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
resourceConsumption	ResourceConsumption	1	aggr	All static and dynamic resources for each implementation are described within the ResourceConsumption class.
swVersion	RevisionLabelString	1	attr	Software version of this implementation. The numbering contains three levels (like major, minor, patch), its values are vendor specific.

Attribute	Datatype	Mul.	Kind	Note
swcBswMapping	SwcBswMapping	0..1	ref	This allows a mapping between an SWC and a BSW behavior to be attached to an implementation description (for AUTOSAR Service, ECU Abstraction and Complex Driver Components). It is up to the methodology to define whether this reference has to be set for the Swc- or BswImplementtion or for both.
usedCodeGenerator	String	0..1	attr	Optional: code generator used.
vendorId	PositiveInteger	1	attr	Vendor ID of this Implementation according to the AUTOSAR vendor list

Table D.118: Implementation

Class	ImplementationDataType			
Package	M2::AUTOSARTemplates::CommonStructure::ImplementationDataTypes			
Note	Describes a reusable data type on the implementation level. This will typically correspond to a typedef in C-code. **Tags:** atp.recommendedPackage=ImplementationDataTypes			
Base	ARElement,ARObject,AtpBlueprint,AtpBlueprintable,AtpClassifier,AtpType,AutosarDataType,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
subElement (ordered)	ImplementationDataTypeElement	*	aggr	Specifies an element of an arrray, struct, or union data type. The aggregation of ImplementionDataTypeElement is subject to variability with the purpose to support the conditional existence of elements inside a ImplementationDataType representing a structure. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
symbolProps	SymbolProps	0..1	aggr	This represents the SymbolProps for the ImplementationDataType. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=shortName
typeEmitter	NameToken	0..1	attr	This attribute is used to control which part of the AUTOSAR toolchain is supposed to trigger data type definitions.

Table D.119: ImplementationDataType

Class	**ImplementationDataTypeElement**			
Package	M2::AUTOSARTemplates::CommonStructure::ImplementationDataTypes			
Note	Declares a data object which is locally aggregated. Such an element can only be used within the scope where it is aggregated. This element either consists of further subElements or it is further defined via its swDataDefProps. There are several use cases within the system of ImplementationDataTypes fur such a local declaration: • It can represent the elements of an array, defining the element type and array size • It can represent an element of a struct, defining its type • It can be the local declaration of a debug element.			
Base	ARObject,Identifiable,MultilanguageReferrable,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
arraySize	PositiveInteger	0..1	attr	The existence of this attributes (if bigger than 0) defines the size of an array and declares that this ImplementationDataTypeElement represents the type of each single array element. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
arraySizeSemantics	ArraySizeSemanticsEnum	0..1	attr	This attribute controls the meaning of the value of the array size.
subElement	ImplementationDataTypeElement	*	aggr	Element of an array, struct, or union in case of a nested declaration (i.e. without using "typedefs"). The aggregation of ImplementionDataTypeElement is subject to variability with the purpose to support the conditional existence of elements inside a ImplementationDataType representing a structure. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
swDataDefProps	SwDataDefProps	0..1	aggr	The properties of this ImplementationDataTypeElementt.

Table D.120: ImplementationDataTypeElement

Class	**ImplementationProps (abstract)**			
Package	M2::AUTOSARTemplates::CommonStructure::Implementation			
Note	Defines a symbol to be used as (depending on the concrete case) either a complete replacement or a prefix when generating code artifacts.			
Base	ARObject,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
symbol	CIdentifier	1	ref	The symbol to be used as (depending on the concrete case) either a complete replacement or a prefix.

Attribute	***Datatype***	***Mul.***	***Kind***	***Note***

Table D.121: ImplementationProps

Class	**IncludedDataTypeSet**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::IncludedDataTypes			
Note	An includedDataTypeSet declares that a set of AutosarDataType is used by the software component for its implementation and the AutosarDataType becomes part of the contract. This information is required if the AutosarDataType is not used for any DataPrototype owned by this software component or if the enumeration literals, lowerLimit and upperLimit constants shall be generated with a literalPrefix. The optional literalPrefix is used to add a common prefix on enumeration literals, lowerLimit and upperLimit constants created by the RTE.			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
dataType	AutosarDataType	1..*	ref	AutosarDataType belonging to the includedDataTypeSet
literalPrefix	Identifier	0..1	ref	LiteralPrefix defines a common prefix for all AutosarDataTypes of the includedDataTypeSet to be added on enumeration literals, lowerLimit and upperLimit constants created by the RTE.

Table D.122: IncludedDataTypeSet

Class	**IncludedModeDeclarationGroupSet**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::ModeDeclarationGroup			
Note	An IncludedModeDeclarationGroupSet declares that a set of ModeDeclarationGroups used by the software component for its implementation and consequently these ModeDeclarationGroups become part of the contract.			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
modeDeclarationGroup	ModeDeclarationGroup	1..*	ref	This represents the referenced ModeDeclarationGroup.
prefix	Identifier	0..1	ref	The prefix shall be used by the RTE generator as a prefix for the creation of symbols related to the referenced ModeDeclarationGroups, e.g RTE_TRANSITION_<ModeDeclarationGroup>.

Table D.123: IncludedModeDeclarationGroupSet

Class	InitEvent			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::RTE Events			
Note	This RTEEvent is supposed to be used for initialization purposes, i.e. for starting and restarting a partition. It is not guaranteed that all RunnableEntities referenced by this InitEvent are executed before the 'regular' RunnableEntities are executed for the first time. The execution order depends on the task mapping.			
Base	ARObject,AbstractEvent,AtpClassifier,AtpFeature,AtpStructure Element,Identifiable,MultilanguageReferrable,RTEEvent,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
–	–	–	–	–

Table D.124: InitEvent

Class	InstantiationDataDefProps			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior:: 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 DataPrototypes (especially VariableDataPrototypes, ParameterDataPrototypes). 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 VariableDataPrototypes and ParameterDataPrototypes to guarantee type compatibility on C-implementation level (e.g. data of a Port is stored in PIM or NvRom Block shall be from same data type as NvRAM Block). This class overcomes such a restriction if applied properly.			
Base	ARObject			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
parameterInstance	AutosarParameterRef	0..1	aggr	This is the particular ParameterDataPrototypes on which the swDataDefProps shall be applied.
swDataDefProps	SwDataDefProps	1	aggr	These are the particular data definition properties which shall be applied
variableInstance	AutosarVariableRef	0..1	aggr	This is the particular VariableDataPrototypes on which the swDataDefProps shall be applied.

Table D.125: InstantiationDataDefProps

Class	InstantiationRTEEventProps (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Composition			
Note	This meta class represents the ability to refine the properties of RTEEvents for particular instances of a software component.			
Base	ARObject			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**

Attribute	Datatype	Mul.	Kind	Note
refinedEvent	RTEEvent	1	iref	This instance ref denotes the Timing Event for which the period shall be refined on an instance level.
shortLabel	Identifier	1	ref	The main purpose of the shortLabel is to contribute to the splitkey of aggregations that are «atpSplitable».

Table D.126: InstantiationRTEEventProps

Class	InternalBehavior (abstract)			
Package	M2::AUTOSARTemplates::CommonStructure::InternalBehavior			
Note	Common base class (abstract) for the internal behavior of both software components and basic software modules/clusters.			
Base	ARObject,AtpClassifier,AtpFeature,AtpStructureElement,Identifiable,MultilanguageReferrable,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
constantMemory	ParameterDataPrototype	*	aggr	Describes a read only memory object containing characteristic value(s) implemented by this InternalBehavior. The shortName of ParameterElementPrototype has to be equal to the "C' identifier of the described constant. The characteristic value(s) might be shared between SwComponentPrototypes 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:** vh.latestBindingTime=preCompileTime
constantValueMapping	ConstantSpecificationMappingSet	*	ref	Reference to the ConstanSpecificationMapping to be applied for the particular InternalBehavior
dataTypeMapping	DataTypeMappingSet	*	ref	Reference to the DataTypeMapping to be applied for the particular InternalBehavior
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:** vh.latestBindingTime=preCompileTime

Attribute	Datatype	Mul.	Kind	Note
exclusiveAreaNestingOrder	ExclusiveAreaNestingOrder	*	aggr	This represents the set of ExclusiveAreaNestingOrder owned by the InternalBehavior. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=preCompileTime
staticMemory	VariableDataPrototype	*	aggr	Describes a read and writeable static memory object representing measurment variables implemented by this software component. 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 DataElementPrototype 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 components implementations. Typically different algorithms in the implementation are requiring different number of memory objects. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime

Table D.127: InternalBehavior

Class	InternalTriggerOccurredEvent			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::RTEEvents			
Note	The event is raised when the referenced internal trigger have been occurred.			
Base	ARObject,AbstractEvent,AtpClassifier,AtpFeature,AtpStructureElement,Identifiable,MultilanguageReferrable,RTEEvent,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
eventSource	InternalTriggeringPoint	1	ref	Internal Triggering Point that triggers the event.

Table D.128: InternalTriggerOccurredEvent

Class	InternalTriggeringPoint			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::Trigger			
Note	If a RunnableEntity owns a InternalTriggeringPoint it is entitled to trigger the execution of RunnableEntities of the corresponding software-component.			
Base	ARObject,AtpClassifier,AtpFeature,AtpStructureElement,Identifiable,MultilanguageReferrable,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
swImplPolicy	SwImplPolicyEnum	0..1	attr	This attribute, when set to value queued, allows for a queued processing of Triggers.

Attribute	***Datatype***	***Mul.***	***Kind***	***Note***

Table D.129: InternalTriggeringPoint

Class	**InvalidationPolicy**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	Specifies whether the component can actively invalidate a particular dataElement. If no invalidationPolicy points to a dataElement this is considered to yield the identical result as if the handleInvalid attribute was set to dontInvalidate.			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
dataElement	VariableDataPrototype	1	ref	Reference to the dataElement for which the InvalidationPolicy applies.
handleInvalid	HandleInvalidEnum	0..1	attr	This attribute defines the action performed upon a reception timeout violation.

Table D.130: InvalidationPolicy

Class	**McDataInstance**			
Package	M2::AUTOSARTemplates::CommonStructure::MeasurementCalibrationSupport			
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			
Attribute	***Datatype***	***Mul.***	***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.

Attribute	Datatype	Mul.	Kind	Note
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.
instanceInMemory	ImplementationElementInParameterInstanceRef	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.
mcDataAccessDetails	McDataAccessDetails	0..1	aggr	Refers to "upstream" information on how the RTE uses this data instance. Use Case: Rapid Prototyping
mcDataAssignment	RoleBasedMcDataAssignment	*	aggr	An assignment between McDataInstances.
resultingProperties	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.
role	Identifier	0..1	ref	An optional attribute to be used for additional information on the role of this data instance, for example in the context of rapid prototyping.
subElement	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

Attribute	Datatype	Mul.	Kind	Note
symbol	SymbolString	0..1	ref	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 D.131: McDataInstance

Class	McParameterElementGroup			
Package	M2::AUTOSARTemplates::CommonStructure::MeasurementCalibrationSupport			
Note	Denotes a group of calibration parameters which are handled by the RTE as one data structure.			
Base	ARObject			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
ramLocation	VariableDataPrototype	1	ref	Refers to the RAM location of this parameter group. To be used for the init-RAM method.
romLocation	ParameterDataPrototype	1	ref	Refers to the ROM location of this parameter group. To be used for the init-RAM method.
shortLabel	Identifier	1	ref	Assigns a name to this element. **Tags:** xml.sequenceOffset=-100

Table D.132: McParameterElementGroup

Class	**McSupportData**			
Package	M2::AUTOSARTemplates::CommonStructure::MeasurementCalibrationSupport			
Note	Root element for all measurement and calibration support data related to one Implementation artifact on an ECU. There shall be one such element related to the RTE implementation (if it owns MC data) and a separate one for each module or component, which owns private MC data.			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
emulationSupport	McSwEmulationMethodSupport	*	aggr	Describes the calibration method used by the RTE. This information is not needed for A2L generation, but to setup software emulation in the ECU. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
mcParameterInstance	McDataInstance	*	aggr	A data instance to be used for calibration. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=postBuild
mcVariableInstance	McDataInstance	*	aggr	A data instance to be used for measurement. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=postBuild
measurableSystemConstantValues	SwSystemconstantValueSet	*	ref	Sets of system constant values to be transferred to the MCD system, because the system constants have been specified with "swCalibrationAccess" = readonly.

Table D.133: McSupportData

Class	**McSwEmulationMethodSupport**			
Package	M2::AUTOSARTemplates::CommonStructure::MeasurementCalibrationSupport			
Note	This denotes the method used by the RTE to handle the calibration data. It is published by the RTE generator and can be used e.g. to generate the corresponding emulation method in a Complex Driver. According to the actual method given by the category attribute, not all attributes are always needed: • double pointered method: only baseReference is mandatory • single pointered method: only referenceTable is mandatory • initRam method: only elementGroup(s) are mandatory Note: For single/double pointered method the group locations are implicitly accessed via the reference table and their location can be found from the initial values in the M1 model of the respective pointers. Therefore, the description of elementGroups is not needed in these cases. Likewise, for double pointered method the reference table description can be accessed via the M1 model under baseReference.			
Base	ARObject			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
category	Identifier	1	ref	Identifies the actual method. The possible names shall correspond to the symbols of the ECU configuration parameter for the calibration method of the RTE, and can include vendor specific methods. **Tags:** xml.sequenceOffset=-90
baseReference	VariableDataPrototype	0..1	ref	Refers to the base pointer in case of the double-pointered method.
elementGroup	McParameterElementGroup	*	aggr	Denotes the grouping of calibration parameters in the actual RTE code. Depending on the category, this information maybe required to set up the emulation code.
referenceTable	VariableDataPrototype	0..1	ref	Refers to the pointer table in case of the single-pointered method.
shortLabel	Identifier	1	ref	Assigns a name to this element. **Tags:** xml.sequenceOffset=-100

Table D.134: McSwEmulationMethodSupport

Enumeration	**MemoryAllocationKeywordPolicyType**
Package	M2::AUTOSARTemplates::CommonStructure::AuxillaryObjects
Note	Enumeration to specify the name pattern of the Memory Allocation Keyword.
Literal	*Description*
addrMethodShortName	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.

addrMethodShortNameAndAlignment	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 the alignment attribute of the MemorySection. This requests a separation of objects in memory dependent from the alignment and is not applicable for SwAddrMethods referred by RunnableEntitys and BswSchedulableEntitys.

Table D.135: MemoryAllocationKeywordPolicyType

Class	**MemorySection**			
Package	M2::AUTOSARTemplates::CommonStructure::ResourceConsumption::MemorySectionUsage			
Note	Provides a description of an abstract memory section used in the Implementation for code or data. It shall be declared by the Implementation Description of the module or component, which actually allocates the memory in its code. This means in case of data prototypes which are allocated by the RTE, that the generated Implementation Description of the RTE shall contain the corresponding MemorySections. The attribute "symbol" (if symbol is missing: "shortName") defines the module or component specific section name used in the code. For details see the document "Specification of Memory Mapping". Typically the section name is build according the pattern: <SwAddrMethod shortName>[_<further specialization nominator>][_<alignment>] where • **[<SwAddrMethod shortName>]** is the shortName of the referenced SwAddrMethod • **[_<further specialization nominator>]** is an optional infix to indicate the specialization in the case that several MemorySections for different purpose of the same Implementation Description referring to the same or equally named SwAddrMethods. • **[_<alignment>]** is the alignment attributes value and is only applicable in the case that the memoryAllocationKeywordPolicy value of the referenced SwAddrMethod is set to addrMethodShortNameAndAlignment MemorySection used to Implement the code of RunnableEntitys and BswSchedulableEntitys shall have a symbol (if missing: shortName) identical to the referred SwAddrMethod to conform to the generated RTE header files. In addition to the section name described above, a prefix is used in the corresponding macro code in order to define a name space. This prefix is by default given by the shortName of the BswModuleDescription resp. the SwcComponentType. It can be superseded by the prefix attribute.			
Base	ARObject, Identifiable, MultilanguageReferrable, Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
alignment	AlignmentType	0..1	attr	The attribute describes the alignment of objects within this memory section.

Attribute	Datatype	Mul.	Kind	Note
executable Entity	ExecutableEntity	*	ref	Reference to the ExecutableEntitites located in this section. This allows to locate different ExecutableEntitities in different sections even if the associated SwAddrmethod is the same. This is applicable to code sections only.
memClass Symbol	CIdentifier	0..1	ref	Defines a specific symbol in order to generate the compiler abstraction "memclass" code for this MemorySection. The existence of this attribute supersedes the usage of swAddrmethod.shortName for this purpose. The complete name of the "memclass" preprocessor symbol is constructed as <prefix>_<memClassSymbol> where prefix is defined in the same way as for the enclosing MemorySection. See also AUTOSAR_SWS_CompilerAbstraction SWS_COMPILER_00040.
option	Identifier	*	ref	This attribute introduces the ability to specify further intended properties of this MemorySection. The following two values are standardized (to be used for code sections only and exclusively to each other): • INLINE - The code section is declared with the compiler abstraction macro INLINE. • LOCAL_INLINE - The code section is declared with the compiler abstraction macro LOCAL_INLINE In both cases (INLINE and LOCAL_INLINE) the inline expansion depends on the compiler specific implementation of these macros. Depending on this, the code section either corresponds to an actual section in memory or is put into the section of the caller. See AUTOSAR_SWS_CompilerAbstraction for more details.
prefix	SectionNamePrefix	0..1	ref	The prefix used to set the memory section's namespace in the code. The existence of a prefix element supersedes rules for a default prefix (such as the BswModuleDescription's shortName). This allows the user to define several name spaces for memory sections within the scope of one module, cluster or SWC.
size	PositiveInteger	0..1	attr	The size in bytes of the section.

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
swAddrmethod	SwAddrMethod	1	ref	This association indicates that this module specific (abstract) memory section is part of an overall SwAddrMethod, referred by the upstream declarations (e.g. calibration parameters, data element prototypes, code entities) which share a common addressing strategy. This can be evaluated for the ECU configuration of the build support. This association shall always be declared by the Implementation description of the module or component, which allocates the memory in its code. This means in case of data prototypes which are allocated by the RTE, that the software components only declare the grouping of its data prototypes to SwAddrMethods, and the generated Implementation Description of the RTE actually sets up this association.
symbol	Identifier	0..1	ref	Defines the section name as explained in the main description. By using this attribute for code generation (instead of the shortName) it is possible to define several different MemorySections having the same name - e.g. symbol = CODE - but using different sectionNamePrefixes.

Table D.136: MemorySection

Class	**ModeAccessPoint**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::ModeDeclarationGroup			
Note	A ModeAccessPoint is required by a RunnableEntity owned by a Mode Manager or Mode User. Its semantics implies the ability to access the current mode (provided by the RTE) of a ModeDeclarationGroupPrototype's ModeDeclarationGroup.			
Base	ARObject			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
ident	ModeAccessPointIdent	0..1	aggr	The aggregation in the role ident provides the ability to make the ModeAccessPoint identifiable. From the semantical point of view, the ModeAccessPoint is considered a first-class Identifiable and therefore the aggregation in the role ident shall always exist (until it may be possible to let ModeAccessPoint directly inherit from Identifiable). **Tags:** atp.Status=shallBecomeMandatory xml.sequenceOffset=-100
modeGroup	ModeDeclarationGroupPrototype	0..1	iref	The mode declaration group that is accessed by this runnable. **Tags:** xml.typeElement=true

Attribute	Datatype	Mul.	Kind	Note

Table D.137: ModeAccessPoint

Enumeration	ModeActivationKind
Package	M2::AUTOSARTemplates::CommonStructure::ModeDeclaration
Note	Kind of mode switch condition used for activation of an event, as further described for each enumeration field.
Literal	**Description**
onEntry	On entering the referred mode.
onExit	On exiting the referred mode.
onTransition	On transition of the 1st referred mode to the 2nd referred mode.

Table D.138: ModeActivationKind

Class	ModeDeclaration			
Package	M2::AUTOSARTemplates::CommonStructure::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,Multilanguage Referrable,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
value	PositiveInteger	0..1	attr	The RTE shall take the value of this attribute for generating the source code representation of this ModeDeclaration.

Table D.139: ModeDeclaration

Class	ModeDeclarationGroup			
Package	M2::AUTOSARTemplates::CommonStructure::ModeDeclaration			
Note	A collection of Mode Declarations. Also, the initial mode is explicitly identified. **Tags:** atp.recommendedPackage=ModeDeclarationGroups			
Base	ARElement,ARObject,AtpBlueprint,AtpBlueprintable,AtpClassifier,Atp Type,CollectableElement,Identifiable,MultilanguageReferrable,Packageable Element,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
initialMode	ModeDeclaratio n	1	ref	The initial mode of the ModeDeclarationGroup. This mode is active before any mode switches occurred.
modeDecl aration	ModeDeclaratio n	1..*	aggr	The ModeDeclarations collected in this ModeDeclarationGroup. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=blueprintDerivation Time

Attribute	Datatype	Mul.	Kind	Note
modeManagerErrorBehavior	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).
modeTransition	ModeTransition	*	aggr	This represents the avaliable ModeTransitions of the ModeDeclarationGroup
modeUserErrorBehavior	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).
onTransitionValue	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.

Table D.140: ModeDeclarationGroup

Class	**ModeDeclarationGroupPrototype**			
Package	M2::AUTOSARTemplates::CommonStructure::ModeDeclaration			
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			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
swCalibrationAccess	SwCalibrationAccessEnum	0..1	attr	This allows for specifying whether or not the enclosing ModeDeclarationGroupPrototype can be measured at run-time.
type	ModeDeclarationGroup	1	tref	The "collection of ModeDeclarations" (= ModeDeclarationGroup) supported by a component **Stereotypes:** isOfType

Table D.141: ModeDeclarationGroupPrototype

Class	**ModeDeclarationMappingSet**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	This meta-class implements a container for ModeDeclarationGroupMappings **Tags:** atp.recommendedPackage=PortInterfaceMappingSets			
Base	ARElement,ARObject,AtpClassifier,AtpType,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
modeDeclarationMapping	ModeDeclarationMapping	1..*	aggr	This represents the collection of ModeDeclarationMappings owned by the enclosing ModeDeclarationMappingSet.

Table D.142: ModeDeclarationMappingSet

Class	**ModeErrorBehavior**			
Package	M2::AUTOSARTemplates::CommonStructure::ModeDeclaration			
Note	This represents the ability to define the error behavior in the context of mode handling.			
Base	ARObject			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
defaultMode	ModeDeclaration	0..1	ref	This represents the ModeDeclaration that is considered the error mode in the context of the enclosing ModeDeclarationGroup.
errorReactionPolicy	ModeErrorReactionPolicyEnum	1	attr	This represents the ability to define the policy in terms of which default model shall apply in case an error occurs.

Table D.143: ModeErrorBehavior

Enumeration	**ModeErrorReactionPolicyEnum**
Package	M2::AUTOSARTemplates::CommonStructure::ModeDeclaration
Note	This represents the ability to specify the reaction on a mode error.
Literal	**Description**
defaultMode	This represents the ability to switch to the defaultMode in case of a mode error.
lastMode	This represents the ability to keep the last mode in case of a mode error.

Table D.144: ModeErrorReactionPolicyEnum

Class	**ModeInterfaceMapping**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	Defines the mapping of ModeDeclarationGroupPrototypes in context of two different ModeInterfaces.			
Base	ARObject,AtpBlueprint,AtpBlueprintable,Identifiable,MultilanguageReferrable,PortInterfaceMapping,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
modeMapping	ModeDeclarationGroupPrototypeMapping	1	aggr	Mapping of two ModeDeclarationGroupPrototypes in two different ModeInterfaces

Table D.145: ModeInterfaceMapping

Class	**ModeRequestTypeMap**			
Package	M2::AUTOSARTemplates::CommonStructure::ModeDeclaration			
Note	Specifies a mapping between a ModeDeclarationGroup and an ImplementationDataType. This ImplementationDataType shall be used to implement the ModeDeclarationGroup.			
Base	ARObject			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
implementationDataType	ImplementationDataType	1	ref	This is the corresponding ImplementationDataType. It shall be modeled along the idea of an "unsigned integer-like" data type.

Attribute	Datatype	Mul.	Kind	Note
modeGroup	ModeDeclarationGroup	1	ref	This is the corresponding ModeDeclarationGroup.

Table D.146: ModeRequestTypeMap

Class	ModeSwitchInterface			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	A mode switch interface declares a ModeDeclarationGroupPrototype to be sent and received. **Tags:** atp.recommendedPackage=PortInterfaces			
Base	ARElement,ARObject,AtpBlueprint,AtpBlueprintable,AtpClassifier,AtpType,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,PortInterface,Referrable			
Attribute	Datatype	Mul.	Kind	Note
modeGroup	ModeDeclarationGroupPrototype	1	aggr	The ModeDeclarationGroupPrototype of this mode interface.

Table D.147: ModeSwitchInterface

Class	ModeSwitchPoint			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::ModeDeclarationGroup			
Note	A ModeSwitchPoint is required by a RunnableEntity owned a Mode Manager. Its semantics implies the ability to initiate a mode switch.			
Base	ARObject,AtpClassifier,AtpFeature,AtpStructureElement,Identifiable,MultilanguageReferrable,Referrable			
Attribute	Datatype	Mul.	Kind	Note
modeGroup	ModeDeclarationGroupPrototype	0..1	iref	The mode declaration group that is switched by this runnable.

Table D.148: ModeSwitchPoint

Class	ModeSwitchReceiverComSpec			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication			
Note	Communication attributes of RPortPrototypes with respect to mode communication			
Base	ARObject,RPortComSpec			
Attribute	Datatype	Mul.	Kind	Note
enhancedModeApi	Boolean	0..1	attr	This controls the creation of the enhanced mode API that returns information about the previous mode and the next mode. If set to "true" the enhanced mode API is supposed to be generated. For more details please refer to the SWS_RTE.

Attribute	Datatype	Mul.	Kind	Note
modeGroup	ModeDeclarationGroupPrototype	0..1	ref	ModeDeclarationGroupPrototype (of the same PortInterface) to which these communication attributes apply. **Tags:** atp.Status=shallBecomeMandatory
supportsAsynchronousModeSwitch	Boolean	1	attr	This attribute controls the behavior of the corresponding RPortPrototype with respect to the question whether it can deal with asynchronous mode switch requests, i.e. if set to true, the RPortPrototype is able to deal with an asynchronous mode switch request.

Table D.149: ModeSwitchReceiverComSpec

Class	ModeSwitchSenderComSpec			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication			
Note	Communication attributes of PPortPrototypes with respect to mode communication			
Base	ARObject,PPortComSpec			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
enhancedModeApi	Boolean	0..1	attr	This controls the creation of the enhanced mode API that returns information about the previous mode and the next mode. If set to "true" the enhanced mode API is supposed to be generated. For more details please refer to the SWS_RTE.
modeGroup	ModeDeclarationGroupPrototype	1	ref	ModeDeclarationGroupPrototype (of the same PortInterface) to which these communication attributes apply.
modeSwitchedAck	ModeSwitchedAckRequest	0..1	aggr	If this aggregation exists an acknowledgement for the successful processing of the mode switch request is required.
queueLength	PositiveInteger	1	attr	Length of call queue on the mode user side. The queue is implemented by the RTE. The value shall be greater or equal to 1. Setting the value of queueLength to 1 implies that incoming requests are rejected while another request that arrived earlier is being processed.

Table D.150: ModeSwitchSenderComSpec

Class	ModeSwitchedAckEvent			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::RTEEvents			
Note	The event is raised when the referenced modes have been received or an error occurs.			
Base	ARObject,AbstractEvent,AtpClassifier,AtpFeature,AtpStructureElement,Identifiable,MultilanguageReferrable,RTEEvent,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
eventSource	ModeSwitchPoint	1	ref	Mode switch point that triggers the event.

Attribute	Datatype	Mul.	Kind	Note

Table D.151: ModeSwitchedAckEvent

Class	ModeSwitchedAckRequest			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication			
Note	Requests acknowledgements that a mode switch has been proceeded successfully			
Base	ARObject			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
timeout	TimeValue	1	attr	Number of seconds before an error is reported or in case of allowed redundancy, the value is sent again.

Table D.152: ModeSwitchedAckRequest

Class	NonqueuedReceiverComSpec			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication			
Note	Communication attributes specific to non-queued receiving.			
Base	ARObject,RPortComSpec,ReceiverComSpec			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
aliveTimeout	TimeValue	1	attr	Specify the amount of time (in seconds) after which the software component (via the RTE) needs to be notified if the corresponding data item have not been received according to the specified timing description. If the aliveTimeout attribute is 0 no timeout monitoring shall be performed.
enableUpdate	Boolean	1	attr	This attribute controls whether application code is entitled to check whether the value of the corresponding VariableDataPrototype has been updated.
filter	DataFilter	0..1	aggr	The applicable filter algorithm for filtering the value of the corresponding dataElement.
handleNeverReceived	Boolean	1	attr	This attribute specifies whether for the corresponding VariableDataPrototype the "never received" flag is available. If yes, the RTE is supposed to assume that initially the VariableDataPrototype has not been received before. After the first reception of the corresponding VariableDataPrototype the flag is cleared. • If the value of this attribute is set to "true" the flag is required. • If set to "false", the RTE shall not support the "never received" functionality for the corresponding VariableDataPrototype.
handleTimeoutType	HandleTimeoutEnum	1	attr	This attribute controls the behavior with respect to the handling of timeouts.

Attribute	Datatype	Mul.	Kind	Note
initValue	ValueSpecification	0..1	aggr	Initial value to be used in case the sending component is not yet initialized. If the sender also specifies an initial value the receiver's value will be used.

Table D.153: NonqueuedReceiverComSpec

Class	NonqueuedSenderComSpec			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication			
Note	Communication attributes for non-queued sender/receiver communication (sender side)			
Base	ARObject,PPortComSpec,SenderComSpec			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
initValue	ValueSpecification	1	aggr	Initial value to be sent if sender component is not yet fully initialized, but receiver needs data already.

Table D.154: NonqueuedSenderComSpec

Class	NumericalRuleBasedValueSpecification			
Package	M2::AUTOSARTemplates::CommonStructure::Constants			
Note	This meta-class is used to support a rule-based initialization approach for data types with an array-nature (ImplementationDataType of category ARRAY).			
Base	ARObject,AbstractRuleBasedValueSpecification,ValueSpecification			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
ruleBasedValues	RuleBasedValueSpecification	1	aggr	This represents the rule based value specification for the array. **Tags:** xml.roleElement=true; xml.roleWrapperElement=false; xml.typeWrapperElement=false

Table D.155: NumericalRuleBasedValueSpecification

Class	NumericalValueSpecification			
Package	M2::AUTOSARTemplates::CommonStructure::Constants			
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			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
value	Numerical	1	attr	This is the value itself. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime

Table D.156: NumericalValueSpecification

Class	NvBlockDataMapping			
Package	M2::AUTOSARTemplates::SWComponentTemplate::NvBlockComponent			
Note	Defines the mapping between the VariableDataPrototypes in the NvBlockComponents ports and the VariableDataPrototypes of the RAM Block. The data types of the referenced VariableDataPrototypes in the ports and the referenced sub-element (inside a CompositeDataType) of the VariableDataPrototype representing the RAM Block shall be compatible.			
Base	ARObject			
Attribute	Datatype	Mul.	Kind	Note
nvRamBlockElement	AutosarVariableRef	1	aggr	Reference to a VariableDataPrototype of a Ram Block.
readNvData	AutosarVariableRef	0..1	aggr	Reference to a VariableDataPrototype of a pPort of the NvBlockComponent providing read access to the NvRam Mirror. If there is no port providing read access (write-only) the reference can be omitted.
writtenNvData	AutosarVariableRef	0..1	aggr	Reference to a VariableDataPrototype of a rPort of the NvBlockComponent providing write access to the NvRam Mirror. If there is no port providing write access (read-only) the reference can be omitted.
writtenReadNvData	AutosarVariableRef	0..1	aggr	Reference to a VariableDataPrototype of a PRPortPrototype of the NvBlockSwComponentType providing write and read access to the NvRam Mirror.

Table D.157: NvBlockDataMapping

Class	NvBlockDescriptor			
Package	M2::AUTOSARTemplates::SWComponentTemplate::NvBlockComponent			
Note	Specifies the properties of exactly on NvBlock.			
Base	ARObject,Identifiable,MultilanguageReferrable,Referrable			
Attribute	Datatype	Mul.	Kind	Note
clientServerPort	RoleBasedPortAssignment	*	aggr	The RoleBasedPortAssignement defines which client server port of the NvBlockSwComponentType serves for which kind of service or notification. In case of notifications one common callback function is provided by the RTE for each individual kind of notification defined by the "role". The aggregation of RoleBasedPortAssignment is subject to variability with the purpose to support the conditional existence of ports. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
constantValueMapping	ConstantSpecificationMappingSet	*	ref	Reference to the ConstanSpecificationMapping to be applied for the particular NvBlock

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
dataTypeMapping	DataTypeMappingSet	*	ref	Reference to the DataTypeMapping to be applied for the particular NvBlock
instantiationDataDefProps	InstantiationDataDefProps	*	aggr	The purpose of InstantiationDataDefProps are the refinement of some data def properties of individual instantiations within the context of a NvBlockSwComponentType. The aggregation of InstantiationDataDefProps is subject to variability with the purpose to support the conditional existence of ports, component internal memory objects and those attributes. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
nvBlockDataMapping	NvBlockDataMapping	1..*	aggr	Defines the mapping between the VariableDataPrototypes in the NvBlockComponents ports and the VariableDataPrototypes of the RAM Block. The aggregation of NvBlockDataMapping is subject to variability with the purpose to support the conditional existence of nv data ports. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
nvBlockNeeds	NvBlockNeeds	1	aggr	Specifies the abstract needs on the configuration of the NvRam Manager for the single NvRam Block described by this NvBlockDescriptor. Please note that the attributes nDataSets and nRomBlocks are not relevant for this aggregation because the RTE will allocate just one block anyway. In a different context, however, they do make sense.
ramBlock	VariableDataPrototype	1	aggr	Defines the RAM Block of the NvBlock provided by NvBlockSwComponentType.
romBlock	ParameterDataPrototype	0..1	aggr	Defines the ROM Block of the NvBlock provided by NvBlockSwComponentType.

Table D.158: NvBlockDescriptor

Class	**NvBlockNeeds**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	Specifies the abstract needs on the configuration of a single Nv block.			
Base	ARObject,Identifiable,MultilanguageReferrable,Referrable,ServiceNeeds			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
calcRamBlockCrc	Boolean	0..1	attr	Defines if CRC (re)calculation for the permanent RAM block is required.
checkStaticBlockId	Boolean	0..1	attr	Defines if the Static Block Id check shall be enabled.

Attribute	Datatype	Mul.	Kind	Note
nDataSets	PositiveInteger	0..1	attr	Number of data sets to be provided by the NVRAM manager for this block. This is the total number of ROM blocks and NV Blocks.
nRomBlocks	PositiveInteger	0..1	attr	Number of ROM blocks to be provided by the NVRAM manager for this block. Please note that these multiple ROM Blocks are given in a contiguous area.
ramBlockStatusControl	RamBlockStatusControlEnum	0..1	attr	This attribute defines how the management of the ramBlock status is controlled.
readonly	Boolean	0..1	attr	True: data of this block are write protected for normal operation (but protection can be disabled) false: no restriction
reliability	NvBlockNeedsReliabilityEnum	0..1	attr	Reliability against data loss on the non-volatile medium.
resistantToChangedSw	Boolean	0..1	attr	Defines whether an Nv block shall be treated resistant to configuration changes (true) or not (false). For details how to handle initialization in the latter case, please refer to the NVRAM specification.
restoreAtStart	Boolean	0..1	attr	Defines whether the associated RAM mirror block shall be implicitly restored during startup by the basic SW or not. Only relevant if a RAM mirror block is associated with this port (for Software Components the latter is modeled via SwcServiceDependency).
storeAtShutdown	Boolean	0..1	attr	Defines whether or not the associated RAM mirror block shall be implicitly stored during shutdown by the basic SW. This is only relevant if a RAM mirror block is associated with this port (for software-components the latter is modeled by means of a SwcServiceDependency).
writeOnlyOnce	Boolean	0..1	attr	Defines write protection after first write: true: This block is prevented from being changed/erased or being replaced with the default ROM data after first initialization by the software-component. false: No such restriction.
writeVerification	Boolean	0..1	attr	Defines if Write Verification shall be enabled for this Nv Block.
writingFrequency	PositiveInteger	0..1	attr	Provides the amount of updates to this block from the application point of view. It has to be provided in "number of write access per year".
writingPriority	NvBlockNeedsWritingPriorityEnum	0..1	attr	Requires the priority of writing this block in case of concurrent requests to write other blocks.

Table D.159: NvBlockNeeds

Class	**NvBlockSwComponentType**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	The NvBlockSwComponentType defines non volatile data which data can be shared between SwComponentPrototypes. The non volatile data of the NvBlockSwComponentType are accessible via provided and required ports. **Tags:** atp.recommendedPackage=SwComponentTypes			
Base	ARElement,ARObject,AtomicSwComponentType,AtpBlueprint,AtpBlueprintable,AtpClassifier,AtpType,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable,SwComponentType			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
nvBlockDescriptor	NvBlockDescriptor	*	aggr	Specification of the properties of exactly one NvBlock. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=preCompileTime

Table D.160: NvBlockSwComponentType

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

Table D.161: NvDataInterface

Class	**NvRequireComSpec**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication			
Note	Communication attributes of RPortPrototypes with respect to Nv data communication on the required side.			
Base	ARObject,RPortComSpec			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
initValue	ValueSpecification	0..1	aggr	The initial value owned by the NvComSpec
variable	VariableDataPrototype	1	ref	The VariableDataPrototype the ComSpec applies for.

Table D.162: NvRequireComSpec

Class	**OperationInvokedEvent**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::RTE Events			
Note	The OperationInvokedEvent references the ClientServerOperation invoked by the client.			
Base	ARObject,AbstractEvent,AtpClassifier,AtpFeature,AtpStructure Element,Identifiable,MultilanguageReferrable,RTEEvent,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
operation	ClientServerOp eration	0..1	iref	The operation to be executed as the consequence of the event.

Table D.163: OperationInvokedEvent

Class	**PPortPrototype**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	Component port providing a certain port interface.			
Base	ARObject,AbstractProvidedPortPrototype,AtpBlueprintable,AtpFeature,Atp Prototype,Identifiable,MultilanguageReferrable,PortPrototype,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
providedInt erface	PortInterface	1	tref	The interface that this port provides. **Stereotypes:** isOfType

Table D.164: PPortPrototype

Class	**PRPortPrototype**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	This kind of PortPrototype can take the role of both a required and a provided PortPrototype.			
Base	ARObject,AbstractProvidedPortPrototype,AbstractRequiredPortPrototype,Atp Blueprintable,AtpFeature,AtpPrototype,Identifiable,MultilanguageReferrable,Port Prototype,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
providedR equiredInte rface	PortInterface	1	tref	This represents the PortInterface used to type the PRPortPrototype **Stereotypes:** isOfType

Table D.165: PRPortPrototype

Class	**ParameterAccess**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::Data Elements			
Note	The presence of a ParameterAccess implies that a RunnableEntity needs access to a ParameterDataPrototype.			
Base	ARObject,AtpClassifier,AtpFeature,AtpStructureElement,Identifiable,Multilanguage Referrable,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*

Attribute	Datatype	Mul.	Kind	Note
accessedParameter	AutosarParameterRef	1	aggr	Refernce to the accessed calibration parameter.
swDataDefProps	SwDataDefProps	0..1	aggr	This allows denote instance and access specific properties, mainly input values and common axis.

Table D.166: ParameterAccess

Class	ParameterDataPrototype			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::DataPrototypes			
Note	A parameter element used for parameter interface and internal behavior, supporting signal like parameter and characteristic value communication patterns and parameter and characteristic value definition.			
Base	ARObject,AtpFeature,AtpPrototype,AutosarDataPrototype,DataPrototype,Identifiable,MultilanguageReferrable,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
initValue	ValueSpecification	0..1	aggr	Specifies initial value(s) of the ParameterDataPrototype

Table D.167: ParameterDataPrototype

Class	ParameterInterface			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
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,AtpBlueprintable,AtpClassifier,AtpType,CollectableElement,DataInterface,Identifiable,MultilanguageReferrable,PackageableElement,PortInterface,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
parameter	ParameterDataPrototype	1..*	aggr	The ParameterDataPrototype of this ParameterInterface.

Table D.168: ParameterInterface

Class	ParameterProvideComSpec			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication			
Note	"Communication" specification that applies to parameters on the provided side of a connection.			
Base	ARObject,PPortComSpec			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
initValue	ValueSpecification	0..1	aggr	The initial value applicable for the corresponding ParameterDataPrototype.
parameter	ParameterDataPrototype	1	ref	The ParameterDataPrototype to which the ParameterComSpec applies.

Table D.169: ParameterProvideComSpec

Class	ParameterRequireComSpec			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication			
Note	"Communication" specification that applies to parameters on the required side of a connection.			
Base	ARObject,RPortComSpec			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
initValue	ValueSpecification	0..1	aggr	The initial value applicable for the corresponding ParameterDataPrototype.
parameter	ParameterDataPrototype	1	ref	The ParameterDataPrototype to which the ParameterRequireComSpec applies.

Table D.170: ParameterRequireComSpec

Class	ParameterSwComponentType			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	The ParameterSwComponentType defines parameters and characteristic values accessible via provided Ports. The provided values are the same for all connected SwComponentPrototypes **Tags:** atp.recommendedPackage=SwComponentTypes			
Base	ARElement,ARObject,AtpBlueprint,AtpBlueprintable,AtpClassifier,AtpType,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable,SwComponentType			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
constantMapping	ConstantSpecificationMappingSet	*	ref	Reference to the ConstanSpecificationMapping to be applied for the particular ParameterSwComponentType
dataTypeMapping	DataTypeMappingSet	*	ref	Reference to the DataTypeMapping to be applied for the particular ParameterSwComponentType
instantiationDataDefProps	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 PortPrototypes **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime

Table D.171: ParameterSwComponentType

Class	**PerInstanceMemory**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::PerInstanceMemory			
Note	Defines a 'C' 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 software-component defines NVRAM access via permanent blocks.			
Base	ARObject,AtpClassifier,AtpFeature,AtpStructureElement,Identifiable,MultilanguageReferrable,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
initValue	String	0..1	attr	Specifies initial value(s) of the PerInstanceMemory
swDataDefProps	SwDataDefProps	0..1	aggr	This represents the ability to to allocate RAM at specific memory sections, for example, to support the RAM block recovery strategy by mapping to uninitialized RAM.
type	CIdentifier	1	ref	The name of the "C"-type
typeDefinition	String	1	attr	A definition of the type with the syntax of a 'C' typedef.

Table D.172: PerInstanceMemory

Class	**PortAPIOption**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::PortAPIOptions			
Note	Options how to generate the signatures of calls for an AtomicSwComponentType in order to communicate over a PortPrototype (for calls into a RunnableEntity as well as for calls from a RunnableEntity to the PortPrototype).			
Base	ARObject			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
enableTakeAddress	Boolean	1	attr	If set to true, the software-component is able to use the API reference for deriving a pointer to an object.
indirectAPI	Boolean	1	attr	If set to true this attribute specifies an "indirect API" to be generated for the associated port which means that the SWC is able to access the actions on a port via a pointer to an object representing a port. This allows e.g. iterating over ports in a loop. This option has no effect for PPortPrototypes of client/server interfaces.
port	PortPrototype	1	ref	The option is valid for generated functions related to communication over this port
portArgValue (ordered)	PortDefinedArgumentValue	*	aggr	An argument value defined by this port.

Table D.173: PortAPIOption

Class	**PortDefinedArgumentValue**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::PortAPI Options			
Note	A PortDefinedArgumentValue is passed to a RunnableEntity dealing with the ClientServerOperations provided by a given PortPrototype. Note that this is restricted to PPortPrototypes of a ClientServerInterface.			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
value	ValueSpecification	1	aggr	Specifies the actual value.
valueType	ImplementationDataType	1	tref	The implementation type of this argument value. It should not be composite type or a pointer. **Stereotypes:** isOfType

Table D.174: PortDefinedArgumentValue

Class	**PortInterface (abstract)**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	Abstract base class for an interface that is either provided or required by a port of a software component.			
Base	ARElement,ARObject,AtpBlueprint,AtpBlueprintable,AtpClassifier,AtpType,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
isService	Boolean	1	attr	This flag is set if the PortInterface is to be used for communication between an • ApplicationSwComponentType or • ServiceProxySwComponentType or • SensorActuatorSwComponentType or • ComplexDeviceDriverSwComponentType or • EcuAbstractionSwComponentType and a ServiceSwComponentType (namely an AUTOSAR Service) located on the same ECU. Otherwise the flag is not set.
serviceKind	ServiceProviderEnum	0..1	attr	This attribute provides further details about the nature of the applied service.

Table D.175: PortInterface

Class	**PortInterfaceMapping (abstract)**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	Specifies one PortInterfaceMapping to support the connection of Ports typed by two different PortInterfaces with PortInterface elements having unequal names and/or unequal semantic (resolution or range).			
Base	ARObject,AtpBlueprint,AtpBlueprintable,Identifiable,Multilanguage Referrable,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
–	–	–	–	–

Table D.176: PortInterfaceMapping

Class	**PortPrototype (abstract)**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	Base class for the ports of an AUTOSAR software component. The aggregation of PortPrototypes is subject to variability with the purpose to support the conditional existence of ports.			
Base	ARObject,AtpBlueprintable,AtpFeature,AtpPrototype,Identifiable,Multilanguage Referrable,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
clientServerAnnotation	ClientServerAnnotation	*	aggr	Annotation of this PortPrototype with respect to client/server communication.
delegatedPortAnnotation	DelegatedPortAnnotation	0..1	aggr	Annotations on this delegated port.
ioHwAbstractionServerAnnotation	IoHwAbstractionServerAnnotation	*	aggr	Annotations on this IO Hardware Abstraction port.
modePortAnnotation	ModePortAnnotation	*	aggr	Annotations on this mode port.
nvDataPortAnnotation	NvDataPortAnnotation	*	aggr	Annotations on this non voilatile data port.
parameterPortAnnotation	ParameterPortAnnotation	*	aggr	Annotations on this parameter port.
senderReceiverAnnotation	SenderReceiverAnnotation	*	aggr	Collection of annotations of this ports sender/receiver communication.
triggerPortAnnotation	TriggerPortAnnotation	*	aggr	Annotations on this trigger port.

Table D.177: PortPrototype

Class	**PostBuildVariantCondition**			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling			
Note	This class specifies the value which must be assigned to a particular variant criterion in order to bind the variation point. If multiple criterion/value pairs are specified, they shall all match to bind the variation point. In other words binding can be represented by `(criterion1 == value1) && (condition2 == value2) ...`			
Base	ARObject			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
matchingCriterion	PostBuildVariantCriterion	1	ref	This is the criterion which needs to match the value in order to make the PostbuildVariantCondition to be true.
value	Integer	1	attr	This is the particular value of the post-build variant criterion. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime

Table D.178: PostBuildVariantCondition

Class	**PostBuildVariantCriterion**			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling			
Note	This class specifies one particular PostBuildVariantSelector. **Tags:** atp.recommendedPackage=PostBuildVariantCriterions			
Base	ARElement,ARObject,AtpDefinition,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
compuMethod	CompuMethod	1	ref	The compuMethod specifies the possible values for the variant criterion serving as an enumerator.

Table D.179: PostBuildVariantCriterion

Class	**PostBuildVariantCriterionValue**			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling			
Note	This class specifies a the value which must be assigned to a particular variant criterion in order to bind the variation point. If multiple criterion/value pairs are specified, they all must must match to bind the variation point.			
Base	ARObject			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
annotation	Annotation	*	aggr	This provides the ability to add information why the value is set like it is. **Tags:** xml.sequenceOffset=30

Attribute	Datatype	Mul.	Kind	Note
value	Integer	1	attr	This is the particular value of the post-build variant criterion. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=20
variantCriterion	PostBuildVariantCriterion	1	ref	This association selects the variant criterion whose value is specified. **Tags:** xml.sequenceOffset=10

Table D.180: PostBuildVariantCriterionValue

Class	PostBuildVariantCriterionValueSet			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling			
Note	This meta-class represents the ability to denote one set of postBuildVariantCriterionValues. **Tags:** atp.recommendedPackage=PostBuildVariantCriterionValueSets			
Base	ARElement,ARObject,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
postBuildVariantCriterionValue	PostBuildVariantCriterionValue	*	aggr	This is is one particular postbuild variant criterion/value pair being part of the PostBuildVariantSet.

Table D.181: PostBuildVariantCriterionValueSet

Class	PredefinedVariant			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling			
Note	This specifies one predefined variant. It is characterized by the union of all system constant values and post-build variant criterion values aggregated within all referenced system constant value sets and post build variant criterion value sets plus the value sets of the included variants. **Tags:** atp.recommendedPackage=PredefinedVariants			
Base	ARElement,ARObject,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
includedVariant	PredefinedVariant	*	ref	The associated variants are considered part of this PredefinedVariant. This means the settings of the included variants are included in the settings of the referencing PredefinedVariant. Nevertheless the included variants might be included in several predefined variants.
postBuildVariantCriterionValueSet	PostBuildVariantCriterionValueSet	*	ref	This is the postBuildVariantCriterionValueSet contributing to the predefinded variant.

Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
swSystemconstantValueSet	SwSystemconstantValueSet	*	ref	This ist the set of Systemconstant Values contributing to the predefined variant.

Table D.182: PredefinedVariant

Class	**QueuedReceiverComSpec**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication			
Note	Communication attributes specific to queued receiving.			
Base	ARObject,RPortComSpec,ReceiverComSpec			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
queueLength	PositiveInteger	1	attr	Length of queue for received events.

Table D.183: QueuedReceiverComSpec

Class	**QueuedSenderComSpec**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication			
Note	Communication attributes specific to distribution of events (PPortPrototype, SenderReceiverInterface and dataElement carries an "event").			
Base	ARObject,PPortComSpec,SenderComSpec			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
–	–	–	–	–

Table D.184: QueuedSenderComSpec

Class	**RPortPrototype**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	Component port requiring a certain port interface.			
Base	ARObject,AbstractRequiredPortPrototype,AtpBlueprintable,AtpFeature,AtpPrototype,Identifiable,MultilanguageReferrable,PortPrototype,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
requiredInterface	PortInterface	1	tref	The interface that this port requires, i.e. the port depends on another port providing the specified interface. **Stereotypes:** isOfType

Table D.185: RPortPrototype

Class	RTEEvent (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::RTE Events			
Note	Abstract base class for all RTE-related events			
Base	ARObject,AbstractEvent,AtpClassifier,AtpFeature,AtpStructure Element,Identifiable,MultilanguageReferrable,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
disabledMode	ModeDeclaration	*	iref	Reference to the Modes that disable the Event. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=contextPort, contextMode DeclarationGroupPrototype, targetMode Declaration
startOnEvent	RunnableEntity	0..1	ref	RunnableEntity starts when the corresponding RTEEvent occurs.

Table D.186: RTEEvent

Class	ReceiverComSpec (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication			
Note	Receiver-specific communication attributes (RPortPrototype typed by SenderReceiverInterface).			
Base	ARObject,RPortComSpec			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
compositeNetworkRepresentation	CompositeNetworkRepresentation	*	aggr	This represents a CompositeNetworkRepresentation defined in the context of a ReceiverComSpec. The purpose of this aggregation is to be able to specify the network representation of leaf elements of ApplicationCompositeDataTypes.
dataElement	VariableDataPrototype	1	ref	Data element these attributes belong to.
externalReplacement	AutosarDataPrototype	0..1	ref	This reference is used to reference the AutosarDataPrototype to be taken for sourcing an external replacement in the out-of-range handling.
handleOutOfRange	HandleOutOfRangeEnum	1	attr	This attribute controls how values that are out of the specified range are handled according to the values of HandleOutOfRangeEnum.
handleOutOfRangeStatus	HandleOutOfRangeStatusEnum	0..1	attr	Control the way how return values are created in case of an out-of-range situation.

Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
maxDeltaCounterInit	PositiveInteger	0..1	attr	Initial maximum allowed gap between two counter values of two consecutively received valid Data, i.e. how many subsequent lost data is accepted. For example, if the receiver gets Data with counter 1 and MaxDeltaCounterInit is 1, then at the next reception the receiver can accept Counters with values 2 and 3, but not 4. Note that if the receiver does not receive new Data at a consecutive read, then the receiver increments the tolerance by 1. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
maxNoNewOrRepeatedData	PositiveInteger	0..1	attr	The maximum amount of missing or repeated Data which the receiver does not expect to exceed under normal communication conditions.
networkRepresentation	SwDataDefProps	0..1	aggr	A networkRepresentation is used to define how the dataElement is mapped to a communication bus.
syncCounterInit	PositiveInteger	0..1	attr	Number of Data required for validating the consistency of the counter that shall be received with a valid counter (i.e. counter within the allowed lock-in range) after the detection of an unexpected behaviour of a received counter.
usesEndToEndProtection	Boolean	1	attr	This indicates whether the corresponding dataElement shall be transmitted using end-to-end protection. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime

Table D.187: ReceiverComSpec

Class	**ReferenceValueSpecification**			
Package	M2::AUTOSARTemplates::CommonStructure::Constants			
Note	Specifies a reference to a data prototype to be used as an initial value for a pointer in the software.			
Base	ARObject,ValueSpecification			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
referenceValue	DataPrototype	1	ref	The referenced data prototype.

Table D.188: ReferenceValueSpecification

Class	**Referrable (abstract)**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Identifiable			
Note	Instances of this class can be referred to by their identifier (while adhering to namespace borders).			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
shortName	Identifier	1	ref	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. **Tags:** xml.enforceMinMultiplicity=true; xml.sequenceOffset=-100

Table D.189: Referrable

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

Table D.190: RevisionLabelString

Class	**RoleBasedPortAssignment**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::ServiceMapping			
Note	This class specifies an assignment of a role to a particular service port (RPortPrototype or PPortPrototype) of an AtomicSwComponentType. With this assignment, the role of the service port can be mapped to a specific ServiceNeeds element, so that a tool is able to create the correct connector.			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***

Attribute	Datatype	Mul.	Kind	Note
portPrototype	PortPrototype	1	ref	Service port used in the assigned role. This port shall either belong to the same AtomicSoftwareComponent as the SwcInternalBehavior which owns the ServiceDependency or to the same NvBlockComponentType as the NvBlockDescriptor.
role	Identifier	1	ref	This is the role of the assigned Port in the given context. The value shall be a shortName of the Blueprint of a PortInterface as standardized in the Software Specification of the related AUTOSAR Service.

Table D.191: RoleBasedPortAssignment

Class	RootSwCompositionPrototype			
Package	M2::AUTOSARTemplates::SystemTemplate			
Note	The RootSwCompositionPrototype represents the top-level-composition of software components within a given System. According to the use case of the System, this may for example be the a more or less complete VFB description, the software of a System Extract or the software of a flat ECU Extract with only atomic SWCs. 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 PortPrototypes, PortInterfaces, VariableDataPrototypes, SwcInternalBehavior etc.), and their ports are interconnected using SwConnectorPrototypes.			
Base	ARObject,AtpFeature,AtpPrototype,Identifiable,MultilanguageReferrable,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
calibrationParameterValueSet	CalibrationParameterValueSet	*	ref	Used CalibrationParameterValueSet for instance specific initialization of calibration parameters. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=calibrationParameterValueSet
flatMap	FlatMap	0..1	ref	The FlatMap used in the scope of this RootSwCompositionPrototype. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=flatMap
softwareComposition	CompositionSwComponentType	1	tref	We assume that there is exactly one top-level composition that includes all Component instances of the system **Stereotypes:** isOfType

Attribute	Datatype	Mul.	Kind	Note

Table D.192: RootSwCompositionPrototype

Class	**RuleBasedValueSpecification**			
Package	M2::AUTOSARTemplates::CommonStructure::Constants			
Note	This meta-class is used to support a rule-based initialization approach for data types with an array-nature (ApplicationArrayDataType and ImplementationDataType of category ARRAY) or a compound ApplicationPrimitiveDataType (which also boils down to an array-nature).			
Base	ARObject			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
arguments	RuleArguments	1	aggr	This represents the arguments for the RuleBasedValueSpecification. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=30
rule	Identifier	1	ref	This denotes the name of the rule of the RuleBasedValueSpecification. The rule determines the calculation specification according which the arguments are used to calculated the values. **Tags:** xml.sequenceOffset=20

Table D.193: RuleBasedValueSpecification

Class	**RunnableEntity**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior			
Note	A RunnableEntity represents the smallest code-fragment that is provided by an AtomicSwComponentType and are executed under control of the RTE. RunnableEntities are for instance set up to respond to data reception or operation invocation on a server.			
Base	ARObject,AtpClassifier,AtpFeature,AtpStructureElement,Executable Entity,Identifiable,MultilanguageReferrable,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
argument (ordered)	RunnableEntity Argument	*	aggr	This represents the formal definition of a an argument to a RunnableEntity.
asynchronousServerCallResultPoint	AsynchronousServerCallResultPoint	*	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:** vh.latestBindingTime=preCompileTime

Attribute	Datatype	Mul.	Kind	Note
canBeInvokedConcurrently	Boolean	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. Note that the default value of this attribute is set to "false".
dataReadAccess	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:** vh.latestBindingTime=preCompileTime
dataReceivePointByArgument	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:** vh.latestBindingTime=preCompileTime
dataReceivePointByValue	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 dataReceivePointByValue 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:** vh.latestBindingTime=preCompileTime

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
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:** vh.latestBindingTime=preCompileTime
dataWriteAccess	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:** vh.latestBindingTime=preCompileTime
externalTriggeringPoint	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:** vh.latestBindingTime=preCompileTime
internalTriggeringPoint	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:** vh.latestBindingTime=preCompileTime
modeAccessPoint	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:** vh.latestBindingTime=preCompileTime

Attribute	Datatype	Mul.	Kind	Note
modeSwitchPoint	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:** vh.latestBindingTime=preCompileTime
parameterAccess	ParameterAccess	*	aggr	The presence of a ParameterAccess implies that a RunnableEntity needs read only access to a ParameterDataPrototype which may either be local or within a PortPrototype. 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 ParameterAccess (points) in the implementation. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
readLocalVariable	VariableAccess	*	aggr	The presence of a readLocalVariable implies that a RunnableEntity needs read access to a VariableDataPrototype 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 explicitInterRunnableVariable or the variant existence of readLocalVariable (points) in the implementation. **Stereotypes:** atpVariation **Tags:** 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:** vh.latestBindingTime=preCompileTime
symbol	CIdentifier	1	ref	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.

Attribute	Datatype	Mul.	Kind	Note
writtenLocalVariable	VariableAccess	*	aggr	The presence of a writtenLocalVariable implies that a RunnableEntity needs write access to a VariableDataPrototype 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 explicitInterRunnableVariable or the variant existence of writtenLocalVariable (points) in the implementation. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime

Table D.194: RunnableEntity

Class	RunnableEntityArgument			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::RunnableEntity			
Note	This meta-class represents the ability to provide specific information regarding the arguments to a RunnableEntity.			
Base	ARObject			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
symbol	CIdentifier	1	ref	This represents the symbol to be generated into the actual signature on the level of the C programming language.

Table D.195: RunnableEntityArgument

Class	RunnableEntityGroup			
Package	M2::AUTOSARTemplates::SWComponentTemplate::ImplicitCommunicationBehavior			
Note	This meta-class represents the ability to define a collection of RunnableEntities. The collection can be nested.			
Base	ARObject,AtpClassifier,AtpFeature,AtpStructureElement,Identifiable,MultilanguageReferrable,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
runnableEntity	RunnableEntity	*	iref	This represents a collection of RunnableEntitys that belong to the enclosing RunnableEntityGroup. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
runnableEntityGroup	RunnableEntityGroup	*	iref	This represents the ability to define nested groups of RunnableEntitys. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime

Table D.196: RunnableEntityGroup

Class	**Sdg**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::SpecialData			
Note	Sdg (SpecialDataGroup) is a generic model which can be used to keep arbitrary information which is not explicitly modeled in the meta-model. Sdg can have various contents as defined by sdgContentsType. Special Data should only be used moderately since all elements should be defined in the meta-model. Thereby SDG should be considered as a temporary solution when no explicit model is available. If an sdgCaption is available, it is possible to establish a reference to the sdg structure.			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
gid	NameToken	1	attr	This attributes specifies an identifier. Gid comes from the SGML/XML-Term "Generic Identifier" which is the element name in XML. The role of this attribute is the same as the name of an XML - element. **Tags:** xml.attribute=true
sdgCaption	SdgCaption	0..1	aggr	This aggregation allows to assign the properties of Identifiable to the sdg. By this, a shortName etc. can be assigned to the Sdg. **Tags:** xml.sequenceOffset=20
sdgCaptionRef	SdgCaption	0..1	ref	This association allows to reuse an already existing caption. **Tags:** xml.name=SDG-CAPTION-REF; xml.sequenceOffset=25
sdgContentsType	SdgContents	0..1	aggr	This is the content of the Sdg. **Tags:** xml.roleElement=false; xml.roleWrapperElement=false; xml.sequenceOffset=30; xml.typeElement=false; xml.typeWrapperElement=false

Table D.197: Sdg

Class	**ScaleConstr**			
Package	M2::AUTOSARTemplates::CommonStructure::GlobalConstraints			
Note	This meta-class represents the ability to specify constraints as a list of intervals (called scales).			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
desc	MultiLanguageOverviewParagraph	0..1	aggr	<desc> represents a general but brief description of the object in question. **Tags:** xml.sequenceOffset=30

Attribute	Datatype	Mul.	Kind	Note
lowerLimit	Limit	0..1	ref	This specifies the lower limit of the scale. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=40
shortLabel	Identifier	0..1	ref	This element specifies a short name for the scaleConstr. This can for example be used to create more specific messages of a constraint checker. The constraints cannot be associated in the meta-model, therefore shortLabel is somehow a substitute for shortName. **Tags:** xml.sequenceOffset=20
upperLimit	Limit	0..1	ref	This specifies the upper limit of a the scale. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=50
validity	ScaleConstrValidityEnum	0..1	attr	Specifies if the values defined by the scales are considered to be valid. If the attribute is missing then the default value is "VALID". **Tags:** xml.attribute=true

Table D.198: ScaleConstr

Class	SectionNamePrefix			
Package	M2::AUTOSARTemplates::CommonStructure::ResourceConsumption::MemorySectionUsage			
Note	A prefix to be used for generated code artifacts defining a memory section name in the source code of the using module.			
Base	ARObject,ImplementationProps,Referrable			
Attribute	Datatype	Mul.	Kind	Note
implementedIn	DependencyOnArtifact	0..1	ref	Optional reference that allows to Indicate the code artifact (header file) containing the preprocessor implementation of memory sections with this prefix. The usage of this link supersedes the usage of a memory mapping header with the default name (derived from the BswModuleDescription's shortName).

Table D.199: SectionNamePrefix

Class	SenderComSpec (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication			
Note	Communication attributes for a sender port (PPortPrototype typed by SenderReceiverInterface).			
Base	ARObject,PPortComSpec			
Attribute	Datatype	Mul.	Kind	Note
compositeNetworkRepresentation	CompositeNetworkRepresentation	*	aggr	This represents a CompositeNetworkRepresentation defined in the context of a SenderComSpec.
dataElement	VariableDataPrototype	1	ref	Data element these quality of service attributes apply to.
handleOutOfRange	HandleOutOfRangeEnum	1	attr	This attribute controls how out-of-range values shall be dealt with.
networkRepresentation	SwDataDefProps	0..1	aggr	A networkRepresentation is used to define how the dataElement is mapped to a communication bus.
transmissionAcknowledge	TransmissionAcknowledgementRequest	0..1	aggr	Requested transmission acknowledgement for data element.
usesEndToEndProtection	Boolean	1	attr	This indicates whether the corresponding dataElement shall be transmitted using end-to-end protection. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime

Table D.200: SenderComSpec

Class	SenderReceiverInterface			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	A sender/receiver interface declares a number of data elements to be sent and received. **Tags:** atp.recommendedPackage=PortInterfaces			
Base	ARElement,ARObject,AtpBlueprint,AtpBlueprintable,AtpClassifier,AtpType,CollectableElement,DataInterface,Identifiable,MultilanguageReferrable,PackageableElement,PortInterface,Referrable			
Attribute	Datatype	Mul.	Kind	Note
dataElement	VariableDataPrototype	1..*	aggr	The data elements of this SenderReceiverInterface.
invalidationPolicy	InvalidationPolicy	*	aggr	InvalidationPolicy for a particular dataElement

Table D.201: SenderReceiverInterface

Class	SenderReceiverToSignalGroupMapping			
Package	M2::AUTOSARTemplates::SystemTemplate::DataMapping			
Note	Mapping of a sender receiver communication data element with a composite datatype to a signal group.			
Base	ARObject,DataMapping			
Attribute	Datatype	Mul.	Kind	Note
dataElement	VariableDataPrototype	1	iref	Reference to a data element with a composite datatype which is mapped to a signal group.
signalGroup	SystemSignalGroup	1	ref	Reference to the signal group, which contain all primitive datatypes of the composite type
typeMapping	SenderRecCompositeTypeMapping	1	aggr	The CompositeTypeMapping maps the the ApplicationArrayElements and ApplicationRecordElements to Signals of the SignalGroup.

Table D.202: SenderReceiverToSignalGroupMapping

Class	SenderReceiverToSignalMapping			
Package	M2::AUTOSARTemplates::SystemTemplate::DataMapping			
Note	Mapping of a sender receiver communication data element with a primitive datatype to a signal.			
Base	ARObject,DataMapping			
Attribute	Datatype	Mul.	Kind	Note
dataElement	VariableDataPrototype	1	iref	Reference to the data element, which ought to be sent over the Communication bus.
systemSignal	SystemSignal	1	ref	Reference to the system signal used to carry the data element.

Table D.203: SenderReceiverToSignalMapping

Class	SensorActuatorSwComponentType			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	The SensorActuatorSwComponentType introduces the possibility to link from the software representation of a sensor/actuator to its hardware description provided by the ECU Resource Template. **Tags:** atp.recommendedPackage=SwComponentTypes			
Base	ARElement,ARObject,AtomicSwComponentType,AtpBlueprint,AtpBlueprintable,AtpClassifier,AtpType,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable,SwComponentType			
Attribute	Datatype	Mul.	Kind	Note
sensorActuator	HwDescriptionEntity	1	ref	Reference from the Sensor Actuator Software Component Type to the description of the actual hardware.

Table D.204: SensorActuatorSwComponentType

Enumeration	ServerArgumentImplPolicyEnum
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface

Note	This defines how the argument type of the servers RunnableEntity is implemented.
Literal	***Description***
useArgument Type	The argument type of the RunnableEntity is derived from the AutosarDataType of the ArgumentPrototype.
useArray BaseType	The argument type of the RunnableEntity is derived from the AutosarDataType of the elements of the array that corresponds to the ArgumentPrototype. This represents the base type of the array in C.
useVoid	The argument type of the RunnableEntity is void.

Table D.205: ServerArgumentImplPolicyEnum

Class	**ServerCallPoint (abstract)**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::ServerCall			
Note	If a RunnableEntity owns a ServerCallPoint it is entitled to invoke a particular ClientServerOperation of a specific RPortPrototype of the corresponding AtomicSwComponentType			
Base	ARObject,AtpClassifier,AtpFeature,AtpStructureElement,Identifiable,Multilanguage Referrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
operation	ClientServerOp eration	0..1	iref	The operation that is called by this runnable.
timeout	TimeValue	1	attr	Time in seconds before the server call times out and returns with an error message. It depends on the call type (synchronous or asynchronous) how this is reported.

Table D.206: ServerCallPoint

Class	**ServerComSpec**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication			
Note	Communication attributes for a server port (PPortPrototype and ClientServerInterface).			
Base	ARObject,PPortComSpec			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
operation	ClientServerOp eration	1	ref	Operation these communication attributes apply to.
queueLeng th	PositiveInteger	1	attr	Length of call queue on the server side. The queue is implemented by the RTE. The value shall be greater or equal to 1. Setting the value of queueLength to 1 implies that incoming requests are rejected while another request that arrived earlier is being processed.

Table D.207: ServerComSpec

Class	**ServiceProxySwComponentType**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	This class provides the ability to express a software-component which provides access to an internal service for remote ECUs. It acts as a proxy for the service providing access to the service. An important use case is the request of vehicle mode switches: Such requests can be communicated via sender-receiver interfaces across ECU boundaries, but the mode manager being responsible to perform the mode switches is an AUTOSAR Service which is located in the Basic Software and is not visible in the VFB view. To handle this situation, a ServiceProxySwComponentType will act as proxy for the mode manager. It will have R-Ports to be connected with the mode requestors on VFB level and Service-Ports to be connected with the local mode manager at ECU integration time. Apart from the semantics, a ServiceProxySwComponentType has these specific properties: • A prototype of it can be mapped to more than one ECUs in the system description. • Exactly one additional instance of it will be created in the ECU-Extract per ECU to which the prototype has been mapped. • For remote communication, it can have only R-Ports with sender-receiver interfaces and 1:n semantics. • There shall be no connectors between two prototypes of any ServiceProxySwComponentType. **Tags:** atp.recommendedPackage=SwComponentTypes			
Base	ARElement,ARObject,AtomicSwComponentType,AtpBlueprint,AtpBlueprintable,AtpClassifier,AtpType,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable,SwComponentType			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table D.208: ServiceProxySwComponentType

Class	**ServiceSwComponentType**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	ServiceSwComponentType is used for configuring services for a given ECU. Instances of this class are only to be created in ECU Configuration phase for the specific purpose of the service configuration. **Tags:** atp.recommendedPackage=SwComponentTypes			
Base	ARElement,ARObject,AtomicSwComponentType,AtpBlueprint,AtpBlueprintable,AtpClassifier,AtpType,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable,SwComponentType			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table D.209: ServiceSwComponentType

Class	**SubElementMapping**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	This meta-class allows for the definition of mappings of elements of a composite data type.			
Base	ARObject			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
firstElement	SubElementRef	0..1	aggr	This represents the first element referenced in the scope of the mapping. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
secondElement	SubElementRef	0..1	aggr	This represents the second element referenced in the scope of the mapping. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
textTableMapping	TextTableMapping	0..2	aggr	This allows for the text-table translation of individual elements of a composite data type.

Table D.210: SubElementMapping

Class	**SwAddrMethod**			
Package	M2::AUTOSARTemplates::CommonStructure::AuxillaryObjects			
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,AtpBlueprintable,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
memoryAllocationKeywordPolicy	MemoryAllocationKeywordPolicyType	0..1	attr	Enumeration to specify the name pattern of the Memory Allocation Keyword.
option	Identifier	*	ref	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 MemMapAddressingModeSet.

Attribute	Datatype	Mul.	Kind	Note
sectionInitializationPolicy	SectionInitializationPolicyType	0..1	attr	Specifies the expected initialization of the variables (inclusive those which are implementing VariableDataPrototypes). 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 AutosarDataPrototypes 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 addresssing method.

Table D.211: SwAddrMethod

Class	SwBaseType			
Package	M2::AUTOSARTemplates::CommonStructure::BaseTypes			
Note	This meta-class represents a base type used within ECU software. **Tags:** atp.recommendedPackage=BaseTypes			
Base	ARElement,ARObject,AtpBlueprint,AtpBlueprintable,BaseType,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table D.212: SwBaseType

Enumeration	SwCalibrationAccessEnum
Package	M2::AUTOSARTemplates::CommonStructure::DataDefProperties
Note	Determines the access rights to a data object w.r.t. measurement and calibration.
Literal	***Description***
notAccessible	The element will not be accessible via MCD tools, i.e. will not appear in the ASAP file.
readOnly	The element will only appear as read-only in an ASAP file.
readWrite	The element will appear in the ASAP file with both read and write access.

Table D.213: SwCalibrationAccessEnum

Class	SwComponentPrototype			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Composition			
Note	Role of a software component within a composition.			
Base	ARObject,AtpFeature,AtpPrototype,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
type	SwComponentType	1	tref	Type of the instance. **Stereotypes:** isOfType

Attribute	Datatype	Mul.	Kind	Note

Table D.214: SwComponentPrototype

Class	**SwComponentType (abstract)**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	Base class for AUTOSAR software components.			
Base	ARElement,ARObject,AtpBlueprint,AtpBlueprintable,AtpClassifier,AtpType,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
consistencyNeeds	ConsistencyNeeds	*	aggr	This represents the colelction of ConsistencyNeeds owned by the enclosing SwComponentType. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=preCompileTime
port	PortPrototype	*	aggr	The ports through which this component can communicate. The aggregation of PortPrototype is subject to variability with the purpose to support the conditional existence of PortPrototypes. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=preCompileTime
portGroup	PortGroup	*	aggr	A port group being part of this component. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
swComponentDocumentation	SwComponentDocumentation	0..1	aggr	This adds a documentation to the SwComponentType. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=swComponentDocumentation, variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=-10
unitGroup	UnitGroup	*	ref	This allows for the specification of which UnitGroups are relevant in the context of referencing SwComponentType.

Table D.215: SwComponentType

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

Table D.216: SwConnector

Class	≪atpVariation≫ SwDataDefProps			
Package	M2::AUTOSARTemplates::CommonStructure::DataDefProperties			
Note	This class is a collection of properties relevant for data objects under various aspects. One could consider this class as a "pattern of inheritance by aggregation". The properties can be applied to all objects of all classes in which SwDataDefProps is aggregated. Note that not all of the attributes or associated elements are useful all of the time. Hence, the process definition (e.g. expressed with an OCL or a Document Control Instance MSR-DCI) has the task of implementing limitations. SwDataDefProps covers various aspects: • Structure of the data element for calibration use cases: is it a single value, a curve, or a map, but also the recordLayouts which specify how such elements are mapped/converted to the DataTypes in the programming language (or in AUTOSAR). This is mainly expressed by properties like swRecordLayout and swCalprmAxisSet • Implementation aspects, mainly expressed by swImplPolicy, swVariableAccessImplPolicy, swAddrMethod, swPointerTagetProps, 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, invalidValue • Code generation policy provided by swRecordLayout **Tags:** vh.latestBindingTime=codeGenerationTime			
Base	ARObject			
Attribute	Datatype	Mul.	Kind	Note

Attribute	Datatype	Mul.	Kind	Note
additionalNativeTypeQualifier	NativeDeclarationString	0..1	attr	This attribute is used to declare native qualifiers of the programming language which can neither be deduced from the baseType (e.g. because the data object describes a pointer) nor from other more abstract attributes. Examples are qualifiers like "volatile", "strict" or "enum" of the C-language. All such declarations have to be put into one string. **Tags:** xml.sequenceOffset=235
annotation	Annotation	*	aggr	This aggregation allows to add annotations (yellow pads ...) related to the current data object. **Tags:** xml.roleElement=true; xml.roleWrapperElement=true; xml.sequenceOffset=20; xml.typeElement=false; xml.typeWrapperElement=false
baseType	SwBaseType	0..1	ref	Base type associated with the containing data object. **Tags:** xml.sequenceOffset=50
compuMethod	CompuMethod	0..1	ref	Computation method associated with the semantics of this data object. **Tags:** xml.sequenceOffset=180
dataConstr	DataConstr	0..1	ref	Data constraint for this data object. **Tags:** xml.sequenceOffset=190
displayFormat	DisplayFormatString	0..1	attr	This property describes how a number is to be rendered e.g. in documents or in a measurement and calibration system. **Tags:** xml.sequenceOffset=210

Attribute	Datatype	Mul.	Kind	Note
implementationDataType	ImplementationDataType	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 SwPointerTargetProps), 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:** xml.sequenceOffset=215
invalidValue	ValueSpecification	0..1	aggr	Optional value to express invalidity of the actual data element. **Tags:** xml.sequenceOffset=255
mcFunction	Identifier	0..1	ref	Specifies the name of a "Function" (in the sense of the MC system) to which this data object belongs. This corresponds to the Function in ASAM MCD 2MC /ASAP2 which defines the characteristic resp. which provides the measurement as output. The function name is only used for support of MC systems. It can be predefined on the level of software component design. If it is not predefined, it could be filled out with a reasonable name, e.g. the component prototype name, from the ECU extract. Note: This attribute is deprecated because an explicit model of MC functions can be set up by using the meta-class McFunction. **Tags:** atp.Status=obsolete xml.sequenceOffset=257
swAddrMethod	SwAddrMethod	0..1	ref	Addressing method related to this data object. Via an association to the same SwAddrMethod it can be specified that several DataPrototypes shall be located in the same memory without already specifying the memory section itself. **Tags:** xml.sequenceOffset=30

Attribute	Datatype	Mul.	Kind	Note
swAlignment	AlignmentType	0..1	attr	The attribute describes the intended alignment of the DataPrototype. If the attribute is not defined the alignment is determined by the swBaseType size and the memoryAllocationKeywordPolicy of the referenced SwAddrMethod. **Tags:** xml.sequenceOffset=33
swBitRepresentation	SwBitRepresentation	0..1	aggr	Description of the binary representaion in case of a bit variable. **Tags:** xml.sequenceOffset=60
swCalibrationAccess	SwCalibrationAccessEnum	0..1	attr	Specifies the read or write access by MCD tools for this data object. **Tags:** xml.sequenceOffset=70
swCalprmAxisSet	SwCalprmAxisSet	0..1	aggr	This specifies the properties of the axes in case of a curve or map etc. This is mainly applicable to calibration parameters. **Tags:** xml.sequenceOffset=90
swComparisonVariable	SwVariableRefProxy	*	aggr	Variables used for comparison in an MCD process. **Tags:** xml.sequenceOffset=170; xml.typeElement=false
swDataDependency	SwDataDependency	0..1	aggr	Describes how the value of the data object has to be calculated from the value of another data object (by the MCD system). **Tags:** xml.sequenceOffset=200
swHostVariable	SwVariableRefProxy	0..1	aggr	Contains a reference to a variable which serves as a host-variable for a bit variable. Only applicable to bit objects. **Tags:** xml.sequenceOffset=220; xml.typeElement=false
swImplPolicy	SwImplPolicyEnum	0..1	attr	Implementation policy for this data object. **Tags:** xml.sequenceOffset=230

Attribute	Datatype	Mul.	Kind	Note
swIntendedResolution	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
swInterpolationMethod	Identifier	0..1	ref	This is a keyword identifying the mathematical method to be applied for interpolation. The keyword needs to be related to the interpolation routine which needs to be invoked. **Tags:** xml.sequenceOffset=250
swIsVirtual	Boolean	0..1	attr	This element distinguishes virtual objects. Virtual objects do not appear in the memory, their derivation is much more dependent on other objects and hence they shall have a swDataDependency . **Tags:** xml.sequenceOffset=260
swPointerTargetProps	SwPointerTargetProps	0..1	aggr	Specifies that the containing data object is a pointer to another data object. **Tags:** xml.sequenceOffset=280
swRecordLayout	SwRecordLayout	0..1	ref	Record layout for this data object. **Tags:** xml.sequenceOffset=290
swRefreshTiming	MultidimensionalTime	0..1	aggr	This element specifies the frequency in which the object involved shall be or is called or calculated. This timing can be collected from the task in which write access processes to the variable run. But this cannot be done by the MCD system. So this attribute can be used in an early phase to express the desired refresh timing and later on to specify the real refresh timing. **Tags:** xml.sequenceOffset=300

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
swTextProps	SwTextProps	0..1	aggr	the specific properties if the data object is a text object. **Tags:** xml.sequenceOffset=120
swValueBlockSize	Numerical	0..1	attr	This represents the size of a Value Block **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=80
unit	Unit	0..1	ref	Physical unit associated with the semantics of this data object. This attribute applies if no compuMethod is specified. If both units (this as well as via compuMethod) are specified the units shall be compatible. **Tags:** xml.sequenceOffset=350
valueAxisDataType	ApplicationPrimitiveDataType	0..1	ref	The referenced ApplicationPrimitiveDataType represents the primitive data type of the value axis within a compound primitive (e.g. curve, map). It supersedes CompuMethod, Unit, and BaseType. **Tags:** xml.sequenceOffset=355

Table D.217: SwDataDefProps

Enumeration	**SwImplPolicyEnum**
Package	M2::AUTOSARTemplates::CommonStructure::DataDefProperties
Note	Specifies the implementation strategy with respect to consistency mechanisms of variables.
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.
fixed	This data element is fixed. In particular this indicates, that it might also be implemented e.g. as in place data, (#DEFINE).
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.
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.
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.

Table D.218: SwImplPolicyEnum

Class	**SwPointerTargetProps**			
Package	M2::AUTOSARTemplates::CommonStructure::DataDefProperties			
Note	This element defines, that the data object (which is specified by the aggregating element) contains a reference to another data object or to a function in the CPU code. This corresponds to a pointer in the C-language. The attributes of this element describe the category and the detailed properties of the target which is either a data description or a function signature.			
Base	ARObject			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
functionPointerSignature	BswModuleEntry	0..1	ref	The referenced BswModuleEntry serves as the signature of a function pointer definition. Primary use case: function pointer passed as argument to other function. **Tags:** xml.sequenceOffset=40
swDataDefProps	SwDataDefProps	0..1	aggr	The properties of the target data type. **Tags:** xml.sequenceOffset=30
targetCategory	Identifier	0..1	ref	This specifies the category of the target: • In case of a data pointer, it shall specify the category of the referenced data. • In case of a function pointer, it could be used to denote the category of the referenced BswModuleEntry. Since currently no categories for BswModuleEntry are defined it will be empty. **Tags:** xml.sequenceOffset=5

Table D.219: SwPointerTargetProps

Class	**SwRecordLayout**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::RecordLayout			
Note	Defines how the data objects (variables, calibration parameters etc.) are to be stored in the ECU memory. As an example, this definition specifies the sequence of axis points in the ECU memory. Iterations through axis values are stored within the sub-elements swRecordLayoutGroup. **Tags:** atp.recommendedPackage=SwRecordLayouts			
Base	ARElement,ARObject,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
swRecordLayoutGroup	SwRecordLayoutGroup	1	aggr	This is the top level record layout group. **Tags:** xml.roleElement=true; xml.roleWrapperElement=false; xml.sequenceOffset=20; xml.typeElement=false; xml.typeWrapperElement=false

Table D.220: SwRecordLayout

Class	SwServiceArg			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceProcessTask			
Note	Specifies the properties of a data object exchanged during the call of an SwService, e.g. an argument or a return value.			
Base	ARObject,Identifiable,MultilanguageReferrable,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
direction	ArgumentDirectionEnum	0..1	attr	Specifies the direction of the data transfer. The direction shall indicate the direction of the actual information that is being consumed by the caller and/or the callee, not the direction of formal arguments in C. The attribute is optional for backwards compatibility reasons. For example, if a pointer is used to pass a memory address for the expected result, the direction shall be "out". If a pointer is used to pass a memory address with content to be read by the callee, its direction shall be "in". **Tags:** xml.sequenceOffset=10
swArraysize	ValueList	0..1	aggr	This turns the argument of the service to an array. **Tags:** xml.sequenceOffset=20
swDataDefProps	SwDataDefProps	0..1	aggr	Data properties of this SwServiceArg. **Tags:** xml.sequenceOffset=30

Table D.221: SwServiceArg

Class	SwSystemconst			
Package	M2::AUTOSARTemplates::CommonStructure::SystemConstant			
Note	This element defines a system constant which serves an input to select a particular variation point. In particular a system constant serves as an operand of the binding function (swSyscond) in a Variation point. Note that the binding process can only happen if a value was assigned to to the referenced system constants. **Tags:** atp.recommendedPackage=SwSystemconsts			
Base	ARElement,ARObject,AtpDefinition,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
swDataDefProps	SwDataDefProps	0..1	aggr	This denotes the data defintion properties of the system constant. In particular it is the limits and - in case the system constant is an enumeration - the compu method. **Tags:** xml.sequenceOffset=40

Table D.222: SwSystemconst

Class	≪atpMixedString≫ **SwSystemconstDependentFormula (abstract)**			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling			
Note	This class represents an expression depending on system constants.			
Base	ARObject,FormulaExpression			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
sysc	SwSystemconst	1	ref	This refers to a system constant. The internal (coded) value of the system constant shall be used. **Tags:** xml.sequenceOffset=50
syscString	SwSystemconst	1	ref	syscString indicates that the referenced systm constant shall be evaluated as a string according to [TPS_SWCT_01431].

Table D.223: SwSystemconstDependentFormula

Class	**SwSystemconstValue**			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling			
Note	This meta-class assigns a particular value to a system constant.			
Base	ARObject			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
annotation	Annotation	*	aggr	This provides the ability to add information why the value is set like it is. **Tags:** xml.sequenceOffset=30
swSystemconst	SwSystemconst	1	ref	This is the system constant to which the value applies. **Tags:** xml.sequenceOffset=10
value	Numerical	1	attr	This is the particular value of a system constant. It is specified as Numerical. Further restrictions may apply by the definition of the system constant. The value attribute defines the internal value of the SwSystemconst as it is processed in the Formula Language. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=20

Table D.224: SwSystemconstValue

Class	SwSystemconstantValueSet			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling			
Note	This meta-class represents the ability to specify a set of system constant values. **Tags:** atp.recommendedPackage=SwSystemconstantValueSets			
Base	ARElement,ARObject,CollectableElement,Identifiable,Multilanguage Referrable,PackageableElement,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
swSystem constantVa lue	SwSystemconst Value	*	aggr	This is one particular value of a system constant.

Table D.225: SwSystemconstantValueSet

Class	≪atpMixed≫ SwValues			
Package	M2::AUTOSARTemplates::CommonStructure::CalibrationValue			
Note	This meta-class represents a list of values. These values can either be the input values of a curve (abscissa values) or the associated values (ordinate values). In case of multidimensional structures, the values are ordered such that the lowest index runs the fastest. In particular for maps and cuboids etc. the resulting long value list can be subsectioned using ValueGroup. But the processing needs to be done as if vg is not there. Note that numerical values and textual values should not be mixed.			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
v	Numerical	1	attr	This is a non variant Value. It is provided for sake of Compatibility to ASAM CDF. **Tags:** xml.sequenceOffset=40
vf	Numerical	1	attr	This allows to specify the value as VariationPoint. It is distinguished to non variant for sake of compatibility to ASAM CDF 2.0. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=20
vg	ValueGroup	1	aggr	This allows to have intersections in the values in order to support specific rendering (eg. using stylesheets). For tools it is important that the v values are always processed in the same (flattened) order and the tool is able to interpret it without respecting vg. **Tags:** xml.sequenceOffset=50

Attribute	Datatype	Mul.	Kind	Note
vt	VerbatimString	1	ref	This represents the values of textual data elements (Strings). Note that vt uses the | to separate the values for the different bitfield masks in case that the semantics of the related DataPrototype is described by means of a BITFIELD_TEXTTABLE in the associated CompuMethod. **Tags:** xml.sequenceOffset=30
vtf	NumericalOrText	1	aggr	Thias aggregation represents the ability to provide a value that is either numerical or text which existence is subject to variability. From the formal point of view, the aggregation needs to have the multiplicity 1 because SwValues is modelled with stereotype «atpMixed». Nevertheless, the existence of vtf is optional and subject to constraints. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime

Table D.226: SwValues

Class	SwcBswMapping			
Package	M2::AUTOSARTemplates::CommonStructure::SwcBswMapping			
Note	Maps an SwcInternalBehavior to an BswInternalBehavior. This is required to coordinate the API generation and the scheduling for AUTOSAR Service Components, ECU Abstraction Components and Complex Driver Components by the RTE and the BSW scheduling mechanisms. **Tags:** atp.recommendedPackage=SwcBswMappings			
Base	ARElement,ARObject,AtpClassifier,AtpFeature,AtpStructureElement,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
bswBehavior	BswInternalBehavior	1	ref	The mapped BswInternalBehavior
runnableMapping	SwcBswRunnableMapping	*	aggr	A mapping between a pair of SWC and BSW runnables. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
swcBehavior	SwcInternalBehavior	1	ref	The mapped SwcInternalBehavior.
synchronizedModeGroup	SwcBswSynchronizedModeGroupPrototype	*	aggr	A pair of SWC and BSW mode group prototypes to be synchronized by the scheduler. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime

Attribute	Datatype	Mul.	Kind	Note
synchronizedTrigger	SwcBswSynchronizedTrigger	*	aggr	A pair of SWC and BSW Triggers to be synchronized by the scheduler. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime

Table D.227: SwcBswMapping

Class	**SwcBswRunnableMapping**			
Package	M2::AUTOSARTemplates::CommonStructure::SwcBswMapping			
Note	Maps a BswModuleEntity to a RunnableEntity if it is implemented as part of a BSW module (in the case of an AUTOSAR Service, a Complex Driver or an ECU Abstraction). The mapping can be used by a tool to find relevant information on the behavior, e.g. whether the bswEntity shall be running in interrupt context.			
Base	ARObject			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
bswEntity	BswModuleEntity	1	ref	The mapped BswModuleEntity
swcRunnable	RunnableEntity	1	ref	The mapped SWC runnable.

Table D.228: SwcBswRunnableMapping

Class	**SwcBswSynchronizedTrigger**			
Package	M2::AUTOSARTemplates::CommonStructure::SwcBswMapping			
Note	Synchronizes a Trigger provided by a component via a port with a Trigger provided by a BSW module or cluster.			
Base	ARObject			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
bswTrigger	Trigger	1	ref	The BSW Trigger.
swcTrigger	Trigger	1	iref	The SWC Trigger provided by a particular port.

Table D.229: SwcBswSynchronizedTrigger

Class	**SwcImplementation**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::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			
Base	ARElement,ARObject,CollectableElement,Identifiable,Implementation,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
behavior	SwcInternalBehavior	1	ref	The internal behavior implemented by this Implementation.

Attribute	Datatype	Mul.	Kind	Note
perInstanceMemorySize	PerInstanceMemorySize	*	aggr	Allows a definition of the size of the per-instance memory for this implementation. The aggregation of PerInstanceMemorySize 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:** vh.latestBindingTime=preCompileTime
requiredRTEVendor	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 D.230: SwcImplementation

Class	SwcInternalBehavior			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior			
Note	The SwcInternalBehavior of an AtomicSwComponentType describes the relevant aspects of the software-component with respect to the RTE, i.e. the RunnableEntities and the RTEEvents they respond to.			
Base	ARObject,AtpClassifier,AtpFeature,AtpStructureElement,Identifiable,InternalBehavior,MultilanguageReferrable,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
arTypedPerInstanceMemory	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 components implementations. Typically different algorithms in the implementation are requiring different number of memory objects. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime

Attribute	Datatype	Mul.	Kind	Note
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 RTE events. Note: the number of RTE events might vary due to the conditional existence of PortPrototypes using DataReceivedEvents or due to different scheduling needs of algorithms. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variation Point.shortLabel vh.latestBindingTime=preCompileTime
explicitInterRunnableVariable	VariableDataPrototype	*	aggr	Implement state message semantics for establishing communication among runnables of the same component. The aggregation of explicitInterRunnableVariable 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:** vh.latestBindingTime=preCompileTime
handleTerminationAndRestart	HandleTerminationAndRestartEnum	1	attr	This attribute controls the behavior with respect to stopping and restarting. The corresponding AtomicSwComponentType may either not support stop and restart, or support only stop, or support both stop and restart.
implicitInterRunnableVariable	VariableDataPrototype	*	aggr	Implement state message semantics for establishing communication among runnables of the same component. The aggregation of implicitInterRunnableVariable 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:** vh.latestBindingTime=preCompileTime
includedDataTypeSet	IncludedDataTypeSet	*	aggr	The includedDataTypeSet is used by a software component for its implementation.
includedModeDeclarationGroupSet	IncludedModeDeclarationGroupSet	*	aggr	This aggregation represents the included ModeDeclarationGroups

Attribute	Datatype	Mul.	Kind	Note
instantiationDataDefProps	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 PortPrototypes and component local memories like "perInstanceParameter" or "arTypedPerInstanceMemory". **Stereotypes:** atpVariation **Tags:** 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:** 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:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, 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:** vh.latestBindingTime=preCompileTime

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
runnable	RunnableEntity	1..*	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 RunnableEntities. Note: the number of RunnableEntities might vary due to the conditional existence of PortPrototypes using DataReceivedEvents or due to different scheduling needs of algorithms. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variation Point.shortLabel vh.latestBindingTime=preCompileTime
serviceDep endency	SwcServiceDep endency	*	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 SwcInternalBehavior 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 «atpSplitable». **Stereotypes:** atpVariation **Tags:** atp.Splitkey=serviceDependency.short Name, variationPoint.shortLabel vh.latestBindingTime=preCompileTime
sharedPar ameter	ParameterData Prototype	*	aggr	Defines parameter(s) or characteristic value(s) shared between SwComponentPrototypes of the same SwComponentType The aggregation of sharedParameter is subject to variability with the purpose to support variability in the software components implementations. Typically different algorithms in the implementation are requiring different number of memory objects. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variation Point.shortLabel vh.latestBindingTime=preCompileTime
supportsM ultipleInsta ntiation	Boolean	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).

Attribute	Datatype	Mul.	Kind	Note
variationPointProxy	VariationPointProxy	*	aggr	Proxy of a variation points in the C/C++ implementation.

Table D.231: SwcInternalBehavior

Class	SwcModeManagerErrorEvent			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::RTEEvents			
Note	This represents the ability to react on errors occurring during mode handling.			
Base	ARObject,AbstractEvent,AtpClassifier,AtpFeature,AtpStructureElement,Identifiable,MultilanguageReferrable,RTEEvent,Referrable			
Attribute	Datatype	Mul.	Kind	Note
modeGroup	ModeDeclarationGroupPrototype	1	iref	This represents the ModeDeclarationGroupPrototype for which the error behavior of the mode manager applies.

Table D.232: SwcModeManagerErrorEvent

Class	SwcModeSwitchEvent			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::RTEEvents			
Note	This event is raised upon a received mode change.			
Base	ARObject,AbstractEvent,AtpClassifier,AtpFeature,AtpStructureElement,Identifiable,MultilanguageReferrable,RTEEvent,Referrable			
Attribute	Datatype	Mul.	Kind	Note
activation	ModeActivationKind	1	attr	Specifies if the event is activated on entering or exiting the referenced Mode.
mode (ordered)	ModeDeclaration	1..2	iref	Reference to one or two Modes that initiate the SwcModeSwitchEvent.

Table D.233: SwcModeSwitchEvent

Class	SwcServiceDependency			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::ServiceMapping			
Note	Specialization of ServiceDependency in the context of an SwcInternalBehavior. It allows to associate ports, port groups and (in special cases) data defined for an atomic software component to a given ServiceNeeds element.			
Base	ARObject,AtpClassifier,AtpFeature,AtpStructureElement,Identifiable,MultilanguageReferrable,Referrable,ServiceDependency			
Attribute	Datatype	Mul.	Kind	Note
assignedData	RoleBasedDataAssignment	*	aggr	Defines the role of an associated data object of the same component. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime

Attribute	Datatype	Mul.	Kind	Note
assignedPort	RoleBasedPortAssignment	*	aggr	Defines the role of an associated port of the same component. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=assignedPort, variationPoint.shortLabel vh.latestBindingTime=preCompileTime
representedPortGroup	PortGroup	0..1	ref	This reference specifies an association between the ServiceNeeeds and a PortGroup, for example to request a communication mode which applies for communication via these ports. The referred PortGroup shall be local to this atomic SWC, but via the links between the PortGroups, a tool can evaluate this information such that all the ports linked via this port group on the same ECU can be found.
serviceNeeds	ServiceNeeds	1	aggr	The associated ServiceNeeds.

Table D.234: SwcServiceDependency

Class	SymbolProps			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	This meta-class represents the ability to attach with the symbol attribute a symbolic name that is conform to C language requirements to another meta-class, e.g. AtomicSwComponentType, that is a potential subject to a name clash on the level of RTE source code.			
Base	ARObject,ImplementationProps,Referrable			
Attribute	Datatype	Mul.	Kind	Note
–	–	–	–	–

Table D.235: SymbolProps

Class	SynchronousServerCallPoint			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::ServerCall			
Note	This means that the RunnableEntity is supposed to perform a blocking wait for a response from the server.			
Base	ARObject,AtpClassifier,AtpFeature,AtpStructureElement,Identifiable,MultilanguageReferrable,Referrable,ServerCallPoint			
Attribute	Datatype	Mul.	Kind	Note
calledFromWithinExclusiveArea	ExclusiveAreaNestingOrder	0..1	ref	This indicates that the call point is located at the deepest level inside one or more ExclusiveAreas that are nested in the given order.

Table D.236: SynchronousServerCallPoint

Class	**SystemMapping**			
Package	M2::AUTOSARTemplates::SystemTemplate			
Note	The system mapping aggregates all mapping aspects (mapping of SW components to ECUs, mapping of data elements to signals, and mapping constraints).			
Base	ARObject,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
dataMapping	DataMapping	*	aggr	The data mappings defined. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=postBuild
ecuResourceMapping	ECUMapping	*	aggr	Mapping of hardware related topology elements onto their counterpart definitions in the ECU Resource Template. atpVariation: The ECU Resource type might be variable. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=systemDesignTime
mappingConstraint	MappingConstraint	*	aggr	Constraints that limit the mapping freedom for the mapping of SW components to ECUs. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=systemDesignTime
pncMapping	PncMapping	*	aggr	**Stereotypes:** atpVariation**Tags:** vh.latestBindingTime=systemDesignTime
resourceEstimation	EcuResourceEstimation	*	aggr	Resource estimations for this set of mappings, zero or one per ECU instance. atpVariation: Used ECUs are variable. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=systemDesignTime
signalPathConstraint	SignalPathConstraint	*	aggr	Constraints that limit the mapping freedom for the mapping of data elements to signals. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=systemDesignTime
swImplMapping	SwcToImplMapping	*	aggr	The mappings of AtomicSoftwareComponentInstances to Implementations. atpVariation: Derived, because SwcToEcuMapping is variable. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
swMapping	SwcToEcuMapping	*	aggr	The mappings of SW components to ECUs. atpVariation: SWC shall be mapped to other ECUs. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=systemDesignTime

Table D.237: SystemMapping

Class	SystemSignal			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::FibexCore::CoreCommunication			
Note	The system signal represents the communication system's view of data exchanged between SW components which reside on different ECUs. The system signals allow to represent this communication in a flattened structure, with exactly one system signal defined for each data element prototype sent and received by connected SW component instances. **Tags:** atp.recommendedPackage=SystemSignals			
Base	ARElement,ARObject,CollectableElement,Identifiable,Multilanguage Referrable,PackageableElement,Referrable			
Attribute	Datatype	Mul.	Kind	Note
dynamicLength	Boolean	1	attr	The length of dynamic length signals is variable in run-time. Only a maximum length of such a signal is specified in the configuration (attribute length in ISignal element).
physicalProps	SwDataDefProps	0..1	aggr	Specification of the physical representation.

Table D.238: SystemSignal

Class	SystemSignalGroup			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::FibexCore::CoreCommunication			
Note	A signal group refers to a set of signals that must always be kept together. A signal group is used to guarantee the atomic transfer of AUTOSAR composite data types. The SystemSignalGroup defines a signal grouping on VFB level. On cluster level the Signal grouping is described by the ISignalGroup element. **Tags:** atp.recommendedPackage=SystemSignalGroups			
Base	ARElement,ARObject,CollectableElement,Identifiable,Multilanguage Referrable,PackageableElement,Referrable			
Attribute	Datatype	Mul.	Kind	Note
systemSignal	SystemSignal	*	ref	Reference to a set of SystemSignals that must always be kept together.

Table D.239: SystemSignalGroup

Class	TextTableMapping			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	Defines the mapping of two DataPrototypes typed by AutosarDataTypes that refer to CompuMethods of category TEXTTABLE.			
Base	ARObject			
Attribute	Datatype	Mul.	Kind	Note
identicalMapping	Boolean	1	attr	If identicalMapping is set == true the values of the two referenced DataPrototypes do not need any conversion of the values.
mappingDirection	MappingDirectionEnum	1	attr	Specifies the conversion direction for which the TextTableMapping is applicable.

Attribute	Datatype	Mul.	Kind	Note
valuePair	TextTableValuePair	*	aggr	Defines a pair of values which are translated into each other.

Table D.240: TextTableMapping

Class	TextValueSpecification			
Package	M2::AUTOSARTemplates::CommonStructure::Constants			
Note	The purpose of TextValueSpecification is to define the labels that correspond to enumeration values.			
Base	ARObject,ValueSpecification			
Attribute	Datatype	Mul.	Kind	Note
value	VerbatimString	1	ref	This is the value itself. Note that vt uses the | operator to separate the values for the different bitfield masks in case that the semantics of the related DataPrototype is described by means of a BITFIELD_TEXTTABLE in the associated CompuMethod.

Table D.241: TextValueSpecification

Class	TimingEvent			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::RTE Events			
Note	TimingEvent references the RunnableEntity that need to be started in response to the TimingEvent			
Base	ARObject,AbstractEvent,AtpClassifier,AtpFeature,AtpStructure Element,Identifiable,MultilanguageReferrable,RTEEvent,Referrable			
Attribute	Datatype	Mul.	Kind	Note
period	TimeValue	1	attr	Period of timing event in seconds. The value of this attribute shall be greater than zero.

Table D.242: TimingEvent

Class	TransmissionAcknowledgementRequest			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication			
Note	Requests transmission acknowledgement that data has been sent successfully. Success/failure is reported via a SendPoint of a RunnableEntity.			
Base	ARObject			
Attribute	Datatype	Mul.	Kind	Note
timeout	TimeValue	1	attr	Number of seconds before an error is reported or in case of allowed redundancy, the value is sent again.

Table D.243: TransmissionAcknowledgementRequest

Class	**Trigger**			
Package	M2::AUTOSARTemplates::CommonStructure::TriggerDeclaration			
Note	A trigger which is provided (i.e. released) or required (i.e. used to activate something) in the given context.			
Base	ARObject,AtpClassifier,AtpFeature,AtpStructureElement,Identifiable,MultilanguageReferrable,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
swImplPolicy	SwImplPolicyEnum	0..1	attr	This attribute, when set to value queued, allows for a queued processing of Triggers.
triggerPeriod	MultidimensionalTime	0..1	aggr	Optional definition of a period in case of a periodically (time or angle) driven external trigger.

Table D.244: Trigger

Class	**TriggerInterface**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	A trigger interface declares a number of triggers that can be sent by an trigger source. **Tags:** atp.recommendedPackage=PortInterfaces			
Base	ARElement,ARObject,AtpBlueprint,AtpBlueprintable,AtpClassifier,AtpType,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,PortInterface,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
trigger	Trigger	1..*	aggr	The Trigger of this trigger interface.

Table D.245: TriggerInterface

Class	**TriggerInterfaceMapping**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	Defines the mapping of unequal named Triggers in context of two different TriggerInterfaces.			
Base	ARObject,AtpBlueprint,AtpBlueprintable,Identifiable,MultilanguageReferrable,PortInterfaceMapping,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
triggerMapping	TriggerMapping	1..*	aggr	Mapping of two Trigger in two different TriggerInterface

Table D.246: TriggerInterfaceMapping

Class	**Unit**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::Units			
Note	This is a physical measurement unit. All units that might be defined should stem from SI units. In order to convert one unit into another factor and offset are defined. For the calculation from SI-unit to the defined unit the factor (factorSiToUnit) and the offset (offsetSiToUnit) are applied: unit = siUnit * factorSiToUnit + offsetSiToUnit For the calculation from a unit to SI-unit the reciprocal of the factor (factorSiToUnit) and the negation of the offset (offsetSiToUnit) are applied: siUnit = (unit - offsetSiToUnit) / factorSiToUnit **Tags:** atp.recommendedPackage=Units			
Base	ARElement,ARObject,CollectableElement,Identifiable,Multilanguage Referrable,PackageableElement,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
displayName	SingleLanguage UnitNames	0..1	aggr	This specifies how the unit shall be displayed in documents or in user interfaces of tools.The displayName corresponds to the Unit.Display in an ASAM MCD-2MC file. **Tags:** xml.sequenceOffset=20
factorSiToUnit	Float	0..1	attr	This is the factor for the conversion from and to siUnits. **Tags:** xml.sequenceOffset=30
offsetSiToUnit	Float	0..1	attr	This is the offset for the conversion from and to siUnits. **Tags:** xml.sequenceOffset=40
physicalDimension	PhysicalDimension	0..1	ref	This association represents the physical dimension to which the unit belongs to. Note that only values with units of the same physical dimensions might be converted. **Tags:** xml.sequenceOffset=50

Table D.247: Unit

Class	≪**atpMixed**≫ **ValueList**			
Package	M2::AUTOSARTemplates::CommonStructure::DataDefProperties			
Note	This is a generic list of numerical values.			
Base	ARObject			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
v	Numerical	1	attr	This is a particular numerical value without variation. **Tags:** xml.sequenceOffset=30

Attribute	Datatype	Mul.	Kind	Note
vf (ordered)	Numerical	*	attr	This is one entry in the list of numerical values **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.roleElement=true; xml.roleWrapperElement=false; xml.typeElement=false; xml.typeWrapperElement=false

Table D.248: ValueList

Class	**ValueSpecification (abstract)**			
Package	M2::AUTOSARTemplates::CommonStructure::Constants			
Note	Base class for expressions leading to a value which can be used to initialize a data object.			
Base	ARObject			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
shortLabel	Identifier	0..1	ref	This can be used to identify particular value specifications for human readers, for example elements of a record type.

Table D.249: ValueSpecification

Class	**VariableAccess**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::DataElements			
Note	The presence of a VariableAccess implies that a RunnableEntity needs access to a VariableDataPrototype. The kind of access is specified by the role in which the class is used.			
Base	ARObject,AtpClassifier,AtpFeature,AtpStructureElement,Identifiable,MultilanguageReferrable,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
accessedVariable	AutosarVariableRef	1	aggr	This denotes the accessed variable.
scope	VariableAccessScopeEnum	0..1	attr	This attribute allows for constraining the scope of the corresponding communication. For example, it possible to express whether the communication is intended to cross the boundary of an ECU or whether it is intended not to cross the boundary of a single partition.

Table D.250: VariableAccess

Class	VariableAndParameterInterfaceMapping			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	Defines the mapping of VariableDataPrototypes or ParameterDataPrototypes in context of two different SenderReceiverInterfaces, NvDataInterfaces or ParameterInterfaces.			
Base	ARObject,AtpBlueprint,AtpBlueprintable,Identifiable,MultilanguageReferrable,PortInterfaceMapping,Referrable			
Attribute	Datatype	Mul.	Kind	Note
dataMapping	DataPrototypeMapping	1..*	aggr	Defines the mapping of two particular VariableDataPrototypes or ParameterDataPrototypes with unequal names and/or unequal semantic (resolution or range) in context of two different SenderReceiverInterfaces, NvDataInterfaces or ParameterInterfaces

Table D.251: VariableAndParameterInterfaceMapping

Class	VariableDataPrototype			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::DataPrototypes			
Note	A VariableDataPrototype is used to contain values in an ECU application. This means that most likely a VariableDataPrototype allocates "static" memory on the ECU. In some cases optimization strategies might lead to a situation where the memory allocation can be avoided. In particular, the value of a VariableDataPrototype is likely to change as the ECU on which it is used executes.			
Base	ARObject,AtpFeature,AtpPrototype,AutosarDataPrototype,DataPrototype,Identifiable,MultilanguageReferrable,Referrable			
Attribute	Datatype	Mul.	Kind	Note
initValue	ValueSpecification	0..1	aggr	Specifies initial value(s) of the VariableDataPrototype

Table D.252: VariableDataPrototype

Class	VariationPoint			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling			
Note	This meta-class represents the ability to express a "structural variation point". The container of the variation point is part of the selected variant if swSyscond evaluates to true and each postBuildVariantCriterion is fulfilled.			
Base	ARObject			
Attribute	Datatype	Mul.	Kind	Note
desc	MultiLanguageOverviewParagraph	0..1	aggr	This allows to describe shortly the purpose of the variation point. **Tags:** xml.sequenceOffset=20

Attribute	Datatype	Mul.	Kind	Note
blueprintCondition	DocumentationBlock	0..1	aggr	This represents a description that documents how the variation point shall be resolved when deriving objects from the blueprint. Note that variationPoints are not allowed within a blueprintCondition. **Tags:** xml.sequenceOffset=28
formalBlueprintCondition	BlueprintFormula	0..1	aggr	This denotes a formal blueprintCondition. This shall be not in contradiction with blueprintCondition. It is recommanded only to use one of the two. **Tags:** xml.sequenceOffset=29
postBuildVariantCondition	PostBuildVariantCondition	*	aggr	This is the set of post build variant conditions which all shall be fulfilled in order to (postbuild) bind the variation point. **Tags:** xml.sequenceOffset=40
sdg	Sdg	0..1	aggr	An optional special data group is attached to every variation point. These data can be used by external software systems to attach application specific data. For example, a variant management system might add an identifier, an URL or a specific classifier. **Tags:** xml.sequenceOffset=50
shortLabel	Identifier	0..1	ref	This provides a name to the particular variation point to support the RTE generator. It is necessary for supporting splitable aggregations and if binding time is later than codeGenerationTime, as well as some RTE conditions. It needs to be unique with in the enclosing Identifiables with the same ShortName. **Tags:** xml.sequenceOffset=10
swSyscond	ConditionByFormula	0..1	aggr	This condition acts as Binding Function for the VariationPoint. Note that the mulitplicity is 0..1 in order to support pure postBuild variants. **Tags:** xml.sequenceOffset=30

Table D.253: VariationPoint

Class	**VariationPointProxy**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::Variant Handling			
Note	The VariationPointProxy represents variation points of the C/C++ implementation. In case of bindingTime = compileTime the RTE provides defines which can be used for Pre Processor directives to implement compileTime variability.			
Base	ARObject,Identifiable,MultilanguageReferrable,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
conditionAccess	ConditionByFormula	0..1	aggr	This condition acts as Binding Function for the VariationPoint.
implementationDataType	ImplementationDataType	0..1	ref	This association to ImplementationDataType shall be taken as an implementation hint by the RTE generator.
postBuildValueAccess	PostBuildVariantCriterion	0..1	ref	This represents the applicable PostBuildVariantCriterion in the context of a VariationPointProxy. Note that the technical details how to access the particular postBuildValueAccess are still considered internal to the RTE and are consequently not standardized.
postBuildVariantCondition	PostBuildVariantCondition	*	aggr	This represents that applicable PostBuoldVariantCondition in the context of aVariationPointProxy.
valueAccess	AttributeValueVariationPoint	0..1	aggr	This value acts as Binding Function for the VariationPoint.

Table D.254: VariationPointProxy

Class	**WaitPoint**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::RTE Events			
Note	This defines a wait-point for which the RunnableEntity can wait.			
Base	ARObject,Identifiable,MultilanguageReferrable,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
timeout	TimeValue	1	attr	Time in seconds before the WaitPoint times out and the blocking wait call returns with an error indicating the timeout.
trigger	RTEEvent	1	ref	This is the RTEEvent this WaitPoint is waiting for.

Table D.255: WaitPoint

Class	≪atpVariation≫ **SwDataDefProps**			
Package	M2::AUTOSARTemplates::CommonStructure::DataDefProperties			
Note	This class is a collection of properties relevant for data objects under various aspects. One could consider this class as a "pattern of inheritance by aggregation". The properties can be applied to all objects of all classes in which SwDataDefProps is aggregated. Note that not all of the attributes or associated elements are useful all of the time. Hence, the process definition (e.g. expressed with an OCL or a Document Control Instance MSR-DCI) has the task of implementing limitations. SwDataDefProps covers various aspects: • Structure of the data element for calibration use cases: is it a single value, a curve, or a map, but also the recordLayouts which specify how such elements are mapped/converted to the DataTypes in the programming language (or in AUTOSAR). This is mainly expressed by properties like swRecordLayout and swCalprmAxisSet • Implementation aspects, mainly expressed by swImplPolicy, swVariableAccessImplPolicy, swAddrMethod, swPointerTagetProps, 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, invalidValue • Code generation policy provided by swRecordLayout **Tags:** vh.latestBindingTime=codeGenerationTime			
Base	ARObject			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
additionalNativeTypeQualifier	NativeDeclarationString	0..1	attr	This attribute is used to declare native qualifiers of the programming language which can neither be deduced from the baseType (e.g. because the data object describes a pointer) nor from other more abstract attributes. Examples are qualifiers like "volatile", "strict" or "enum" of the C-language. All such declarations have to be put into one string. **Tags:** xml.sequenceOffset=235
annotation	Annotation	*	aggr	This aggregation allows to add annotations (yellow pads ...) related to the current data object. **Tags:** xml.roleElement=true; xml.roleWrapperElement=true; xml.sequenceOffset=20; xml.typeElement=false; xml.typeWrapperElement=false
baseType	SwBaseType	0..1	ref	Base type associated with the containing data object. **Tags:** xml.sequenceOffset=50

Attribute	Datatype	Mul.	Kind	Note
compuMethod	CompuMethod	0..1	ref	Computation method associated with the semantics of this data object. **Tags:** xml.sequenceOffset=180
dataConstr	DataConstr	0..1	ref	Data constraint for this data object. **Tags:** xml.sequenceOffset=190
displayFormat	DisplayFormatString	0..1	attr	This property describes how a number is to be rendered e.g. in documents or in a measurement and calibration system. **Tags:** xml.sequenceOffset=210
implementationDataType	ImplementationDataType	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 SwPointerTargetProps), 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:** xml.sequenceOffset=215
invalidValue	ValueSpecification	0..1	aggr	Optional value to express invalidity of the actual data element. **Tags:** xml.sequenceOffset=255

Attribute	Datatype	Mul.	Kind	Note
mcFunction	Identifier	0..1	ref	Specifies the name of a "Function" (in the sense of the MC system) to which this data object belongs. This corresponds to the Function in ASAM MCD 2MC /ASAP2 which defines the characteristic resp. which provides the measurement as output. The function name is only used for support of MC systems. It can be predefined on the level of software component design. If it is not predefined, it could be filled out with a reasonable name, e.g. the component prototype name, from the ECU extract. Note: This attribute is deprecated because an explicit model of MC functions can be set up by using the meta-class McFunction. **Tags:** atp.Status=obsolete xml.sequenceOffset=257
swAddrMethod	SwAddrMethod	0..1	ref	Addressing method related to this data object. Via an association to the same SwAddrMethod it can be specified that several DataPrototypes shall be located in the same memory without already specifying the memory section itself. **Tags:** xml.sequenceOffset=30
swAlignment	AlignmentType	0..1	attr	The attribute describes the intended alignment of the DataPrototype. If the attribute is not defined the alignment is determined by the swBaseType size and the memoryAllocationKeywordPolicy of the referenced SwAddrMethod. **Tags:** xml.sequenceOffset=33
swBitRepresentation	SwBitRepresentation	0..1	aggr	Description of the binary representaion in case of a bit variable. **Tags:** xml.sequenceOffset=60
swCalibrationAccess	SwCalibrationAccessEnum	0..1	attr	Specifies the read or write access by MCD tools for this data object. **Tags:** xml.sequenceOffset=70
swCalprmAxisSet	SwCalprmAxisSet	0..1	aggr	This specifies the properties of the axes in case of a curve or map etc. This is mainly applicable to calibration parameters. **Tags:** xml.sequenceOffset=90
swComparisonVariable	SwVariableRefProxy	*	aggr	Variables used for comparison in an MCD process. **Tags:** xml.sequenceOffset=170; xml.typeElement=false

Attribute	Datatype	Mul.	Kind	Note
swDataDependency	SwDataDependency	0..1	aggr	Describes how the value of the data object has to be calculated from the value of another data object (by the MCD system). **Tags:** xml.sequenceOffset=200
swHostVariable	SwVariableRefProxy	0..1	aggr	Contains a reference to a variable which serves as a host-variable for a bit variable. Only applicable to bit objects. **Tags:** xml.sequenceOffset=220; xml.typeElement=false
swImplPolicy	SwImplPolicyEnum	0..1	attr	Implementation policy for this data object. **Tags:** xml.sequenceOffset=230
swIntendedResolution	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
swInterpolationMethod	Identifier	0..1	ref	This is a keyword identifying the mathematical method to be applied for interpolation. The keyword needs to be related to the interpolation routine which needs to be invoked. **Tags:** xml.sequenceOffset=250
swIsVirtual	Boolean	0..1	attr	This element distinguishes virtual objects. Virtual objects do not appear in the memory, their derivation is much more dependent on other objects and hence they shall have a swDataDependency . **Tags:** xml.sequenceOffset=260
swPointerTargetProps	SwPointerTargetProps	0..1	aggr	Specifies that the containing data object is a pointer to another data object. **Tags:** xml.sequenceOffset=280
swRecordLayout	SwRecordLayout	0..1	ref	Record layout for this data object. **Tags:** xml.sequenceOffset=290

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
swRefresh Timing	Multidimensiona lTime	0..1	aggr	This element specifies the frequency in which the object involved shall be or is called or calculated. This timing can be collected from the task in which write access processes to the variable run. But this cannot be done by the MCD system. So this attribute can be used in an early phase to express the desired refresh timing and later on to specify the real refresh timing. **Tags:** xml.sequenceOffset=300
swTextPro ps	SwTextProps	0..1	aggr	the specific properties if the data object is a text object. **Tags:** xml.sequenceOffset=120
swValueBl ockSize	Numerical	0..1	attr	This represents the size of a Value Block **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=80
unit	Unit	0..1	ref	Physical unit associated with the semantics of this data object. This attribute applies if no compuMethod is specified. If both units (this as well as via compuMethod) are specified the units shall be compatible. **Tags:** xml.sequenceOffset=350
valueAxisD ataType	ApplicationPrimi tiveDataType	0..1	ref	The referenced ApplicationPrimitiveDataType represents the primitive data type of the value axis within a compound primitive (e.g. curve, map). It supersedes CompuMethod, Unit, and BaseType. **Tags:** xml.sequenceOffset=355

Table D.256: SwDataDefProps

E Referenced ECUC Configuration Parameters

ComGroupSignal

SWS Item	[ECUC_Com_00520]
Container Name	ComGroupSignal
Description	This container contains the configuration parameters of group signals. I.e. signals that are included within a signal group. **Attributes:** postBuildChangeable=true
Configuration Parameters	

Name	ComBitPosition [ECUC_Com_00259]		
Description	Starting position within the I-PDU. This parameter refers to the position in the I-PDU and not in the shadow buffer. If the endianness conversion is configured to Opaque the parameter ComBitPosition shall define the bit0 of the first byte like in little endian byte order		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default Value			
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

Name	ComBitSize [ECUC_Com_00158]		
Description	Size in bits, for integer signal types. For ComSignalType UINT8_N and UINT8_DYN the size shall be configured by ComSignalLength. For ComSignalTypes FLOAT32 and FLOAT64 the size is already defined by the signal type and therefore may be omitted.		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	0 .. 64		
Default Value			
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

Name	ComHandleId [ECUC_Com_00165]		
Description	The numerical value used as the ID. For signals it is required by the API calls Com_UpdateShadowSignal, Com_ReceiveShadowSignal and Com_InvalidateShadowSignal. For signals groups it is required by the Com_SendSignalGroup and Com_ReceiveSignalGroup calls.		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 65535		
Default Value			
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	–	
Scope / Dependency	scope: ECU		

Name	ComSignalDataInvalidValue [ECUC_Com_00391]		
Description	Defines the data invalid value of the signal. In case the ComSignalType is UINT8, UINT16, UINT32, SINT8, SINT16, SINT32 the string shall be interpreted as defined in the chapter Integer Type in the AUTOSAR EcuC specification. In case the ComSignalType is FLOAT32, FLOAT64 the string shall be interpreted as defined in the chapter Float Type in the AUTOSAR EcuC specification. In case the ComSignalType is BOOLEAN the string shall be interpreted as defined in the chapter Boolean Type in the AUTOSAR EcuC specification. In case the ComSignal is a UINT8_N, UINT8_DYN the string shall be interpreted as a decimal representation of the characters separated by blanks, e.g. "97 98 100" means a string "abd", where the char "a" is in byte 0(lowest address), "b" is in byte 1, and "d" is in byte 2 and (highest address). For the ComSignalType UINT8_DYN the dynamic length shall be set to the number of configured characters. An empty string "" shall be interpretated as 0-sized dynamic signal.		
Multiplicity	0..1		
Type	EcucStringParamDef		
Default Value			
Regular Expression			
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	–	
Scope / Dependency	scope: local dependency: In case of UINT8_N the length of ComSignalDataInvalidValue has to be the same as ComSignalLength.		

Name	ComSignalEndianness [ECUC_Com_00157]	
Description	Defines the endianness of the signal's network representation.	
Multiplicity	1	
Type	EcucEnumerationParamDef	
Range	BIG_ENDIAN	

	LITTLE_ENDIAN		
	OPAQUE		
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

Name	ComSignalInitValue [ECUC_Com_00170]		
Description	Initial value for this signal. In case of UINT8_N the default value is a string of length ComSignalLength with all bytes set to 0x00. In case of UINT8_DYN the initial size shall be 0. In case the ComSignalType is UINT8, UINT16, UINT32, SINT8, SINT16, SINT32 the string shall be interpreted as defined in the chapter Integer Type in the AUTOSAR EcuC specification. In case the ComSignalType is FLOAT32, FLOAT64 the string shall be interpreted as defined in the chapter Float Type in the AUTOSAR EcuC specification. In case the ComSignalType is BOOLEAN the string shall be interpreted as defined in the chapter Boolean Type in the AUTOSAR EcuC specification. In case the ComSignal is a UINT8_N, UINT8_DYN the string shall be interpreted as a decimal representation of the characters separated by blanks, e.g. "97 98 100" means a string "abd", where the char "a" is in byte 0(lowest address), "b" is in byte 1, and "d" is in byte 2 and (highest address). For the ComSignalType UINT8_DYN the dynamic length shall be set to the number of configured characters. An empty string "" shall be interpretated as 0-sized dynamic signal.		
Multiplicity	0..1		
Type	EcucStringParamDef		
Default Value	0		
Regular Expression			
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local dependency: In case of UINT8_N the length of ComSignalInitValue has to be the same as ComSignalLength.		

Name	ComSignalLength [ECUC_Com_00437]	
Description	Description: For ComSignalType UINT8_N this parameter specifies the length n in bytes. For ComSignalType UINT8_DYN it specifies the maximum length in bytes. For all other types this parameter shall be ignored. Range: 0..8 for normal CAN/ LIN I-PDUs, 0..254 for normal FlexRay I-PDUs, and 0..4294967295 for I-PDUs with ComIPduType TP.	
Multiplicity	0..1	
Type	EcucIntegerParamDef	
Range	0 .. 4294967295	
Default Value		

Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	–	
Scope / Dependency	scope: local		

Name	ComSignalType [ECUC_Com_00127]		
Description	The AUTOSAR type of the signal. Whether or not the signal is signed or unsigned can be found by examining the value of this attribute. This type could also be used to reserved appropriate storage in AUTOSAR COM.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	BOOLEAN		
	FLOAT32		
	FLOAT64		
	SINT16		
	SINT32		
	SINT8		
	UINT16		
	UINT32		
	UINT8		
	UINT8_DYN		
	UINT8_N		
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	–	
Scope / Dependency	scope: local		

Name	ComSystemTemplateSystemSignalRef [ECUC_Com_00002]		
Description	Reference to the ISignalToIPduMapping that contains a reference to the ISignal (System Template) which this ComSignal (or ComGroupSignal) represents.		
Multiplicity	0..1		
Type	Foreign reference to I-SIGNAL-TO-I-PDU-MAPPING		
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

Name	ComTransferProperty [ECUC_Com_00560]			
Description	Optionally defines whether this group signal shall contribute to the TRIGGERED_ON_CHANGE transfer property of the signal group. If at least one group signal of a signal group has the "ComTransferProperty" configured all other group signals of that signal group shall have the attribute configured as well.			
Multiplicity	0..1			
Type	EcucEnumerationParamDef			
Range	PENDING	A change of the value of this group signal shall not be considered in the evaluation of the signal groups ComTransferProperty.		
	TRIGGERED_ON_CHANGE	A change of the value of this group signal shall be considered in the evaluation of the signal groups ComTransferProperty.		
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE	
	Link time	X	VARIANT-LINK-TIME	
	Post-build time	X	VARIANT-POST-BUILD	
Scope / Dependency	scope: local			

Included Containers		
Container Name	**Multiplicity**	**Scope / Depedency**
ComFilter	0..1	This container contains the configuration parameters of the AUTOSAR COM module's Filters. Note: On sender side the container is used to specify the transmission mode conditions.

ComIPdu

SWS Item	[ECUC_Com_00340]
Container Name	ComIPdu
Description	Contains the configuration parameters of the AUTOSAR COM module's I-PDUs. **Attributes:** postBuildChangeable=true
Configuration Parameters	

Name	ComIPduCallout [ECUC_Com_00387]
Description	This parameter defines the existence and the name of a callout function for the corresponding I-PDU. If this parameter is omitted no I-PDU callout shall take place for the corresponding I-PDU.
Multiplicity	0..1
Type	EcucFunctionNameDef
Default Value	
Regular Expression	

Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	ComIPduCancellationSupport [ECUC_Com_00709]		
Description	Defines for I-PDUs with ComIPduType NORMAL: If the underlying IF-modul supports cancellation of transmit requests. Defines for I-PDUs with ComIPduType TP: If the underlying TP-module supports RX and TX cancellation of ongoing requests.		
Multiplicity	0..1		
Type	EcucBooleanParamDef		
Default Value	false		
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU dependency: This parameter shall not be set to true if ComCancellationSupport is set to false		

Name	ComIPduDirection [ECUC_Com_00493]		
Description	The direction defines if this I-PDU, and therefore the contributing signals and signal groups, shall be sent or received.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	RECEIVE		
	SEND		
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	–	
Scope / Dependency	scope: local dependency: If configured to Sent also a ComTxIpdu container shall be included, see ECUC_Com_00496		

Name	ComIPduGroupRef [ECUC_Com_00206]		
Description	Reference to the I-PDU groups this I-PDU belongs to.		
Multiplicity	0..*		
Type	Reference to ComIPduGroup		
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

Name	ComIPduHandleId [ECUC_Com_00175]		
Description	The numerical value used as the ID of this I-PDU. The ComIPduHandleId is required by the API calls Com_RxIndication, Com_TpRxIndication, Com_StartOfReception and Com_CopyRxData to receive I-PDUs from the PduR (ComIP-duDirection: Receive), as well as the PduId passed to an Rx-I-PDU-callout. For Tx-I-PDUs (ComIPduDirection: Send), this handle Id is used for the APIs calls Com_TxConfirmation, Com_TriggerTransmit, Com_TriggerIPDUSend, Com_CopyTxData and Com_TpTxConfirmation to transmit respectively confirm transmissions of I-PDUs, as well as the PduId passed to the Tx-I-PDU-callout configured with ComIPduCallout and/or ComIPduTriggerTransmitCallout.		
Multiplicity	0..1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 65535		
Default Value			
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	–	
Scope / Dependency	scope: ECU		

Name	ComIPduSignalGroupRef [ECUC_Com_00519]		
Description	References to all signal groups contained in this I-Pdu		
Multiplicity	0..*		
Type	Reference to ComSignalGroup		
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

Name	ComIPduSignalProcessing [ECUC_Com_00119]		
Description	For the definition of the two modes Immediate and Deferred.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	DEFERRED	signal indication / confirmations are deferred for example to a cyclic task	
	IMMEDIATE	the signal indications / confirmations are performed in Com_RxIndication/ Com_TxConfirmation	
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

Name	ComIPduSignalRef [ECUC_Com_00518]		
Description	References to all signals contained in this I-PDU.		
Multiplicity	0..*		
Type	Reference to ComSignal		
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

Name	ComIPduTriggerTransmitCallout [ECUC_Com_00765]		
Description	If there is a trigger transmit callout defined for this I-PDU this parameter contains the name of the callout function.		
Multiplicity	0..1		
Type	EcucFunctionNameDef		
Default Value			
Regular Expression			
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	ComIPduType [ECUC_Com_00761]		
Description	Defines if this I-PDU is a normal I-PDU that can be sent unfragmented or if this is a large I-PDU that shall be sent via the Transport Protocol of the underlying bus.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	NORMAL	sent or received via normal L-PDU	
	TP	sent or received via TP	
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

Name	ComPduIdRef [ECUC_Com_00711]		
Description	Reference to the "global" Pdu structure to allow harmonization of handle IDs in the COM-Stack.		
Multiplicity	1		
Type	Reference to Pdu		
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	–	
Scope / Dependency			

Included Containers		
Container Name	**Multiplicity**	**Scope / Depedency**

ComIPduCounter	0..1	This optional container contains the configuration parameters of PDU Counter.
ComIPduReplication	0..1	This optional container contains the information needed for each I-PDU replicated.
ComTxIPdu	0..1	This container contains additional transmission related configuration parameters of the AUTOSAR COM module's I-PDUs.

ComSignal

SWS Item	[ECUC_Com_00344]
Container Name	ComSignal
Description	Contains the configuration parameters of the AUTOSAR COM module's signals. **Attributes:** postBuildChangeable=true
Configuration Parameters	

Name	ComBitPosition [ECUC_Com_00259]		
Description	Starting position within the I-PDU. This parameter refers to the position in the I-PDU and not in the shadow buffer. If the endianness conversion is configured to Opaque the parameter ComBitPosition shall define the bit0 of the first byte like in little endian byte order		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default Value			
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

Name	ComBitSize [ECUC_Com_00158]		
Description	Size in bits, for integer signal types. For ComSignalType UINT8_N and UINT8_DYN the size shall be configured by ComSignalLength. For ComSignalTypes FLOAT32 and FLOAT64 the size is already defined by the signal type and therefore may be omitted.		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	0 .. 64		
Default Value			
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

Name	ComDataInvalidAction [ECUC_Com_00314]		
Description	This parameter defines the action performed upon reception of an invalid signal. Relating to signal groups the action in case if one of the included signals is an invalid signal. If Replace is used the ComSignalInitValue will be used for the replacement.		
Multiplicity	0..1		
Type	EcucEnumerationParamDef		
Range	NOTIFY		
	REPLACE	Literal for DataInvalidAction	
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	–	
Scope / Dependency	scope: local		

Name	ComErrorNotification [ECUC_Com_00499]		
Description	Only valid on sender side: Name of Com_CbkTxErr callback function to be called. If this parameter is omitted no error notification shall take place.		
Multiplicity	0..1		
Type	EcucFunctionNameDef		
Default Value			
Regular Expression			
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	–	
Scope / Dependency	scope: local		

Name	ComFirstTimeout [ECUC_Com_00183]		
Description	Defines the length of the first deadline monitoring timeout period in seconds. This timeout is used immediately after start (or restart) of the deadline monitoring service. The timeout period of the successive periods is configured by ECUC_Com_00263.		
Multiplicity	0..1		
Type	EcucFloatParamDef		
Range	0 .. 3600		
Default Value			
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

Name	ComHandleId [ECUC_Com_00165]		
Description	The numerical value used as the ID. For signals it is required by the API calls Com_UpdateShadowSignal, Com_ReceiveShadowSignal and Com_InvalidateShadowSignal. For signals groups it is required by the Com_SendSignalGroup and Com_ReceiveSignalGroup calls.		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 65535		
Default Value			
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	–	
Scope / Dependency	scope: ECU		

Name	ComInitialValueOnly [ECUC_Com_00811]		
Description	This parameter defines that the respective signal's initial value shall be put into the respective PDU but there will not be any update of the value through the RTE. Thus the Com implementation does not need to expect any API calls for this signal (group).		
Multiplicity	0..1		
Type	EcucBooleanParamDef		
Default Value	false		
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	–	
Scope / Dependency	scope: local		

Name	ComInvalidNotification [ECUC_Com_00315]		
Description	Only valid on receiver side: Name of Com_CbkInv callback function to be called. Name of the function which notifies the RTE about the reception of an invalidated signal/ signal group. Only applicable if ComDataInvalidAction is configured to NOTIFY.		
Multiplicity	0..1		
Type	EcucFunctionNameDef		
Default Value			
Regular Expression			
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	–	
Scope / Dependency	scope: local		

Name	ComNotification [ECUC_Com_00498]		
Description	On sender side: Name of Com_CbkTxAck callback function to be called. On receiver side: Name of Com_CbkRxAck callback function to be called. If this parameter is omitted no notification shall take place.		
Multiplicity	0..1		
Type	EcucFunctionNameDef		
Default Value			
Regular Expression			
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	–	
Scope / Dependency	scope: local		

Name	ComRxDataTimeoutAction [ECUC_Com_00412]		
Description	This parameter defines the action performed upon expiration of the reception deadline monitoring timer.		
Multiplicity	0..1		
Type	EcucEnumerationParamDef		
Range	NONE	no replacement shall take place	
	REPLACE	signals shall be replaced by their ComSignalInitValue	
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	–	
Scope / Dependency	scope: local		

Name	ComSignalDataInvalidValue [ECUC_Com_00391]
Description	Defines the data invalid value of the signal. In case the ComSignalType is UINT8, UINT16, UINT32, SINT8, SINT16, SINT32 the string shall be interpreted as defined in the chapter Integer Type in the AUTOSAR EcuC specification. In case the ComSignalType is FLOAT32, FLOAT64 the string shall be interpreted as defined in the chapter Float Type in the AUTOSAR EcuC specification. In case the ComSignalType is BOOLEAN the string shall be interpreted as defined in the chapter Boolean Type in the AUTOSAR EcuC specification. In case the ComSignal is a UINT8_N, UINT8_DYN the string shall be interpreted as a decimal representation of the characters separated by blanks, e.g. "97 98 100" means a string "abd", where the char "a" is in byte 0(lowest address), "b" is in byte 1, and "d" is in byte 2 and (highest address). For the ComSignalType UINT8_DYN the dynamic length shall be set to the number of configured characters. An empty string "" shall be interpretated as 0-sized dynamic signal.
Multiplicity	0..1
Type	EcucStringParamDef
Default Value	
Regular Expression	

Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	–	
Scope / Dependency	scope: local dependency: In case of UINT8_N the length of ComSignalDataInvalidValue has to be the same as ComSignalLength.		

Name	ComSignalEndianness [ECUC_Com_00157]		
Description	Defines the endianness of the signal's network representation.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	BIG_ENDIAN		
	LITTLE_ENDIAN		
	OPAQUE		
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

Name	ComSignalInitValue [ECUC_Com_00170]		
Description	Initial value for this signal. In case of UINT8_N the default value is a string of length ComSignalLength with all bytes set to 0x00. In case of UINT8_DYN the initial size shall be 0. In case the ComSignalType is UINT8, UINT16, UINT32, SINT8, SINT16, SINT32 the string shall be interpreted as defined in the chapter Integer Type in the AUTOSAR EcuC specification. In case the ComSignalType is FLOAT32, FLOAT64 the string shall be interpreted as defined in the chapter Float Type in the AUTOSAR EcuC specification. In case the ComSignalType is BOOLEAN the string shall be interpreted as defined in the chapter Boolean Type in the AUTOSAR EcuC specification. In case the ComSignal is a UINT8_N, UINT8_DYN the string shall be interpreted as a decimal representation of the characters separated by blanks, e.g. "97 98 100" means a string "abd", where the char "a" is in byte 0(lowest address), "b" is in byte 1, and "d" is in byte 2 and (highest address). For the ComSignalType UINT8_DYN the dynamic length shall be set to the number of configured characters. An empty string "" shall be interpretated as 0-sized dynamic signal.		
Multiplicity	0..1		
Type	EcucStringParamDef		
Default Value	0		
Regular Expression			
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local dependency: In case of UINT8_N the length of ComSignalInitValue has to be the same as ComSignalLength.		

Name	ComSignalLength [ECUC_Com_00437]		
Description	Description: For ComSignalType UINT8_N this parameter specifies the length n in bytes. For ComSignalType UINT8_DYN it specifies the maximum length in bytes. For all other types this parameter shall be ignored. Range: 0..8 for normal CAN/ LIN I-PDUs, 0..254 for normal FlexRay I-PDUs, and 0..4294967295 for I-PDUs with ComIPduType TP.		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default Value			
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	–	
Scope / Dependency	scope: local		

Name	ComSignalType [ECUC_Com_00127]		
Description	The AUTOSAR type of the signal. Whether or not the signal is signed or unsigned can be found by examining the value of this attribute. This type could also be used to reserved appropriate storage in AUTOSAR COM.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	BOOLEAN		
	FLOAT32		
	FLOAT64		
	SINT16		
	SINT32		
	SINT8		
	UINT16		
	UINT32		
	UINT8		
	UINT8_DYN		
	UINT8_N		
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	–	
Scope / Dependency	scope: local		

Name	ComSystemTemplateSystemSignalRef [ECUC_Com_00002]		
Description	Reference to the ISignalToIPduMapping that contains a reference to the ISignal (System Template) which this ComSignal (or ComGroupSignal) represents.		
Multiplicity	0..1		
Type	Foreign reference to I-SIGNAL-TO-I-PDU-MAPPING		
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

Name	ComTimeout [ECUC_Com_00263]		
Description	Defines the length of the deadline monitoring timeout period in seconds. The period for the first timeout period can be configured separately by ECUC_Com_00183.		
Multiplicity	0..1		
Type	EcucFloatParamDef		
Range	0 .. 3600		
Default Value			
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

Name	ComTimeoutNotification [ECUC_Com_00552]		
Description	On sender side: Name of Com_CbkTxTOut callback function to be called. On receiver side: Name of Com_CbkRxTOut callback function to be called.		
Multiplicity	0..1		
Type	EcucFunctionNameDef		
Default Value			
Regular Expression			
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	–	
Scope / Dependency	scope: local		

Name	ComTransferProperty [ECUC_Com_00232]	
Description	Defines if a write access to this signal can trigger the transmission of the corresponding I-PDU. If the I-PDU is triggered, depends also on the transmission mode of the corresponding I-PDU.	
Multiplicity	0..1	
Type	EcucEnumerationParamDef	
Range	PENDING	A write access to this signal never triggers the transmission of the corresponding I-PDU.

	TRIGGERED	Depending on the transmission mode, a write access to this signal can trigger the transmission of the corresponding I-PDU.	
	TRIGGERED_ON_CHANGE	Depending on the transmission mode, a write access to this signal can trigger the transmission of the corresponding I-PDU, but only in case the written value is different to the locally stored (last written or init) value.	
	TRIGGERED_ON_CHANGE_WITHOUT_REPETITION	Depending on the transmission mode, a write access to this signal can trigger the transmission of the corresponding I-PDU just once without a repetition, but only in case the written value is different to the locally stored (last written or init) value.	
	TRIGGERED_WITHOUT_REPETITION	Depending on the transmission mode, a write access to this signal can trigger the transmission of the corresponding I-PDU just once without a repetition.	
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

Name	ComUpdateBitPosition [ECUC_Com_00257]		
Description	Bit position of update-bit inside I-PDU. If this attribute is omitted then there is no update-bit. This setting must be consistently on sender and on receiver side. Range: 0..63 for CAN and LIN 0..2031 for FlexRay		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default Value			
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

Included Containers		
Container Name	**Multiplicity**	**Scope / Depedency**
ComFilter	0..1	This container contains the configuration parameters of the AUTOSAR COM module's Filters. Note: On sender side the container is used to specify the transmission mode conditions.

ComSignalGroup

SWS Item	[ECUC_Com_00345]		
Container Name	ComSignalGroup		
Description	Contains the configuration parameters of the AUTOSAR COM module's signal groups. **Attributes:** postBuildChangeable=true		
Configuration Parameters			

Name	ComDataInvalidAction [ECUC_Com_00314]		
Description	This parameter defines the action performed upon reception of an invalid signal. Relating to signal groups the action in case if one of the included signals is an invalid signal. If Replace is used the ComSignalInitValue will be used for the replacement.		
Multiplicity	0..1		
Type	EcucEnumerationParamDef		
Range	NOTIFY		
	REPLACE	Literal for DataInvalidAction	
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	–	
Scope / Dependency	scope: local		

Name	ComErrorNotification [ECUC_Com_00499]		
Description	Only valid on sender side: Name of Com_CbkTxErr callback function to be called. If this parameter is omitted no error notification shall take place.		
Multiplicity	0..1		
Type	EcucFunctionNameDef		
Default Value			
Regular Expression			
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	–	
Scope / Dependency	scope: local		

Name	ComFirstTimeout [ECUC_Com_00183]	
Description	Defines the length of the first deadline monitoring timeout period in seconds. This timeout is used immediately after start (or restart) of the deadline monitoring service. The timeout period of the successive periods is configured by ECUC_Com_00263.	
Multiplicity	0..1	
Type	EcucFloatParamDef	
Range	0 .. 3600	
Default Value		

Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

Name	ComHandleId [ECUC_Com_00165]		
Description	The numerical value used as the ID. For signals it is required by the API calls Com_UpdateShadowSignal, Com_ReceiveShadowSignal and Com_InvalidateShadowSignal. For signals groups it is required by the Com_SendSignalGroup and Com_ReceiveSignalGroup calls.		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 65535		
Default Value			
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	–	
Scope / Dependency	scope: ECU		

Name	ComInitialValueOnly [ECUC_Com_00811]		
Description	This parameter defines that the respective signal's initial value shall be put into the respective PDU but there will not be any update of the value through the RTE. Thus the Com implementation does not need to expect any API calls for this signal (group).		
Multiplicity	0..1		
Type	EcucBooleanParamDef		
Default Value	false		
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	–	
Scope / Dependency	scope: local		

Name	ComInvalidNotification [ECUC_Com_00315]
Description	Only valid on receiver side: Name of Com_CbkInv callback function to be called. Name of the function which notifies the RTE about the reception of an invalidated signal/ signal group. Only applicable if ComDataInvalidAction is configured to NOTIFY.
Multiplicity	0..1
Type	EcucFunctionNameDef
Default Value	
Regular Expression	

Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	–	
Scope / Dependency	scope: local		

Name	ComNotification [ECUC_Com_00498]		
Description	On sender side: Name of Com_CbkTxAck callback function to be called. On receiver side: Name of Com_CbkRxAck callback function to be called. If this parameter is omitted no notification shall take place.		
Multiplicity	0..1		
Type	EcucFunctionNameDef		
Default Value			
Regular Expression			
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	–	
Scope / Dependency	scope: local		

Name	ComRxDataTimeoutAction [ECUC_Com_00412]		
Description	This parameter defines the action performed upon expiration of the reception deadline monitoring timer.		
Multiplicity	0..1		
Type	EcucEnumerationParamDef		
Range	NONE	no replacement shall take place	
	REPLACE	signals shall be replaced by their ComSignalInitValue	
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	–	
Scope / Dependency	scope: local		

Name	ComSystemTemplateSignalGroupRef [ECUC_Com_00001]		
Description	Reference to the ISignalToIPduMapping that contains a reference to the ISignalGroup (SystemTemplate) which this ComSignalGroup represents.		
Multiplicity	0..1		
Type	Foreign reference to I-SIGNAL-TO-I-PDU-MAPPING		
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

Name	ComTimeout [ECUC_Com_00263]		
Description	Defines the length of the deadline monitoring timeout period in seconds. The period for the first timeout period can be configured separately by ECUC_Com_00183.		
Multiplicity	0..1		
Type	EcucFloatParamDef		
Range	0 .. 3600		
Default Value			
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

Name	ComTimeoutNotification [ECUC_Com_00552]		
Description	On sender side: Name of Com_CbkTxTOut callback function to be called. On receiver side: Name of Com_CbkRxTOut callback function to be called.		
Multiplicity	0..1		
Type	EcucFunctionNameDef		
Default Value			
Regular Expression			
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	–	
Scope / Dependency	scope: local		

Name	ComTransferProperty [ECUC_Com_00232]	
Description	Defines if a write access to this signal can trigger the transmission of the corresponding I-PDU. If the I-PDU is triggered, depends also on the transmission mode of the corresponding I-PDU.	
Multiplicity	0..1	
Type	EcucEnumerationParamDef	
Range	PENDING	A write access to this signal never triggers the transmission of the corresponding I-PDU.
	TRIGGERED	Depending on the transmission mode, a write access to this signal can trigger the transmission of the corresponding I-PDU.
	TRIGGERED_ON_CHANGE	Depending on the transmission mode, a write access to this signal can trigger the transmission of the corresponding I-PDU, but only in case the written value is different to the locally stored (last written or init) value.

	TRIGGERED_ON_CHANGE_WITHOUT_REPETITION	Depending on the transmission mode, a write access to this signal can trigger the transmission of the corresponding I-PDU just once without a repetition, but only in case the written value is different to the locally stored (last written or init) value.	
	TRIGGERED_WITHOUT_REPETITION	Depending on the transmission mode, a write access to this signal can trigger the transmission of the corresponding I-PDU just once without a repetition.	
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

Name	ComUpdateBitPosition [ECUC_Com_00257]		
Description	Bit position of update-bit inside I-PDU. If this attribute is omitted then there is no update-bit. This setting must be consistently on sender and on receiver side. Range: 0..63 for CAN and LIN 0..2031 for FlexRay		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default Value			
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

Included Containers		
Container Name	**Multiplicity**	**Scope / Depedency**
ComGroupSignal	0..*	This container contains the configuration parameters of group signals. I.e. signals that are included within a signal group.

EcucPartition

SWS Item	[ECUC_EcuC_00005]
Container Name	EcucPartition
Description	Definition of one Partition on this ECU. One Partition will be implemented using one Os-Application. **Attributes:** postBuildChangeable=false
Configuration Parameters	

Name	EcucPartitionBswModuleDistinguishedPartition [ECUC_EcuC_00068]		
Description	This maps the abstract partition of the Bsw Module to a concrete Partition existing in the ECU.		
Multiplicity	0..*		
Type	Foreign reference to BSW-DISTINGUISHED-PARTITION		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency			

Name	EcucPartitionBswModuleExecution [ECUC_EcuC_00037]		
Description	Denotes that this partition will execute BSW Modules. BSW Modules can only be executed in such partitions.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value			
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency			

Name	EcucPartitionSoftwareComponentInstanceRef [ECUC_EcuC_00036]		
Description	References the SW Component instances from the Ecu Extract that shall be executed in this partition.		
Multiplicity	0..*		
Type	Instance reference to SW-COMPONENT-PROTOTYPE context: ROOT-SW-COMPOSITION-PROTOTYPE		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency			

Name	PartitionCanBeRestarted [ECUC_EcuC_00006]		
Description	Specifies the requirement whether the Partition can be restarted. If set to true all software executing in this partition shall be capable of handling a restart.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value			
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency			

No Included Containers

NvMBlockDescriptor

SWS Item	[ECUC_NvM_00061]
Container Name	NvMBlockDescriptor
Description	Container for a management structure to configure the composition of a given NVRAM Block Management Type. Its multiplicity describes the number of configured NVRAM blocks, one block is required to be configured. The NVRAM block descriptors are condensed in the NVRAM block descriptor table.
Configuration Parameters	

Name	NvMBlockCrcType {NVM_BLOCK_CRC_TYPE} [ECUC_NvM_00476]		
Description	Defines CRC data width for the NVRAM block. Default: NVM_CRC16, i.e. CRC16 will be used if NVM_BLOCK_USE_CRC==true		
Multiplicity	0..1		
Type	EcucEnumerationParamDef		
Range	NVM_CRC16	(Default) CRC16 will be used if NVM_BLOCK_USE_CRC==true.	
	NVM_CRC32	CRC32 is selected for this NVRAM block if NVM_BLOCK_USE_CRC==true.	
	NVM_CRC8	CRC8 is selected for this NVRAM block if NVM_BLOCK_USE_CRC==true.	
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	–	
Scope / Dependency	scope: local dependency: NVM_BLOCK_USE_CRC, NVM_CALC_RAM_BLOCK_CRC		

Name	NvMBlockHeaderInclude [ECUC_NvM_00554]		
Description	Defines the header file where the owner of the NVRAM block has the declarations of the permanent RAM data block, ROM data block (if configured) and the callback function prototype for each configured callback. If no permanent RAM block, ROM block or callback functions are configured then this configuration parameter shall be ignored.		
Multiplicity	0..1		
Type	EcucStringParamDef		
Default Value			
Regular Expression			
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	–	
Scope / Dependency	scope: local		

Name	NvMBlockJobPriority {NVM_BLOCK_JOB_PRIORITY} [ECUC_NvM_00477]		
Description	Defines the job priority for a NVRAM block (0 = Immediate priority).		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 255		
Default Value			
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	–	
Scope / Dependency	scope: local		

Name	NvMBlockManagementType {NVM_BLOCK_MANAGEMENT_TYPE} [ECUC_NvM_00062]		
Description	Defines the block management type for the NVRAM block.[NVM137]		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	NVM_BLOCK_DATASET	NVRAM block is configured to be of dataset type.	
	NVM_BLOCK_NATIVE	NVRAM block is configured to be of native type.	
	NVM_BLOCK_REDUNDANT	NVRAM block is configured to be of redundant type.	
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	–	
Scope / Dependency	scope: local		

Name	NvMBlockUseCrc {NVM_BLOCK_USE_CRC} [ECUC_NvM_00036]		
Description	Defines CRC usage for the NVRAM block, i.e. memory space for CRC is reserved in RAM and NV memory. true: CRC will be used for this NVRAM block. false: CRC will not be used for this NVRAM block.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value			
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	–	
Scope / Dependency	scope: local		

Name	NvMBlockUseSetRamBlockStatus [ECUC_NvM_00552]		
Description	Defines if NvMSetRamBlockStatusApi shall be used for this block or not. Note: If NvMSetRamBlockStatusApi is disabled this configuration parameter shall be ignored. true: calling of NvMSetRamBlockStatus for this RAM block shall set the status of the RAM block. false: calling of NvMSetRamBlockStatus for this RAM block shall be ignored.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value			
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	–	
Scope / Dependency	scope: local		

Name	NvMBlockUseSyncMechanism {NVM_BLOCK_USE_SYNC_MECHANISM} [ECUC_NvM_00519]		
Description	Defines whether an explicit synchronization mechanism with a RAM mirror and callback routines for transferring data to and from NvM module's RAM mirror is used for NV block. true if synchronization mechanism is used, false otherwise.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value	false		
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	–	
Scope / Dependency	scope: local		

Name	NvMBlockWriteProt {NVM_BLOCK_WRITE_PROT} [ECUC_NvM_00033]		
Description	Defines an initial write protection of the NV block true: Initial block write protection is enabled. false: Initial block write protection is disabled.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value			
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	–	
Scope / Dependency	scope: local		

Name	NvMBswMBlockStatusInformation {NVM_BSWM_BLOCK_STATUS_INFORMATION} [ECUC_NvM_00551]		
Description	This parameter specifies whether BswM is informed about the current status of the specified block. True: Call BswM_NvM_CurrentBlockMode on changes False: Dont inform BswM at all		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value	false		
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	–	
Scope / Dependency	scope: local		

Name	NvMCalcRamBlockCrc {NVM_CALC_RAM_BLOCK_CRC} [ECUC_NvM_00119]		
Description	Defines CRC (re)calculation for the permanent RAM block or NVRAM blocks which are configured to use explicit synchronization mechanism. true: CRC will be (re)calculated for this permanent RAM block. false: CRC will not be (re)calculated for this permanent RAM block.		
Multiplicity	0..1		
Type	EcucBooleanParamDef		
Default Value			
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	–	
Scope / Dependency	scope: local dependency: NVM_BLOCK_USE_CRC		

Name	NvMInitBlockCallback {NVM_INIT_BLOCK_CALLBACK} [ECUC_NvM_00116]		
Description	Entry address of a block specific callback routine which shall be called if no ROM data is available for initialization of the NVRAM block. If not configured, no specific callback routine shall be called for initialization of the NVRAM block with default data.		
Multiplicity	0..1		
Type	EcucFunctionNameDef		
Default Value			
Regular Expression			
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	–	
Scope / Dependency	scope: local		

Name	NvMMaxNumOfReadRetries {NVM_MAX_NUM_OF_READ_RETRIES} [ECUC_NvM_00533]		
Description	Defines the maximum number of read retries.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 7		
Default Value	0		
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	–	
Scope / Dependency	scope: local		

Name	NvMMaxNumOfWriteRetries {NVM_MAX_NUM_OF_WRITE_RETRIES} [ECUC_NvM_00499]		
Description	Defines the maximum number of write retries for a NVRAM block with [ECUC_NvM_00061]. Regardless of configuration a consistency check (and maybe write retries) are always forced for each block which is processed by the request NvM_WriteAll and NvM_WriteBlock.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 7		
Default Value			
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	–	
Scope / Dependency	scope: local		

Name	NvMNvBlockBaseNumber {NVM_NV_BLOCK_BASE_NUMBER} [ECUC_NvM_00478]		
Description	Configuration parameter to perform the link between the NVM_NVRAM_BLOCK_IDENTIFIER used by the SW-Cs and the FEE_BLOCK_NUMBER expected by the memory abstraction modules. The parameter value equals the FEE_BLOCK_NUMBER or EA_BLOCK_NUMBER shifted to the right by NvMDatasetSelectionBits bits. (ref. to chapter 7.1.2.1). Calculation Formula: value = TargetBlockReference.[Ea/Fee]BlockConfiguration.[Ea/Fee]BlockNumber » NvMDatasetSelectionBits		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 65534		
Default Value			
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	–	
Scope / Dependency	scope: local dependency: FEE_BLOCK_NUMBER, EA_BLOCK_NUMBER		

Name	NvMNvBlockLength {NVM_NV_BLOCK_LENGTH} [ECUC_NvM_00479]		
Description	Defines the NV block data length in bytes. Note: The implementer can add the attribute 'withAuto' to the parameter definition which indicates that the length can be calculated by the generator automatically (e.g. by using the sizeof operator). When 'withAuto' is set to 'true' for this parameter definition the 'isAutoValue' can be set to 'true'. If 'isAutoValue' is set to 'true' the actual value will not be considered during ECU Configuration but will be (re-)calculated by the code generator and stored in the value attribute afterwards.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 65535		
Default Value			
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	–	
Scope / Dependency	scope: local		

Name	NvMNvBlockNum {NVM_NV_BLOCK_NUM} [ECUC_NvM_00480]		
Description	Defines the number of multiple NV blocks in a contiguous area according to the given block management type. 1-255 For NVRAM blocks to be configured of block management type NVM_BLOCK_DATASET. The actual range is limited according to NVM444. 1 For NVRAM blocks to be configured of block management type NVM_BLOCK_NATIVE 2 For NVRAM blocks to be configured of block management type NVM_BLOCK_REDUNDANT		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 255		
Default Value			
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	–	
Scope / Dependency	scope: local dependency: NVM_BLOCK_MANAGEMENT_TYPE		

Name	NvMNvramBlockIdentifier {NVM_NVRAM_BLOCK_IDENTIFIER} [ECUC_NvM_00481]		
Description	Identification of a NVRAM block via a unique block identifier. Implementation Type: NvM_BlockIdType. min = 1 max = 2ˆ (16- NVM_DATASET_SELECTION_BITS)-1 Reserved NVRAM block IDs: 0 -> to derive multi block request results via NvM_GetErrorStatus 1 -> redundant NVRAM block which holds the configuration ID (generation tool should check that this block is correctly configured from type,CRC and size point of view)		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	1 .. 65535		
Default Value			
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	–	
Scope / Dependency	scope: local dependency: NVM_DATASET_SELECTION_BITS		

Name	NvMNvramDeviceId {NVM_NVRAM_DEVICE_ID} [ECUC_NvM_00035]		
Description	Defines the NVRAM device ID where the NVRAM block is located. Calculation Formula: value = TargetBlockReference.[Ea/Fee]BlockConfiguration.[Ea/Fee]DeviceIndex		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 254		
Default Value			
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	–	
Scope / Dependency	scope: local dependency: EA_DEVICE_INDEX, FEE_DEVICE_INDEX		

Name	NvMRamBlockDataAddress {NVM_RAM_BLOCK_DATA_ADDRESS} [ECUC_NvM_00482]
Description	Defines the start address of the RAM block data. If this is not configured, no permanent RAM data block is available for the selected block management type.
Multiplicity	0..1
Type	EcucStringParamDef
Default Value	
Regular Expression	

Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	–	
Scope / Dependency	scope: local		

Name	NvMReadRamBlockFromNvCallback {NVM_READ_RAM_BLOCK_FROM_NVM} [ECUC_NvM_00521]		
Description	Entry address of a block specific callback routine which shall be called in order to let the application copy data from the NvM module's mirror to RAM block. Implementation type: Std_ReturnType E_OK: copy was successful E_NOT_OK: copy was not successful, callback routine to be called again		
Multiplicity	0..1		
Type	EcucFunctionNameDef		
Default Value			
Regular Expression			
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	–	
Scope / Dependency	scope: local		

Name	NvMResistantToChangedSw {NVM_RESISTANT_TO_CHANGED_SW} [ECUC_NvM_00483]		
Description	Defines whether a NVRAM block shall be treated resistant to configuration changes or not. If there is no default data available at configuration time then the application shall be responsible for providing the default initialization data. In this case the application has to use NvM_GetErrorStatus()to be able to distinguish between first initialization and corrupted data. true: NVRAM block is resistant to changed software. false: NVRAM block is not resistant to changed software.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value			
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	–	
Scope / Dependency	scope: local		

Name	NvMRomBlockDataAddress {NVM_ROM_BLOCK_DATA_ADDRESS} [ECUC_NvM_00484]		
Description	Defines the start address of the ROM block data. If not configured, no ROM block is available for the selected block management type.		
Multiplicity	0..1		
Type	EcucStringParamDef		
Default Value			
Regular Expression			
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	–	
Scope / Dependency	scope: local		

Name	NvMRomBlockNum {NVM_ROM_BLOCK_NUM} [ECUC_NvM_00485]		
Description	Defines the number of multiple ROM blocks in a contiguous area according to the given block management type. 0-255 For NVRAM blocks to be configured of block management type NVM_BLOCK_DATASET. The actual range is limited according to NVM444. 0-1 For NVRAM blocks to be configured of block management type NVM_BLOCK_NATIVE 0-1 For NVRAM blocks to be configured of block management type NVM_BLOCK_REDUNDANT		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 255		
Default Value			
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	–	
Scope / Dependency	scope: local dependency: NVM_BLOCK_MANAGEMENT_TYPE, NVM_NV_BLOCK_NUM		

Name	NvMSelectBlockForReadAll {NVM_SELECT_BLOCK_FOR_READALL} [ECUC_NvM_00117]		
Description	Defines whether a NVRAM block shall be processed during NvM_ReadAll or not. This configuration parameter has only influence on those NVRAM blocks which are configured to have a permanent RAM block or which are configured to use explicit synchronization mechanism. true: NVRAM block shall be processed by NvM_ReadAll false: NVRAM block shall not be processed by NvM_ReadAll		
Multiplicity	0..1		
Type	EcucBooleanParamDef		
Default Value			
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	–	
Scope / Dependency	scope: local dependency: NVM_RAM_BLOCK_DATA_ADDRESS		

Name	NvMSelectBlockForWriteAll {NVM_SELECT_BLOCK_FOR_WRITEALL} [ECUC_NvM_00549]		
Description	Defines whether a NVRAM block shall be processed during NvM_WriteAll or not. This configuration parameter has only influence on those NVRAM blocks which are configured to have a permanent RAM block or which are configured to use explicit synchronization mechanism. true: NVRAM block shall be processed by NvM_WriteAll false: NVRAM block shall not be processed by NvM_WriteAll		
Multiplicity	0..1		
Type	EcucBooleanParamDef		
Default Value			
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	–	
Scope / Dependency	scope: local dependency: NVM_RAM_BLOCK_DATA_ADDRESS		

Name	NvMSingleBlockCallback {NVM_SINGLE_BLOCK_CALLBACK} [ECUC_NvM_00506]		
Description	Entry address of the block specific callback routine which shall be invoked on termination of each asynchronous single block request [NVM113].		
Multiplicity	0..1		
Type	EcucFunctionNameDef		
Default Value			
Regular Expression			
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	–	
Scope / Dependency	scope: local		

Name	NvMStaticBlockIDCheck {NVM_STATIC_BLOCK_ID_CHECK} [ECUC_NvM_00532]		
Description	Defines if the Static Block ID check is enabled. false: Static Block ID check is disabled. true: Static Block ID check is enabled.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value	false		
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	–	
Scope / Dependency	scope: local		

Name	NvMWriteBlockOnce {NVM_WRITE_BLOCK_ONCE} [ECUC_NvM_00072]		
Description	Defines write protection after first write. The NVRAM manager sets the write protection bit after the NV block was written the first time. This means that some of the NV blocks in the NVRAM should never be erased nor be replaced with the default ROM data after first initialization. [NVM276]. true: Defines write protection after first write is enabled. false: Defines write protection after first write is disabled.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value			
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	–	
Scope / Dependency	scope: local		

Name	NvMWriteRamBlockToNvCallback {NVM_WRITE_RAM_BLOCK_TO_NVM} [ECUC_NvM_00520]		
Description	Entry address of a block specific callback routine which shall be called in order to let the application copy data from RAM block to NvM module's mirror. Implementation type: Std_ReturnType E_OK: copy was successful E_NOT_OK: copy was not successful, callback routine to be called again		
Multiplicity	0..1		
Type	EcucFunctionNameDef		
Default Value			
Regular Expression			
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	–	
Scope / Dependency	scope: local		

Name	NvMWriteVerification {NVM_WRITE_VERIFICATION} [ECUC_NvM_00534]		
Description	Defines if Write Verification is enabled. false: Write verification is disabled. true: Write Verification is enabled.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value	false		
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	–	
Scope / Dependency	scope: local		

Name	NvMWriteVerificationDataSize {NVM_WRITE_VERIFICATION_DATA_SIZE} [ECUC_NvM_00538]		
Description	Defines the number of bytes to compare in each step when comparing the content of a RAM Block and a block read back.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 65535		
Default Value			
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	–	
Scope / Dependency	scope: local		

Included Containers		
Container Name	**Multiplicity**	**Scope / Depedency**
NvMTargetBlock Reference [NvMEaRef, NvMFeeRef]	1	This parameter is just a container for the parameters for EA and FEE

OsAlarm

SWS Item	[ECUC_Os_00003]
Container Name	OsAlarm {ALARM}
Description	An OsAlarm may be used to asynchronously inform or activate a specific task. It is possible to start alarms automatically at system start-up depending on the application mode.
Configuration Parameters	

Name	OsAlarmAccessingApplication {ACCESSING_APPLICATION} [ECUC_Os_00004]		
Description	Reference to applications which have an access to this object.		
Multiplicity	0..*		
Type	Reference to OsApplication		
Configuration Class	Pre-compile time	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency			

Name	OsAlarmCounterRef {COUNTER} [ECUC_Os_00005]		
Description	Reference to the assigned counter for that alarm		
Multiplicity	1		
Type	Reference to OsCounter		
Configuration Class	Pre-compile time	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Included Containers		
Container Name	**Multiplicity**	**Scope / Depedency**
OsAlarmAction [OsAlarmActivateTask, OsAlarmCallback, OsAlarmIncrementCounter, OsAlarmSetEvent]	1	This container defines which type of notification is used when the alarm expires.
OsAlarmAutostart	0..1	If present this container defines if an alarm is started automatically at system start-up depending on the application mode.

OsApplication

SWS Item	[ECUC_Os_00114]
Container Name	OsApplication {APPLICATION}
Description	An AUTOSAR OS must be capable of supporting a collection of OS objects (tasks, interrupts, alarms, hooks etc.) that form a cohesive functional unit. This collection of objects is termed an OS-Application. All objects which belong to the same OS-Application have access to each other. Access means to allow to use these objects within API services. Access by other applications can be granted separately.
Configuration Parameters	

Name	OsAppAlarmRef [ECUC_Os_00231]		
Description	Specifies the OsAlarms that belong to the OsApplication.		
Multiplicity	0..*		
Type	Reference to OsAlarm		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: ECU dependency: Required for scalability class 3 and 4		

Name	OsAppCounterRef [ECUC_Os_00234]		
Description	References the OsCounters that belong to the OsApplication.		
Multiplicity	0..*		
Type	Reference to OsCounter		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: ECU dependency: Required for scalability class 3 and 4.		

Name	OsAppEcucPartitionRef [ECUC_Os_00392]		
Description	Denotes which "EcucPartition" is implemented by this "OSApplication".		
Multiplicity	0..1		
Type	Reference to EcucPartition		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency			

Name	OsAppIsrRef [ECUC_Os_00221]		
Description	references which OsIsrs belong to the OsApplication		
Multiplicity	0..*		
Type	Reference to OsIsr		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: ECU dependency: Required for scalability class 3 and 4.		

Name	OsAppScheduleTableRef [ECUC_Os_00230]		
Description	References the OsScheduleTables that belong to the OsApplication.		
Multiplicity	0..*		
Type	Reference to OsScheduleTable		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: ECU dependency: Required for scalability class 3 and 4.		

Name	OsAppTaskRef [ECUC_Os_00116]		
Description	references which OsTasks belong to the OsApplication		
Multiplicity	0..*		
Type	Reference to OsTask		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: ECU dependency: Required for scalability class 3 and 4		

Name	OsApplicationCoreAssignment {CORE} [ECUC_Os_01020] (Obsolete. Use OsApplicationCoreRef instead.)		
Description	ID of the core onto which the OsApplication is bound. Please note that this attribute is deprecated and replaced by the OsApplicationCoreRef. **Tags:** atp.Status=obsolete atp.StatusComment=This parameter is replaced by the OsApplication CoreRef. atp.StatusRevisionBegin=4.1.1		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	0 .. 65534		
Default Value			
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	OsApplicationCoreRef [ECUC_Os_00393]		
Description	Reference to the Core Definition in the Ecuc Module where the CoreId is defined. This reference is used to describe to which Core the OsApplication is bound.		
Multiplicity	0..1		
Type	Symbolic name reference to EcucCoreDefinition		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	OsRestartTask {RESTARTTASK} [ECUC_Os_00120]		
Description	Optionally one task of an OS-Application may be defined as Restart Task. Multiplicity = 1: Restart Task is activated by the Operating System if the protection hook requests it. Multiplicity = 0: No task is automatically started after a protection error happened.		
Multiplicity	0..1		
Type	Reference to OsTask		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: ECU dependency: Required for scalability class 3 and 4.		

Name	OsTrusted {TRUSTED} [ECUC_Os_00115]		
Description	Parameter to specify if an OS-Application is trusted or not. true: OS-Application is trusted false: OS-Application is not trusted (default)		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value	false		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: ECU dependency: Required for scalability class 3 and 4.		

Included Containers		
Container Name	**Multiplicity**	**Scope / Depedency**
OsApplicationHooks	1	Container to structure the OS-Application-specific hooks
OsApplicationTrusted Function	0..*	Container to structure the configuration parameters of trusted functions

OsCounter

SWS Item	[ECUC_Os_00026]
Container Name	OsCounter {COUNTER}
Description	Configuration information for the counters that belong to the OsApplication.
Configuration Parameters	

Name	OsCounterAccessingApplication {ACCESSING_APPLICATION} [ECUC_Os_00031]		
Description	Reference to applications which have an access to this object.		
Multiplicity	0..*		
Type	Reference to OsApplication		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	OsCounterMaxAllowedValue {MAXALLOWEDVALUE} [ECUC_Os_00027]		
Description	Maximum possible allowed value of the system counter in ticks.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 18446744073709551615		
Default Value			
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	OsCounterMinCycle {MINCYCLE} [ECUC_Os_00028]		
Description	The MINCYCLE attribute specifies the minimum allowed number of counter ticks for a cyclic alarm linked to the counter.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 18446744073709551615		
Default Value			
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	OsCounterTicksPerBase {TICKSPERBASE} [ECUC_Os_00029]		
Description	The TICKSPERBASE attribute specifies the number of ticks required to reach a counterspecific unit. The interpretation is implementation-specific.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default Value			
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	OsCounterType {TYPE} [ECUC_Os_00255]		
Description	This parameter contains the natural type or unit of the counter.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	HARDWARE	This counter is driven by some hardware e.g. a hardware timer unit.	
	SOFTWARE	The counter is driven by some software which calls the IncrementCounter service.	
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: ECU		

Name	OsSecondsPerTick [ECUC_Os_00030]		
Description	Time of one counter tick in seconds.		
Multiplicity	0..1		
Type	EcucFloatParamDef		
Range	0 .. INF		
Default Value			
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: ECU		

Included Containers		
Container Name	**Multiplicity**	**Scope / Depedency**
OsDriver	0..1	This Container contains the information who will drive the counter. This configuration is only valid if the counter has OsCounterType set to HARDWARE. If the container does not exist (multiplicity=0) the timer is managed by the OS internally (OSINTERNAL). If the container exists the OS can use the GPT interface to manage the timer. The user have to supply the GPT channel. If the counter is driven by some other (external to the OS) source (like a TPU for example) this must be described as a vendor specific extension.
OsTimeConstant	0..*	Allows the user to define constants which can be e.g. used to compare time values with timer tick values. `A time value will be converted to a timer tick value during generation and can later on accessed via the OsConstName. The conversation is done by rounding time values to the nearest fitting tick value.`

OsEvent

SWS Item	[ECUC_Os_00033]
Container Name	OsEvent {EVENT}
Description	Representation of OS events in the configuration context. Adopted from the OSEK OIL specification.
Configuration Parameters	

Name	OsEventMask {MASK} [ECUC_Os_00034]		
Description	If event mask would be set to AUTO in OIL, this parameter should be omitted here.		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	0 .. 18446744073709551615		
Default Value			
Configuration Class	Pre-compile time	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

No Included Containers

OsScheduleTable

SWS Item	[ECUC_Os_00141]
Container Name	OsScheduleTable {SCHEDULETABLE}
Description	An OsScheduleTable addresses the synchronization issue by providing an encapsulation of a statically defined set of alarms that cannot be modified at runtime.
Configuration Parameters	

Name	OsSchTblAccessingApplication {ACCESSING_APPLICATION} [ECUC_Os_00054]		
Description	Reference to applications which have an access to this object.		
Multiplicity	0..*		
Type	Reference to OsApplication		
Configuration Class	Pre-compile time	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	OsScheduleTableCounterRef {COUNTER} [ECUC_Os_00145]		
Description	This parameter contains a reference to the counter which drives the schedule table.		
Multiplicity	1		
Type	Reference to OsCounter		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: ECU		

Name	OsScheduleTableDuration [ECUC_Os_00053]		
Description	This parameter defines the modulus of the schedule table (in ticks).		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 18446744073709551615		
Default Value			
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	OsScheduleTableRepeating {REPEATING} [ECUC_Os_00144]		
Description	true: first expiry point on the schedule table shall be processed at final expiry point delay ticks after the final expiry point is processed. false: the schedule table processing stops when the final expiry point is processed.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value			
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: ECU		

Included Containers		
Container Name	**Multiplicity**	**Scope / Depedency**
OsScheduleTable Autostart	0..1	This container specifies if and how the schedule table is started on startup of the Operating System. The options to start a schedule table correspond to the API calls to start schedule tables during runtime.
OsScheduleTableExpiry Point	1..*	The point on a Schedule Table at which the OS activates tasks and/or sets events
OsScheduleTableSync	0..1	This container specifies the synchronization parameters of the schedule table.

OsScheduleTableExpiryPoint

SWS Item	[ECUC_Os_00143]
Container Name	OsScheduleTableExpiryPoint {ACTION}
Description	The point on a Schedule Table at which the OS activates tasks and/or sets events
Configuration Parameters	

Name	OsScheduleTblExpPointOffset [ECUC_Os_00062]		
Description	The offset from zero (in ticks) at which the expiry point is to be processed.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 18446744073709551615		
Default Value			
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency			

Included Containers		
Container Name	**Multiplicity**	**Scope / Depedency**
OsScheduleTableEventSetting	0..*	Event that is triggered by that schedule table.
OsScheduleTableTaskActivation	0..*	Task that is triggered by that schedule table.
OsScheduleTblAdjustableExpPoint	0..1	Adjustable expiry point

OsTask

SWS Item	[ECUC_Os_00073]
Container Name	OsTask {TASK}
Description	This container represents an OSEK task.
Configuration Parameters	

Name	OsTaskAccessingApplication {ACCESSING_APPLICATION} [ECUC_Os_00077]		
Description	Reference to applications which have an access to this object.		
Multiplicity	0..*		
Type	Reference to OsApplication		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	OsTaskActivation {ACTIVATION} [ECUC_Os_00074]		
Description	This attribute defines the maximum number of queued activation requests for the task. A value equal to "1" means that at any time only a single activation is permitted for this task. Note that the value must be a natural number starting at 1.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default Value			
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	OsTaskEventRef {EVENT} [ECUC_Os_00078]		
Description	This reference defines the list of events the extended task may react on.		
Multiplicity	0..*		
Type	Reference to OsEvent		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	OsTaskPriority {PRIORITY} [ECUC_Os_00075]		
Description	The priority of a task is defined by the value of this attribute. This value has to be understood as a relative value, i.e. the values show only the relative ordering of the tasks. OSEK OS defines the lowest priority as zero (0); larger values correspond to higher priorities.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default Value			
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	OsTaskResourceRef {RESOURCE} [ECUC_Os_00079]		
Description	This reference defines a list of resources accessed by this task.		
Multiplicity	0..*		
Type	Reference to OsResource		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	OsTaskSchedule {SCHEDULE} [ECUC_Os_00076]			
Description	The OsTaskSchedule attribute defines the preemptability of the task. If this attribute is set to NON, no internal resources may be assigned to this task.			
Multiplicity	1			
Type	EcucEnumerationParamDef			
Range	FULL	Task is preemptable.		
	NON	Task is not preemptable.		
Configuration Class	**Pre-compile time**	X	All Variants	
	Link time	–		
	Post-build time	–		
Scope / Dependency	scope: local			

Included Containers		
Container Name	**Multiplicity**	**Scope / Depedency**
OsTaskAutostart	0..1	This container determines whether the task is activated during the system start-up procedure or not for some specific application modes. If the task shall be activated during the system start-up, this container is present and holds the references to the application modes in which the task is auto-started.
OsTaskTimingProtection	0..1	This container contains all parameters regarding timing protection of the task.

Module Name	EcuC	
Module Description	Virtual module to collect ECU Configuration specific / global configuration information.	
Included Containers		
Container Name	**Multiplicity**	**Scope / Dependency**
EcucConfigSet	0..1	This container contains the configuration parameters and sub containers of the global PduCollection. This container is a MultipleConfigurationContainer, i.e. this container and its sub-containers exist once per configuration set.
EcucHardware	0..1	Hardware definition of this Ecu.
EcucPartitionCollection	0..1	Collection of Partitions defined for this ECU.
EcucUnitGroupAssignment	0..1	Collection of UnitGroup references to support the generation of ASAM MCD file.
EcucVariationResolver	0..1	Collection of PredefinedVariant elements containing definition of values for SwSystemconst which shall be applied when resolving the variability during ECU Configuration.

F Examples

This chapter contains more detailed information for examples which were shown inside the preceding chapters of the specification.

F.1 ModeDeclarationGroupMapping

The example for **Mapping of ModeDeclarations** in chapter 4.4.4 is based on the following ARXML:

```
<?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http:
    //autosar.org/schema/r4.0" xsi:schemaLocation="http://autosar.org/schema
    /r4.0␣AUTOSAR_4-0-4.xsd">
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>Demo</SHORT-NAME>
      <DESC>
        <L-2 L="EN">Example about Connection of Mode Managers and Mode
            Users with different number of ModeDeclarations</L-2>
      </DESC>
      <CATEGORY>EXAMPLE</CATEGORY>
      <AR-PACKAGES>
        <AR-PACKAGE>
          <SHORT-NAME>SwComponentTypes</SHORT-NAME>
          <ELEMENTS>
            <APPLICATION-SW-COMPONENT-TYPE>
              <SHORT-NAME>ModeManager</SHORT-NAME>
              <PORTS>
                <P-PORT-PROTOTYPE>
                  <SHORT-NAME>EcuState</SHORT-NAME>
                  <PROVIDED-COM-SPECS>
                    <MODE-SWITCH-SENDER-COM-SPEC>
                      <ENHANCED-MODE-API>true</ENHANCED-MODE-API>
                      <MODE-GROUP-REF DEST="MODE-DECLARATION-GROUP-
                          PROTOTYPE">/Demo/PortInterfaces/EcuStatesExtended/
                          EcuStatesExtended</MODE-GROUP-REF>
                      <QUEUE-LENGTH>1</QUEUE-LENGTH>
                    </MODE-SWITCH-SENDER-COM-SPEC>
                  </PROVIDED-COM-SPECS>
                  <PROVIDED-INTERFACE-TREF DEST="MODE-SWITCH-INTERFACE">/
                      Demo/PortInterfaces/EcuStatesExtended</PROVIDED-
                      INTERFACE-TREF>
                </P-PORT-PROTOTYPE>
              </PORTS>
            </APPLICATION-SW-COMPONENT-TYPE>
            <APPLICATION-SW-COMPONENT-TYPE>
              <SHORT-NAME>ModeUser</SHORT-NAME>
              <PORTS>
                <R-PORT-PROTOTYPE>
                  <SHORT-NAME>EcuState</SHORT-NAME>
                  <REQUIRED-COM-SPECS>
                    <MODE-SWITCH-RECEIVER-COM-SPEC>
```

```
          <ENHANCED-MODE-API>1</ENHANCED-MODE-API>
          <SUPPORTS-ASYNCHRONOUS-MODE-SWITCH>false</SUPPORTS-
             ASYNCHRONOUS-MODE-SWITCH>
        </MODE-SWITCH-RECEIVER-COM-SPEC>
      </REQUIRED-COM-SPECS>
      <REQUIRED-INTERFACE-TREF DEST="MODE-SWITCH-INTERFACE">/
         Demo/PortInterfaces/EcuStatesBasic</REQUIRED-INTERFACE
         -TREF>
    </R-PORT-PROTOTYPE>
  </PORTS>
</APPLICATION-SW-COMPONENT-TYPE>
<COMPOSITION-SW-COMPONENT-TYPE>
  <SHORT-NAME>DemoEcu</SHORT-NAME>
  <COMPONENTS>
    <SW-COMPONENT-PROTOTYPE>
      <SHORT-NAME>ModeManager</SHORT-NAME>
      <TYPE-TREF DEST="APPLICATION-SW-COMPONENT-TYPE">/Demo/
         SwComponentTypes/ModeManager</TYPE-TREF>
    </SW-COMPONENT-PROTOTYPE>
    <SW-COMPONENT-PROTOTYPE>
      <SHORT-NAME>ModeUser</SHORT-NAME>
      <TYPE-TREF DEST="APPLICATION-SW-COMPONENT-TYPE">/Demo/
         SwComponentTypes/ModeUser</TYPE-TREF>
    </SW-COMPONENT-PROTOTYPE>
  </COMPONENTS>
  <CONNECTORS>
    <ASSEMBLY-SW-CONNECTOR>
      <SHORT-NAME>ModeManager_EcuState_ModeUser_EcuState</SHORT
         -NAME>
      <MAPPING-REF DEST="MODE-INTERFACE-MAPPING">/Demo/
         PortInterfaceMappingSets/ModeSwitchInterfaceMapping/
         EcuStatesExtended_2_EcuStatesBasic</MAPPING-REF>
      <PROVIDER-IREF>
        <CONTEXT-COMPONENT-REF DEST="SW-COMPONENT-PROTOTYPE">/
           Demo/SwComponentTypes/DemoEcu/ModeManager</CONTEXT-
           COMPONENT-REF>
        <TARGET-P-PORT-REF DEST="P-PORT-PROTOTYPE">/Demo/
           SwComponentTypes/ModeManager/EcuState</TARGET-P-PORT
           -REF>
      </PROVIDER-IREF>
      <REQUESTER-IREF>
        <CONTEXT-COMPONENT-REF DEST="SW-COMPONENT-PROTOTYPE">/
           Demo/SwComponentTypes/DemoEcu/ModeUser</CONTEXT-
           COMPONENT-REF>
        <TARGET-R-PORT-REF DEST="R-PORT-PROTOTYPE">/Demo/
           SwComponentTypes/ModeUser/EcuState</TARGET-R-PORT-
           REF>
      </REQUESTER-IREF>
    </ASSEMBLY-SW-CONNECTOR>
  </CONNECTORS>
</COMPOSITION-SW-COMPONENT-TYPE>
</ELEMENTS>
</AR-PACKAGE>
<AR-PACKAGE>
  <SHORT-NAME>PortInterfaces</SHORT-NAME>
  <ELEMENTS>
```

```
        <MODE-SWITCH-INTERFACE>
          <SHORT-NAME>EcuStatesBasic</SHORT-NAME>
          <MODE-GROUP>
            <SHORT-NAME>EcuStatesBasic</SHORT-NAME>
            <SW-CALIBRATION-ACCESS>READ-ONLY</SW-CALIBRATION-ACCESS>
            <TYPE-TREF DEST="MODE-DECLARATION-GROUP">/Demo/
               ModeDeclarationGroups/EcuStatesBasic</TYPE-TREF>
          </MODE-GROUP>
        </MODE-SWITCH-INTERFACE>
        <MODE-SWITCH-INTERFACE>
          <SHORT-NAME>EcuStatesExtended</SHORT-NAME>
          <MODE-GROUP>
            <SHORT-NAME>EcuStatesExtended</SHORT-NAME>
            <SW-CALIBRATION-ACCESS>READ-ONLY</SW-CALIBRATION-ACCESS>
            <TYPE-TREF DEST="MODE-DECLARATION-GROUP">/Demo/
               ModeDeclarationGroups/EcuStatesExtended</TYPE-TREF>
          </MODE-GROUP>
        </MODE-SWITCH-INTERFACE>
      </ELEMENTS>
    </AR-PACKAGE>
    <AR-PACKAGE>
      <SHORT-NAME>ModeDeclarationGroups</SHORT-NAME>
      <ELEMENTS>
        <MODE-DECLARATION-GROUP>
          <SHORT-NAME>EcuStatesBasic</SHORT-NAME>
          <CATEGORY>EXPLICIT_ORDER</CATEGORY>
          <INITIAL-MODE-REF DEST="MODE-DECLARATION">/Demo/
             ModeDeclarationGroups/EcuStatesBasic/STARTUP</INITIAL-MODE
             -REF>

          <MODE-DECLARATIONS>
            <MODE-DECLARATION>
              <SHORT-NAME>STARTUP</SHORT-NAME>
              <DESC>
                <L-2 L="EN">Startup phase of the Ecu</L-2>
              </DESC>
              <VALUE>1</VALUE>
            </MODE-DECLARATION>
            <MODE-DECLARATION>
              <SHORT-NAME>RUN</SHORT-NAME>
              <DESC>
                <L-2 L="EN">Run phase of the Ecu</L-2>
              </DESC>
              <VALUE>2</VALUE>
            </MODE-DECLARATION>
            <MODE-DECLARATION>
              <SHORT-NAME>POST_RUN</SHORT-NAME>
              <DESC>
                <L-2 L="EN">post run phase of the Ecu</L-2>
              </DESC>
              <VALUE>3</VALUE>
            </MODE-DECLARATION>
            <MODE-DECLARATION>
              <SHORT-NAME>SHUTDOWN</SHORT-NAME>
              <DESC>
                <L-2 L="EN">shutdown phase of the Ecu</L-2>
```

```
      </DESC>
      <VALUE>4</VALUE>
    </MODE-DECLARATION>
  </MODE-DECLARATIONS>
  <MODE-TRANSITIONS>
  <MODE-TRANSITION>
    <SHORT-NAME>STARTUP_RUN</SHORT-NAME>
    <ENTERED-MODE-REF DEST="MODE-DECLARATION">/Demo/
       ModeDeclarationGroups/EcuStatesBasic/RUN</ENTERED-MODE-
       REF>
    <EXITED-MODE-REF DEST="MODE-DECLARATION">/Demo/
       ModeDeclarationGroups/EcuStatesBasic/STARTUP</EXITED-
       MODE-REF>
  </MODE-TRANSITION>
  <MODE-TRANSITION>
    <SHORT-NAME>STARTUP_POST_RUN</SHORT-NAME>
    <ENTERED-MODE-REF DEST="MODE-DECLARATION">/Demo/
       ModeDeclarationGroups/EcuStatesBasic/POST_RUN</ENTERED-
       MODE-REF>
    <EXITED-MODE-REF DEST="MODE-DECLARATION">/Demo/
       ModeDeclarationGroups/EcuStatesBasic/STARTUP</EXITED-
       MODE-REF>
  </MODE-TRANSITION>
  <MODE-TRANSITION>
    <SHORT-NAME>RUN_POST_RUN</SHORT-NAME>
    <ENTERED-MODE-REF DEST="MODE-DECLARATION">/Demo/
       ModeDeclarationGroups/EcuStatesBasic/POST_RUN</ENTERED-
       MODE-REF>
    <EXITED-MODE-REF DEST="MODE-DECLARATION">/Demo/
       ModeDeclarationGroups/EcuStatesBasic/RUN</EXITED-MODE-
       REF>
  </MODE-TRANSITION>
  <MODE-TRANSITION>
    <SHORT-NAME>POST_RUN_SHUTDOWN</SHORT-NAME>
    <ENTERED-MODE-REF DEST="MODE-DECLARATION">/Demo/
       ModeDeclarationGroups/EcuStatesBasic/SHUTDOWN</ENTERED-
       MODE-REF>
    <EXITED-MODE-REF DEST="MODE-DECLARATION">/Demo/
       ModeDeclarationGroups/EcuStatesBasic/POST_RUN</EXITED-
       MODE-REF>
  </MODE-TRANSITION>
  </MODE-TRANSITIONS>
  <ON-TRANSITION-VALUE>0</ON-TRANSITION-VALUE>
</MODE-DECLARATION-GROUP>
<MODE-DECLARATION-GROUP>
  <SHORT-NAME>EcuStatesExtended</SHORT-NAME>
  <CATEGORY>ALPHABETIC_ORDER</CATEGORY>
  <INITIAL-MODE-REF DEST="MODE-DECLARATION">/Demo/
     ModeDeclarationGroups/EcuStatesExtended/StartUp</INITIAL-
     MODE-REF>
  <MODE-DECLARATIONS>
    <MODE-DECLARATION>
      <SHORT-NAME>StartUp</SHORT-NAME>
      <DESC>
        <L-2 L="EN">Start up phase of the Ecu</L-2>
      </DESC>
```

```
</MODE-DECLARATION>
<MODE-DECLARATION>
  <SHORT-NAME>Run</SHORT-NAME>
  <DESC>
    <L-2 L="EN">Run phase of the Ecu</L-2>
  </DESC>
</MODE-DECLARATION>
<MODE-DECLARATION>
  <SHORT-NAME>PostRun1</SHORT-NAME>
  <DESC>
    <L-2 L="EN">First post run phase of the Ecu</L-2>
  </DESC>
</MODE-DECLARATION>
<MODE-DECLARATION>
  <SHORT-NAME>PostRun2</SHORT-NAME>
  <DESC>
    <L-2 L="EN">Second post run phase of the Ecu</L-2>
  </DESC>
</MODE-DECLARATION>
<MODE-DECLARATION>
  <SHORT-NAME>ShutDown</SHORT-NAME>
  <DESC>
    <L-2 L="EN">Shut down phase of the Ecu</L-2>
  </DESC>
</MODE-DECLARATION>
<MODE-DECLARATION>
  <SHORT-NAME>Sleep</SHORT-NAME>
  <DESC>
    <L-2 L="EN">Sleep mode of the Ecu with reduced
        functionality</L-2>
  </DESC>
</MODE-DECLARATION>
<MODE-DECLARATION>
  <SHORT-NAME>Hibernate</SHORT-NAME>
  <DESC>
    <L-2 L="EN">Hibernate mode of the Ecu with extreme
        reduced functionality</L-2>
  </DESC>
</MODE-DECLARATION>
</MODE-DECLARATIONS>
</MODE-DECLARATION-GROUP>
</ELEMENTS>
</AR-PACKAGE>
<AR-PACKAGE>
  <SHORT-NAME>PortInterfaceMappingSets</SHORT-NAME>
  <ELEMENTS>
    <MODE-DECLARATION-MAPPING-SET>
      <SHORT-NAME>EcuStateMapping</SHORT-NAME>
      <MODE-DECLARATION-MAPPINGS>
        <MODE-DECLARATION-MAPPING>
          <SHORT-NAME>StartUp_2_STARTUP_</SHORT-NAME>
          <FIRST-MODE-REFS>
            <FIRST-MODE-REF DEST="MODE-DECLARATION">/Demo/
                ModeDeclarationGroups/EcuStatesExtended/StartUp</
                FIRST-MODE-REF>
          </FIRST-MODE-REFS>
```

```
      <SECOND-MODE-REF DEST="MODE-DECLARATION">/Demo/
          ModeDeclarationGroups/EcuStatesBasic/STARTUP</SECOND-
          MODE-REF>
    </MODE-DECLARATION-MAPPING>
    <MODE-DECLARATION-MAPPING>
      <SHORT-NAME>Run_2_RUN</SHORT-NAME>
      <FIRST-MODE-REFS>
        <FIRST-MODE-REF DEST="MODE-DECLARATION">/Demo/
            ModeDeclarationGroups/EcuStatesExtended/Run</FIRST-
            MODE-REF>
      </FIRST-MODE-REFS>
      <SECOND-MODE-REF DEST="MODE-DECLARATION">/Demo/
          ModeDeclarationGroups/EcuStatesBasic/RUN</SECOND-MODE-
          REF>
    </MODE-DECLARATION-MAPPING>
    <MODE-DECLARATION-MAPPING>
      <SHORT-NAME>PostRunX_2_POST_RUN</SHORT-NAME>
      <FIRST-MODE-REFS>
        <FIRST-MODE-REF DEST="MODE-DECLARATION">/Demo/
            ModeDeclarationGroups/EcuStatesExtended/PostRun1</
            FIRST-MODE-REF>
        <FIRST-MODE-REF DEST="MODE-DECLARATION">/Demo/
            ModeDeclarationGroups/EcuStatesExtended/PostRun2</
            FIRST-MODE-REF>
      </FIRST-MODE-REFS>
      <SECOND-MODE-REF DEST="MODE-DECLARATION">/Demo/
          ModeDeclarationGroups/EcuStatesBasic/POST_RUN</SECOND-
          MODE-REF>
    </MODE-DECLARATION-MAPPING>
    <MODE-DECLARATION-MAPPING>
      <SHORT-NAME>ShutDown_2_SHUTDOWN</SHORT-NAME>
      <FIRST-MODE-REFS>
        <FIRST-MODE-REF DEST="MODE-DECLARATION">/Demo/
            ModeDeclarationGroups/EcuStatesExtended/ShutDown</
            FIRST-MODE-REF>
      </FIRST-MODE-REFS>
      <SECOND-MODE-REF DEST="MODE-DECLARATION">/Demo/
          ModeDeclarationGroups/EcuStatesBasic/SHUTDOWN</SECOND-
          MODE-REF>
    </MODE-DECLARATION-MAPPING>
    <MODE-DECLARATION-MAPPING>
      <SHORT-NAME>Sleep_Hibernate_2_SHUTDOWN</SHORT-NAME>
      <FIRST-MODE-REFS>
        <FIRST-MODE-REF DEST="MODE-DECLARATION">/Demo/
            ModeDeclarationGroups/EcuStatesExtended/Sleep</FIRST
            -MODE-REF>
        <FIRST-MODE-REF DEST="MODE-DECLARATION">/Demo/
            ModeDeclarationGroups/EcuStatesExtended/Hibernate</
            FIRST-MODE-REF>
      </FIRST-MODE-REFS>
      <SECOND-MODE-REF DEST="MODE-DECLARATION">/Demo/
          ModeDeclarationGroups/EcuStatesBasic/SHUTDOWN</SECOND-
          MODE-REF>
    </MODE-DECLARATION-MAPPING>
  </MODE-DECLARATION-MAPPINGS>
</MODE-DECLARATION-MAPPING-SET>
```

```
          <PORT-INTERFACE-MAPPING-SET>
            <SHORT-NAME>ModeSwitchInterfaceMapping</SHORT-NAME>
            <PORT-INTERFACE-MAPPINGS>
              <MODE-INTERFACE-MAPPING>
                <SHORT-NAME>EcuStatesExtended_2_EcuStatesBasic</SHORT-
                   NAME>
                <MODE-MAPPING>
                  <FIRST-MODE-GROUP-REF DEST="MODE-DECLARATION-GROUP-
                     PROTOTYPE">/Demo/PortInterfaces/EcuStatesExtended/
                     EcuStatesExtended</FIRST-MODE-GROUP-REF>
                  <MODE-DECLARATION-MAPPING-SET-REF DEST="MODE-
                     DECLARATION-MAPPING-SET">/Demo/
                     PortInterfaceMappingSets/EcuStateMapping</MODE-
                     DECLARATION-MAPPING-SET-REF>
                  <SECOND-MODE-GROUP-REF DEST="MODE-DECLARATION-GROUP-
                     PROTOTYPE">/Demo/PortInterfaces/EcuStatesBasic/
                     EcuStatesBasic</SECOND-MODE-GROUP-REF>
                </MODE-MAPPING>
              </MODE-INTERFACE-MAPPING>
            </PORT-INTERFACE-MAPPINGS>
          </PORT-INTERFACE-MAPPING-SET>
        </ELEMENTS>
      </AR-PACKAGE>
    </AR-PACKAGES>
  </AR-PACKAGE>
</AR-PACKAGES>
</AUTOSAR>
```

F.2 Stability need for received data

The example for **Stability need for received data** in example 4.6 is based on the following ARXML:

```
<?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http:
    //autosar.org/schema/r4.0" xsi:schemaLocation="http://autosar.org/schema
    /r4.0␣AUTOSAR_4-0-4.xsd">
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>Demo</SHORT-NAME>
      <CATEGORY>EXAMPLE</CATEGORY>
      <AR-PACKAGES>
        <AR-PACKAGE>
          <SHORT-NAME>SwComponentTypes</SHORT-NAME>
          <ELEMENTS>
            <COMPOSITION-SW-COMPONENT-TYPE>
              <SHORT-NAME>COMP_1</SHORT-NAME>
              <DESC><L-2 L="EN">Stability need for received data (see SWS
                  RTE)</L-2></DESC>
              <CONSISTENCY-NEEDSS>
              <CONSISTENCY-NEEDS>
                <SHORT-NAME>CN_BC</SHORT-NAME>
                <DPG-DOES-NOT-REQUIRE-COHERENCYS>
```

```
<DATA-PROTOTYPE-GROUP>
  <SHORT-NAME>CN_BC_DG1</SHORT-NAME>
  <IMPLICIT-DATA-ACCESS-IREFS>
  <IMPLICIT-DATA-ACCESS-IREF>
  <CONTEXT-SW-COMPONENT-PROTOTYPE-REF DEST="SW-COMPONENT-
     PROTOTYPE">/Demo/SwComponentTypes/COMP_1/ASWC_B</
     CONTEXT-SW-COMPONENT-PROTOTYPE-REF>
  <CONTEXT-PORT-PROTOTYPE-REF DEST="R-PORT-PROTOTYPE">/Demo
     /SwComponentTypes/ASWC_B/A</CONTEXT-PORT-PROTOTYPE-REF
     >
  <TARGET-VARIABLE-DATA-PROTOTYPE-REF DEST="VARIABLE-DATA-
     PROTOTYPE">/Demo/PortInterfaces/A/A</TARGET-VARIABLE-
     DATA-PROTOTYPE-REF>
  </IMPLICIT-DATA-ACCESS-IREF>

  <IMPLICIT-DATA-ACCESS-IREF>
  <CONTEXT-SW-COMPONENT-PROTOTYPE-REF DEST="SW-COMPONENT-
     PROTOTYPE">/Demo/SwComponentTypes/COMP_1/ASWC_C</
     CONTEXT-SW-COMPONENT-PROTOTYPE-REF>
  <CONTEXT-PORT-PROTOTYPE-REF DEST="R-PORT-PROTOTYPE">/Demo
     /SwComponentTypes/ASWC_C/A</CONTEXT-PORT-PROTOTYPE-REF
     >
  <TARGET-VARIABLE-DATA-PROTOTYPE-REF DEST="VARIABLE-DATA-
     PROTOTYPE">/Demo/PortInterfaces/A/A</TARGET-VARIABLE-
     DATA-PROTOTYPE-REF>
  </IMPLICIT-DATA-ACCESS-IREF>

  <IMPLICIT-DATA-ACCESS-IREF>
  <CONTEXT-SW-COMPONENT-PROTOTYPE-REF DEST="SW-COMPONENT-
     PROTOTYPE">/Demo/SwComponentTypes/COMP_1/ASWC_B</
     CONTEXT-SW-COMPONENT-PROTOTYPE-REF>
  <CONTEXT-PORT-PROTOTYPE-REF DEST="R-PORT-PROTOTYPE">/Demo
     /SwComponentTypes/ASWC_B/B</CONTEXT-PORT-PROTOTYPE-REF
     >
  <TARGET-VARIABLE-DATA-PROTOTYPE-REF DEST="VARIABLE-DATA-
     PROTOTYPE">/Demo/PortInterfaces/B/B</TARGET-VARIABLE-
     DATA-PROTOTYPE-REF>
  </IMPLICIT-DATA-ACCESS-IREF>


  <IMPLICIT-DATA-ACCESS-IREF>
  <CONTEXT-SW-COMPONENT-PROTOTYPE-REF DEST="SW-COMPONENT-
     PROTOTYPE">/Demo/SwComponentTypes/COMP_1/ASWC_C</
     CONTEXT-SW-COMPONENT-PROTOTYPE-REF>
  <CONTEXT-PORT-PROTOTYPE-REF DEST="R-PORT-PROTOTYPE">/Demo
     /SwComponentTypes/ASWC_C/B</CONTEXT-PORT-PROTOTYPE-REF
     >
  <TARGET-VARIABLE-DATA-PROTOTYPE-REF DEST="VARIABLE-DATA-
     PROTOTYPE">/Demo/PortInterfaces/B/B</TARGET-VARIABLE-
     DATA-PROTOTYPE-REF>
  </IMPLICIT-DATA-ACCESS-IREF>

  </IMPLICIT-DATA-ACCESS-IREFS>
</DATA-PROTOTYPE-GROUP>
</DPG-DOES-NOT-REQUIRE-COHERENCYS>
<REG-REQUIRES-STABILITYS>
```

```
        <RUNNABLE-ENTITY-GROUP>
          <SHORT-NAME>CN_BC_RG1</SHORT-NAME>
          <RUNNABLE-ENTITY-IREFS>
          <RUNNABLE-ENTITY-IREF>
          <CONTEXT-SW-COMPONENT-PROTOTYPE-REF DEST="SW-COMPONENT-
             PROTOTYPE">/Demo/SwComponentTypes/COMP_1/ASWC_B</
             CONTEXT-SW-COMPONENT-PROTOTYPE-REF>
          <TARGET-RUNNABLE-ENTITY-REF DEST="RUNNABLE-ENTITY">/Demo/
             SwComponentTypes/ASWC_B/IB_ASWC_B/ASWC_B_RUN1</TARGET-
             RUNNABLE-ENTITY-REF>
          </RUNNABLE-ENTITY-IREF>
          <RUNNABLE-ENTITY-IREF>
          <CONTEXT-SW-COMPONENT-PROTOTYPE-REF DEST="SW-COMPONENT-
             PROTOTYPE">/Demo/SwComponentTypes/COMP_1/ASWC_C</
             CONTEXT-SW-COMPONENT-PROTOTYPE-REF>
          <TARGET-RUNNABLE-ENTITY-REF DEST="RUNNABLE-ENTITY">/Demo/
             SwComponentTypes/ASWC_C/IB_ASWC_C/ASWC_C_RUN1</TARGET-
             RUNNABLE-ENTITY-REF>
          </RUNNABLE-ENTITY-IREF>
          </RUNNABLE-ENTITY-IREFS>
        </RUNNABLE-ENTITY-GROUP>
        </REG-REQUIRES-STABILITYS>
      </CONSISTENCY-NEEDS>
      </CONSISTENCY-NEEDSS>
      <COMPONENTS>
        <SW-COMPONENT-PROTOTYPE>
          <SHORT-NAME>ASWC_A</SHORT-NAME>
          <TYPE-TREF DEST="APPLICATION-SW-COMPONENT-TYPE">/Demo/
             SwComponentTypes/ASWC_A</TYPE-TREF>
        </SW-COMPONENT-PROTOTYPE>
        <SW-COMPONENT-PROTOTYPE>
          <SHORT-NAME>ASWC_B</SHORT-NAME>
          <TYPE-TREF DEST="APPLICATION-SW-COMPONENT-TYPE">/Demo/
             SwComponentTypes/ASWC_B</TYPE-TREF>
        </SW-COMPONENT-PROTOTYPE>
        <SW-COMPONENT-PROTOTYPE>
          <SHORT-NAME>ASWC_C</SHORT-NAME>
          <TYPE-TREF DEST="APPLICATION-SW-COMPONENT-TYPE">/Demo/
             SwComponentTypes/ASWC_C</TYPE-TREF>
        </SW-COMPONENT-PROTOTYPE>
      </COMPONENTS>
    </COMPOSITION-SW-COMPONENT-TYPE>
    <APPLICATION-SW-COMPONENT-TYPE>
      <SHORT-NAME>ASWC_A</SHORT-NAME>
      <PORTS>
        <P-PORT-PROTOTYPE>
          <SHORT-NAME>A</SHORT-NAME>
          <PROVIDED-INTERFACE-TREF DEST="SENDER-RECEIVER-INTERFACE"
             >/Demo/PortInterfaces/A</PROVIDED-INTERFACE-TREF>
        </P-PORT-PROTOTYPE>
        <P-PORT-PROTOTYPE>
          <SHORT-NAME>B</SHORT-NAME>
          <PROVIDED-INTERFACE-TREF DEST="SENDER-RECEIVER-INTERFACE"
             >/Demo/PortInterfaces/B</PROVIDED-INTERFACE-TREF>
        </P-PORT-PROTOTYPE>
      </PORTS>
```

```xml
<INTERNAL-BEHAVIORS>
  <SWC-INTERNAL-BEHAVIOR>
    <SHORT-NAME>IB_ASWC_A</SHORT-NAME>
    <RUNNABLES>
      <RUNNABLE-ENTITY>
        <SHORT-NAME>ASWC_A_RUN1</SHORT-NAME>
        <DATA-WRITE-ACCESSS>
          <VARIABLE-ACCESS>
            <SHORT-NAME>DWP_ASWC_A_RUN1_A_A</SHORT-NAME>
            <ACCESSED-VARIABLE>
              <AUTOSAR-VARIABLE-IREF>
                <PORT-PROTOTYPE-REF DEST="P-PORT-PROTOTYPE">/
                    Demo/SwComponentTypes/ASWC_A/A</PORT-
                    PROTOTYPE-REF>
                <TARGET-DATA-PROTOTYPE-REF DEST="VARIABLE-
                    DATA-PROTOTYPE">/Demo/PortInterfaces/A/A</
                    TARGET-DATA-PROTOTYPE-REF>
              </AUTOSAR-VARIABLE-IREF>
            </ACCESSED-VARIABLE>
          </VARIABLE-ACCESS>
          <VARIABLE-ACCESS>
            <SHORT-NAME>DWP_ASWC_A_RUN1_B_B</SHORT-NAME>
            <ACCESSED-VARIABLE>
              <AUTOSAR-VARIABLE-IREF>
                <PORT-PROTOTYPE-REF DEST="P-PORT-PROTOTYPE">/
                    Demo/SwComponentTypes/ASWC_A/B</PORT-
                    PROTOTYPE-REF>
                <TARGET-DATA-PROTOTYPE-REF DEST="VARIABLE-
                    DATA-PROTOTYPE">/Demo/PortInterfaces/B/B</
                    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>
<APPLICATION-SW-COMPONENT-TYPE>
  <SHORT-NAME>ASWC_B</SHORT-NAME>
  <PORTS>
    <R-PORT-PROTOTYPE>
      <SHORT-NAME>A</SHORT-NAME>
      <REQUIRED-INTERFACE-TREF DEST="SENDER-RECEIVER-INTERFACE"
          >/Demo/PortInterfaces/A</REQUIRED-INTERFACE-TREF>
    </R-PORT-PROTOTYPE>
    <R-PORT-PROTOTYPE>
      <SHORT-NAME>B</SHORT-NAME>
      <REQUIRED-INTERFACE-TREF DEST="SENDER-RECEIVER-INTERFACE"
          >/Demo/PortInterfaces/B</REQUIRED-INTERFACE-TREF>
    </R-PORT-PROTOTYPE>
  </PORTS>
  <INTERNAL-BEHAVIORS>
    <SWC-INTERNAL-BEHAVIOR>
      <SHORT-NAME>IB_ASWC_B</SHORT-NAME>
```

```
          <RUNNABLES>
            <RUNNABLE-ENTITY>
              <SHORT-NAME>ASWC_B_RUN1</SHORT-NAME>
              <DATA-READ-ACCESSS>
                <VARIABLE-ACCESS>
                  <SHORT-NAME>DWP_ASWC_B_RUN1_A_A</SHORT-NAME>
                  <ACCESSED-VARIABLE>
                    <AUTOSAR-VARIABLE-IREF>
                      <PORT-PROTOTYPE-REF DEST="R-PORT-PROTOTYPE">/
                         Demo/SwComponentTypes/ASWC_B/A</PORT-
                         PROTOTYPE-REF>
                      <TARGET-DATA-PROTOTYPE-REF DEST="VARIABLE-
                         DATA-PROTOTYPE">/Demo/PortInterfaces/A/A</
                         TARGET-DATA-PROTOTYPE-REF>
                    </AUTOSAR-VARIABLE-IREF>
                  </ACCESSED-VARIABLE>
                </VARIABLE-ACCESS>
                <VARIABLE-ACCESS>
                  <SHORT-NAME>DWP_ASWC_B_RUN1_B_B</SHORT-NAME>
                  <ACCESSED-VARIABLE>
                    <AUTOSAR-VARIABLE-IREF>
                      <PORT-PROTOTYPE-REF DEST="R-PORT-PROTOTYPE">/
                         Demo/SwComponentTypes/ASWC_B/B</PORT-
                         PROTOTYPE-REF>
                      <TARGET-DATA-PROTOTYPE-REF DEST="VARIABLE-
                         DATA-PROTOTYPE">/Demo/PortInterfaces/B/B</
                         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>
    <APPLICATION-SW-COMPONENT-TYPE>
      <SHORT-NAME>ASWC_C</SHORT-NAME>
      <PORTS>
        <R-PORT-PROTOTYPE>
          <SHORT-NAME>A</SHORT-NAME>
          <REQUIRED-INTERFACE-TREF DEST="SENDER-RECEIVER-INTERFACE"
             >/Demo/PortInterfaces/A</REQUIRED-INTERFACE-TREF>
        </R-PORT-PROTOTYPE>
        <R-PORT-PROTOTYPE>
          <SHORT-NAME>B</SHORT-NAME>
          <REQUIRED-INTERFACE-TREF DEST="SENDER-RECEIVER-INTERFACE"
             >/Demo/PortInterfaces/B</REQUIRED-INTERFACE-TREF>
        </R-PORT-PROTOTYPE>
      </PORTS>
      <INTERNAL-BEHAVIORS>
        <SWC-INTERNAL-BEHAVIOR>
          <SHORT-NAME>IB_ASWC_C</SHORT-NAME>
          <RUNNABLES>
            <RUNNABLE-ENTITY>
              <SHORT-NAME>ASWC_C_RUN1</SHORT-NAME>
```

```
                <DATA-READ-ACCESSS>
                  <VARIABLE-ACCESS>
                    <SHORT-NAME>DWP_ASWC_C_RUN1_A_A</SHORT-NAME>
                    <ACCESSED-VARIABLE>
                      <AUTOSAR-VARIABLE-IREF>
                        <PORT-PROTOTYPE-REF DEST="R-PORT-PROTOTYPE">/
                            Demo/SwComponentTypes/ASWC_C/A</PORT-
                            PROTOTYPE-REF>
                        <TARGET-DATA-PROTOTYPE-REF DEST="VARIABLE-
                            DATA-PROTOTYPE">/Demo/PortInterfaces/A/A</
                            TARGET-DATA-PROTOTYPE-REF>
                      </AUTOSAR-VARIABLE-IREF>
                    </ACCESSED-VARIABLE>
                  </VARIABLE-ACCESS>
                  <VARIABLE-ACCESS>
                    <SHORT-NAME>DWP_ASWC_C_RUN1_B_B</SHORT-NAME>
                    <ACCESSED-VARIABLE>
                      <AUTOSAR-VARIABLE-IREF>
                        <PORT-PROTOTYPE-REF DEST="R-PORT-PROTOTYPE">/
                            Demo/SwComponentTypes/ASWC_C/B</PORT-
                            PROTOTYPE-REF>
                        <TARGET-DATA-PROTOTYPE-REF DEST="VARIABLE-
                            DATA-PROTOTYPE">/Demo/PortInterfaces/B/B</
                            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>
    </ELEMENTS>
  </AR-PACKAGE>
  <AR-PACKAGE>
    <SHORT-NAME>PortInterfaces</SHORT-NAME>
    <ELEMENTS>
      <SENDER-RECEIVER-INTERFACE>
        <SHORT-NAME>A</SHORT-NAME>
        <DATA-ELEMENTS>
          <VARIABLE-DATA-PROTOTYPE>
            <SHORT-NAME>A</SHORT-NAME>
          </VARIABLE-DATA-PROTOTYPE>
        </DATA-ELEMENTS>
      </SENDER-RECEIVER-INTERFACE>
      <SENDER-RECEIVER-INTERFACE>
        <SHORT-NAME>B</SHORT-NAME>
        <DATA-ELEMENTS>
          <VARIABLE-DATA-PROTOTYPE>
            <SHORT-NAME>B</SHORT-NAME>
          </VARIABLE-DATA-PROTOTYPE>
        </DATA-ELEMENTS>
      </SENDER-RECEIVER-INTERFACE>
    </ELEMENTS>
  </AR-PACKAGE>
```

```
      </AR-PACKAGES>
    </AR-PACKAGE>
  </AR-PACKAGES>
</AUTOSAR>
```

F.3 CompuMethod with bitfield texttable conversion

The following `CompuMethod` of `category` `BITFIELD_TEXTTABLE`

Listing F.1: example for bit field text table `CompuMethod`

```
<COMPU-METHOD>
  <SHORT-NAME>Texttable</SHORT-NAME>
  <CATEGORY>BITFIELD_TEXTTABLE</CATEGORY>
  <COMPU-INTERNAL-TO-PHYS>
    <COMPU-SCALES>
      <!-- problem -->
      <COMPU-SCALE>
        <SHORT-LABEL>problem</SHORT-LABEL>
        <SYMBOL>problem_flat_tire</SYMBOL>
        <MASK>0b11110000</MASK>
        <LOWER-LIMIT INTERVAL-TYPE="CLOSED">0b00000000</LOWER-LIMIT>
        <UPPER-LIMIT INTERVAL-TYPE="CLOSED">0b00000000</UPPER-LIMIT>
        <COMPU-CONST>
          <VT>flat tire</VT>
        </COMPU-CONST>
      </COMPU-SCALE>
      <COMPU-SCALE>
        <SHORT-LABEL>problem</SHORT-LABEL>
        <SYMBOL>problem_low_pressure</SYMBOL>
        <MASK>0b11110000</MASK>
        <LOWER-LIMIT INTERVAL-TYPE="CLOSED">0b00010000</LOWER-LIMIT>
        <UPPER-LIMIT INTERVAL-TYPE="CLOSED">0b00010000</UPPER-LIMIT>
        <COMPU-CONST>
          <VT>low pressure</VT>
        </COMPU-CONST>
      </COMPU-SCALE>
      <COMPU-SCALE>
        <SHORT-LABEL>problem</SHORT-LABEL>
        <SYMBOL>problem_unbalanced</SYMBOL>
        <MASK>0b11110000</MASK>
        <LOWER-LIMIT INTERVAL-TYPE="CLOSED">0b00100000</LOWER-LIMIT>
        <UPPER-LIMIT INTERVAL-TYPE="CLOSED">0b00100000</UPPER-LIMIT>
        <COMPU-CONST>
          <VT>unbalanced</VT>
        </COMPU-CONST>
      </COMPU-SCALE>
      <COMPU-SCALE>
        <SHORT-LABEL>problem</SHORT-LABEL>
        <SYMBOL>problem_unknown</SYMBOL>
        <MASK>0b11110000</MASK>
        <LOWER-LIMIT INTERVAL-TYPE="CLOSED">0b00110000</LOWER-LIMIT>
        <UPPER-LIMIT INTERVAL-TYPE="CLOSED">0b00110000</UPPER-LIMIT>
        <COMPU-CONST>
```

```
        <VT>unknown</VT>
      </COMPU-CONST>
    </COMPU-SCALE>
    <COMPU-SCALE>
      <SHORT-LABEL>problem</SHORT-LABEL>
      <SYMBOL>problem_invalid</SYMBOL>
      <MASK>0b11110000</MASK>
      <LOWER-LIMIT INTERVAL-TYPE="CLOSED">0b11110000</LOWER-LIMIT>
      <UPPER-LIMIT INTERVAL-TYPE="CLOSED">0b11110000</UPPER-LIMIT>
      <COMPU-CONST>
        <VT>invalid</VT>
      </COMPU-CONST>
    </COMPU-SCALE>
    <!-- rear right -->
    <COMPU-SCALE>
      <SHORT-LABEL>rearRight</SHORT-LABEL>
      <SYMBOL>rearRight_no</SYMBOL>
      <MASK>0b11001000</MASK>
      <LOWER-LIMIT INTERVAL-TYPE="CLOSED">0b00000000</LOWER-LIMIT>
      <UPPER-LIMIT INTERVAL-TYPE="CLOSED">0b00000000</UPPER-LIMIT>
      <COMPU-CONST>
        <VT>no</VT>
      </COMPU-CONST>
    </COMPU-SCALE>
    <COMPU-SCALE>
      <SHORT-LABEL>rearRight</SHORT-LABEL>
      <SYMBOL>rearRight_yes</SYMBOL>
      <MASK>0b11001000</MASK>
      <LOWER-LIMIT INTERVAL-TYPE="CLOSED">0b00001000</LOWER-LIMIT>
      <UPPER-LIMIT INTERVAL-TYPE="CLOSED">0b00001000</UPPER-LIMIT>
      <COMPU-CONST>
        <VT>yes</VT>
      </COMPU-CONST>
    </COMPU-SCALE>
    <!-- rear left -->
    <COMPU-SCALE>
      <SHORT-LABEL>rearLeft</SHORT-LABEL>
      <SYMBOL>rearLeft_no</SYMBOL>
      <MASK>0b11000100</MASK>
      <LOWER-LIMIT INTERVAL-TYPE="CLOSED">0b00000000</LOWER-LIMIT>
      <UPPER-LIMIT INTERVAL-TYPE="CLOSED">0b00000000</UPPER-LIMIT>
      <COMPU-CONST>
        <VT>no</VT>
      </COMPU-CONST>
    </COMPU-SCALE>
    <COMPU-SCALE>
      <SHORT-LABEL>rearLeft</SHORT-LABEL>
      <SYMBOL>rearLeft_yes</SYMBOL>
      <MASK>0b11000100</MASK>
      <LOWER-LIMIT INTERVAL-TYPE="CLOSED">0b00000100</LOWER-LIMIT>
      <UPPER-LIMIT INTERVAL-TYPE="CLOSED">0b00000100</UPPER-LIMIT>
      <COMPU-CONST>
        <VT>yes</VT>
      </COMPU-CONST>
    </COMPU-SCALE>
    <!-- front right -->
```

```
        <COMPU-SCALE>
          <SHORT-LABEL>frontRight</SHORT-LABEL>
          <SYMBOL>frontRight_no</SYMBOL>
          <MASK>0b11000010</MASK>
          <LOWER-LIMIT INTERVAL-TYPE="CLOSED">0b00000000</LOWER-LIMIT>
          <UPPER-LIMIT INTERVAL-TYPE="CLOSED">0b00000000</UPPER-LIMIT>
          <COMPU-CONST>
            <VT>no</VT>
          </COMPU-CONST>
        </COMPU-SCALE>
        <COMPU-SCALE>
          <SHORT-LABEL>frontRight</SHORT-LABEL>
          <SYMBOL>frontRight_yes</SYMBOL>
          <MASK>0b11000010</MASK>
          <LOWER-LIMIT INTERVAL-TYPE="CLOSED">0b00000010</LOWER-LIMIT>
          <UPPER-LIMIT INTERVAL-TYPE="CLOSED">0b00000010</UPPER-LIMIT>
          <COMPU-CONST>
            <VT>yes</VT>
          </COMPU-CONST>
        </COMPU-SCALE>
        <!-- front left -->
        <COMPU-SCALE>
          <SHORT-LABEL>frontLeft</SHORT-LABEL>
          <SYMBOL>frontLeft_no</SYMBOL>
          <MASK>0b11000001</MASK>
          <LOWER-LIMIT INTERVAL-TYPE="CLOSED">0b00000000</LOWER-LIMIT>
          <UPPER-LIMIT INTERVAL-TYPE="CLOSED">0b00000000</UPPER-LIMIT>
          <COMPU-CONST>
            <VT>no</VT>
          </COMPU-CONST>
        </COMPU-SCALE>
        <COMPU-SCALE>
          <SHORT-LABEL>frontLeft</SHORT-LABEL>
          <SYMBOL>frontLeft_yes</SYMBOL>
          <MASK>0b11000001</MASK>
          <LOWER-LIMIT INTERVAL-TYPE="CLOSED">0b00000001</LOWER-LIMIT>
          <UPPER-LIMIT INTERVAL-TYPE="CLOSED">0b00000001</UPPER-LIMIT>
          <COMPU-CONST>
            <VT>yes</VT>
          </COMPU-CONST>
        </COMPU-SCALE>
      </COMPU-SCALES>
    </COMPU-INTERNAL-TO-PHYS>
  </COMPU-METHOD>
```

results in this definitions:

Listing F.2: literals for bit field text table CompuMethod

```
/* [SWS_Rte_07410] unique "shortLabel" / "mask" pair "problem" / 0
   b11110000 */
#ifndef problem_BflMask
#define problem_BflMask 240U
#endif /* problem_BflMask */

```

```
/* [SWS_Rte_07411] unique "shortLabel" / "mask" pair "problem" / 0
    b11110000  with a single contiguous bit field*/
#ifndef problem_BflPn
#define problem_BflPn 4U
#endif /* problem_BflPn */

/* [SWS_Rte_07412] unique "shortLabel" / "mask" pair "problem" / 0
    b11110000  with a single contiguous bit field*/
#ifndef problem_BflLn
#define problem_BflLn 4U
#endif /* problem_BflLn */

/* [SWS_Rte_03810] CompuScale with point range "0b00000000", symbol
    attribute "problem_flat_tire"*/
#ifndef problem_flat_tire
#define problem_flat_tire 0U
#endif /* problem_flat_tire */

/* [SWS_Rte_03810] CompuScale with point range "0b00010000", symbol
    attribute "problem_low_pressure"*/
#ifndef problem_low_pressure
#define problem_low_pressure 16U
#endif /* problem_low_pressure */

/* [SWS_Rte_03810] CompuScale with point range "0b00100000", symbol
    attribute "problem_unbalanced"*/
#ifndef problem_unbalanced
#define problem_unbalanced 32U
#endif /* problem_unbalanced */

/* [SWS_Rte_03810] CompuScale with point range "0b00110000", symbol
    attribute "problem_unknown"*/
#ifndef problem_unknown
#define problem_unknown 48U
#endif /* problem_unknown */

/* [SWS_Rte_03810] CompuScale with point range "0b11110000", symbol
    attribute "problem_invalid"*/
#ifndef problem_invalid
#define problem_invalid 240U
#endif /* problem_invalid */

/* [SWS_Rte_07410] unique "shortLabel" / "mask" pair "rearRight" / 0
    b11001000 */
#ifndef rearRight_BflMask
#define rearRight_BflMask 200U
#endif /* rearRight_BflMask */

/* [SWS_Rte_07411] unique "shortLabel" / "mask" pair "rearRight" / 0
    b11001000  but not a single contiguous bit field*/

/* [SWS_Rte_07412] unique "shortLabel" / "mask" pair "rearRight" / 0
    b11001000  bot not a single contiguous bit field*/

/* [SWS_Rte_03810] CompuScale with point range "0b00000000", symbol
    attribute "rearRight_no"*/
```

```
#ifndef rearRight_no
#define rearRight_no 0U
#endif /* rearRight_no */

/* [SWS_Rte_03810] CompuScale with point range "0b00001000", symbol
    attribute "rearRight_yes"*/
#ifndef rearRight_yes
#define rearRight_yes 8U
#endif /* rearRight_yes */

/* [SWS_Rte_07410] unique "shortLabel" / "mask" pair "rearLeft" / 0
    b11000100 */
#ifndef rearLeft_BflMask
#define rearLeft_BflMask 200U
#endif /* rearLeft_BflMask */

/* [SWS_Rte_07411] unique "shortLabel" / "mask" pair "rearLeft" / 0
    b11000100  but not a single contiguous bit field*/

/* [SWS_Rte_07412] unique "shortLabel" / "mask" pair "rearLeft" / 0
    b11000100  bot not a single contiguous bit field*/

/* [SWS_Rte_03810] CompuScale with point range "0b00000000", symbol
    attribute "rearLeft_no"*/
#ifndef rearLeft_no
#define rearLeft_no 0U
#endif /* rearLeft_no */

/* [SWS_Rte_03810] CompuScale with point range "0b00000100", symbol
    attribute "rearLeft_yes"*/
#ifndef rearLeft_yes
#define rearLeft_yes 4U
#endif /* rearLeft_yes */

/* [SWS_Rte_07410] unique "shortLabel" / "mask" pair "frontRight" / 0
    b11000010 */
#ifndef frontRight_BflMask
#define frontRight_BflMask 194U
#endif /* frontRight_BflMask */

/* [SWS_Rte_07411] unique "shortLabel" / "mask" pair "frontRight" / 0
    b11000010  but not a single contiguous bit field*/

/* [SWS_Rte_07412] unique "shortLabel" / "mask" pair "frontRight" / 0
    b11000010  bot not a single contiguous bit field*/

/* [SWS_Rte_03810] CompuScale with point range "0b00000000", symbol
    attribute "frontRight_no"*/
#ifndef frontRight_no
#define frontRight_no 0U
#endif /* frontRight_no */

/* [SWS_Rte_03810] CompuScale with point range "0b00000010", symbol
    attribute "frontRight_yes"*/
#ifndef frontRight_yes
#define frontRight_yes 2U
```

```
#endif /* frontRight_yes */

/* [SWS_Rte_07410] unique "shortLabel" / "mask" pair "frontLeft" / 0
    b11000001 */
#ifndef frontLeft_BflMask
#define frontLeft_BflMask 193U
#endif /* frontLeft_BflMask */

/* [SWS_Rte_07411] unique "shortLabel" / "mask" pair "frontLeft" / 0
    b11000001  but not a single contiguous bit field*/

/* [SWS_Rte_07412] unique "shortLabel" / "mask" pair "frontLeft" / 0
    b11000001  bot not a single contiguous bit field*/

/* [SWS_Rte_03810] CompuScale with point range "0b00000000", symbol
    attribute "frontLeft_no"*/
#ifndef frontLeft_no
#define frontLeft_no 0U
#endif /* frontLeft_no */

/* [SWS_Rte_03810] CompuScale with point range "0b00000001", symbol
    attribute "frontLeft_yes"*/
#ifndef frontLeft_yes
#define frontLeft_yes 1U
#endif /* frontLeft_yes */
```

G Changes History

G.1 Changes in Rel. 4.0 Rev. 2 compared to Rel. 4.0 Rev. 1

G.1.1 Deleted SWS Items

The following SWS Items were removed in Rel. 4.0 Rev. 2: rte_sws_1254, rte_sws_3552, rte_sws_3557, rte_sws_3559, rte_sws_3563, rte_sws_3564, rte_sws_3568, rte_sws_3588, rte_sws_3593, rte_sws_3743, rte_sws_5512.

G.1.2 Changed SWS Items

The following SWS Items were changed in Rel. 4.0 Rev. 2: [SWS_Rte_01086], [SWS_Rte_01111], [SWS_Rte_01113], [SWS_Rte_01114], [SWS_Rte_01118], [SWS_Rte_01156], [SWS_Rte_01355], [SWS_Rte_02517], [SWS_Rte_02527], [SWS_Rte_02528], [SWS_Rte_02613], [SWS_Rte_02615], [SWS_Rte_02679], [SWS_Rte_02728], [SWS_Rte_02730], [SWS_Rte_02747], [SWS_Rte_02752], [SWS_Rte_02753], [SWS_Rte_03001], [SWS_Rte_03560], [SWS_Rte_03562], [SWS_Rte_03567], [SWS_Rte_03598], [SWS_Rte_03599], [SWS_Rte_03774], [SWS_Rte_03827], [SWS_Rte_03837], [SWS_Rte_03930], [SWS_Rte_03953], [SWS_Rte_03954], [SWS_Rte_03955], [SWS_Rte_03956], [SWS_Rte_03957], [SWS_Rte_05021], [SWS_Rte_05156], SWS_Rte_05506, [SWS_Rte_05509], [SWS_Rte_06010], [SWS_Rte_06633], [SWS_Rte_07020], [SWS_Rte_07021], [SWS_Rte_07041], [SWS_Rte_07184], [SWS_Rte_07187], [SWS_Rte_07195], [SWS_Rte_07262], [SWS_Rte_07280], [SWS_Rte_07282], [SWS_Rte_07293], [SWS_Rte_07294], [SWS_Rte_07375], [SWS_Rte_07376], [SWS_Rte_07409], [SWS_Rte_07586], [SWS_Rte_07589], [SWS_Rte_07632], [SWS_Rte_07636], [SWS_Rte_07637], [SWS_Rte_07667], [SWS_Rte_07680], [SWS_Rte_07683], rte_sws_ext_3811.

G.1.3 Added SWS Items

The following SWS Items were added in Rel. 4.0 Rev. 2: [SWS_Rte_02761], rte_sws_3850, rte_sws_3851, [SWS_Rte_03852], [SWS_Rte_03853], [SWS_Rte_07045], [SWS_Rte_07046], [SWS_Rte_07047], [SWS_Rte_07048], [SWS_Rte_07049], [SWS_Rte_07050], [SWS_Rte_07051], [SWS_Rte_07052], [SWS_Rte_07053], [SWS_Rte_07054], [SWS_Rte_07055], [SWS_Rte_07056], [SWS_Rte_07057], [SWS_Rte_07058], [SWS_Rte_07059], [SWS_Rte_07060], [SWS_Rte_07061], [SWS_Rte_07062], [SWS_Rte_07063], [SWS_Rte_07064], [SWS_Rte_07065], [SWS_Rte_07066], [SWS_Rte_07067], [SWS_Rte_07068], [SWS_Rte_07069], [SWS_Rte_07070], [SWS_Rte_07071], [SWS_Rte_07072], [SWS_Rte_07073], [SWS_Rte_07074], [SWS_Rte_07075], [SWS_Rte_07076], [SWS_Rte_07077], [SWS_Rte_07078], [SWS_Rte_07079], [SWS_Rte_07080], [SWS_Rte_07081], [SWS_Rte_08000], [SWS_Rte_08001], [SWS_Rte_08002], [SWS_Rte_08300], [SWS_Rte_08301], [SWS_Rte_08302].

G.2 Changes in Rel. 4.0 Rev. 3 compared to Rel. 4.0 Rev. 2

G.2.1 Deleted SWS Items

The following SWS Items were removed in Rel. 4.0 Rev. 3: rte_sws_3838, rte_sws_3844, rte_sws_3850, rte_sws_5171, rte_sws_7106, rte_sws_7108, rte_sws_7164, rte_sws_7165, rte_sws_7168, rte_sws_7176, rte_sws_7674.

G.2.2 Changed SWS Items

The following SWS Items were changed in Rel. 4.0 Rev. 3: [SWS_Rte_01018], [SWS_Rte_01019], [SWS_Rte_01020], [SWS_Rte_01156], [SWS_Rte_01171], [SWS_Rte_01238], [SWS_Rte_01239], [SWS_Rte_01248], [SWS_Rte_01249], [SWS_Rte_01300], [SWS_Rte_02500], [SWS_Rte_02568], [SWS_Rte_02576], [SWS_Rte_02627], [SWS_Rte_02628], [SWS_Rte_02629], [SWS_Rte_02631], [SWS_Rte_02648], [SWS_Rte_02659], [SWS_Rte_02662], [SWS_Rte_02664], [SWS_Rte_02675], [SWS_Rte_02732], [SWS_Rte_03526], [SWS_Rte_03714], [SWS_Rte_03731], [SWS_Rte_03782], [SWS_Rte_03793], [SWS_Rte_03809], [SWS_Rte_03810], [SWS_Rte_03813], [SWS_Rte_03827], [SWS_Rte_03828], [SWS_Rte_03829], [SWS_Rte_03831], [SWS_Rte_03832], [SWS_Rte_03833], [SWS_Rte_03837], [SWS_Rte_03839], [SWS_Rte_03840], [SWS_Rte_03841], [SWS_Rte_03842], [SWS_Rte_03843], [SWS_Rte_03845], [SWS_Rte_03846], [SWS_Rte_03847], [SWS_Rte_03848], [SWS_Rte_03849], [SWS_Rte_03851], [SWS_Rte_03907], [SWS_Rte_03949], [SWS_Rte_04526], [SWS_Rte_05051], [SWS_Rte_05052], SWS_Rte_05059, [SWS_Rte_05062], [SWS_Rte_05078], [SWS_Rte_05127], [SWS_Rte_05128], [SWS_Rte_06513], [SWS_Rte_06515], [SWS_Rte_06518], [SWS_Rte_06519], [SWS_Rte_06520], [SWS_Rte_06530], [SWS_Rte_06532], [SWS_Rte_06535], [SWS_Rte_06536], [SWS_Rte_07022], [SWS_Rte_07030], [SWS_Rte_07036], [SWS_Rte_07037], [SWS_Rte_07038], [SWS_Rte_07047], [SWS_Rte_07048], [SWS_Rte_07069], [SWS_Rte_07104], [SWS_Rte_07109], [SWS_Rte_07110], [SWS_Rte_07111], [SWS_Rte_07113], [SWS_Rte_07114], [SWS_Rte_07116], [SWS_Rte_07133], [SWS_Rte_07136], [SWS_Rte_07144], [SWS_Rte_07148], [SWS_Rte_07149], [SWS_Rte_07157], [SWS_Rte_07162], [SWS_Rte_07163], [SWS_Rte_07166], [SWS_Rte_07175], [SWS_Rte_07182], [SWS_Rte_07185], [SWS_Rte_07190], [SWS_Rte_07194], [SWS_Rte_07195], [SWS_Rte_07200], [SWS_Rte_07203], [SWS_Rte_07214], [SWS_Rte_07224], [SWS_Rte_07250], [SWS_Rte_07253], [SWS_Rte_07255], [SWS_Rte_07260], [SWS_Rte_07261], [SWS_Rte_07263], [SWS_Rte_07266], [SWS_Rte_07282], [SWS_Rte_07292], [SWS_Rte_07293], [SWS_Rte_07294], [SWS_Rte_07295], [SWS_Rte_07310], [SWS_Rte_07315], [SWS_Rte_07381], [SWS_Rte_07382], [SWS_Rte_07383], [SWS_Rte_07501], [SWS_Rte_07503], [SWS_Rte_07504], [SWS_Rte_07543], [SWS_Rte_07544], [SWS_Rte_07552], [SWS_Rte_07554], [SWS_Rte_07555], [SWS_Rte_07556], [SWS_Rte_07670], [SWS_Rte_07682], [SWS_Rte_08300].

G.2.3 Added SWS Items

The following SWS Items were added in Rel. 4.0 Rev. 3: [SWS_Rte_03854], [SWS_Rte_03855], [SWS_Rte_03856], [SWS_Rte_03857], [SWS_Rte_03858], [SWS_Rte_03859], [SWS_Rte_03860], [SWS_Rte_03861], [SWS_Rte_06700], [SWS_Rte_06701], [SWS_Rte_06702], [SWS_Rte_06703], [SWS_Rte_06704], [SWS_Rte_06705], [SWS_Rte_06706], [SWS_Rte_06707], [SWS_Rte_06708], [SWS_Rte_06709], [SWS_Rte_06710], [SWS_Rte_06711], [SWS_Rte_06712], [SWS_Rte_06713], [SWS_Rte_06714], [SWS_Rte_06715], [SWS_Rte_06716], [SWS_Rte_06717], [SWS_Rte_06718], [SWS_Rte_06719], [SWS_Rte_06720], [SWS_Rte_06721], [SWS_Rte_06722], [SWS_Rte_06723], [SWS_Rte_06724], [SWS_Rte_06725], [SWS_Rte_06726], [SWS_Rte_07082], [SWS_Rte_07083], [SWS_Rte_07084], [SWS_Rte_07085], [SWS_Rte_07086], [SWS_Rte_07087], [SWS_Rte_07088], [SWS_Rte_07089], [SWS_Rte_07090], [SWS_Rte_07091], [SWS_Rte_07092], [SWS_Rte_07093], [SWS_Rte_07094], [SWS_Rte_07095], [SWS_Rte_07096], [SWS_Rte_07097], [SWS_Rte_07099], [SWS_Rte_07593], [SWS_Rte_07594], [SWS_Rte_07595], [SWS_Rte_07596], [SWS_Rte_07692], [SWS_Rte_07693], [SWS_Rte_07694], [SWS_Rte_07920], [SWS_Rte_07921], [SWS_Rte_07922], [SWS_Rte_07923], [SWS_Rte_07924], [SWS_Rte_08004], [SWS_Rte_08005], [SWS_Rte_08007], [SWS_Rte_08008], [SWS_Rte_08009], [SWS_Rte_08016], [SWS_Rte_08017], [SWS_Rte_08018], [SWS_Rte_08020], [SWS_Rte_08021], [SWS_Rte_08022], [SWS_Rte_08023], [SWS_Rte_08024], [SWS_Rte_08025], [SWS_Rte_08026], [SWS_Rte_08027], [SWS_Rte_08028], [SWS_Rte_08029], [SWS_Rte_08030], [SWS_Rte_08031], [SWS_Rte_08032], [SWS_Rte_08033], [SWS_Rte_08034], [SWS_Rte_08035], [SWS_Rte_08036], [SWS_Rte_08037], [SWS_Rte_08038], [SWS_Rte_08039], [SWS_Rte_08040], [SWS_Rte_08041], [SWS_Rte_08042], [SWS_Rte_08043], [SWS_Rte_08044], [SWS_Rte_08045], [SWS_Rte_08303], [SWS_Rte_08304], [SWS_Rte_08305], [SWS_Rte_08306], [SWS_Rte_08307], [SWS_Rte_08308], [SWS_Rte_08400], [SWS_Rte_08401], [SWS_Rte_08402], [SWS_Rte_08403], [SWS_Rte_08404], [SWS_Rte_08500], [SWS_Rte_08501], SWS_Rte_08503, [SWS_Rte_08504], [SWS_Rte_08505], [SWS_Rte_08506], [SWS_Rte_08507], [SWS_Rte_08509], [SWS_Rte_08510], rte_sws_ext_7597, rte_sws_ext_7598, rte_sws_ext_8502, rte_sws_ext_8508.

G.3 Changes in Rel. 4.1 Rev. 1 compared to Rel. 4.0 Rev. 3

G.3.1 Renamed SWS Items

The external requirements are redefined as AUTOSAR constraints.

rte_sws_ext_3811	[constr_9004]	Usage of WaitPoints is restricted depending on *ExclusiveAreaImplMechanism*
rte_sws_ext_7598	[constr_9005]	The references RteSwcTriggerSourceRef has to be consistent with the RteSoftwareComponentInstanceRef
rte_sws_ext_7597	[constr_9006]	The references RteBswTriggerSourceRef has to be consistent with the RteBswImplementationRef
rte_sws_ext_7547	[constr_9007]	*issuedTrigger* and *BswTriggerDirectImplementation* are mutually exclusive
rte_sws_ext_7040	[constr_9008]	The same Trigger in a *Trigger Sink* must not be connected to multiple *Trigger Source*s
rte_sws_ext_7550	[constr_9009]	Synchronized Trigger shall not be referenced by more than one type of access method
rte_sws_ext_7521	[constr_9010]	Worst case execution time shall be less than the GCD
rte_sws_ext_7351	[constr_9011]	NvMBlockDescriptor related to a RAM Block of a NvBlockSwComponentType shall use NvMBlockUseSyncMechanism
rte_sws_ext_7816	[constr_9012]	Category 1 interrupts shall not access the RTE
rte_sws_ext_2542	[constr_9013]	Exactly one mode or one mode transition shall be active
rte_sws_ext_7565	[constr_9014]	*ModeSwitchPoint*(s) and *managedModeGroup*(s) are mutually exclusive for synchronized *ModeDeclarationGroupPrototype*s
rte_sws_ext_7818	[constr_9015]	Rte_Write API may only be used by the runnable that describe its usage
rte_sws_ext_7819	[constr_9016]	Rte_Send API may only be used by the runnable that describes its usage
rte_sws_ext_2681	[constr_9017]	Rte_Switch API may only be used by the runnable that describes its usage
rte_sws_ext_2682	[constr_9018]	Rte_Invalidate API may only be used by the runnable that describe its usage
rte_sws_ext_2687	[constr_9019]	Rte_Feedback API may only be used by the runnable that describe its usage
rte_sws_ext_2726	[constr_9020]	Rte_SwitchAck API may only be used by the runnable that describe its usage
rte_sws_ext_2683	[constr_9021]	Rte_Read API may only be used by the runnable that describe its usage
rte_sws_ext_7397	[constr_9022]	Rte_DRead API may only be used by the runnable that describe its usage
rte_sws_ext_2684	[constr_9023]	Rte_Receive API may only be used by the runnable that describe its usage
rte_sws_ext_2685	[constr_9024]	Rte_Call API may only be used by the runnable that describe its usage
rte_sws_ext_2686	[constr_9025]	Blocking Rte_Result API may only be used by the runnable that describe the WaitPoint

rte_sws_ext_7679	[constr_9026]	Rte_IWriteRef may not return values written in previous executions
rte_sws_ext_2601	[constr_9027]	Rte_IStatus API shall only be used by a RunnableEntity describing an access to the data or which is triggered by an error event related to this data
rte_sws_ext_7171	[constr_9028]	Rte_Enter and Rte_Exit API may only be used by runnables describing its usage
rte_sws_ext_7172	[constr_9029]	Nested call of Rte_Enter and Rte_Exit is restricted
rte_sws_ext_7568	[constr_9030]	Rte_Mode API may only be used by the runnable that describe its usage
rte_sws_ext_8502	[constr_9031]	Rte_Mode API may only be used by the runnable that describe its usage
rte_sws_ext_7202	[constr_9032]	Rte_Trigger API may only be used by the runnable that describe its usage
rte_sws_ext_7205	[constr_9033]	Rte_IrTrigger API may only be used by the runnable that describe its usage
rte_sws_ext_7603	[constr_9034]	Rte_IsUpdated API may only be used by the runnable that describe the access to the corresponding data
rte_sws_ext_2582	[constr_9035]	Rte_Start shall be called only once
rte_sws_ext_7577	[constr_9036]	Rte_Start API may only be used after call of SchM_Init
rte_sws_ext_2714	[constr_9037]	Rte_Start API shall be called on every core
rte_sws_ext_2583	[constr_9038]	Rte_Stop shall be called before BSW shutdown
rte_sws_ext_7332	[constr_9039]	Rte_PartitionTerminated shall be called only once
rte_sws_ext_7618	[constr_9040]	Rte_PartitionRestarting shall be called only once
rte_sws_ext_7337	[constr_9041]	Rte_RestartPartition shall be called from RestartTask
rte_sws_ext_1190	[constr_9042]	Array Implementation Data Types needs at least one element
rte_sws_ext_1192	[constr_9043]	Structure Implementation Data Types needs at least one element
rte_sws_ext_7147	[constr_9044]	*Union Implementation Data Type* shall include at least two elements
rte_sws_ext_2704	[constr_9045]	The upper two bits of the of the server return value are reserved
rte_sws_ext_7285	[constr_9046]	SchM_Enter and SchM_Exit API may only be used by BswModuleEntitys describing its usage
rte_sws_ext_7529	[constr_9047]	Nested call of SchM_Enter and SchM_Exit API is restricted
rte_sws_ext_7189	[constr_9048]	SchM_Exit API may only be used by BswModuleEntitys that describe its usage
rte_sws_ext_7257	[constr_9049]	SchM_Switch API may only be used by BswModuleEntitys that describe its usage
rte_sws_ext_7587	[constr_9050]	SchM_Mode API may only be used by BswModuleEntitys that describe its usage
rte_sws_ext_8508	[constr_9051]	SchM_Mode API may only be used by BswModuleEntitys that describe its usage

rte_sws_ext_7567	[constr_9052]	SchM_SwitchAck API may only be used by BswModuleEntitys that describe its usage
rte_sws_ext_7265	[constr_9053]	SchM_Trigger API may only be used by the BswModuleEntitys that describe its usage
rte_sws_ext_7268	[constr_9054]	SchM_ActMainFunction API may only be used by the BswModuleEntitys that describe its usage
rte_sws_ext_7272	[constr_9055]	SchM_Init shall be called only once
rte_sws_ext_7576	[constr_9056]	SchM_Deinit API may only be used after the was RTE finalized
rte_sws_ext_7276	[constr_9057]	SchM_Deinit shall be called before shut down of BSW
rte_sws_ext_7287	[constr_9058]	BswSchedulableEntity is not allowed to have service arguments or return value
rte_sws_ext_7512	[constr_9059]	Usage of *Basic Software Scheduler* API prerequisites the include of the *Module Interlink Header File*

Table G.1: external requirements converted to constraints

rte_sws_7649	[constr_9000]	Rte_IFeedback API may only be used by the RunnableEntitys that describe its usage

Table G.2: requirements converted to constraints

G.3.2 Added constraints

The following constraints were added in Rel. 4.1 Rev. 1: [constr_3510],[constr_9060],[constr_9061],[constr_9062],[constr_9063],[constr_9064]

G.3.3 Deleted SWS Items

The following SWS items were removed in Rel. 4.1 Rev. 1: SWS_Rte_02652, SWS_Rte_02731, SWS_Rte_03555, SWS_Rte_03569, SWS_Rte_03581, SWS_Rte_03747, SWS_Rte_03803, SWS_Rte_05020, SWS_Rte_05033, SWS_Rte_05054, SWS_Rte_05055, SWS_Rte_05056, SWS_Rte_05057, SWS_Rte_05058, SWS_Rte_05059, SWS_Rte_05066, SWS_Rte_05067, SWS_Rte_05110, SWS_Rte_05163, SWS_Rte_06028, SWS_Rte_07296, SWS_Rte_07649, SWS_Rte_07656, SWS_Rte_07657, SWS_Rte_07658, SWS_Rte_07665, SWS_Rte_07687, SWS_Rte_07688, SWS_Rte_07690, SWS_Rte_07691, SWS_Rte_08503.

G.3.4 Changed SWS Items

The following SWS items were changed in Rel. 4.1 Rev. 1: [SWS_Rte_01003], [SWS_Rte_01019], [SWS_Rte_01058], [SWS_Rte_01060], [SWS_Rte_01061], [SWS_Rte_01064], [SWS_Rte_01065], [SWS_Rte_01071], [SWS_Rte_01072], [SWS_Rte_01083], [SWS_Rte_01091], [SWS_Rte_01092], [SWS_Rte_01102], [SWS_Rte_01111], [SWS_Rte_01118], [SWS_Rte_01120], [SWS_Rte_01123], [SWS_Rte_01126], [SWS_Rte_01150], [SWS_Rte_01206], [SWS_Rte_01252], [SWS_Rte_01284], [SWS_Rte_01285], [SWS_Rte_01286], [SWS_Rte_01317], [SWS_Rte_01354], [SWS_Rte_01358], [SWS_Rte_01360], [SWS_Rte_01368], [SWS_Rte_02516], [SWS_Rte_02530], [SWS_Rte_02544], [SWS_Rte_02571], [SWS_Rte_02579], [SWS_Rte_02594], [SWS_Rte_02599], [SWS_Rte_02600], [SWS_Rte_02610], [SWS_Rte_02611], [SWS_Rte_02612], [SWS_Rte_02613], [SWS_Rte_02614], [SWS_Rte_02615], [SWS_Rte_02619], [SWS_Rte_02623], [SWS_Rte_02628], [SWS_Rte_02631], [SWS_Rte_02649], [SWS_Rte_02651], [SWS_Rte_02679], [SWS_Rte_02702], [SWS_Rte_02707], [SWS_Rte_02709], [SWS_Rte_02712], [SWS_Rte_02713], [SWS_Rte_02725], [SWS_Rte_02736], [SWS_Rte_02739], [SWS_Rte_02747], [SWS_Rte_02757], [SWS_Rte_02759], [SWS_Rte_02760], [SWS_Rte_03001], [SWS_Rte_03002], [SWS_Rte_03004], [SWS_Rte_03005], [SWS_Rte_03012], [SWS_Rte_03503], [SWS_Rte_03504], [SWS_Rte_03526], [SWS_Rte_03527], [SWS_Rte_03550], [SWS_Rte_03553], [SWS_Rte_03560], [SWS_Rte_03565], [SWS_Rte_03589], [SWS_Rte_03595], [SWS_Rte_03598], [SWS_Rte_03602], [SWS_Rte_03603], [SWS_Rte_03714], [SWS_Rte_03741], [SWS_Rte_03744], [SWS_Rte_03755], [SWS_Rte_03760], [SWS_Rte_03764], [SWS_Rte_03770], [SWS_Rte_03775], [SWS_Rte_03776], [SWS_Rte_03788], [SWS_Rte_03800], [SWS_Rte_03809], [SWS_Rte_03827], [SWS_Rte_03828], [SWS_Rte_03843], [SWS_Rte_03849], [SWS_Rte_03857], [SWS_Rte_03927], [SWS_Rte_03928], [SWS_Rte_03952], [SWS_Rte_03955], [SWS_Rte_03970], [SWS_Rte_04508], [SWS_Rte_04515], [SWS_Rte_04516], [SWS_Rte_04518], [SWS_Rte_05021], [SWS_Rte_05026], [SWS_Rte_05048], [SWS_Rte_05052], [SWS_Rte_05065], [SWS_Rte_05084], [SWS_Rte_05085], [SWS_Rte_05090], [SWS_Rte_05111], [SWS_Rte_05131], [SWS_Rte_05145], [SWS_Rte_05146], [SWS_Rte_05147], [SWS_Rte_05164], [SWS_Rte_05189], SWS_Rte_05506, [SWS_Rte_05509], [SWS_Rte_06532], [SWS_Rte_06533], [SWS_Rte_06713], [SWS_Rte_06714], [SWS_Rte_06715], [SWS_Rte_06718], [SWS_Rte_07006], [SWS_Rte_07008], [SWS_Rte_07031], [SWS_Rte_07047], [SWS_Rte_07048], [SWS_Rte_07054], [SWS_Rte_07056], [SWS_Rte_07059], [SWS_Rte_07075], [SWS_Rte_07092], [SWS_Rte_07093], [SWS_Rte_07099], [SWS_Rte_07101], [SWS_Rte_07122], [SWS_Rte_07135], [SWS_Rte_07140], [SWS_Rte_07152], [SWS_Rte_07170], [SWS_Rte_07175], [SWS_Rte_07178], [SWS_Rte_07187], [SWS_Rte_07194], [SWS_Rte_07195], [SWS_Rte_07200], [SWS_Rte_07203], [SWS_Rte_07251], [SWS_Rte_07254], [SWS_Rte_07270], [SWS_Rte_07282], [SWS_Rte_07283], [SWS_Rte_07289], [SWS_Rte_07290], [SWS_Rte_07293], [SWS_Rte_07294], [SWS_Rte_07346], [SWS_Rte_07367], [SWS_Rte_07384], [SWS_Rte_07385], [SWS_Rte_07387], [SWS_Rte_07390], [SWS_Rte_07394], [SWS_Rte_07396], [SWS_Rte_07530], [SWS_Rte_07559], [SWS_Rte_07562], [SWS_Rte_07563], [SWS_Rte_07575], [SWS_Rte_07586], [SWS_Rte_07590], [SWS_Rte_07621], [SWS_Rte_07647], [SWS_Rte_07648], [SWS_Rte_07654], [SWS_Rte_07655], [SWS_Rte_07675], [SWS_Rte_07680], [SWS_Rte_08001], [SWS_Rte_08002], [SWS_Rte_08016], [SWS_Rte_08039], [SWS_Rte_08301], [SWS_Rte_08500], [SWS_Rte_08505].

G.3.5 Added SWS Items

The following SWS items were added in Rel. 4.1 Rev. 1: [SWS_Rte_03862], [SWS_Rte_06727], [SWS_Rte_06728], [SWS_Rte_06729], [SWS_Rte_06730], [SWS_Rte_06731], [SWS_Rte_06732], [SWS_Rte_06733], [SWS_Rte_06734], [SWS_Rte_06735], [SWS_Rte_06736], [SWS_Rte_06737], [SWS_Rte_06738], [SWS_Rte_06739], [SWS_Rte_06740], [SWS_Rte_06741], [SWS_Rte_06742], [SWS_Rte_06743], [SWS_Rte_06744], [SWS_Rte_06745], [SWS_Rte_06746], [SWS_Rte_06747], [SWS_Rte_06748], [SWS_Rte_06749], [SWS_Rte_06750], [SWS_Rte_06751], [SWS_Rte_06752], [SWS_Rte_06753], [SWS_Rte_06754], [SWS_Rte_06755], [SWS_Rte_06756], [SWS_Rte_06757], [SWS_Rte_06758], [SWS_Rte_06759], [SWS_Rte_06760], [SWS_Rte_06761], [SWS_Rte_06762], [SWS_Rte_06764], [SWS_Rte_06765], [SWS_Rte_06766], [SWS_Rte_06767], [SWS_Rte_06768], [SWS_Rte_06769], [SWS_Rte_06770], [SWS_Rte_06771], [SWS_Rte_06772], [SWS_Rte_06773], [SWS_Rte_06774], [SWS_Rte_06775], [SWS_Rte_06776], [SWS_Rte_06777], [SWS_Rte_06778], [SWS_Rte_06779], [SWS_Rte_06780], [SWS_Rte_06781], [SWS_Rte_06782], [SWS_Rte_06783], [SWS_Rte_06784], [SWS_Rte_06785], [SWS_Rte_06786], [SWS_Rte_06787], [SWS_Rte_06788], [SWS_Rte_06789], [SWS_Rte_06791], [SWS_Rte_06792], [SWS_Rte_06793], [SWS_Rte_06794], [SWS_Rte_06795], [SWS_Rte_06796], [SWS_Rte_06797], [SWS_Rte_07828], [SWS_Rte_07829], [SWS_Rte_07830], [SWS_Rte_07831], [SWS_Rte_07832], [SWS_Rte_07833], [SWS_Rte_07834], [SWS_Rte_07835], [SWS_Rte_07836], [SWS_Rte_07837], [SWS_Rte_07838], [SWS_Rte_07839], [SWS_Rte_07840], [SWS_Rte_07841], [SWS_Rte_07925], [SWS_Rte_07926], [SWS_Rte_07927], [SWS_Rte_08046], [SWS_Rte_08047], [SWS_Rte_08048], [SWS_Rte_08049], [SWS_Rte_08050], [SWS_Rte_08051], [SWS_Rte_08052], [SWS_Rte_08053], [SWS_Rte_08054], [SWS_Rte_08055], [SWS_Rte_08056], [SWS_Rte_08057], [SWS_Rte_08058], [SWS_Rte_08059], [SWS_Rte_08060], [SWS_Rte_08061], [SWS_Rte_08062], [SWS_Rte_08063], [SWS_Rte_08064], [SWS_Rte_08065], [SWS_Rte_08066], [SWS_Rte_08067], [SWS_Rte_08068], [SWS_Rte_08069], [SWS_Rte_08070], [SWS_Rte_08071], [SWS_Rte_08072], [SWS_Rte_08073], [SWS_Rte_08309], [SWS_Rte_08310], [SWS_Rte_08311], [SWS_Rte_08405], [SWS_Rte_08406], [SWS_Rte_08407], [SWS_Rte_08408], [SWS_Rte_08409], [SWS_Rte_08410], [SWS_Rte_08411], [SWS_Rte_08412], [SWS_Rte_08511], [SWS_Rte_08512], [SWS_Rte_08513], [SWS_Rte_08514], [SWS_Rte_08600], [SWS_Rte_08601], [SWS_Rte_08700], [SWS_Rte_08701], [SWS_Rte_08702], [SWS_Rte_08703], [SWS_Rte_08704], [SWS_Rte_08705], [SWS_Rte_08706], [SWS_Rte_08707], [SWS_Rte_08708], [SWS_Rte_08709], [SWS_Rte_08710], [SWS_Rte_08711], [SWS_Rte_08712], [SWS_Rte_08713], [SWS_Rte_08725], [SWS_Rte_08726], [SWS_Rte_08727], [SWS_Rte_08728], [SWS_Rte_08729], [SWS_Rte_08730], [SWS_Rte_08731], [SWS_Rte_08732], [SWS_Rte_08733], [SWS_Rte_08734], [SWS_Rte_08735], [SWS_Rte_08736], [SWS_Rte_08737], [SWS_Rte_08738], [SWS_Rte_08739], [SWS_Rte_08740], [SWS_Rte_08741], [SWS_Rte_08742], [SWS_Rte_08743], [SWS_Rte_08744], [SWS_Rte_08745], [SWS_Rte_08746], [SWS_Rte_08747], [SWS_Rte_08748], [SWS_Rte_08749], [SWS_Rte_08750], [SWS_Rte_08751], [SWS_Rte_08752], [SWS_Rte_08753], [SWS_Rte_08754], [SWS_Rte_08755], [SWS_Rte_08756], [SWS_Rte_08757], [SWS_Rte_08758], [SWS_Rte_08759], [SWS_Rte_08761], [SWS_Rte_08762], [SWS_Rte_08763], [SWS_Rte_08764], [SWS_Rte_08765], [SWS_Rte_08766].

G.4 Changes in Rel. 4.1 Rev. 2 compared to Rel. 4.1 Rev. 1

G.4.1 Added Traceables in 4.1.2

[SWS_Rte_01371] [SWS_Rte_01372] [SWS_Rte_07410] [SWS_Rte_07411] [SWS_Rte_07412] [SWS_Rte_07842] [SWS_Rte_07843] [SWS_Rte_07844] [SWS_Rte_07928] [SWS_Rte_08074] [SWS_Rte_08075] [SWS_Rte_08076] [SWS_Rte_08312] [SWS_Rte_08313] [SWS_Rte_08314] [SWS_Rte_08315] [SWS_Rte_08316] [SWS_Rte_08317] [SWS_Rte_08413] [SWS_Rte_08414] [SWS_Rte_08415] [SWS_Rte_08416] [SWS_Rte_08767] [SWS_Rte_08768] [SWS_Rte_08769] [SWS_Rte_08770] [SWS_Rte_08771] [SWS_Rte_08772] [SWS_Rte_08773] [SWS_Rte_08774] [SWS_Rte_08775] [SWS_Rte_08776] [SWS_Rte_08800] [SWS_Rte_08801]

G.4.2 Changed Traceables in 4.1.2

[SWS_Rte_01003] [SWS_Rte_01296] [SWS_Rte_01297] [SWS_Rte_01358] [SWS_Rte_01360] [SWS_Rte_01368] [SWS_Rte_02549] [SWS_Rte_02600] [SWS_Rte_02678] [SWS_Rte_03012] [SWS_Rte_03526] [SWS_Rte_03527] [SWS_Rte_03571] [SWS_Rte_03755] [SWS_Rte_03788] [SWS_Rte_03809] [SWS_Rte_03810] [SWS_Rte_03813] [SWS_Rte_03832] [SWS_Rte_03843] [SWS_Rte_03849] [SWS_Rte_03851] [SWS_Rte_03862] [SWS_Rte_03970] [SWS_Rte_04508] [SWS_Rte_05052] [SWS_Rte_05088] [SWS_Rte_05089] [SWS_Rte_05090] [SWS_Rte_05097] [SWS_Rte_05129] [SWS_Rte_05147] [SWS_Rte_05177] [SWS_Rte_05184] [SWS_Rte_05191] [SWS_Rte_05503] [SWS_Rte_06727] [SWS_Rte_06731] [SWS_Rte_06732] [SWS_Rte_06737] [SWS_Rte_06738] [SWS_Rte_06780] [SWS_Rte_07006] [SWS_Rte_07027] [SWS_Rte_07085] [SWS_Rte_07101] [SWS_Rte_07135] [SWS_Rte_07170] [SWS_Rte_07175] [SWS_Rte_07188] [SWS_Rte_07196] [SWS_Rte_07260] [SWS_Rte_07261] [SWS_Rte_07385] [SWS_Rte_07538] [SWS_Rte_07620] [SWS_Rte_07621] [SWS_Rte_07654] [SWS_Rte_07662] [SWS_Rte_07694]

[SWS_Rte_07831] [SWS_Rte_07832] [SWS_Rte_07927] [SWS_Rte_08017] [SWS_Rte_08018] [SWS_Rte_08020] [SWS_Rte_08021] [SWS_Rte_08022] [SWS_Rte_08023] [SWS_Rte_08043] [SWS_Rte_08044] [SWS_Rte_08045] [SWS_Rte_08064] [SWS_Rte_08072] [SWS_Rte_08403] [SWS_Rte_08404] [SWS_Rte_08407] [SWS_Rte_08501] [SWS_Rte_08507] [SWS_Rte_08513] [SWS_Rte_08514] [SWS_Rte_08733] [SWS_Rte_08743]

G.4.3 Deleted Traceables in 4.1.2

[SWS_Rte_02673] [SWS_Rte_05001] [SWS_Rte_05506]

G.4.4 Added Constraints in 4.1.2

Id	Heading
[constr_9080]	The *shortNames* of *PortInterfaces* shall be unique within a software component if it supports multiple instantiation or `indirectAPI` attribute is set to 'true'
[constr_9081]	Mapping to partition vs the value of VariableAccess.scope

Table G.3: Added Constraints in 4.1.2

G.4.5 Changed Constraints in 4.1.2

Id	Heading
[constr_9020]	The blocking `Rte_SwitchAck` API may only be used by the runnable that describes its usage.

Table G.4: Changed Constraints in 4.1.2

G.4.6 Deleted Constraints in 4.1.2

none

G.5 Changes in Rel. 4.1 Rev. 3 compared to Rel. 4.1 Rev. 2

G.5.1 Added Traceables in 4.1.3

[SWS_Rte_01373] [SWS_Rte_01374] [SWS_Rte_01375] [SWS_Rte_06030] [SWS_Rte_06031] [SWS_Rte_06032] [SWS_Rte_06551] [SWS_Rte_06552] [SWS_Rte_06553] [SWS_Rte_06790] [SWS_Rte_06798] [SWS_Rte_06799] [SWS_Rte_06800] [SWS_Rte_06801] [SWS_Rte_06802] [SWS_Rte_06803] [SWS_Rte_06804] [SWS_Rte_06805] [SWS_Rte_06806] [SWS_Rte_06807] [SWS_Rte_06808] [SWS_Rte_06809] [SWS_Rte_06810] [SWS_Rte_07845] [SWS_Rte_07846] [SWS_Rte_07847] [SWS_Rte_07848] [SWS_Rte_07849]

[SWS_Rte_07850] [SWS_Rte_07851] [SWS_Rte_08077] [SWS_Rte_08078] [SWS_Rte_08079] [SWS_Rte_08318] [SWS_Rte_08319] [SWS_Rte_08320] [SWS_Rte_08321] [SWS_Rte_08322] [SWS_Rte_08777] [SWS_Rte_08778] [SWS_Rte_08779] [SWS_Rte_08780] [SWS_Rte_08781] [SWS_Rte_08782] [SWS_Rte_08783] [SWS_Rte_08784] [SWS_Rte_08785] [SWS_Rte_08786]

G.5.2 Changed Traceables in 4.1.3

[SWS_Rte_01071] [SWS_Rte_01072] [SWS_Rte_01083] [SWS_Rte_01091] [SWS_Rte_01092] [SWS_Rte_01102] [SWS_Rte_01111] [SWS_Rte_01118] [SWS_Rte_01120] [SWS_Rte_01123] [SWS_Rte_01206] [SWS_Rte_01252] [SWS_Rte_01354] [SWS_Rte_02568] [SWS_Rte_02599] [SWS_Rte_02614] [SWS_Rte_02619] [SWS_Rte_02628] [SWS_Rte_02631] [SWS_Rte_02659] [SWS_Rte_02667] [SWS_Rte_02725] [SWS_Rte_02740] [SWS_Rte_02741] [SWS_Rte_02743] [SWS_Rte_02744] [SWS_Rte_02745] [SWS_Rte_03527] [SWS_Rte_03550] [SWS_Rte_03553] [SWS_Rte_03560] [SWS_Rte_03565] [SWS_Rte_03741] [SWS_Rte_03744] [SWS_Rte_03800] [SWS_Rte_03813] [SWS_Rte_03832] [SWS_Rte_03858] [SWS_Rte_03859] [SWS_Rte_03928] [SWS_Rte_05129] [SWS_Rte_05501] [SWS_Rte_05509] [SWS_Rte_06536] [SWS_Rte_07026] [SWS_Rte_07038] [SWS_Rte_07039] [SWS_Rte_07057] [SWS_Rte_07195] [SWS_Rte_07200] [SWS_Rte_07203] [SWS_Rte_07214] [SWS_Rte_07216] [SWS_Rte_07223] [SWS_Rte_07224] [SWS_Rte_07367] [SWS_Rte_07394] [SWS_Rte_07554] [SWS_Rte_07640] [SWS_Rte_07680] [SWS_Rte_07928] [SWS_Rte_08066] [SWS_Rte_08314] [SWS_Rte_08315] [SWS_Rte_08316] [SWS_Rte_08800]

G.5.3 Deleted Traceables in 4.1.3

[SWS_Rte_03012] [SWS_Rte_03790] [SWS_Rte_04525] [SWS_Rte_05116] [SWS_Rte_05134]

G.5.4 Added Constraints in 4.1.3

Id	Heading
[constr_9082]	`RtePositionInTask` and `RteBswPositionInTask` values shall be unique in a particular context

Table G.5: Added Constraints in 4.1.3

G.5.5 Changed Constraints in 4.1.3

none

G.5.6 Deleted Constraints in 4.1.3

Id	Heading
[constr_9004]	Usage of `WaitPoint`s is restricted depending on *ExclusiveAreaImplMechanism*

Table G.6: Deleted Constraints in 4.1.3