Elektrobit

EB tresos® AutoCore Generic 8 RTE documentation

product release 8.8.4

Elektrobit Automotive GmbH
Am Wolfsmantel 46
91058 Erlangen, Germany
Phone: +49 9131 7701 0
Fax: +49 9131 7701 6333
Email: info.automotive@elektrobit.com

Technical support

https://www.elektrobit.com/support

Legal disclaimer

Confidential information.

ALL RIGHTS RESERVED. No part of this publication may be copied in any form, by photocopy, microfilm, retrieval system, or by any other means now known or hereafter invented without the prior written permission of Elektrobit Automotive GmbH.

All brand names, trademarks, and registered trademarks are property of their rightful owners and are used only for description.
Copyright 2021, Elektrobit Automotive GmbH.

Table of Contents

1. Overview of EB tresos AutoCore Generic 8 RTE documentation

Welcome to the EB tresos AutoCore Generic 8 RTE (ACG8 RTE) product documentation.

This document provides:

▶   Chapter 3, "ACG8 RTE release notes": release notes for the ACG8 RTE modules

▶   Chapter 4, "ACG8 RTE user's guide": containing background information and instructions

▶   Chapter 5, "ACG8 RTE module references": information about configuration parameters and the application programming interface

2. Supported features

The following table defines a list of scenarios for the use of transformers that have been developed during the course of the analysis of the data transformation concept in AUTOSAR. The table indicates whether data transformation is already supported in certain scenarios by the ACG8 RTE product.

Scenario	Data transformation support
Explicit non-queued sender/receiver communication	yes
Implicit non-queued sender/receiver communication	yes
Sender/receiver communication with signal fan-out	yes
Sender/receiver communication with signal fan-in	yes
Sender/receiver communication with mixed inter-ECU and intra-ECU communication	yes
Sender/receiver communication via LdCom without TriggerTransmit	yes
Sender/receiver communication via LdCom with TriggerTransmit (no signal fan-out)	yes
Transformers with out-of-place buffering	yes
Transformers with in-place buffering	yes
Sender/receiver communication with specification of dynamic array size profiles	yes
Transformer optimization for transformers with out-of-place buffering only	yes
Transformer optimization for transformers with in-place buffering	yes
Sender/receiver communication with portAPIOption.errorHandling set to transformerErrorHandling (additional transformerError argument) for Rte_Read/Write	yes
Transformer return values (hard/soft transformer error) in APIs Rte_Feedback(), Rte_IFeedback() and Rte_IStatus()	yes
Configuration of non-serializing transformers with access to the original data	yes
Sender/receiver communication via LdCom with TriggerTransmit and signal fan-out where each ISignal has its own transformer	yes
Sender/receiver communication behavior at receiver side during startup (no data received)	yes
Sender/receiver communication with specification of PRPortPrototype elements for data transmission OR data reception only	yes
Sender/receiver communication with more than one non-serializing transformer (real transformer chains)	yes
Sender/receiver communication with specification of executeDespiteDataUnavailability=true for receivers which periodically poll received data, e.g. for correct E2E state handling if no data is received	yes

Scenario	Data transfor-mation support
Support for customer transformer	yes
Queued sender/receiver communication	yes
Sender/receiver communication with inter-partition/inter-ECU communication (intra-core)	yes
Sender/receiver communication with specification of PortInterfaceMapping elements	yes
Sender/receiver communication with additional transformerError argument portAPIOption.errorHandling set to transformerErrorHandling for Rte_DRead/IStatus/Invalidate	no
Sender/receiver communication with data invalidation with primitive data type (both sender and receiver side)	no
Sender/receiver communication with data invalidation with complex data type (both sender and receiver side)	no
Sender/receiver communication with data invalidation and transmission via LdCom	no
Sender/receiver communication with specification of PRPortPrototype elements for both data transmission AND data reception	no
Sender/receiver communication with specification of XfrmVariableDataPrototypeInstanceRef, e.g. for multiple E2E protected receivers at same ECU	yes
Support for security transformer	yes
Sender/receiver communication with specification for communication timeout for Inter-ECU communication (Rte_IStatus, Rte_Read)	yes
Sender/receiver communication with data conversion	no
Trigger communication with inter-ECU communication	yes
Trigger communication with inter-partition/inter-ECU communication	no
One SystemSignal(Group) mapped to multiple sending software components	no
Client/server communication with intra-partition/inter-ECU communication	yes
Client/server communication with inter-partition/inter-ECU communication (intra-core)	yes
Client/server communication with autonomous error reaction (intra-partition and inter-partition)	yes
Client/server communication with TransformerHardErrorEvents (intra-partition and inter-partition)	yes
Intra-ECU non-queued 1:1 sender/receiver communication	yes
Intra-ECU 1:n sender/receiver communication	no
Intra-ECU n:1 sender/receiver communication	no
Intra-ECU queued sender/receiver communication	no

The following table contains scenarios for the use of RIPS that were developed during the analysis of the RIPS concept in AUTOSAR. The table indicates whether RIPS is already supported in certain scenarios by the ACG8 RTE product.

Scenario	RIPS support
Explicit non-queued sender/receiver communication	yes
Explicit queued sender/receiver communication	yes
Implicit sender/receiver communication with component data structure	yes
Implicit sender/receiver communication with component data structure optimization	yes
RIPS fill and flush routine	yes
Mode disabling support for RIPS fill and flush routine	yes
OsSchedulePoint support for RIPS fill and flush routine	yes
RIPS lifecycle API	yes
Sender/receiver communication with inter-ECU signal/signalGroup mapping	no
Sender/receiver communication with mixed inter-ECU and intra-ECU communication	no
Sender/receiver communication with data conversion	no
Data Communication Graphs involving NvBlockSwComponents	no
Client-Server communication	no
Handling of communication status conversion	no
Support for inter-runnable variable communication	no
Inter-Partition communication with the parameter RteRipsGlobalCopyInstantiationPolicy configured to RTE_RIPS_INSTANTIATION_BY_PLUGIN	yes
Inter-Partition communication with the parameter RteRipsGlobalCopyInstantiationPolicy configured to RTE_RIPS_INSTANTIATION_BY_RTE	no
Support for exclusive area protection of software components	no
Support for exclusive area protection of basic software modules	yes
Exclusive area in the role runsInsideExclusiveArea	yes
Exclusive area in the role canEnterExclusiveArea where the ExclusiveAreaPolicy.apiPrinciple set to common	yes
Exclusive area in the role canEnterExclusiveArea where the ExclusiveAreaPolicy.apiPrinciple set to perExecutable	no
Support for mode management communication	no
Support for transformers	no
Support for measurement	no
Bypass support	no

Scenario	RIPS support
Support for parameter interface	yes

3. ACG8 RTE release notes

3.1. Overview

This chapter provides the ACG8 RTE product specific release notes. General release notes that are applicable to all products are provided in the EB tresos AutoCore Generic documentation. Refer to the general release notes in addition to the product release notes documented here.

3.2. Scope of the release

3.2.1. Configuration tool

Your release of EB tresos AutoCore is compatible with the release of the EB tresos Studio configuration tool:

▶   EB tresos Studio: 28.2.0 b211016-0103

3.2.2. AUTOSAR modules

The following table lists the AUTOSAR modules that are part of this ACG8 RTE release.

Module name	AUTOSAR version and revision	SWS version and revision	Module version	Supplier
Rte	4.0.3 []	3.2.0 [0003]	6.6.3	Elektrobit Automotive GmbH

Table 3.1. Hardware-Independent Modules specified by the AUTOSAR standard

3.2.3. EB (Elektrobit) modules

The following table lists all modules which are part of this release but are not specified by the AUTOSAR standard. These modules include tooling developed by EB or they may hold files shared by all other modules.

Module name	Module version	Supplier
No EB modules available		

Table 3.2. Modules not specified by the AUTOSAR standard

3.2.4. MCAL modules and EB tresos AutoCore OS

For information about MCAL modules and OS, refer to the respective documentation, which is available as PDF at `$TRESOS_BASE/doc/3.0_EB_tresos_AutoCore_OS` and `$TRESOS_BASE/doc/5.0_MCAL_-modules`[1]. It is also available in the online help in EB tresos Studio. Browse to the folders `EB tresos AutoCore OS` and `MCAL modules`.

3.3. Module release notes

3.3.1. Rte module release notes

► AUTOSAR R4.0 Rev 3

► AUTOSAR SWS document version: 3.2.0

► Module version: 6.6.3.B466224

► Supplier: Elektrobit Automotive GmbH

3.3.1.1. Change log

This chapter lists the changes between different versions.

Module version 6.6.3
2021-10-20

► Support for time-triggered execution of runnables for DataSendCompletedEvent

► Added support for the generation of review instructions to easily identify memory writes access for freedom from interference analyses (released)

► Added support for the transmission preparation callback (Rte_COMCbkTxPrep_mn) for inter-partition inter-ECU communication. To activate this feature, set the configuration parameter RteGeneration/SendSignalQueueStrategy to "OnePerMainFunction"

► ASCRTE-6563 Fixed known issue: The Rte may not compile if mixed Rte and BSW events are mapped to a mode user task in case of synchronous inter-partition mode communication

► ASCRTE-6556 Fixed known issue: The Rte Generator may not generate the ISR body for events mapped to ISRs if the Event Mapping tab of the Rte Editor is used

---

[1] `$TRESOS_BASE` is the location at which you installed EB tresos Studio.

► Improved the software component template code generation by adding the proper CODE memory sections

Module version 6.6.2
2021-09-24

► ASCRTE-6513 Fixed known issue: The Rte_Send/Write API uses a wrong buffer if multiple variable data prototypes are mapped to the same system signal and the senders are not located in the Com partition

► Improved the runtime of the Rte generator for software components with many client-server provide ports

► Added a verifier to reject configuration with 1:n client to server connection

► Added a verifier to report an error if the server is synchronously connected to more than 1 client, and at least 1 client is mapped to a different partition than the server and 1 partition-local client is mapped to the same task as server

► The Rte generator now allows direct server calls where the server runnable has a runsInsideExclusiveArea

► ASCRTE-6500 Fixed known issue: Under certain conditions, an unqueued sender-receiver data element with a structure data type is not received intra-partition inter-ECU if it is directly read from Com

► Added support for Rte_IrvIWriteRef API

Module version 6.6.1
2021-08-20

► The Rte will no longer export the alignment attribute of a memory section if the memoryAllocationKeyword-Policy of the SwAddrMethod is set to ADDR-METHOD-SHORT-NAME

► Avoided exception when there is no valid BswOsTaskRef in case of multi-core client/server communication and Det enabled

► ASCRTE-6533 Fixed known issue: The Rte Bswmd generator may fail with an error on attempt to create a McDataInstance for an ApplicationArrayDataType with a DynamicArraySizeProfile VSA_LINEAR or VSA_-SQUARE

Module version 6.6.0
2021-07-28

► Improved verifier in case of single receive signal port runnable not mapped to a task

► ASCRTE-6501 Fixed known issue: Under certain conditions the tasks in a task chain are not activated if all TimingEvents define the same offset greater than 0

► Improved RTE_516 Verifier Warning to be more descriptive to the user (included the minimum Queue length required)

► Changed verification of the BswModuleDescription.moduleID from mandatory to optional

Module version 6.5.3

2021-07-02

► Improved support for variable size arrays

► ASCRTE-6446 Fixed known issue: If one return signal is used for n:1 inter-ECU client-server communication, the response from the server might get lost

► ASCRTE-6476 Fixed known issue: Missing memory sections for VFB tracing function declarations

► ASCRTE-6473 Fixed known issue: The Rte generator fails with error in case of inter-partition inter-ECU SR communication on receiver side where no runnable has access to the data element and a time-out is configured

► ASCRTE-6485 Fixed known issue: Under certain conditions the Rte falsely applies network representation conversion to inter-partition inter-ECU sender-receiver communication on sender side

Module version 6.5.2

2021-05-28

► ASCRTE-6453 Fixed known issue: The Rte Generator may falsely export elements with duplicate IDs in the RteDataModel.xml

► Added option to always invoke the transformer chain for required sender-receiver data elements if ExecuteDespiteDataUnavailability is true. To enable this option, set the configuration parameter Rte/RteGeneration/ExecuteTransformersDespiteLocalDataQueued (for queued communication) resp. Rte/RteGeneration/ExecuteTransformersDespiteLocalDataUnqueued (for unqueued communication) to true

► Improved sharing of implicit sender-receiver buffers by allocating the buffers in highest to lowest status element order

Module version 6.5.1

2021-04-30

► ASCRTE-6439 Fixed known issue: Data type of Rte variable "Rte_AliveTimeoutCounter_<Sig>" is wrongly generated in case of 1:n sender-receiver communication with different aliveTimeout values

► Improved sharing of implicit sender-receiver buffers between sender and receiver with different compatible data types

Module version 6.5.0

2021-04-16

► Improved: The Rte_Write/Rte_Send API in case of LdCom TP usage (EcuC_TpTransmit) and locked tx buffer

► Improved the Rte generator to trigger autonomous error reaction irrespective of the transformer class

► Improved startup and initialization of NvBlockSwComponents: Restore default data if both ROM and initialization callback are configured

► ASCRTE-6407 Fixed known issue: The Rte_Read() API may copy wrong data when inter-partition-inter-ECU sender-receiver communication is used with transformers and function elision is enabled

► ASCRTE-6410 Fixed known issue: The Rte does not apply data conversion to an implicit intra-ECU sender-receiver communication under certain conditions

Module version 6.4.11

2021-03-12

► ASCRTE-6360 Fixed known issue: The Rte does not trigger DataSendCompletedEvent if an Rte_LdCom-CbkTpTxConfirmation() callback arrives with a value different from E_OK

► The Rte falsely reports a warning for arrays of anonymous structs

► Added measurement support for Nv RomBlocks

► Added support for retrieving partition context for LdCom on a per-PDU basis

► ASCRTE-6355 Fixed known issue: The Rte_Send() API might not return RTE_E_LIMIT if a queue was full in case of 1:n inter-partition queued sender-receiver communication

► Added support for the generation of review instructions to easily identify memory writes access for freedom from interference analyses (draft)

► Added support to generate graphical representation of the internal Rte generator model

Module version 6.4.10

2021-02-17

► Improved DET checking/reporting code in Enter/Exit APIs

► Added support for mapping of variable data prototypes to parameter data prototypes

► ASCRTE-6342 Fixed known issue: The Rte Generator fails with an error if an ArrayImplementation-DataType is used for parameter access in combination with RIPS

► ASCRTE-6336 Fixed known issue: The Rte Generator might ignore system data mappings of receive signals/signalGroups to PRPortPrototypes

► Added support for memory sections with alignment when SWAddrMethod or SwAlignment are not configured

► ASCRTE-6339 Fixed known issue: The Rte Generator fails with an error when generating the Rte_Ld-ComCbkRxIndication for a variable size array

▶ ASCRTE-6327 Fixed known issue: The Rte Generator is not generating the definition of Rte_Transformer-Error when contract phase is generated

▶ Added support for retrieving partition context for COM on a per-PDU basis

▶ Added support for Rte_IReadRef API

Module version 6.4.9

2021-01-22

▶ ASCRTE-6291 Fixed known issue: Rte Editor does not support the n:1 use case for RteBswRequiredSenderReceiverConnection

Module version 6.4.8

2020-12-18

▶ ASCRTE-6292 Fixed known issue: The SvcAs configures duplicate XfrmImplementationMappings in case of inter-ECU client-server communication with multiple clients

▶ ASCRTE-6296 Fixed known issue: Under certain conditions the Rte verifier does not report an error if an NvBlockSwComponentType is connected to another component type from a different partition

▶ Improved precision of data conversion formulas with floating-point operands

▶ Added support for synchronous mode switching procedure over multi partitions/cores

Module version 6.4.7

2020-10-30

▶ ASCRTE-6235 Fixed known issue: The Rte_IStatus API might be empty in case of implicit 1:n sender-receiver communication with different aliveTimeout values

▶ Added support for inter-partition direct server call

▶ ASCRTE-6228 Fixed known issue: The Rte might not activate the runnable triggered by an AsynchronousServerCallReturnsEvent if a timeout occured in case of non-blocking result reception

▶ ASCRTE-6241 Fixed known issue: The Rte generates an incorrect call to RIPS APIs Rips_Enter()/Rips_-Exit() under certain conditions

▶ ASCRTE-6204 Fixed known issue: The Rte may generate wrong event handling in case a server uses multiple instantiation

▶ ASCRTE-6223 Fixed known issue: The Rte Generator fails if an AsynchronousServerCallReturnsEvent exists for an unconnected non-blocking AsynchronousServerCallPoint with time-out

▶ Added support for mapping of ModeSwitchEvents for initialMode switches to RteInitializationRunnable-Batch

► Improved: The Rte now aborts the execution of the SvcAs in case of an error

► ASCRTE-6188 Fixed known issue: The Rte may generate extern definitions of the same data handle variable in multiple C-files in case of implicit inter-partition 1:n sender-receiver communication

► Added: The verification of the mapping constraints (component clustering and separation) for EcuExtract

► ASCRTE-6249 Fixed known issue: The Rte might not activate the runnable triggered by ModeSwitchedAckEvent in case of mode user partitions without ModeSwitchEvents

► ASCRTE-6265 Fixed known issue: Under certain conditions OperationInvokedEvents and DataReceivedEvents are not triggered according to their configured RtePeriod/RteBswPeriod

► ASCRTE-6219 Fixed known issue: Under certain conditions the Rte does not trigger ModeSwitchedEvents

► ASCRTE-6277 Fixed known issue: Rte_IRead() API might not read written value in implicit 1:1 sender-receiver communication with transmission acknowledgement enabled

Module version 6.4.6

2020-09-25

► Added support of multiple receivers of SignalGroups with different subelement mappings to primitives

► ASCRTE-6189 Fixed known issue: The Rte Generator fails with an error if a SwServiceArg of category DATA_REFERENCE is used for SchM client-server communication

► Added support for handling the RTE_E_NEVER_RECEIVED error in case of inter-partition communication

Module version 6.4.5

2020-08-28

► ASCRTE-6187 Fixed known issue: The Rte Generator fails with an error if metadata is used for C/S communication over LdCom TP

► Added support of RIPS for ParameterInterfaces

► Removed the verifier which reports a warning when the ComTimeout value is different from the AliveTimeout value

Module version 6.4.4

2020-07-31

► ASCRTE-6171 Fixed known issue: The Rte Generator fails with an error if an application data type which represents a new world variable size array is used for S/R communication

► ASCRTE-6175 Fixed known issue: The Rte rejects configurations if client-server operations are compatible according to TPS_SWCT_01125 and if an ArgumentDataPrototype is typed by an ApplicationArrayDataType

► Added support of multiple receivers with different aliveTimeout values mapped to the same SystemSignal

► ASCRTE-6177 Fixed known issue: The Rte may generate a wrong alignment suffix for the Memory Allo-cation Keyword if a SwAddrMethod is configured for a VariableDataPrototype typed by an array of struct data type

► ASCRTE-6185 Fixed known issue: The Rte Generator fails with an error if metadata is configured for a PDU and a signal contained in this PDU is sent via S/R communication over LdCom

► Added missing task end/chain hook in task chains according to [SWS_Rte_06032]

Module version 6.4.3

2020-06-29

► Changed linking of Rte symbols from object files to the newly introduced Rte_src.lib library file

► ASCRTE-6106 Fixed known issue: The Rte_Feedback() API never returns a transformer error in case of inter-partition inter-ECU communication with transformers

► ASCRTE-6109 Fixed known issue: The Rte may generate an empty Rte_IStatus() macro if an implicit S/R communication buffer (data handle buffer) is shared

► Added support for initial value calculation with RuleBasedValueSpecification

► ASCRTE-6124 Fixed known issue: The Rte does not enter/exit an exclusive area in the role runsInside-ExclusiveArea of a RunnableEntity that is mapped to a BswSchedulableEntity and started by a BswEvent

► ASCRTE-6125 Fixed known issue: An implicit inter-runnable variable is not updated for a RunnableEntity that is mapped to a BswCalledEntity which is started by an BswOperationInvokedEvent and mapped to a task

► Added a verifier to report a warning when the ComTimeout value is different from the AliveTimeout value

► ASCRTE-6086 Fixed known issue: The Rte generator does not consider the value of the parameter RteRip-sPluginFillFlushRoutineFncSymbol

► ASCRTE-6123 Fixed known issue: The Rte may falsely generate a direct access to a data handle buffer in an Rte_Write() API function

► ASCRTE-6117 Fixed known issue: The Rte may update a shared sender-receiver data handle buffer from different partitions

► Added support for DataWriteCompletedEvents and Rte_IFeedback API

► ASCRTE-6147 Fixed known issue: The Rte generates an invalid switch statement in the Com task if a data element to signal fan-out is used and the signals are sent on different Com partitions

Module version 6.4.2

2020-05-27

▶ Improved argument buffering for synchronous client/server communication

▶ Added support for synchronized triggers

▶ ASCRTE-6085 Fixed known issue: The Event Mapping tab of the Rte Editor reports a StackOverflowError when a circular client-server call chain is configured

▶ Added support for variable size array inter-ECU communication without transformers

▶ ASCRTE-6092 Fixed known issue: The Rte verifier may falsely report an error about a wrong availability-Bitfield array size for structs with optional elements

▶ Improved: The Rte shall optionally use the (Ld)Com container's length as transformer buffer length

▶ ASCRTE-6091 Fixed known issue: The Rte_Read() API generates an incorrect call to Com_ReceiveSignal if DirectReadFromCom applies but LdCom is used

Module version 6.4.1

2020-04-24

▶ Improved verifier SWCUTILS_SYSDVFY_658 to allow blocking data reception of the same required VariableDataPrototype from different runnables which are mapped to the same task

▶ Added verifier to check transformer buffer sizes against LdCom signal sizes

▶ ASCRTE-6049 Fixed known issue: The contract phase generation fails with an error if a Bsw module contains a BswInternalTriggeringPoint or an IssuedTrigger

▶ Added export of internal Rte generator model to XML when RteDataModelExport is enabled

▶ ASCRTE-6060 Fixed known issue: The Rte generates an unnecessary cast to void* in the IRead() API if RIPS is enabled

▶ ASCRTE-6064 Fixed known issue: RTE does not generate calls to RIPS APIs IWBufferRef()/IRBufferRef() for implicit S/R communication if RtePluginSupportsIReadIWrite is set to false

▶ ASCRTE-6043 Fixed known issue: An AsynchronousServerCallReturnsEvent triggers the non-blocking result runnable twice if a timeout is defined for the AsynchronousServerCallPoint

▶ Improved the Rte generator to verifiy the consistency of PortAPIOptions and PortDefinedArgumentValues if EventInstances are merged

▶ ASCRTE-6072 Fixed known issue: The Rte invokes configured Rte_Rips_FillFlushRoutines although the parameter RteRipsSupport is set to RTE_RIPS_OFF

▶ Added verifier to check restriction to explicit sending semantics for the usage of DataServices in the context of a SwcServiceDependency that aggregates DiagnosticValueNeeds that in turn is referenced by a DiagnosticIoControlNeeds

Module version 6.4.0

2020-03-25

► Improved the RTE generator to call the RIPS APIs for unconnected Sender/Receiver ports

► Added verifier check for implicit S/R 1:n communication where receivers are not compatible and mapped to the same FID and RIPS is managed by plugin

► Added support for generating Bitfield Texttable definitions according to the AUTOSAR 4.4 RTE spec

Module version 6.3.14

2020-02-21

► Added support for LdCom without transformers and inter-partition communication over Ioc

► ASCRTE-5943 Fixed known issue: The Rte triggers ModeSwitchedAckEvents at the end of the transition to the initial mode

► Improved performance of queue accesses by caching the queue index in a stack variable. This also fixes a possible compiler warning: "the order of volatile accesses is undefined"

► ASCRTE-5990 Fixed known issue: The Rte software component header might include a wrong RIPS software component header

► ASCRTE-5988 Fixed known issue: The Rte Editor does not display all BswEvents under certain conditions

► ASCRTE-5981 Fixed known issue: The Rte generates an incorrect subElement mapping under certain conditions

► ASCRTE-5967 Fixed known issue: The Rte does not allow mapping of ApplicationRecordDataType to ImplementationDataType with optional elements

► ASCRTE-5992 Fixed known issue: Under certain conditions the Rte falsely skips data conversion for inter-ecu sender-receiver communication

► ASCRTE-5970 Fixed known issue: The Rte triggers ModeSwitchedAckEvents and BswModeSwitchedAckEvents wrongly in case of multiple mode user partitions

► ASCRTE-5950 Fixed known issue: Subsequent calls to LdCom_TpTransmit() fail if an Rte_LdComCbkTpTxConfirmation() callback arrives very shortly after LdCom_TpTransmit()

► Generate execution condition for the A1 mapped timing events to ISR if periods are not equal

► ASCRTE-5978 Fixed known issue: Rte_LdComCbkTriggerTransmit() copies wrong data in case of inter-partition inter-ECU communication with an autonomous error reaction

► ASCRTE-5979 Fixed known issue: An autonomous error reaction may corrupt data of outstanding server responses

► Added support for RIPS for BSW explicit queued sender/receiver communication and exclusive area protection

► ASCRTE-5963 Fixed known issue: The Rte_Main.c does not include Rte_UserDefinedExclusiveArea.h in case multiple ExclusiveAreas are shared across different partitions

► Updated the check for Rte Det error reporting with partitioning and shared Schm exclusive areas

▶ Added support for memory optimization for zero initial values

▶ Improved the Rte verifier to report an error if a server RunnableEntity is mapped to a BswSchedulableEntity. It must be mapped to a BswCalledEntity instead

▶ ASCRTE-6002 Fixed known issue: The Rte might return a wrong server result to a client if all other connected clients use result-free asynchronous client-server communication

▶ Improved the Rte generator to allow client tasks to have a higher priority than the server task also in basic intra-partition scenarios

▶ Improved data consistency optimizations and task mapping verifications by considering Os resources of exclusive areas in the runsInsideExclusiveArea role

▶ ASCRTE-6004 Fixed known issue: The Rte generates duplicate macros in the Rte_Buffers.h for 1:n implicit S/R RIPS communication

Module version 6.3.13

2020-01-24

▶ ASCRTE-5942 Fixed known issue: Rte generation fails if an ImplementationDataType is used in a PortInterface and has the same name as one of the standard OsTypes

▶ Improved: The Rte now adds suffixes for float and double literals if initial values are generated but not explicitly configured

▶ Improved: The Rte Generator now verifies the partition of Tx Com signals against the partition of the Bsw task if no ComTaskConfiguration containers are configured

▶ ASCRTE-5954 Fixed known issue: The SvcAs fails with an error in case of a SystemSignal-to-ISignal Tx fan-out with LdCom, even if distinct TransformationTechnology elements are used for each ISignal

▶ ASCRTE-5945 Fixed known issue: The Rte does not ensure data consistency between LdCom_Transmit() and Rte_LdComCbkTriggerTransmit()

Module version 6.3.12

2019-12-06

▶ ASCRTE-5912 Fixed known issue: The server might not be called synchronously if intra-partition or inter-partition-intra-core client-server communication is used and a category 2 executable is mapped to the server task

▶ ASCRTE-5903 Fixed known issue: The SvcAs does not create all CustomXf XfrmImplementationMappings for Tx signals if a transformer chain contains multiple TransformationTechnology elements with transformerClass set to custom

▶ Added support for configurable output directory of Rte_Bswmd.arxml

▶ Added support for LdCom without transformers

▶ ASCRTE-5915 Fixed known issue: The Rte may copy wrong data in Rte_LdComCbkCopyTxData()

▶ ASCRTE-5914 Fixed known issue: The Rte does not lock the LdCom TP transmit buffer sufficiently

▶ Updated the verifier to report a warning when an ImplementationDatType of category ARRAY contains an ImplementationDataTypeElement of category STRUCTURE or UNION

▶ ASCRTE-5904 Fixed known issue: The Rte_Receive() API never returns RTE_E_LOST_DATA in case of inter-partition inter-ECU communication with transformers

▶ Provided ALIGNMENT info for CONST MEMORY-SECTIONs which are generated in Rte_Bswmd.arxml

▶ Added support for RteCalibrationSwAddrMethodRef for software calibration

▶ Added support for mapping timing events to ISRs

Module version 6.3.11
2019-11-08

▶ Add partitioning support for communication timeout with transformers

▶ Improved lock optimizations for Rte/SchM APIs that are accessed from non-interruptible tasks only

Module version 6.3.10
2019-10-11

▶ Improved: Define macro of PR port initial value to be generated once

▶ Changed the parameter RteSwNvRamMappingRef of the container RteNvRamAllocation to optional

▶ ASCRTE-5826 Fixed known issue: Under certain conditions the Rte may send/receive incorrect data to/ from the Com module

▶ Improved verifier for inter Ecu C/S communication with client runnable entity not mapped to a task

▶ Added support for TimingEvent.offset

▶ ASCRTE-5850 Fixed known issue: The SvcAs does not set ComTimeout in case of inter-partition inter-ECU sender-receiver communication

▶ Added support for overlayed errors in Rte_Read(), Rte_Receive() and Rte_IStatus() APIs (intra-partition intra/inter-ECU sender-receiver communication only)

▶ ASCRTE-5851 Fixed known issue: The SvcAs does not set ComSignalInitValue, ComSignalDataInvalid-Value and ComDataInvalidAction in case of inter-partition inter-ECU sender-receiver communication

▶ ASCRTE-5848 Fixed known issue: Rte generates incorrect code in the case of multiple internal triggering points with the same short name triggering the same runnable

▶ ASCRTE-5849 Fixed known issue: Rte generation fails for inter-core asynchronous client-server communication with InterPartitionCommunication set to Mixed

▶ ASCRTE-5822 Fixed known issue: The generated Rte may not compile in case of implicit mixed n:1 in-ter/intra-partition S/R communication with transformers

▶ Added support for Rte(Bsw)ExclusiveAreaOsResourceRef to reference an OsResource in case RteExclu-siveAreaImplMechanism is configured to OS_RESOURCE for this ExclusiveArea

▶ ASCRTE-5845 Fixed known issue: The Rte_IStatus() API does not return any transformer error in case of inter-partition inter-ECU sender-receiver communication

▶ ASCRTE-5854 Fixed known issue: The Rte generator may falsely report a warning if multiple Nonqueue-dReceiverComSpec.initValue of the same port reference data elements with different application data types

▶ Improved the number of enter/exit critical section calls by grouping the SMC channel read/write accesses of the same OS task by their critical section blocks in case of implicit S/R communication

▶ ASCRTE-5879 Fixed known issue: The Rte Generator may fail if RteDevErrorDetect is true and a software component has multiple ports with the same port interface

▶ ASCRTE-5855 Fixed known issue: Det error reporting and partitioning may lead to linkage errors about multiple definitions of Rte_Det_IntLockTask

▶ Improved data consistency optimizations and task mapping verifications by considering internal Os re-sources

Module version 6.3.9

2019-09-06

▶ ASCRTE-5791 Fixed known issue: The function declaration of the RunnableEntity triggered by External-TriggerOccurredEvent is not visible for the trigger source if direct function call and function elision apply

▶ ASCRTE-5814 Fixed known issue: The Rte Generator generates a wrong argument cast in direct server call when server and client use different client-server-interfaces

▶ ASCRTE-5815 Fixed known issue: The Rte Generator falsely reports an error if an ExternalTriggerOc-curredEvent is mapped to an OsIsr in a multi-partition environment

▶ Added support for intra-ECU sender-receiver sub element mapping to a primitive data element

▶ Added support for mode switch acknowledgement for basic software

Module version 6.3.8

2019-08-09

▶ Improved: The Rte now supports casting to compatible data types when calling transformers

▶ Changed the assignment of void pointer arguments to always use TS_MemCpy, e.g. in the Rte_SetMirror and Rte_GetMirror callbacks

Module version 6.3.7

2019-08-06

▶ Added support for compu methods specified through the physical props of SystemSignal

▶ ASCRTE-5752 Fixed known issue: The Rte.bmd file contains an invalid destination reference to the NvM-BlockDescriptor for parameter RteNvmBlockRef

▶ ASCRTE-5775 Fixed known issue: The Rte verifier may falsely report an error about a wrong transformer buffer size if GroupSignals are not byte-aligned

▶ Added inter-partition support for coherency groups

▶ Fixed violation of MISRA rule 20.7 by using FUNC_P2CONST/FUNC_P2VAR macro instead of nested compiler abstraction

▶ ASCRTE-5753 Fixed known issue: The Rte may not properly evaluate the SwDataDefProps if multiple ImplementationDataTypes with the same shortName exist

▶ ASCRTE-5749 Fixed known issue: The Rte does not protect the inter-partition SchM_Switch API against concurrent mode switch notifications under certain conditions

▶ ASCRTE-5761 Fixed known issue: The Rte Generator may not call the server function with an activating event argument

Module version 6.3.6

2019-07-12

▶ Added support for Rte(Bsw)UsedOsEventRef to specify which OsEvent is used to activate the OsTask

Module version 6.3.5

2019-06-14

▶ ASCRTE-5676 Fixed known issue: Data conversion for network representation without an application data type is not performed

▶ ASCRTE-5689 Fixed known issue: The Rte may not apply data consistency for the readers of inter-core SMC channels where only implicit writers exist

▶ Improved EB_FAST_LOCK for multi-core set-ups

▶ Improved separation of the Rte partitions for COM send signals by adding a new configuration parameter (SendSignalQueueStrategy)

▶ Added support for inter-partition inter-ECU communication timeout. The RTE will write the data and RTE_-E_OK in case of successful reception of data and will write RTE_E_MAX_AGE_EXCEEDED as status in case of reception timeout

▶ ASCRTE-5710 Fixed known issue: The Rte may not apply data consistency for the implicit writers of inter-partition n:1 sender-receiver communication

▶ Added limited support for Coherency Groups

▶ ASCRTE-5907 Fixed known issue: The Rte generation may fail in case of inter-core intra-ECU C/S communication where the server runnable is target of multiple OperationInvokedEvents

▶ Added support for multiple mode user partitions

Module version 6.3.4

2019-05-17

▶ Improved Rte generator to support arbitrary Os TaskType definitions

▶ ASCRTE-5649 Fixed known issue: Potential loss of precision for initial/invalid values defined by ApplicationValueSpecifications

▶ Removed verifier warnings for diverse SwBaseType native declarations of type "long long"

▶ Improved verifier for BswCalledEntity SwServiceArgs

▶ ASCRTE-5670 Fixed known issue: The Rte Generator may fail if a received group signal is mapped to a primitive non-queued sender-receiver data element and DirectReadFromCom applies

▶ Added Support for time-triggered execution of runnables for data received events

▶ Improved the Rte generator to not throw a warning message in case R-Port of NVBlockSoftwareComponentType is unconnected and do not define an INIT value

▶ Improved Rte by providing appropriate suffixes for 64-bit integer values

▶ Implemented timeout monitoring for asynchronous client/server communication with non-blocking result calls

Module version 6.3.3

2019-04-18

▶ Enhanced the information displayed for mode switched events in the Rte Editor

▶ ASCRTE-5616 Fixed known issue: The Rte_Read API does not protect receiving of a Com signal against concurrent access if DirectReadFromCom applies

▶ Added support for autonomous error reaction on client side

▶ ASCRTE-5586 Fixed known issue: User-defined ExclusiveArea callouts are not aware of multiple SWC instances

▶ Improved spinlock allocation for cross-core queues in case of unmapped executables

▶ ASCRTE-5637 Fixed known issue: The Rte Editor configures wrong trace hook function names for COM callbacks

▶ Improved client data structure type naming

► Changed the return code of Rte_Read in case of intra-partition inter-ECU sender-receiver communication with data transformation; the RTE_E_SOFT_TRANSFORMER_ERROR and RTE_E_HARD_TRANS-FORMER_ERROR of the previous transformer invocation is returned repeatedly unless new data are available and executeDespiteDataUnavailability is false

► ASCRTE-5614 Fixed known issue: The Rte calculates a wrong deserializing transformer buffer size if subElement mappings/partial records and a non-constant bufferComputation are used

► ASCRTE-5607 Fixed known issue: The generated Rte BSWMD file does not contain the OUT-GOING-CALLBACK entries for defined RteVfbTraceFunctions if RteGeneratorOutput is set to BSW_-SCHEDULER_ONLY

Module version 6.3.2

2019-03-22

► Improved the transformer hard error handling for intra-ECU transformations

► Replaced SMC API by direct buffer access

► Improved allocation of OsResources for ReceiveSignals if no API uses them

Module version 6.3.1

2019-02-15

► Implemented `Rte_IsAvailable` and `Rte_SetAvailable` APIs to query and modify the availability status of optional structure elements

► ASCRTE-5564 Fixed known issue: The Rte generator falsely reports an error if a CompuMethod of category TEXTTABLE contains non-point ranges

► Added support for RteBswRequiredSenderReceiverConnections in the Rte Editor

► ASCRTE-5566 Fixed known issue: The Rte Editor faces delays on the attempt to change the position of an EventToTaskMapping entry

► Improved generation of Rte_DummyDirtyFlag

► Added support for flag ExecuteDespiteDataUnavailability, for unqueued sender-receiver communication

► Improved the Rte Generator by avoiding null pointer exception when allocating a BswModeSwitchEvent with an invalid TargetModeRef

► Implemented verifier to check transformer buffer sizes against Com signal(group) sizes

► ASCRTE-5578 Fixed known issue: The Rte generates an invalid switch statement if a client-server provide port is connected but no ServerCallPoint references the operation

► Improved: The Rte Generator allows non-mapped client executables (i.e. executable with unknown task context) in case of intra-ECU result-free asynchronous client-server communication

▶ Updated the Rte Verifier to report an error if a VariableDataPrototype instance in the role ramBlock is accessed by SW-C instances of different partitions

▶ Added verifier to check that the returnSignal and callSignal of a ClientServerToSignalMapping are mapped to Com signals of type UINT8_N or UINT8_DYN

▶ Improved: The Rte Generator now considers the SwAddrMethod in the memory section of a BswSchedulableEntity/BswCalledEntity, in case it is present

▶ Added configurable generation of timestamp for generated RTE files

▶ ASCRTE-5565 Fixed known issue: The Rte generator falsely reports an error if the Init-Value of an ApplicationPrimitiveDataType does not fall in the internal interval of the CompuMethod

▶ Implemented additional verifier for Implementation data type according to [constr_1383]

▶ Updated the Rte Verifier to report an error if the mode values of a mode declaration group do not fit into the mapped implementation data type

▶ Improved verification of compatibility of Com signal type with base type of NetworkRepresentationProps

Module version 6.3.0

2019-01-25

▶ Improved: The Rte Generator now verifies that no Bsw event triggering any type of Bsw module entity is mapped to the Bsw Os task

▶ Added support for a new configuration parameter "Respect configured task type" to prevent the Rte from ignoring the configured task type BASIC although an EXTENDED task is required and to report an error message instead

▶ Added verifier for missing configured BSW module implementation reference

▶ Added support for data filtering on sender side for inter-ecu sender/receiver communication

▶ Implemented additional verifier for SwcBswMapping according to [constr_4085] and [constr_4084]

▶ Updated the Rte verifier to report a warning if a ComSignal has the wrong direction

Module version 6.2.27

2018-12-21

▶ Added a configuration parameter (HumanReadableBufferNames) to switch between the generation of hashed buffer names and human readable buffer names

▶ Improved: The Rte Generator shall report warning RTE_526 only for the used data element prototypes of an interface that do not have an initial value defined in the port prototype's ComSpec

▶ ASCRTE-5539 Fixed known issue: The Rte generates illegal hexadecimal constants on 64-bit architectures under certain conditions

Module version 6.2.26

2018-12-13

► Improved the Rte Bswmd generator to export the McSupportData as self-contained artifact

► Improved ROM consumption of the Rte by removing the unnecessary zero-based initializations of trans-former rawValue arrays

► Added support for Lock-free queues

► ASCRTE-5525 Fixed known issue: The Rte Generator fails with an error if a signal uses a data type of category VALUE referencing a SWBaseType

► Fixed known issue: The Rte Generator fails with an error when the NvBlockSwComponentType has an unconnected client port in the role NvMNotifyJobFinished

► ASCRTE-5519 Fixed known issue: The Rte_Write API does not update a data handle variable in an N:1 sender-receiver communication under certain circumstances

► ASCRTE-5527 Fixed known issue: If a server runnable handles multiple inter-partition operations with an array argument of different length, the argument's data may be lost

Module version 6.2.25

2018-11-23

► Improved the access of RTE Implementation Plugins (RIPS) by applying the reference type URI-REFER-ENCE for RteRipsPluginConfigurationRef

► Improved sorting of column Position on the Event Mapping tab of the Rte Editor

► Improved verifier for constr_1011 in case of SwBaseType with category VOID

► ASCRTE-5503 Fixed known issue: The Rte generates inconsistent memory sections for a per-instance memory that is mapped to an NvM block

► Added support for time-triggered server executables

► Changed queuing support of triggers to strictly adhere to [SWS_Rte_07087] refined with AUTOSAR 4.4.0

► Improved: The Rte now supports including additional type header files for ImplementationDatTypes whose typeEmitter is different from RTE

► Added support for mixed data conversion for intra-ECU SenderReceiverInterfaces (SCALE_LIN-EAR_AND_TEXTTABLE to TEXTTABLE or LINEAR) and mixed linear conversion for network represen-tation according to rte_sws_3832

Module version 6.2.24

2018-11-02

▶ Improved queue efficiency and memory consumption (e.g. for queued sender-receiver communication). RTE_E_LOST_DATA is now only reported on first reception after the overflow occurred in case of inter-partition queued sender-receiver communication

▶ Added the return code SCHM_E_LOST_DATA for the SchM_Receive API

▶ ASCRTE-5367 Fixed known issue: The Rte does not reset the transformer error status at the beginning of an implicit write

▶ ASCRTE-5462 Fixed known issue: The Rte creates too small buffers for Com reception and transformers if partial record support is used

▶ Added support for multiple data elements in the context of a software component port being all mapped to one data element in the context of a NvBlockSwComponentType port using TextTableMappings

▶ Added warning message in case locking strategy is overruled by the Rte (for OsResources only in case of unmapped runnables)

▶ ASCRTE-5465 Fixed known issue: Possible corruption of mode disabling dependencies in case of multiple inter-partition mode communications

▶ ASCRTE-5474 Fixed known issue: The server may not be called synchronously if an inter-partition intra-core client is mapped to a non-preemptive task

▶ ASCRTE-5461 Fixed known issue: The Rte aborts the execution of all subsequent transformer chains if a transformer returns a hard error

▶ Added support for the RTE Implementation Plugins (RIPS) for explicit/implicit sender receiver communication

▶ Implemented support for inter-partition queued triggering for application software components and basic software modules.

▶ Implemented: Mapping of ExternalTriggerOccurredEvents for category 1 executables to category 2 ISRs

▶ ASCRTE-5452 Fixed known issue: The SchM_Trigger API may activate the triggered executables multiple times in case of 1:n external trigger communication

▶ ASCRTE-5433 Fixed known issue: The Rte Generator fails with an error in case of mixed Rte_IWrite/Rte_Read API and non-mapped reader

▶ Added support for sender-receiver intra-partition data conversion for CompuMethod categories TEXTTABLE and SCALE_LINEAR_AND_TEXTTABLE

▶ ASCRTE-5445 Fixed known issue: Rte_Read/Rte_IRead/Rte_IStatus might return the wrong value until first reception of data if invalidation for complex types is used

▶ Added support for InitEvents triggering runnable entities via RteInitializationRunnableBatch

▶ Added a prefix for Shared Memory Communicator (SMC) inter-core variables

Module version 6.2.23

2018-10-03

▶ Enabled moving of multiple mapped events / runnables at once and editing of position numbers

▶ Changed Rte Bswmd generator to a module dependent pre generator

▶ ASCRTE-5271 Fixed known issue: The Rte may not compile if intra-partition mode management is used and inter-partition events with mode disabling dependencies exist

▶ Updated verification of [constr_4022] to accept BswModuleDescription.implementedEntry according to AUTOSAR 4.3.1

▶ ASCRTE-5420 Fixed known issue: The prioritization rules of attribute SwDataDefProps.swImplPolicy are not handled properly for a VariableAccess to a sender-receiver data element

▶ ASCRTE-5423 Fixed known issue: Os events may be falsely shared for executable with a minimumStartInterval greater zero

▶ Implemented: Support for multiple dirty flags

▶ ASCRTE-5421 Fixed known issue: The Rte generation fails when using floating point values for CompuScale limits

▶ Added support for empty initialization of variable size arrays

▶ ASCRTE-5446 Fixed known issue: The ROM Block Location Symbols are not generated in the Rte_-NvMData.h if SwcServiceDependency.assignedDatas in the role "defaultValue" are used

▶ ASCRTE-5432 Fixed known issue: Rte_Cbk.h might not include Nvm_Types.h when partitioning is enabled with NvBlockSwComponents

Module version 6.2.22
2018-08-24

▶ ASCRTE-5327 Added deviation: A DataReceivedEvent for a queued sender-receiver data element is triggered even if enqueuing of the data failed due to a full receive queue

▶ Implemented: Component Data Structure (CDS) generation optimization regarding data handle buffers and inter runnable variables

▶ ASCRTE-5311 Fixed known issue: The generated Rte BSWMD file does not contain all used memory sections

Module version 6.2.21
2018-08-08

▶ Implemented: A configuration parameter (SpinlockAllocationStrategy) that makes it possible to allocate spinlocks with different strategies

▶ ASCRTE-5387 Fixed known issue: The Rte generator fails with an error if an event with activationReasonRepresentation is non-exclusively mapped to a task

► Improved: The Rte shall find the NvMBlockDescriptor entry of an NvBlockDescriptor via name matching if no RteNvmBlockRef exists for it

► ASCRTE-5378 Fixed known issue: The Rte does not set Xfrm(Inv)TransformerBswModuleEntryRef correctly if name clashes during the allocation of XfrmImplementationMappings occur

Module version 6.2.20

2018-07-27

► Implemented basic task support for Rte and Bsw timing events mapped to the same task in case OneScheduleTablePerPartition is enabled

► ASCRTE-5328 Fixed known issue: The Rte might not send the result of a server runnable that is started by multiple OperationInvokedEvents which are mapped to the same task

► Changed queueing behavior for asynchronous result-free client-server communication by permitting multiple outstanding server invocations for the same client

► Added new optional configuration parameters RteServerQueueLength and RteBswServerQueueLength as defined by AUTOSAR RfC #79150

► Added configuration check for BSW mode communication with mode users on multiple partitions

► Added support for configurable type of buffer length for transformers

Module version 6.2.19

2018-06-28

► Updated the included memory mapping header file for Bsw module entities in the sense of [SWS_Rte_-07830], i.e. <Msn>[_<vi>_<ai>]_MemMap.h

► Added support for InitEvents triggering runnable entities via OS task

► Removed error message RTE_59 which states that the implementation selection could not be found although there is only one SwcImplementation available and thus does not require a configuration of the RteImplementationRef parameter

► Implemented vendor-specific MetaData support of type SOCKET_CONNECTION_ID_16 for inter-ECU client-server communication

► ASCRTE-5259 Fixed known issue: The Rte overwrites the complete RamBlock despite a configured TextTableMapping if two PR-Ports are connected

► ASCRTE-5151 Fixed known issue: The Rte doesn't allocate consistency mechanism for atomic type in receiver buffer despite enabled serializer

► ASCRTE-5269 Fixed known issue: The Rte_NvMData.h may declare parameter buffers as extern and static if the calibration parameter uses a ServiceDependency

► Removed the declaration of a runnable entity from the Application Header File, if a SwcBswRunnableMapping exists for it (AUTOSAR RfC 79717)

► Added support for swAddrMethod respectively swDataDefProps in 'C' typed PerInstanceMemory

► ASCRTE-5214 Fixed known issue: If a server runnable handles multiple operations with an array argument of different length, the argument's data may be lost

► ASCRTE-5274 Fixed known issue: The Rte Generator may generate duplicate event names if Bsw communication with multiple internal behaviors is used

► Implemented check for LdCom usage without DataTransformation

► Removed reporting of message RTE_612 in case of UINT8_N or UINT8_DYN signal type

► Implemented support for NvM API runnables that can now be mapped to a task

► Changed 'Required' attribute in EventMapping-Tab of Rte Editor for BswOperationInvokedEvents and BswInternalTriggerOccuredEvents to false

► ASCRTE-5294 Fixed known issue: The Rte generates an insufficient counter data type for queued Rte or Bsw trigger communication if the configured queue length is 256

► ASCRTE-5297 Fixed known issue: The Rte generates an insufficient counter data type for data filter oneEveryN if the configured period is 256

► Fixed known issue: Opening and Closing of 'NVRAM Allocation' of the Rte Editor erases the NVRAM Allocations made in the Generic Editor if no Service Dependency exists for the mapping

► ASCRTE-5272 Fixed known issue: The NvBlockDescriptor's RunnableEntity for storeImmediate is triggered too often

► Implemented: The Rte shall find the NvMBlockDescriptor entry of an NvBlockDescriptor via name matching if no RteNvmBlockRef exists for it

Module version 6.2.18

2018-05-25

► Improved memory usage for mixed implicit and explicit sender-receiver communication; the explicit Rte_Read API now reads from an existing data handle buffer under certain circumstances

► ASCRTE-5161 Fixed known issue: The optional transformerError argument is not considered for the intra-ECU asynchronous Rte_Call() API

► ASCRTE-5136 Fixed known issue: Under certain conditions the Rte generates an invalid signature for Rte_ComHook_<ComSignal>_SigTx/SigRx trace functions

► Implemented support of maps and curves (compound primitives)

► ASCRTE-5200 Fixed known issue: Floating point constants cause compiler warning

► Implemented support of transport protocol for serialization

► ASCRTE-5167 Fixed known issue: The Rte calls a runnable in a task of another partition or falsely reports error RTE_571 when software components mapped to different partitions have compatible timing events mapped to the same task

► ASCRTE-5252 Fixed known issue: Dirty Flag storeImmediate NvBlockDescriptor's RunnableEntity is triggered too early

Module version 6.2.17

2018-04-20

► Added support for providing the activating event of an ExecutableEntity

► ASCRTE-5202 Fixed known issue: The generated Rte may not compile if an inter-partition BswOperationInvokedEvent is mapped to a task with many other events

► ASCRTE-5218 Fixed known issue: The Rte Generator may generate an insufficient buffer in a Com callback for a multiple receiver transformer scenario

► ASCRTE-5165 Fixed known issue: The generated Rte BSWMD file contains non-existing measurement symbols if IOC partitioning support is enabled

► Implemented checks for LdCom usage without DataTransformation and for configuring the same ISignalToIPduMapping for multiple Com/LdCom containers.

► Improved the Rte generator with ClientIdDefinitionSet support for inter-ECU client-server communication

► ASCRTE-5049 Fixed known issue: The Rte generates non-compilable code if all BSW module instances are mapped to the same partition and the RteGeneratorOutput is BSW_SCHEDULER_ONLY

Module version 6.2.16

2018-03-16

► Added support for setting the value of Rte_TransformerClass to RTE_TRANSFORMER_UNSPECIFIED and Rte_TransformerErrorCode to E_OK for transformer error argument of an Rte Api to which no data transformation applies

► ASCRTE-5129 Fixed known issue: The transformer error parameter for the Rte_Write()/Rte_Send()/Rte_Call()/Rte_Trigger() function is not updated when the partition is not active

► Implemented support for multiple Com instances

► ASCRTE-4991 Fixed known issue: The generated Rte code may cause an OS_E_INTDISABLE error if the Det checks are enabled

► Improved the check for the error message RTE_803

► Removed Permanent RAM Block consideration for SwcServiceDependency / RoleBasedDataAssignment roles different to "ramBlock" or "ramMirror"

Module version 6.2.15

2018-02-23

▶ ASCRTE-5112 Fixed known issue: The Rte Generator may fail if multiple IRead APIs are configured for the same data element and no initial value is specified

▶ Removed unused sender-receiver buffers in case only implicit read accesses and no write accesses are configured

▶ ASCRTE-4993 Fixed known issue: The Rte verifier does not consider the default value of the swImplPolicy

▶ Improved Rte Generator by ignoring unmapped runnables for send/receive signal/signal group data consistency. The new warnings RTE_610 and RTE_611 are reported for affected signals/signal groups

▶ ASCRTE-5089 Fixed known issue: The Event Mapping tab of the Rte Editor reports a NullPointerException if a BswOperationInvokedEvent is configured

▶ Improved NvBlockSwComponentType dirty flag support: A runnable in the context of a NvBlockSwComponentType can now handle multiple NvRamBlocks

▶ Improved Rte Generator to support LINEAR CompuMethods with a factor equal to zero

▶ ASCRTE-5080 Fixed known issue: The values of the predefined error codes are generated with a cast to the type Std_ReturnType

▶ Improved the filtering in Rte Editor for required event mapping for DataReceivedEvent(s) of NvBlockSwComponentType(s)

▶ Improved generation of function like macro Rte_WaitGetClearEvent by calling the Rte trace hooks even if Os defines its own WaitGetClearEvent function

▶ Improved receive buffer sharing for explicit intra-partition 1:n sender-receiver communication

▶ Added support for debounced activation of executable entities

▶ ASCRTE-5135 Fixed known issue: BSW Os task shutdown may be delayed

▶ Improved data conversion by considering the baseType of the NetworkRepresentationProps

Module version 6.2.14

2018-01-19

▶ Improved generation of very large numerical value specifications and data filter attributes

▶ Improved the Rte generator to accept a missing BswImplementation.ResourceConsumption that is mandatory by means of the strict AUTOSAR schema

▶ ASCRTE-5066 Fixed known issue: The Rte generator may fail if the number of elements of an ApplicationRecordDataType of a require port is less than the number of signals of the signal group mapped to the provide port

▶ ASCRTE-4691 Fixed known issue: The timing event execution offset may be shifted after a partition restart

Module version 6.2.13

2017-12-20

▶ ASCRTE-5019 Fixed known issue: The generated Rte may not compile if the Det checks are enabled and multiple clients call the same server by a direct function call

▶ Added support for external replacement invalidation for sender-receiver communication

▶ Added support for handling of Section Name Prefix for BswSchedulableEntitys

▶ ASCRTE-4965 Fixed known issue: The Rte reports an error if a variable size array is initialized with an application value specification without a constant mapping

Module version 6.2.12

2017-12-11

▶ ASCRTE-4979 Fixed known issue: The Rte contract phase does not compile if implicit sender-receiver communication using array type is configured

▶ Improved Rte generation time for data types, especially for single-pointered or double-pointered calibration parameter reference tables

▶ ASCRTE-4995 Fixed known issue: The Rte generator fails if an operation argument is named "Status" and inter-partition-intra-ECU client-server communication is used

▶ ASCRTE-4975 Fixed known issue: Subsequent calls of Rte_Call() always return RTE_E_LIMIT if RTE_-E_HARD_TRANSFORMER_ERROR occurred in the result signal callback (inter-partition)

▶ ASCRTE-4689 Fixed known issue: Com_SendSignal/Com_SendSignalGroup function may be accessed concurrently for the same Com signal/Com signal group

▶ ASCRTE-4973 Fixed known issue: Os schedule table offset is not always taken into account in case of OSEK compatibility mode

▶ Updated Memory Mapping: No additional type qualifier is added to the Memory Allocation Keyword if a SwAddrMethod is configured

▶ Added support for implicit communication for directly called ModeSwitchEvent runnables in the context of asynchronous mode communication

Module version 6.2.11

2017-11-17

▶ ASCRTE-4976 Fixed known issue: The Rte generator fails if PossibleErrorRefs are used for asynchronous inter-partition-inter-ECU client-server communication

▶ Removed falsely reported warnings for constr_9082

▶ ASCRTE-5010 Fixed known issue: The access to the event bit mask of the internal Rte event buffer mechanism may be out-of-bounds

▶ ASCRTE-4326 Fixed known issue: Some inter-partition channels are not protected against concurrent access

▶ Added support for intra partition Bsw client/server calling chains

▶ ASCRTE-4969 Fixed known issue: Enumerations of ImplementationDataTypeElements of category VALUE are not considered

Module version 6.2.10

2017-10-20

▶ Improved task mapping verification for trigger communication

▶ ASCRTE-4948 Fixed known issue: The Rte generates non-compilable code in case of intra-partition sender-receiver communication with complex data type and handleInvalid set to REPLACE

▶ ASCRTE-4964 Fixed known issue: Subsequent calls of Rte_Call() always return RTE_E_LIMIT if RTE_-E_HARD_TRANSFORMER_ERROR occurred in the result signal callback

Module version 6.2.9

2017-10-06

▶ ASCRTE-4846 Fixed known issue: The Rte_Enter and Rte_Exit API return a non-void value if OsResource is used and function elision is enabled

▶ ASCRTE-4782 Fixed known issue: The RTE generates incorrect code for sender/receiver communication with transformers and partial records but no subElementMappings

▶ ASCRTE-4928 Fixed known issue: The Rte does not propagate the transformerError to Rte_Read() for unqueued inter-partition inter-ECU communication

▶ Added support for implicit communication for directly called triggers

▶ ASCRTE-4924 Fixed known issue: Subsequent calls of Rte_Call always return RTE_E_LIMIT if RTE_E_-HARD_TRANSFORMER_ERROR or RTE_E_COM_STOPPED was returned before

▶ ASCRTE-4847 Fixed known issue: Implicit inter-runnable variable buffer is not updated for a RunnableEntity that is mapped to a BswModuleEntity and started by a BSW event

▶ Implemented support for measurement of Nv RAM Block and non volatile data communication

▶ Added dirty flag support for NvBlockSwComponentType

▶ Added support for the Rte_NvMNotifyJobFinished callback

▶ ASCRTE-4899 Fixed known issue: The Rte does not generate any code if an invalid value is specified for a deeply nested element

▶ Implemented memory mapping initialization strategy

▶ ASCRTE-4925 Fixed known issue: The Rte may generate a non-existing memory section in the SchM module interlink header

▶ Added support for Bsw client/server communication

Module version 6.2.8

2017-08-25

▶ ASCRTE-4830 Fixed known issue: An out of bounds access may occur in the APIs Rte_Enter / Rte_Exit if the Det check is enabled in the Rte

▶ ASCRTE-4850 Fixed known issue: The Rte Generator fails if a blocking Rte_Result without timeout references an asynchronous server call point with a timeout

▶ ASCRTE-4893 Fixed known issue: The Rte_Main.h may declare parameter buffers as extern and static if partitioning is enabled

▶ Added support for SenderReceiverToSignalGroupMapping/SenderRecArrayTypeMapping to byte arrays

▶ Added support to configure if the Os supports OsSpinlockLockMethod which allows the Rte to generate optimized spinlocks

▶ ASCRTE-4860 Fixed known issue: The Rte Editor configures wrong trace hook function names for BSW Schedulable entities

▶ ASCRTE-4892 Fixed known issue: The Rte Generator fails if partitioning is disabled and a basic software module instance or software component instance is mapped to an Os application

▶ Added deviation to document that the Rte allows multiple inter-partition synchronous client server calls for the same client

Module version 6.2.7

2017-07-28

▶ Improved sharing of inter-partition S/R channels

▶ ASCRTE-4834 Fixed known issue: The Rte generator may apply a wrong event setting mechanism in case of inter-partition inter-ECU client-server communication

▶ Removed unused Rte_SetEvent macros from Rte_Intern<Partition> header files

▶ ASCRTE-4857 Fixed known issue: ParameterElementGroup types are not generated in Rte_Main.h

▶ ASCRTE-4773 Fixed known issue: The Rte_Read API does not overwrite the whole OUT parameter while reading from a NvRamBlockElement with a bitfield texttable mapping

Module version 6.2.6

2017-06-30

▶ ASCRTE-4737 Fixed known issue: Rte could report a misleading error message for events with mode-disabling dependencies in certain conditions

▶ ASCRTE-4687 Fixed known issue: The Rte_Read API does not protect receiving of a Com signal group against concurrent access if DirectReadFromCom applies

▶ ASCRTE-4781 Fixed known issue: The Rte considers a sender/receiver partial record communication as incompatible if the matching structure elements are not in the same order

▶ Added user-defined exclusive area implementation mechanism

▶ ASCRTE-4768 Fixed known issue: The RTE ignores the PortAPIOption errorHandling for the generation of direct function calls to a server runnable

▶ Added support for intra-ECU sender-receiver communication with data transformation

▶ ASCRTE-4819 Fixed known issue: Implicit S/R communication is not working for a RunnableEntity that is mapped to a BswModuleEntity and started by a BSW event

Module version 6.2.5

2017-05-31

▶ ASCRTE-4747 Fixed known issue: The Rte Generator may fail with an OutOfMemoryError if many direct server calls are configured

▶ ASCRTE-4802 Fixed known issue: The Rte may falsely report the error RTE_538 during execution of the SvcAs

▶ Improved the order of intra-partition unqueued sender/receiver receive buffers definitions to reduce the amount of MemMap.h inclusions

Module version 6.2.4

2017-05-05

▶ Removed filter flag from sender/receiver buffer

▶ Added support for Os alarm and schedule table allocation with Os counter on different core. This allows to use a single Os counter on a partitioned multi-core system

▶ ASCRTE-4665 Fixed known issue: Rte_Read does not return the initial value of a complex inter-ECU communication data under certain conditions

▶ ASCRTE-4683 Fixed known issue: The Rte does not send any data if the S/R data queue overflows in case of inter-partition-inter-ECU communication

▶ ASCRTE-2026 Fixed known issue: If inter-partition mode management is used with mode disabling dependencies, the Rte may not compile

▶ ASCRTE-4695 Fixed known issue: The synchronous client may not wait for the server to finish while an exclusive area is active

▶ ASCRTE-4760 Fixed known issue: The Rte Generator fails if a structure ImplementationDataType contains a pointer element with an invalid target type

▶ Removed the receive buffer structure data types for intra-partition unqueued sender/receiver communication. Each receive buffer element is now a discrete global variable

▶ Updated the Rte Verifier to report an error if a BSW requiredTrigger is connected to more than one BSW providedTriggers

Module version 6.2.3

2017-03-31

▶ Added support for autonomous error reaction for inter-ECU client/server communication with transformers

▶ Added support for TransformerHardErrorEvents for inter-ECU client/server communication

▶ ASCRTE-4698 Fixed known issue: DataReceiveErrorEvents are not triggered if inter-ECU S/R invalidation is handled by the Rte and no data receive point exists

▶ ASCRTE-4651 Fixed known issue: The Rte may not compile if a schedulable entity and a runnable entity with a blocking API are mapped to the same task

▶ ASCRTE-4692 Fixed known issue: The Rte doesn't create the memory section in the Rte_Bswmd.arxml for a RamBlock of a NvBlockSwComponentType

▶ ASCRTE-4704 Fixed known issue: The Rte generates uncompilable MemMap definitions if SectionInitializationPolicies are used according to AUTOSAR specifications

▶ Improved data consistency by locking the interrupts when a spinlock is used

▶ ASCRTE-4621 Fixed known issue: The RTE does not provide NULL to all transformers in a chain if executeDespiteDataUnavailability is enabled

▶ Added suport for bitfield mapping of multiple require variable data prototypes to one provide variable data prototype

▶ Implemented Asynchronous Mode Switch

▶ Added allocation of init and invalid values for ComGroupSignals and ComRxDataTimeoutAction for ComGroupSignals/ComSignals via the service needs assistant

▶ Removed unused Rte_IsModeDisablingDepSet functions from Rte_<partition>.c files

Module version 6.2.2

2017-03-03

▶ ASCRTE-4064 Fixed known issue: Implicit communication with data conversion is not working for send signals/signal groups

▶ Improved measurement support for inter-partition sender/receiver communication in conjunction with the SharedMemory communication mechanism

► ASCRTE-4664 Fixed known issue: The Rte Editor does not provide available NvM blocks

► Improved queued triggering to use basic task where extended task is not mandatory

► Improved generated code relaxing the restriction regarding the SwcInternalBehavior in case of NvBlock-SwComponentType

► ASCRTE-4531 Fixed known issue: DataFilter oneEveryN is not taking into account the offset and period properly

Module version 6.2.1

2017-02-03

► ASCRTE-4131 Fixed known issue: The Rte does not consider the initial value for an unconnected parameter port and reports an error

► ASCRTE-4547 Fixed known issue: The Rte does not generate a Rte_IrTrigger API for each RunnableEntity if the InternalTriggeringPoints have equal short names

► ASCRTE-4599 Fixed known issue: An exclusive area which uses COOPERATIVE_RUNNABLE_PLACE-MENT may be disabled by another exclusive area which uses OS_RESOURCE

► Improved generated code for Rte_Feedback API in case "function elision" is enabled

► Added custom constraint checks for LINEAR CompuMethods in the sense of constr_1375

► Improved task mapping scenarios for runnables with a synchronous server call point that is implemented as direct function call. The runnable may now be mapped to a basic task regardless of the configured timeout value

► ASCRTE-4409 Fixed known issue: The component data structure may not be initialized if the SWC contains two provide ports which refer the same trigger interface

► Updated the inter-partition mode receive emptyQueue

Module version 6.1.172

2017-01-05

► Added support for 64 bit datatypes in sender/receiver, client/server and inter-ECU communication

Module version 6.1.171

2016-12-14

► ASCRTE-4554 Fixed known issue: DataReceiveErrorEvents are not triggered in case of inter-partition inter-ECU S/R invalidation

▶ Consider memory mapping for SMC inter-partition communication

▶ ASCRTE-4525 Fixed known issue: DataFilter newIsWithin is more restrictive than specified

▶ ASCRTE-4528 Fixed known issue: TaskType BASIC leads to an extended task for Task Mapping Scenario B1

▶ ASCRTE-4551 Fixed known issue: If a timeout is specified for an asynchronous server callpoint and no waitpoint is defined, the Rte Generator does not setup an alarm in Rte_Call

▶ ASCRTE-4558 Fixed known issue: The Rte generator fails for inter-ECU client/server communication if an ApplicationRecordDataType argument is mapped to an ImplementationDataType argument

▶ Improved merge of OsEvents for multiple TriggerOccuredEvents

▶ ASCRTE-4573 Fixed known issue: The Rte generator fails if a complex ApplicationArrayDataType with variable size semantics is used

▶ ASCRTE-4483 Fixed known issue: The Rte generates empty Rte_Invalidate API in case of complex inter-ECU sender/receiver communication

▶ ASCRTE-4524 Fixed known issue: Invalidation of primitive inter-partition inter-ECU sender/receiver communication is not working

▶ ASCRTE-4553 Fixed known issue: Inter-ECU S/R invalidation of signal groups does not consider the parameter InterECUInvalidationHandledByRte

▶ ASCRTE-4482 Fixed known issue: The Rte generates a non-blocking instead of a blocking Rte_Feedback API under certain conditions

▶ Implemented function elision for the Rte_Trigger and Rte_IrTrigger API

▶ Added support for memory mapping considering SwAddrMethod, SwAlignment and mapping of RunnableEntities in the Application Header File

▶ Improved declaration of shared memory API in Smc.h by generating partition-specific header files

Module version 6.1.170

2016-11-08

▶ ASCRTE-3346 Fixed known issue: The Rte does not generate Rte_Main.[ch] if partitioning is enabled but only one partition is configured

▶ ASCRTE-4434 Fixed known issue: If COOPERATIVE_RUNNABLE_PLACEMENT is configured for an exclusive area, the Rte might allocate more than one internal resource for a task

▶ ASCRTE-4176 Fixed known issue: Port interface mappings may not be applied for client server interfaces

▶ ASCRTE-4497 Fixed known issue: If limits are defined for a signed data type, the Rte generates an unsigned suffix

▶ Fixed violation for MISRA rule 8.8 by moving external declaration of implicit buffers to Rte_Partitioning.h

▶ ASCRTE-4210 Fixed known issue: The Rte Editor may create duplicate Bsw module instances

▶ Added support for asynchronous inter-ECU client server communication

▶ ASCRTE-4532 Fixed known issue: The Rte generates an invalid EventMaskType typedef under certain circumstances

▶ Extended data consistency for inter-core C/S communication related to bugfix ASCRTE-4445

▶ Fixed MISRA rule 5.5 violations which are related to the partition state and execution counter variables

▶ ASCRTE-4110 Fixed known issue: Some APIs do not consider data conversion correctly

▶ Implemented memory mapping by the size of intra-partition variables if no explicit SwAddrMethod is given

▶ Added inter-partition-inter-ECU-inter-core communication optimization: one Com signal IP channel for each core

▶ ASCRTE-4388 Fixed known issue: The Rte expects a parameter data prototype mapped to a per instance memory in the same RoleBasedDataAssignment

▶ Improved verifier for constr_1094 in case of runnables of NvBlockSwComponentType

Module version 6.1.169

2016-10-10

▶ Improved the resolution of SwDataDefProps for implementation data types of category TYPE_REFERENCE.

▶ ASCRTE-4487 Fixed known issue: The Rte uses a wrong buffer size for Com/LdCom signal reception callbacks when client/server inter-ECU communication is used

▶ Relaxed the verifier for implementation data types with the same short name and DataConstr between C/S arguments

▶ Added configuration support for trigger queue lengths in the Rte Editor

▶ Added support for SubElementMapping for inter-ECU S/R Data Serialization

Module version 6.1.168

2016-09-09

▶ Added support for sharing implicit S/R communication buffers (data handle buffers) between partitions

▶ ASCRTE-4590 Fixed known issue: The elided Rte_Write API macro does not update data handle buffers of implicit readers

▶ ASCRTE-4389 Fixed known issue: The Rte Editor may falsely warn that the schedule table expiry points exceed 5000 entries

▶ ASCRTE-3906 Fixed known issue: The Rte does not consider the activation mechanism for the SchM schedule table

Module version 6.1.167

2016-08-05

▶ ASCRTE-4424 Fixed known issue: The Rte Generator may fail if a runnable is started by multiple OperationInvokedEvents for different client/server operations

▶ Added queueing support for the SchM_ActMainFunction API

▶ ASCRTE-4433 Fixed known issue: The Rte Generator fails if incompatible inter-ECU client/server operations are called

▶ ASCRTE-4337 Fixed known issue: The runnable based scope for application header files might not detect a wrong runnable context

▶ Improved inter-partition intra-core synchronous Rte_Call to not wait for Os events anymore if the client's task priority is lower than the server's task

▶ ASCRTE-4426 Fixed known issue: The Rte Bswmd generation fails if measurement is enabled on a PR-PortPrototype

▶ ASCRTE-4443 Fixed known issue: The Rte uses a wrong buffer size for Com signal reception callbacks when S/R inter-partition/inter-ECU communication is used

▶ ASCRTE-4444 Fixed known issue: The Rte Generator might not set the isUpdated flag in a Com/LdCom signal reception callback

▶ ASCRTE-4445 Fixed known issue: Inter-core inter-partition shared-memory communication wrongly shares Os resources

▶ ASCRTE-4384 Fixed known issue: The Rte resets Bsw modes during Rte_Start

Module version 6.1.166

2016-06-30

▶ ASCRTE-4360 Fixed known issue: The Rte Bswmd generation fails if an McDataInstance refers a CompuMethod without a Unit

▶ ASCRTE-4385 Fixed known issue: Implicit InterRunnableVariable communication might not work

▶ ASCRTE-4387 Fixed known issue: The enhanced Rte_Mode API returns a wrong mode if the mode provider partition is not started and the ModeDeclarationGroup has category EXPLICIT_ORDER

▶ Removed unused global buffers for unaccessed parameter data prototypes

▶ ASCRTE-4397 Fixed known issue: The Rte may not apply data consistency for inter-core SMC channels

▶ Added support for queued SchM_Trigger API

▶ ASCRTE-4362 Fixed known issue: The Rte might use the same Os event for an Bsw and Rte event

▶ ASCRTE-4393 Fixed known issue: Shared memory with inter-core-inter-ECU transformer leads to non-compilable code

▶ ASCRTE-4412 Fixed known issue: The Rte_Trigger API does not return a value for an unconnected queued trigger

Module version 6.1.165

2016-06-07

▶ Added use of SvcAs feature to configure VariableLength parameter of IocChannels

▶ ASCRTE-4329 Fixed known issue: Possible infinite wait in inter-core synchronous Rte_Call

▶ Added use of SvcAs feature to configure XfrmOsApplicationRef parameter of XfrmImplementationMappings

▶ ASCRTE-4288 Fixed known issue: Rte_Read might return RTE_E_OK although the API is unconnected

▶ Added support of SubElementMapping with Transformer

▶ Improved asynchronous inter-partition client/server communication so that the client runnable may now be mapped to a basic task

▶ ASCRTE-4358 Fixed known issue: A client-server request's sequence counter value may be truncated

▶ Added buffer length check for copying raw data to local buffer for re-transformation in (Ld)Com callback

Module version 6.1.164

2016-04-29

▶ ASCRTE-4279 Fixed known issue: Software component without Internal Behavior results in the failure of RTE

▶ Improved allocation of XfrmImplementationMapping so that only a single entry for the same tx signal (group) path is created and XfrmVariableDataPrototypeInstanceRef is left disabled

▶ ASCRTE-4200 Fixed known issue: The verifier wrongly reports an error about incompatible categories between two ImplementationDataTypeElements

▶ Changed dependency to the Platforms plugin from mandatory to optional

▶ ASCRTE-4295 Fixed known issue: The Rte generator fails if calibration support of a ParameterSwComponentType is different than for the connected software component

▶ Added support for the AutosarVariableIRef instance reference in NvRamBlockElements

▶ Implemented relaxation of constr_1175

▶ Added support for inter-partition/inter-ECU client server communication

▶ ASCRTE-3895 Fixed known issue: The enhanced Rte_Mode API does not return the initial mode if no ModeSwitchEvents exist for inter-partition mode management

▶ Removed unused mode variable from the mode provider partition in case of inter-partition mode management with enhanced Rte_Mode API

▶ ASCRTE-4317 Fixed known issue: The verifier wrongly reports an error if an ApplicationArrayDataType is mapped to a variable-size array ImplementationDataType

▶ Added support for variable-size arrays (VSA_LINEAR) for inter-partition queued sender-receiver communication with SMC

▶ ASCRTE-4261 Fixed known issue: An Rte client server reception callback function may retransform invalid raw data

Module version 6.1.162
2016-04-01

▶ ASCRTE-4243 Fixed known issue: The Rte generator fails if incompatible initial values are configured for certain data prototypes with measurement support

▶ ASCRTE-4223 Fixed known issue: Rte.c does not include NvM.h leading to compilation error when a software component uses NvMService through a NvBlockSwComponent

▶ Improved allocation of expiry points for timing events with an offset greater than 0

▶ ASCRTE-4252 Fixed known issue: The Rte does not configure the LdComRxIndication for client/server call signals

▶ ASCRTE-4248 Fixed known issue: An internal error is reported if a task chain is created of non periodic tasks

▶ ASCRTE-4233 Fixed known issue: The Rte creates nested MemMap defines

▶ ASCRTE-4269 Fixed known issue: ComIPChannel identifier defines may not be generated if senders are located on more than one non-Bsw partition

▶ Implemented the definition of partition specific and shared memory sections within the Rte's BSWMD

▶ ASCRTE-4260 Fixed known issue: The Rte does not consider invalid values on array application data types

▶ Shortened names of allocated XfrmImplementationMappings

▶ ASCRTE-4266 Fixed known issue: The Rte Generator does not generate code if inter-core C/S communication with SharedMemory is used

▶ ASCRTE-4281 Fixed known issue: If a task chain is configured, the Rte allocates expiry points for all tasks in the chain

▶ Added support for inter-partition/inter-ECU S/R communication

Module version 6.1.161
2016-03-04

▶ Reworked generation of template/example code for each software component.

▶ Added comment next to global variables which shows the original name before MD5 hashing is applied

► Improved the verification of self referencing datatypes to check indirect referencing

► ASCRTE-4228 Fixed known issue: The Rte editor generates wrong Vfb trace hook function names for certain API functions

Module version 6.1.160

2016-02-05

► Added use of SvcAs feature to configure LdComIPdus

► Added support for fire-and-forget (result-free) asynchronous client/server calls as specified by RfC 70295.

► Improved handling of the Rte basic software module description: The Basic Software Scheduler does not consider an existing Rte Bswmd as subject for code generation anymore

Module version 6.1.159

2016-01-15

► ASCRTE-4069 Fixed known issue: The Rte editor may enable unused configuration parameters

► Removed Det RTE_E_DET_UNINIT logging for the SchM_Enter and SchM_Exit Apis

► ASCRTE-4075 Fixed known issue: Incorrect warning regarding unconnected client server ports

► ASCRTE-4000 Fixed known issue: If transformerError argument exists the Rte template code does not compile

► ASCRTE-4072 Fixed known issue: Rte falsely reports an error when SwcServiceDependency is defined on a PortPrototype which is involved in Nv data management

► ASCRTE-4125 Fixed known issue: Under certain conditions the Rte does not generate enumeration definitions

► Adapted the validation of ModeRequestTypeMaps for support of multiple BswInternalBehaviors

► Updated the verification of constraint constr_4071

► ASCRTE-4145 Fixed known issue: The measurement symbols exported to the Rte Bswmd file may not exist in the generated source code

► Added support for mixed usage of IOC for inter-core and Shared Memory Communicator (SMC) for intra-core communication

► Added system description verifier check for constr_1386 (PortDefinedArgumentValue shall only be defined for AbstractProvidedPortPrototype)

► ASCRTE-4152 Fixed known issue: If inter-partition invalidation is used, it might happen that the receiver reads old data

► ASCRTE-4142 Fixed known issue: SchM_Send and SchM_Receive APIs to do not return a value when a port is unconnected

▶ ASCRTE-4114 Fixed known issue: DataSendCompletedEvents are not properly triggered under certain conditions

▶ ASCRTE-4159 Fixed known issue: An extended Os task may not be activated on startup

▶ ASCRTE-4178 Fixed known issue: The Rte Editor reports a NullPointerException under certain conditions

Module version 6.1.158

2015-11-10

▶ Added use of SvcAs feature to configure Nv blocks and callbacks

▶ Changed parameter name XfrmTransformationBswModuleEntryRef for reading from the Xfrm Ecu configurations to XfrmTransformerBswModuleEntryRef (see AUTOSAR RfC #68531)

▶ Added support for inter-partition task chains and multiple task chains

▶ Improved configuration check regarding the mapping of Bsw events to tasks belonging to the configured Bsw Os application

▶ Improved error messages for value specifications that are not compatible to the data type

▶ ASCRTE-4068 Fixed known issue: The Rte editor does not show any error or warning under several conditions

▶ Extended verifier check for constr_1075 (point 1.h is now checked according to RfC#67491/ASR 4.2.2)

▶ ASCRTE-4073 Fixed known issue: The Rte generator ignores initial values for an arTypedPerInstanceMemory defined by application value specifications

▶ ASCRTE-4071 Fixed known issue: A value specification for a multi-dimensional ApplicationArrayDataType can not be applied to a data prototype

▶ Added use of SvcAs feature to configure XfrmImplementationMappings

▶ ASCRTE-4080 Fixed known issue: Under certain conditions the Rte generator fails if measurement is enabled for a data prototype of complex application data type

▶ Removed erroneous warning if init value of primitive data element and mapped Com signal match

▶ ASCRTE-4041 Fixed known issue: The Rte writes array initial values to the Com configuration with an "U" suffix

▶ Updated the Rte Verifier to report a warning if a timeout is configured and LdCom is used

▶ Removed support for system signal/signal group mappings in the Rte editor. The functionality has been moved to the System Signal Mapping Editor.

▶ Removed Service Port Mapping tab from Rte editor. The functionality has been moved to the Connection Editor.

▶ Improved implicit read/write access for server runnables which are triggered by a direct call. The client(s) can be mapped on different tasks now as long as they do not preempt each other.

▶ Updated the Rte Verifier to report an error if a software component name conflicts with a standard Rte header file

Module version 6.1.157

2015-10-09

▶ ASCRTE-3984 Fixed known issue: Inter-partition mode switch events may lead to a deadlock on startup

▶ Added serialization support for QUEUED data reception

▶ Added supported for queued Bsw sender receiver communication

▶ Added support for specification of XfrmVariableDataPrototypeInstanceRef (multiple receivers)

▶ ASCRTE-4016 Fixed known issue: If no initial value is specified for a C typed per instance memory, the Rte generates non-compilable code

▶ Added support for the use of NumericalValueSpecifications to define the invalidValue of an array data type

▶ Updated the definition of the Rte_Cs_TransactionHandleType according to RfC#69581

▶ ASCRTE-4012 Fixed known issue: Rte miscalculates buffer consumptions for transformers in case the CompuNumerator contains a linear factor

▶ ASCRTE-4014 Fixed known issue: Blocking Rte_Receive may return undefined data even if the queue is empty

▶ Removed the SomeIp returnValue parameter when a server runnable has no application errors

▶ ASCRTE-4023 Fixed known issue: The transformerError argument passed to a runnable may contain undefined values

▶ Updated the system description verifier to reject configurations where executeDespiteDataUnavailability=true and signal fan-in is configured

▶ Exchanged partition IDs which are used for the naming of Rte_Smc_Data_<id>.c files with partition names

▶ Fixed the Rte verifier, which might wrongly report the warning that a category 2 runnable is not exclusively mapped to a task

▶ ASCRTE-4040 Fixed known issue: It is required to specify a DataTypeMap for single elements of an ApplicationCompositeDataType

▶ ASCRTE-4047 Fixed known issue: The Rte editor may remove data mappings from the system model

▶ Added support for the inclusion of multiple instances of a Bsw module into the same partition

Module version 6.1.156

2015-08-24

▶ Added support for the generation of the Bsw Scheduler in multiple partitions

▶ ASCRTE-3974 Fixed known issue: Undefined behavior for mode disabling dependencies which are defined in multiple partitions

▶ Added support for C/S calls to NvMServices

▶ Added verifier checks for Variable-Size Array ImplementationDataTypes

▶ ASCRTE-3983 Fixed known issue: An Rte_Call function may not dereference an INOUT operation argument

▶ ASCRTE-3923 Fixed known issue: Rte template code does not compile

▶ ASCRTE-3980 Fixed known issue: If the Rte contract phase is executed from the command line, the data type verifier is not executed

▶ ASCRTE-3905 Fixed known issue: The Rte does not consider the receiver's initial value if the invalidation policy is set to REPLACE

▶ ASCRTE-3991 Fixed known issue: High memory consumption when many complex application data types are defined

▶ ASCRTE-3955 Fixed known issue: Missing critical section for inter-partition-inter-ECU communication

Module version 6.1.155

2015-07-21

▶ ASCRTE-3933 Fixed known issue: The Rte Editor does not properly handle data mappings of application array data types within other complex types

▶ Updated verification check for constr_1060 according to RfC#68247

▶ ASCRTE-3989 Fixed known issue: Rte Api function arguments may reference an undefined data type

▶ ASCRTE-3963 Fixed known issue: The Rte verifier incorrectly evaluates the invalidation policy of sender receiver interfaces

▶ ASCRTE-3961 Fixed known issue: If an array of a structure implementation data type is specified the generated code does not compile

Module version 6.1.154

2015-06-19

▶ ASCRTE-3725 Fixed known issue: If the invalid value equals the init value, Rte_IStatus()/Rte_Read() returns RTE_E_OK for inter-partition invalidation

▶ ASCRTE-3798 Fixed known issue: A McDataInstance entry is not generated for ParameterDataPrototype elements of a ParameterSwComponentType

▶ ASCRTE-3823 Fixed known issue: Rte_Mode does not return the initial mode if no mode switch point exists for the provided mode declaration group

▶ ASCRTE-3705 Fixed known issue: If an executable entity runs inside several exclusive areas with different interrupt locking mechanisms, the Rte considers only one

▶ ASCRTE-3836 Fixed known issue: The Rte Editor does not store event mappings if software component and basic software instance names are ambiguous

▶ ASCRTE-3862 Fixed known issue: The Rte incorrectly requires the mapping of operation invoked events

▶ Removed the additional measurement buffer in case of 1:1 intra-partition sender/receiver communication

▶ Replaced buffer data structure by flat hierarchy

▶ ASCRTE-3852 Fixed known issue: The initial value of a data element may be wrong in case of unqueued 1:n inter-partition sender/receiver communication

▶ Improved the Rte Bswmd generator to only modify the system model (file SystemModel2.tdb) if necessary

▶ ASCRTE-3535 Fixed known issue: The Rte_Read API may return the last received value from Com instead of the initial value if HandleTimeoutType is set to REPLACE

▶ ASCRTE-3867 Fixed known issue: Rte_Write does not write to a subelement of a complex Nv ram block if the RootVariableDataPrototype is not set

▶ Removed constraint check constr_3131 (see RfC#69207)

▶ ASCRTE-3870 Fixed known issue: The generated Rte code does not compile if background events are used with partitioning enabled

▶ ASCRTE-3776 Fixed known issue: The Rte_Read API returns RTE_E_INVALID if the invalidation policy is set to replace and an alive timeout is specified or handle never received is enabled

▶ ASCRTE-3769 Fixed known issue: The Rte may still activate runnable entities on extended tasks after Rte_Stop has been called

▶ ASCRTE-3830 Fixed known issue: The Rte generates duplicate internal IDs for development error tracing if two software components define the same API function

▶ ASCRTE-3814 Fixed known issue: Inter-partition S/R communication lead to compiler warnings if several receivers use different structure types

▶ ASCRTE-3689 Fixed known issue: Mixing of enhanced and non-enhanced Rte_Mode API in inter-partition mode management fails

▶ ASCRTE-3290 Fixed known issue: The Rte may incorrectly report the error message RTE_503

▶ ASCRTE-3875 Fixed known issue: The Rte does not stop the BSW inter-partition task

▶ ASCRTE-3913 Fixed known issue: The Rte ignores incoming events during a blocking API call

▶ ASCRTE-3931 Fixed known issue: If an array implementation data type is referenced by an implementation data type of category TYPE_REFERENCE an internal error is reported

Module version 6.1.153

2015-04-27

▶ ASCRTE-3287 Fixed known issue: Sender receiver record element mappings to array implementation data type elements will not be recognized by the Rte editor

▶ ASCRTE-3799 Fixed known issue: The BSW main function auto-mapping functionality of the Rte editor does not re-create removed Os tasks

▶ ASCRTE-3800 Fixed known issue: Inter-partition inter-ECU sender-receiver signal fan-out does not work

▶ ASCRTE-3808 Fixed known issue: The Rte generator fails if a base type without a native declaration is used

▶ Added support for constant (e.g. initial) value specifications for complex application data types using RecordValueSpecification and ArrayValueSpecification

▶ Added support for inter-ECU client server communication

▶ ASCRTE-3819 Fixed known issue: Compiler error related to the definition of the data type Rte_TransformerError

▶ ASCRTE-3812 Fixed known issue: The inter partition Rte_Result API returns the wrong value after starting the partition

▶ ASCRTE-3821 Fixed known issue: The Rte may pass invalid data to the LdCom

▶ Added support for network representation conversion of complex sender-receiver data elements.

▶ Modified the allocation of the Rte_Result Api, an AsynchronousServerCallResultPoint is now required to generate the Rte_Result Api

▶ ASCRTE-3827 Fixed known issue: The Bsw Os Task may be unnecessarily activated during Rte_Start

▶ ASCRTE-3820 Fixed known issue: The Rte shall support definitions of limits in certain numerical formats

▶ Added support for SubElementMappings of VariableAndParameterInterfaceMappings

Module version 6.1.152

2015-02-23

▶ ASCRTE-3662 Fixed known issue: Update flag is not cleared when Rte_RestartPartition is called

▶ Added support for Internal Trigger Event Communication

▶ Implemented call of Com[Send/Receive]DynSignal only for ComSignalType UINT8_DYN

▶ Improved generated code for Sender / Receiver Serialization (use-case: usage of LdCom only)

▶ Corrected generated code for Sender / Receiver Serialization (use-case: LdComCbkTriggerTransmit)

▶ ASCRTE-3702 Fixed known issue: The Rte generator aborts with a fatal error if a Bsw trigger event is configured

▶ ASCRTE-3701 Fixed known issue: The Rte_Write API behaves like the Rte_Invalidate API if invalidation is configured and the invalid value is sent

▶ Added OS_SPINLOCK as possible parameter for selection of exclusive area implementation mechanism.

► Improved generated code for Sender / Receiver Serialization (use-case: transformerErroHandling enabled)

► Improved generated code for Sender / Receiver Serialization (use-case: transformerError prioritization)

► Improved generated code for Sender / Receiver Serialization (use-case: Transformer Soft/Hard error in Rte_Feedback/IStatus)

► Improved generated code for Sender / Receiver Serialization (use-case: macro definition for RTE_E_-TRANSFORMER_LIMIT)

► Improved generated code for Sender / Receiver Serialization (use-case: consistent (de-)serialization error in case of multiple receiver software components)

► Improved generated code for Sender / Receiver Serialization (use-case: compiler warning in case of (de)-serialization and array data types)

► Improved generated code for Sender / Receiver Serialization (use-case: startup behavior (Rx - RTE_E_-NEVER_RECEIVED+InitValues, Tx - RteFeedback))

► Improved generated code for Sender / Receiver Serialization (use-case: transformerError arg enabled + intra Ecu receiver)

► Added support for TextValueSpecifications to specify enumerations of the corresponding CompuMethod (category TEXTTABLE or BITFIELD_TEXTTABLE)

► Added support for Bitfield Textable PortInterfaceMapping for NvDataInterfaces.

► Added support for NvBlockSwComponentType NvBlockDescriptor romBlock and callback function Rte_-NvMNotifyInitBlock.

► Added support for network representation conversion of primitive sender-receiver data elements.

Module version 6.1.151

2015-01-08

► ASCRTE-3576 Fixed known issue: The Rte contract phase does not compile if implicit interrunnable variables are configured

► ASCRTE-3580 Fixed known issue: The Rte may try to lock an Os resource from within a partition that is not permitted to access the resource

► ASCRTE-3573 Fixed known issue: Rte_ModeDisablingDep_Type may use an inadequate type when using inter-partition mode management

► ASCRTE-3581 Fixed known issue: The Rte reports an internal error if update is enabled for n:1 S/R mixed inter- and intra-partition communication

► Improved generation of interrupt locks for shared buffers which are not required for tasks which have the highest priority

► ASCRTE-3457 Fixed known issue: The order of invalid values in Application Types header file is arbitrary

▶ Added support for server runnables which are triggered by a direct call that can now have implicit read/ write access

▶ ASCRTE-3592 Fixed known issue: If a task chain is configured and the first task is of task mapping scenario A2, the Rte code does not compile

▶ Added algorithm to generate Buffer names based on element names to ensure predictable symbols

▶ ASCRTE-3375 Fixed known issue: If SchM_Trigger or SchM_ActMainFunction is called the Rte ignores implicit exclusive areas

▶ ASCRTE-3625 Fixed known issue: The software component might receive old data if S/R data transformation is used

▶ Implemented [SWS_Rte_08408] so that a runnable with both implicit read and write access to a variable data prototype in the context of a PR-port uses a single shared read/write buffer

▶ ASCRTE-3638 Fixed known issue: The Rte does not generate code for an implicit exclusive area if EB_- FAST_LOCK is configured

▶ ASCRTE-3621 Fixed known issue: Rte_SwitchAck returns RTE_E_TRANSMIT_ACK although the transition status isn't done

▶ ASCRTE-3657 Fixed known issue: The Rte does not consider precedence of data properties for the generation of the McSupportData

▶ ASCRTE-3653 Fixed known issue: The Rte allocates memory for constant memory if it is referenced by a parameter access

▶ ASCRTE-3661 Fixed known issue: Inter-partition S/R communication lead to compiler warnings if used with two different but compatible structures

▶ ASCRTE-3664 Fixed known issue: A RunnableEntity might be triggered by a BswModuleEntity even if the Rte is not started

▶ Added support for External Trigger Event Communication

▶ Added basic support for NvBlockSwComponentType.

Module version 6.1.150

2014-10-13

▶ ASCRTE-3474 Fixed known issue: The Rte contract phase generator fails with an error if implicit sender receiver communication is configured

▶ ASCRTE-3466 Fixed known issue: If the service needs assistant is executed the Rte treats disabled configuration parameters as enabled

▶ ASCRTE-3475 Fixed known issue: The Rte does not generate the BSW Module Description entities for measurement correctly in case of application array and application primitive data types

▶ Added invalidation support for inter-partition communication

▶ Added configuration parameter `DisableInvalidationDataConsistency` to disable data consistency for invalidation if invalidation policy is set to keep

▶ ASCRTE-3505 Fixed known issue: Complex inter-partition inter-ECU data may be corrupted if the received Com signal group is mapped to different data elements

▶ ASCRTE-3501 Fixed known issue: The Rte contract phase does not compile if implicit sender/receiver communication is configured

▶ ASCRTE-3508 Fixed known issue: The Rte generator fails in attempt to optimize the buffer handling for an implicit receiver connected to an explicit sender

▶ ASCRTE-3506 Fixed known issue: The Rte may exchange corrupted data in case of inter-partition inter-ECU sender/receiver communication using complex data types

▶ ASCRTE-3533 Fixed known issue: The generation of the Rte Bswmd file may take very long

▶ ASCRTE-3538 Fixed known issue: The Rte contract phase generator fails with an error if per-instance memory is configured

▶ ASCRTE-3539 Fixed known issue: The Bsw main function auto-mapping functionality maps ComM main functions to an inadequate task if multiple Flexray channels are configured

▶ ASCRTE-3543 Fixed known issue: The VFB Tracing tab of the Rte Editor does not list trace hook functions for the Rte_SwitchAck API

▶ ASCRTE-3458 Fixed known issue: If two or more SwBaseTypes with the same name exist, the Rte only considers one

▶ ASCRTE-3512 Fixed known issue: The Rte Generator throws an exception if data conversion with floating point coefficients is configured

▶ ASCRTE-3524 Fixed known issue: The Rte Editor removes assembly connectors between service components and application components

▶ Added support for combined provide and require ports as described by AUTOSAR 4.1.3

▶ Added support for data conversion of constants defined using an ApplicationValueSpecification in the context of an ApplicationPrimitiveDataTypes of category VALUE

▶ ASCRTE-3551 Fixed known issue: Missing parenthesis around Rte_IWriteRef definition

▶ Added partial support for serialized inter-ECU sender-receiver communication.

▶ ASCRTE-3561 Fixed known issue: The conversion of the schedule table activation offset into ticks may be imprecise

▶ ASCRTE-3547 Fixed known issue: Under some circumstances the Rte_Write API does not write the measurement buffer

▶ ASCRTE-3565 Fixed known issue: The Rte Bswmd generator removes certain information from data types and data prototypes which are enabled for measurement

Module version 6.1.102

2014-08-08

▶ ASCRTE-3416 Fixed known issue: If the callback Rte_COMCbkRxTOut_<sn> is called by Com, the Rte may not set a configured data receive error event

▶ ASCRTE-3386 Fixed known issue: If multiple events have been set for a BSW task, the Rte considers only one

▶ Added support for measurement of sender/receiver communication (intra-partition and inter-ECU), AUTOSAR-typed per-instance memory, inter-runnable variables, and parameter data prototypes

▶ ASCRTE-3355 Fixed known issue: The Rte Generator rejects configurations containing a `CompuMethod` of category `TEXTTABLE` that references a `Unit`

▶ Updated the Rte Generator to support chaining of tasks

▶ Updated the Rte Generator to optimize the allocation of receive buffers if they are not required for implicit communication

▶ Added function to disable the partition active checks

▶ Updated the Rte configuration to generate only one Rte schedule table for all partitions

▶ Updated the configuration of the Bsw task used for sending Com signals/signal groups to be time-triggered by setting the parameter `BswOsTaskPeriod`

▶ Added configurability to hook functions before/after Bsw schedulable entities are triggered. Additionally the Rte generates hook function before task termination

▶ ASCRTE-3423 Fixed known issue: Under certain conditions the software component header file does not include a required Rte header for development error checks

▶ ASCRTE-3415 Fixed known issue: Two client-server interfaces are not compatible if the server only defines a possible error with code 0 and the client defines other possible errors

▶ ASCRTE-3421 Fixed known issue: The Rte illegal invocation development error detection may use an inadequate unsigned integer type

▶ Removed generation of interrupt locks for Shared Memory IP channels if they are accessed by Com callbacks only and Com callbacks are configured as not interruptible

▶ Removed the generation of function `Rte_Det_CheckIllegalInvocation` from those partitions where the illegal invocation check is not applicable

▶ ASCRTE-3452 Fixed known issue: If inter-partition C/S communication is used with application errors, `Rte_Call`/`Rte_Result` return an undefined error code

▶ ASCRTE-3453 Fixed known issue: Under certain conditions the component data structure is not initialized

Module version 6.1.101

2014-07-14

► Added support for data conversion for primitive type data in the context of intra partition sender receiver communication and the definition of constant values in the context of application data types that are transparent to the application

► Updated verification of more elements such as AutosarDataTypes, CompuMethods and SwPointerTarget-Props if the RteGeneratorOutput is set to BSW_SCHEDULER_ONLY

► Added function to configure whether only the first notification from Com shall be taken into account for transmission acknowledgment after a `Rte_Send` or `Rte_Write` request

► ASCRTE-3371 Fixed known issue: The return value of a blocking Rte_Feedback API is wrong when either a shutdown notification or a timeout occurred

► Updated the Rte Editor to correctly set the parameter `RteNvmRomBlockLocationSymbol` if the rom block location is empty

► ASCRTE-3374 Fixed known issue: If a component supports multiple instantiation the Rte implements only the Rte_Enter/Rte_Exit APIs of the first instance correctly

► ASCRTE-3379 Fixed known issue: If an array data element is used in queued sender receiver communication the Rte uses an incorrect type for the queue implementation

► ASCRTE-3156 Fixed known issue: The Rte generates non-compileable code if the short name of the BSW module description contains an underscore

► ASCRTE-3376 Fixed known issue: If the short name of certain elements are not unique the internal IDs for development error tracing may be duplicated

► ASCRTE-3387 Fixed known issue: If Rte_Stop is called the Rte may terminate a task that executes Bsw schedulable entities

► ASCRTE-3386 Fixed known issue: If multiple events have been set for a BSW task, the Rte considers only one

► ASCRTE-3381 Fixed known issue: The Rte may generate shared buffer groups incorrectly if a shared receive buffer must provide a status field

Module version 6.1.100
2014-04-24

► ASCRTE-3244 Fixed known issue: The Rte may generate mode disabling dependencies for mode switch events even if they are not specified in the software component description

► ASCRTE-3282 Fixed known issue: The `Rte_Read` API may return the wrong initial value if multiple receive ports are connected to the same provide port and different initial values are defined at the receive ports

► ASCRTE-3285 Fixed known issue: The Rte Verifier does not report an error if two variable data prototypes with equal names but incompatible data types are connected

► ASCRTE-3288 Fixed known issue: The Rte Verifier does not check the compatibility of connected variable data prototypes if the categories of the data types are different

▶ ASCRTE-3284 Fixed known issue: The Rte may generate incorrect code if an implementation data type of category `TYPE_REFERENCE` is used to define an invalid value in context of a sender/receiver connection

▶ ASCRTE-3281 Fixed known issue: If an union type is used, the Rte Generator produces code that cannot be compiled

▶ ASCRTE-3307 Fixed known issue: No data consistency mechanism is applied to a variable data prototype with a complex data type that is sent inter-partition inter-ECU

Module version 6.1.57

2014-03-24

▶ ASCRTE-3217 Fixed known issue: The `Rte_Read`/`Rte_IRead` API returns the invalid value if the invalidation policy is set to `KEEP` and the data element is invalidated

▶ ASCRTE-3296 Fixed known issue: The Rte does not send structure data to an intra-partition connected receiver if the names of the struct members are different on sender and receiver side

▶ Added support for reporting of development errors to the DET

▶ ASCRTE-3250 Fixed known issue: The Rte may not execute time triggered runnable entities if several timing events with both equal and different periods and mode disabling dependencies are mapped to the same task

▶ Removed the query of the current task ID in `Rte_Call` and `Rte_Result` APIs at runtime if the client task is well-known at code generation

Module version 6.1.56

2014-02-14

▶ ASCRTE-3211 Fixed known issue: The Rte Generator does not consider mappings from calibration parameters to per instance memories if an `arTypedPerInstanceMemory` is used

▶ ASCRTE-3200 Fixed known issue: If the timeout of a waitpoint for a data received event is zero, the Rte_Receive API performs timeout monitoring with zero ticks.

▶ Changed the parameter names for the runnable function signature, if runnable entity or service arguments are specified.

Module version 6.1.55

2014-01-21

▶ Updated the event type (e.g. `BswTimingEvent`) to be shown in a separate table column on the Rte Editor's **Event Mapping** tab

▶ Implemented error report in Rte Verifier if multiple client server operations trigger the same runnable entity but define incompatible possible errors

▶ ASCRTE-2513 Fixed known issue: If two data received events referencing the same variable data prototype are mapped to different tasks, the Rte only considers one

▶ ASCRTE-3161 Fixed known issue: The Verifier wrongly reports an error if a server port of a composition is not implemented by a runnable entity

▶ ASCRTE-3177 Fixed known issue: If synchronous intra-partition client/server-communication is used and a server runnable of category 2 is mapped to a task, the client may receive a wrong result

▶ Added missing trace hook functions (e.g. for `Rte_IsUpdated`) and changed trace hook signature of `Rte_IRead`

▶ Updated compiler abstraction macros

▶ ASCRTE-3168 Fixed known issue: If the Rte editor is opened and the elements mode declaration, mode declaration group or mode declaration group prototype are not complete, an exception is reported

Module version 6.1.54

2013-11-15

▶ ASCRTE-3099 Fixed known issue: If a runnable entity implicitly sends a variable data prototype with invalidation policy set to keep or replace in case of inter-partition inter-ECU communication, then the Rte Generator fails

▶ ASCRTE-2843 Fixed known issue: If a runnable entity with a blocking Rte API function is not mapped to a task, the Rte Generator crashes with an exception

▶ ASCRTE-2801 Fixed known issue: If an implementation data type of category structure contains an implementation data type element referencing its parent implementation data type the Rte Generator causes a stack overflow error

▶ ASCRTE-3106 Fixed known issue: An executable entity is never executed at runtime if it is triggered by a timing event that has an activation offset equal to its period which in turn is a multiple of the cycle length of the task to which it is mapped to

▶ Improved functionality that starting a schedule table with relative activation mechanism no longer fails if the configured schedule table offset is shorter than one Os counter tick. The minimal offset value `1` is used in this case

▶ ASCRTE-2947 Fixed known issue: The Rte does not support sender receiver to signal group mappings for array/record application data types

▶ Fixed known issue: Violations against the AUTOSAR meta model might be reported twice or elements are verified that are not used by the Rte generator.

▶ Updated the table columns **Period** and **Offset** on the Rte Editor's **Event mapping** tab to use numerical instead of alphabetical sort algorithm

Module version 6.1.53

2013-10-22

▶ Updated the Rte Verifier to allow client/server application errors with `errorCode 0`. If the short name of the application error is not `E_OK`, though a warning is reported

▶ Updated the Rte Verifier to allow operation-invoked events triggering the same runnable which have array typed arguments of different length (see TPS_SWCT_1125)

▶ Changed the Rte types header file to use the compiler abstraction macros

▶ ASCRTE-3049 Fixed known issue: The Rte Generator shares internal buffers for non-queued implicit inter-partition sender/receiver communication across partition boundaries which leads to non-compilable code

▶ ASCRTE-3053 Fixed known issue: Mode-disabling dependencies are not working if more than eight mode-disabling dependencies for different events are specified for an inter-partition mode machine instance

Module version 6.1.52

2013-09-18

▶ ASCRTE-2963 Fixed known issue: If the enhanced `Rte_Mode` API is configured for a software component which supports multiple instantiation or uses the indirect API, the generated Rte code does not compile

▶ ASCRTE-2931 Fixed known issue: The Rte editor removes connections between two service components

▶ ASCRTE-2920 Fixed known issue: The Rte Editor does not group elements of `requiredModeGroup` correctly in the BSW Mode Mapping window

▶ ASCRTE-2944 Fixed known issue: If `TS_IntDisableEnable` is configured as the interrupt blocking function in a shared memory inter-partition environment, the generated code does not compile

▶ ASCRTE-2934 Fixed known issue: If only the Bsw scheduler is generated and the Os parameter `OsSecondsPerTick` is not configured, then the Rte fails with an `ArithmeticException`

▶ ASCRTE-2976 Fixed known issue: The Rte Generator fails if an inter runnable variable is typed by an application data type which is not mapped to a corresponding implementation data type

▶ ASCRTE-3004 Fixed known issue: The Rte generates the API infix in upper case letters in memory mapping and compiler abstraction macros

▶ Updated the Rte Generator to support the category `EXPLICIT_ORDER` for mode declaration groups

Module version 6.1.51

2013-08-20

▶ ASCRTE-2921 Fixed known issue: If only on-transition runnables are configured for one mode machine instance and no mode disabling dependencies are configured, the generated Rte code does not compile

► ASCRTE-2911 Fixed known issue: If data mappings are distributed among different system mappings, the Rte Editor does not save the data mapping correctly

► ASCRTE-2988 Fixed known issue: If timing events have a huge period, the Rte configures an empty schedule table in the Os ECU configuration

► ASCRTE-2914 Fixed known issue: If a system signal is mapped to a data element, but no Com signal references this system signal, the Rte Generator fails

► ASCRTE-2932 Fixed known issue: The **Data Mapping** tab of the Rte Editor does not show application record elements of category `BOOLEAN`

► ASCRTE-2940 Fixed known issue: The generated `Rte_IrvRead` and `Rte_IrvWrite` APIs ignore the given parameter if the inter-runnable variable is of complex type and the APIs are implemented as a C-macro

► Updated the Rte Generator to support `Rte_IsUpdated` for inter-partition and inter-partition/inter-ECU communication

► The Rte Verifier now reports an error if not all mode users of a provided mode declaration group are on the same partition

► ASCRTE-2938 Fixed known issue: The Rte Generator does not generate empty module interlink header files

Module version 6.1.50

2013-06-25

► ASCRTE-2824 Fixed known issue: In task mapping scenario B2, the generated Rte code might not contain all events required by `Rte_WaitGetClearEvent()`

► ASCRTE-2697 Fixed known issue: Initial values for data elements of array types are not written to the Com configuration via the service needs assistant

► ASCRTE-2860 Fixed known issue: The generated Rte may not compile if the value of a text value specification contains a number of characters that is greater or equal to the array size of the underlying data type

► ASCRTE-2814 Fixed known issue: The Rte Editor reports an error if data prototype names collide with Rte Editor internal short names

► Updated the generated template code in the folder `src_appl` to use a pointer to the array base type in case of array data types

Module version 6.1.13

2013-05-17

► Updated the Rte editor to ignore a local variable access which does not refer a valid local variable when opening the **VFB Tracing** tab

► ASCRTE-2815 Fixed known issue: If at least two mode switch points reference the same mode declaration group instance and one requires a blocking `Rte_SwitchAck` API while the other one is not referenced by a mode switch acknowledge event, the Rte generates no code

Module version 6.1.12

2013-04-17

► Updated the Rte Editor tab **Service Port Mapping** to allow to connect one application port to multiple service ports and vice versa

► ASCRTE-2555 Fixed known issue: `Rte_Pim()` incorrectly returns a pointer to array type instead of pointer to the array base type for a per instance memory typed by an AUTOSAR data type

► ASCRTE-2529 Fixed known issue: The code which is generated in the contract phase depends on EB-specific header files

► ASCRTE-2640 Fixed known issue: The Rte does not generate any data filters if the underlying C-type is a char type

► Updated the Rte Verifier to report a warning if the resolution of the parameter `OsSecondsPerTick` of the used Os counter is higher than nano-seconds

► Updated the Rte Verifier to report an error if a configured data filter lacks required attributes, e.g. the `MIN` and `MAX` attributes

► ASCRTE-2672 Fixed known issue: The Rte generates incorrect code in case of intra-partition mode switch procedure

► ASCRTE-2450 Fixed known issue: If inter-partition mode management with IOC is used and no mode switch event is specified, the mode disabling dependencies are not set by the `Rte_Switch` API

► Updated the Rte mode defines to generate even if the corresponding mode declaration group is not used, but an included mode declaration group set is referencing the mode declaration group

► ASCRTE-2672 Fixed known issue: The Rte generates incorrect code in case of intra-partition mode switch procedure

► ASCRTE-2675 Fixed known issue: If a mode-disabling dependency references a mode declaration that is not part of the referenced mode declaration group prototype, the Rte Generator aborts with an exception

► ASCRTE-2685 Fixed known issue: If multiple clients are connected to the same server and an unconnected port triggers the same server runnable or the client operations consist of different argument, the Rte might report a null pointer exception or the generated code does not compile

► ASCRTE-2699 Fixed known issue: If the Rte Generator is executed multiple times without restarting EB tresos Studio, the Rte generates incorrect mode-disabling dependency macros

► ASCRTE-2745 Fixed known issue: The enhanced `Rte_Mode` API function does not set the given previous and next mode parameters if function elision is configured

▶ Updated the Rte to support the execution of runnable entities and BSW schedulable entities on the transition of two modes

▶ Added the Rte Editor and the plugin `SwcUtils` to the Rte. The release notes for both plugins are now within the release notes of the Rte

▶ Updated the combo boxes **Bsw Os application** and **BSW Os task** to always show the values of the Rte configuration after opening the Rte Editor page **Partitioning**.

▶ ASCRTE-2727 Fixed known issue: The Bsw mode machine instance is not correctly initialized and `Rte_-Start()` overwrites the state of the mode machine instance

▶ ASCRTE-2733 Fixed known issue: The Rte does not generate enumeration macros for application data types that are referenced by an included data type set

Module version 6.1.11
2013-02-08

▶ Updated the system description verifier to check for a proper Bsw mode management configuration as described by constr_4022, constr_4059, and constr_4063

▶ Added support of the BSW Scheduler name prefix

▶ ASCRTE-2523 Fixed known issue: If an initial value is specified for an implementation data type that references a structure, the Rte Generator wrongly reports an error

▶ Added support of background events

▶ Added support of Rte mode switch acknowledgment

▶ ASCRTEAS-431 Fixed known issue: The Rte Editor does not show elements of `SwcServiceDependency` which are mapped to per-instance memory

▶ ASCSWCUTILS-191 Fixed known issue: If an implementation data type of category `value` does not specify a base type, the Rte Generator reports a null pointer exception

▶ Added support of the enhanced mode API for inter-partition mode management

▶ ASCRTE-2478 Fixed known issue: If function elision is active and the port for a specific API function is unconnected, the Rte generates the return value `RTE_E_OK`

Module version 6.1.10
2012-12-14

▶ Updated the Rte to use the pointer to array type passing scheme by default if the configuration parameter `ASR32RteWrapper` is set to true,

▶ ASCRTE-2458 Fixed known issue: If the enhanced Mode API is used and the corresponding port is unconnected, the Rte Generator does not generate any code

▶ ASCRTE-2306 Fixed known issue: The Rte Generator may produce duplicate code blocks for reading a signal group from COM within a reception callback

▶ ASCRTE-2452 Fixed known issue: The Rte Generator does not generate any code if a per instance memory is typed by an application data type

▶ Updated the Rte Generator to support elements of `InterRunnableVariable` with complex data types

▶ Added generation of enumeration constants for referenced application primitive data types to the application types header file

▶ ASCRTE-2479 Fixed known issue: The Rte cannot allocate more than one Os schedule table expiry point per millisecond

Module version 6.1.9

2012-11-19

▶ ASCRTEAS-435 Fixed known issue: Rte Editor removes configured BSW Module instances from Rte configuration under certain circumstances

▶ Updated the editor fields **BSW Os task**, **BSW send signal queue length**, and **BSW send signal group queue length** on the **Partitioning** tab to be properly enabled depending on the check box value **BSW Os task required**

▶ ASCSWCUTILS-175 Fixed known issue: The compatibility check of two elements of `Implementation-DataType` or `ImplementationDataTypeElement` of category `ARRAY` is incorrect

▶ ASCRTE-2172 Fixed known issue: The Rte Generator generates non-compilable code when a local parameter is mapped to per-instance memory, but no parameter access is defined

▶ ASCRTEAS-439 Fixed known issue: If the Rte configuration is saved and the **Measurement and Calibration** tab was not touched, the calibration support configuration for parameter software component types is removed

▶ ASCRTE-2441 Fixed known issue: If partitioning is enabled, the Rte does not generate invocations of executable entities mapped to the BSW Os task

▶ ASCRTE-2442 Fixed known issue: If basic software (BSW) events are mapped to an OS task that is not part of the BSW partition, the OS task implementation is added to the BSW partition

▶ Updated the Rte Generator to support data element invalidation if the initial value equals the invalid value

▶ Updated enumeration constants for elements of `CompuMethod` to be generated as defined by AUTOSAR 4.0.3

Module version 6.1.8

2012-10-15

► ASCSWCUTILS-167 Fixed known issue: If it is checked whether a variable data prototype is queued or not, the data type map should not be considered if the variable data prototype references an implementation data type

► ASCRTE-2408 Fixed known issue: The behavior of the Rte Generator is undefined when invalid `SwBase-Type` native declarations are present in the data model

► ASCRTE-2386 Fixed known issue: The system model is not validated if the Rte Contract Phase Generator is executed via the command line

► Updated the system description verifier to check if the native declaration of a `SwBaseType` contains a valid C-data type

► Updated the Rte Generator to support the Basic Software Scheduler Mode Management as defined by AUTOSAR 4.0

► Updated the Rte Editor to support connecting of BSW required triggers with BSW released triggers

► ASCRTE-2412 Fixed known issue: Partial record support does not function in combination with system signal mappings

► Updated macro definitions for range data types, enumeration constants and invalid values according to AUTOSAR 4.0.3

► Fixed several issues with the component template generator

► Updated the Rte Editor to support mapping of provided BSW mode declaration group prototypes to required BSW mode declaration group prototypes

► Updated the Rte Generator to verify the Bsw Scheduler ECU configuration

► ASCRTEAS-425 Fixed known issue: The **Service Port Mapping** tab removes all port interface mappings if a service connector is modified

► Updated the Rte Generator to support the enhanced mode API for intra-partition mode management

Module version 6.1.7
2012-09-18

► ASCRTE-2387 Fixed known issue: If an inter-runnable variable is typed by an application data type which does not have a mapped implementation data type, the Rte Generator will prematurely terminate without errors

► ASCRTE-2363 Fixed known issue: The system signal mapping is not working when a port interface mapping is specified

► ASCRTE-2313 Fixed known issue: The Rte Generator may produce code which causes compiler warnings like *Rte_IsModeDisablingDepSet_X defined but not used*

► Updated the Bsw Scheduler to support the `SchM_ActMainFunction` and `SchM_Trigger` API functions

Module version 6.1.6

2012-08-17

▶ Updated the Bsw Scheduler to support Bsw called entities and Bsw interrupt entities

▶ Updated the names of the preprocessor defines for the Com Handle IDs according to AUTOSAR 4.0 Rev 3 naming scheme

▶ Updated the system description verifier to check Bsw schedulable entities, Bsw called entities and Bsw interrupt entities

▶ ASCRTE-2365 Fixed known issue: The Rte Generator may incorrectly initialize Rte data structures when the initial value is greater than `0x7FFFFFFF`

▶ ASCRTE-2368 Fixed known issue: The Rte Generator throws a null pointer exception when a software base type is incorrectly defined

▶ Improved the evaluation of task mapping for asynchronous client/server calling chains

▶ Updated the system description verifier to report an error if two different interfaces with the same short name have incompatible application errors or if they are referenced by a software component which supports multiple instantiation or the indirect API attribute is set to true for at least one port

Module version 6.1.5

2012-07-16

▶ Removed configuration switch for the AUTOSAR 3.1 SchM Exclusive Area API support

▶ ASCRTE-2154 Fixed known issue: Inter-partition mode disabling dependencies do not work with non-timing events

▶ Updated the Bsw Scheduler to generate an AUTOSAR 3.1 SchM Exclusive Area API wrapper for individual Bsw module instances

▶ Updated the Rte Editor to disable exclusive areas with the implementation mechanism option *Disabled Exclusive Area*

▶ Removed the AUTOSAR 2.1 to AUTOSAR 3.1 Rte module transformer from the Rte add-on

▶ ASCRTE-2341 Fixed known issue: Wrong registration of transformer causes error log entry in Studio when using module upgrade functionality

Module version 6.1.4

2012-06-21

▶ ASCRTEAS-411 Fixed known issue: The Rte Editor does not report any errors if the BSWMD is invalid

▶ ASCRTE-2332 Fixed known issue: Warnings from the Rte Generator are not reported to the user

▶ Updated the Rte Generator to support data mappings to system signals

▶ Updated the Rte to support the usage of partial records (signal degradation) for sender-receiver communication as specified by AUTOSAR 3.1

▶ Updated the Rte editor to support the mapping of system signals and system signal groups to data elements

▶ ASCRTE-2322 Fixed known issue: If a runnable entity makes an inter-partition server call and the runnable is not mapped to a task, the Rte Generator reports an error even if a SWC to BSW mapping exists for that runnable

▶ Updated the Rte Generator to provide the generation of the AUTOSAR 3.2 Rte Wrapper to support AUTOSAR 3.2 software components

▶ Updated the Rte Editor to offer a configuration switch to enable or disable the AUTOSAR 3.2 Rte Wrapper feature

▶ Removed legacy support of the ASR 3.1 SchM Exclusive Area API

Module version 6.1.3

2012-05-16

▶ ASCRTE-2288 Fixed known issue: The generated Rte will not compile if a basic software module description name and a software component type name are equal

▶ Updated the Rte Generator to support the generation of empty `Rte_Start`/`Rte_Stop` functions if the configuration switch **Generate empty Rte_Start/Rte_Stop stubs** is enabled

▶ Updated the Rte Editor to offer a configuration switch to enable or disable the generation of an empty `Rte_Start`/`Rte_Stop function`

▶ ASCRTE-2297 Fixed known issue: The Rte Generator may produce duplicate code blocks for implicit reads and writes within the same task

▶ Updated the system description verifier to handle partial records and subelement mappings

▶ ASCRTE-2307 Fixed known issue: The Rte Generator may not consider the task position of timing events mapped to the same task

Module version 6.1.2

2012-04-23

▶ ASCRTEAS-398 Fixed known issue: The configuration of the partitioning support may not be possible, even if the partitioning support is enabled

▶ ASCRTEAS-399 Fixed known issue: The drop-down list in the tables of the **Implementation Selection** and **Partitioning** tabs show the wrong list of items

▶ ASCSWCUTILS-116 Fixed known issue: The system description verifier does not generate an error when the offset and the period of a `ONEEVERYN` data filter are equal

▶ ASCRTE-2192 Fixed known issue: The Rte Generator will generate a non-blocking `Rte_Result` API function, if an asynchronous server call result point is defined, but no corresponding asynchronous server call returns event exists

▶ ASCRTE-2154 Fixed known issue: Inter-partition mode disabling dependencies do not function with non-timing events

▶ ASCRTE-2215 Fixed known issue: The Rte Generator will produce a wrong type definition for an implementation data type of category `STRUCTURE` that contains an implementation data type element of category `ARRAY`

▶ Changed implementation data type elements of category `ARRAY` with the basetype `uint8` to be grouped to byte arrays

▶ ASCRTE-2287 Fixed known issue: The memory mapping in the Module Interlink header files may cause linker errors

Module version 6.1.1

2012-03-23

▶ ASCRTE-2178 Fixed known issue: If a data write access is specified for a variable data prototype which is not of category `ARRAY`, the resulting `Rte_IWriteRef` API will not return a pointer to the variable

▶ ASCRTE-2009 Fixed known issue: The Rte Generator may produce code which causes compiler warnings like *a value of type x cannot be used to initialize an entity of type y*

▶ ASCRTE-2189 Fixed known issue: The Rte Generator may generate code that will not compile due to unresolved symbols

▶ ASCRTE-2102 Fixed known issue: If a timeout is specified for an asynchronous server call point and no wait point is defined for the corresponding asynchronous server call result point, then the Rte Generator will produce non-compilable code

▶ Updated the Rte Generator to support the generation of the legacy AUTOSAR 3.1 SchM Exclusive Area API as function-like macros

▶ ASCRTE-2193 Fixed known issue: The generated Rte code will produce a compile error, if a Com signal group is mapped to a variable data element at a require port and the data can be directly read from Com

▶ Updated the Rte Editor to offer a configuration switch for enabling or disabling the AUTOSAR 3.1 SchM Exclusive Area API support

▶ ASCRTE-2196 Fixed known issue: The Rte Generator may produce a `NullPointerException`, if two tasks from different partitions hold runnable entities that specify a timeout for their `Rte_Call`, `Rte_Feedback`, `Rte_Receive` or `Rte_Result` API

▶ ASCRTE-2200 Fixed known issue: The `Rte_Start()` API will not correctly initialize the Rte on a multi-core system

▶ ASCRTE-2177 Fixed known issue: The Rte Contract Phase will fail without any error if an application data type is not mapped to an implementation data type

▶ Reception signal timeout callbacks no longer read Com signal values when the attribute *replace by initial value* is not configured for a Com signal

▶ Updated the Rte Editor to provide the feature to auto-map all Bsw timing events to a task

▶ Updated the Rte Generator to support the `SwcBswMapping` to determine the calling context for Bsw schedulable entities

▶ ASCSWCUTILS-130 Fixed known issue: The merge of the ECU resource properties fails with an error during the Rte generation or when running the Service Needs Assistant

▶ ASCRTE-2206 Fixed known issue: The generated Rte code will not compile if two or more runnable entities are mapped to different tasks and each of these has a synchronous server call point for the same operation on the same port

▶ ASCRTE-2197 Fixed known issue: Within the Rte default schedule table, the Rte Generator will not allocate any expiry point, which activates an Os task if there already exists an expiry point activating any other Os task with the same period and offset

▶ Updated the Rte Generator to support the generation of the BSW scheduler and its API functions as defined by AUTOSAR 4.0

▶ Implemented generic BSW scheduler verifier

▶ Updated the Rte Editor to support configuring the BSW scheduler specific parts in the Rte

Module version 6.1.0

2012-02-17

▶ ASCRTE-2113 Fixed known issue: If a client port is connected to a server port, which belongs to the same software component instance, the generated Rte code will produce compiler warnings

▶ Updated the system description verifier to consider the port interface mapping if the compatibility between two interfaces is checked

▶ Updated the Rte Generator to support the port interface mapping as defined by AUTOSAR 4.0

▶ ASCRTE-2173 Fixed known issue: `Rte_NvMData.h` can cause compile errors due to the duplicate definition for the same ROM default value variable

▶ Added support for the `ImplementationDataType` attribute `typeEmitter`

▶ Removed support of the Rte Editor for the generation of the service component description via the **Service Port Mapping** tab.

Module version 6.0.12

2012-01-20

► ASCSWCUTILS-107 Fixed known issue: If a delegation connector connects ports of different types, the system description verifier will stop the verification of the system model and report a class cast exception

► ASCRTE-2013 Fixed known issue: If inter-partition sender/receiver communication is used with data filters configured with the data filter algorithm `NEVER`, the Rte Generator produces code which causes compiler warnings

► ASCRTE-2115 Fixed known issue: The Rte Generator will report an error for application data types that refer to a valid application value specification

► ASCRTE-2121 Fixed known issue: The Rte will not update the mode of a mode machine instance after the execution of a related mode switch event

► ASCRTE-2150 Fixed known issue: The Rte Generator will produce compiler errors like *parse error before '}' token* if a runnable entity defines a server call or server call result point for an operation, for which no operation invoked event exists on server side

► Updated the system description verifier to check `CompuMethod` elements of the category `TEXTTABLE` according to the software component template constraint constr_1134

► Implemented partitioning support in the variant *Shared Memory*

Module version 6.0.11

2011-12-09

► Added additional consistency checks

► ASCRTE-1193 Fixed known issue: An `Rte_Write`, `Rte_Feedback` or `Rte_Invalidate` might behave as if the port was not connected

► ASCRTE-2088 Fixed known issue: When a DBC, FIBEX or LDF file is imported with the option *Enable system model import* enabled, the Rte Generator will fail to generate the Rte

► Added generic Rte Verifier

Module version 6.0.10

2011-11-11

► ASCRTE-2007 Fixed known issue: The Rte Generator produces non-compilable code when inter-ECU invalidation shall be handled by the Rte and the invalid value of a COM signal which shall be invalidated is specified in binary or array format

► ASCRTE-2020 Fixed known issue: If there are `CompuMethod` elements with category `TEXTTABLE` where at least one `CompuScale` defines no upper or lower limit, the Rte Generator will produce a null pointer exception

► ASCRTE-2018 Fixed known issue: If inter-partition client/server communication is used, the API function `Rte_Result()` will not return

▶ ASCRTE-2031 Fixed known issue: If optimization function elidation is enabled, the Rte will transmit a primitive data element only within its own partition

▶ ASCRTE-2008 Fixed known issue: When the optimization *use bit fields for internal buffers* is enabled, the Rte Generator may produce non-compilable code

▶ ASCSWCUTILS-91 Fixed known issue: The system description verifier fails to report an error for incorrect instance references, which refer to an entity that is not part of the enclosing software component type

▶ Updated the Rte Generator to support the update flag feature defined by AUTOSAR 4.0

▶ Updated the Rte Generator to support the Rte_DRead API function as defined by AUTOSAR 4.0

▶ Changed the value of the error code RTE_E_SHUTDOWN_NOTIFICATION

▶ Updated the Rte Generator to support the never received status for sender/receiver communication

▶ ASCRTE-2063 Fixed known issue: The Rte Generator may produce code that causes compiler warnings if USE-VOID is used as ServerArgumentImplPolicy for a complex IN argument of an operation

▶ ASCRTEAS-340 Fixed known issue: The Rte Editor will not save any changes which have been applied to the **General** tab if the partitioning support, event mapping, or service port mapping is configured afterwards

▶ Changed the content of the API mapping and declaration in the component header file to not be re-ordered if the configuration is unchanged

Module version 6.0.9
2011-09-02

▶ Initial AUTOSAR 4.0 version

3.3.1.2. New features

▶ The Rte now supports multiple Com instances.

It is now possible to send/receive signal(group)s over multiple Com instances which are mapped to different partitions.

▶ The Bsw Scheduler now supports client/server communication.

Bsw modules can now exchange variable data over the Bsw Scheduler by using the SchM_Call and SchM_Result APIs.

▶ The Rte now supports autonomous error reaction for inter-ECU client server communication with data serialization

For client server communication it is now possible on the server side to trigger an autonomous error reaction which generates the response of the client server communication automatically without involvement of any runnable.

▶ The Rte now supports the asynchronous mode switch

It is now possible to configure the asynchronous mode switch communication. If all intra-partition require mode declaration group instances support the asynchronous mode switch behavior and if the mode switch events are either mapped to no task or if all of them are mapped to the task of the `Rte_Switch`, then the invocation of runnable entities will be made via direct function call.

▶ The Rte now supports intra-ECU sender-receiver communication with data transformation

For intra-ECU sender-receiver communication it is now possible to specify a DataTranformation between two interfaces by using a DataPrototypeMapping

▶ The Rte now supports the dirty flag mechanism for NvBlockSwComponentTypes

It is now possible to enable the `dirtyFlagSupport` for a `NvBlockDescriptor`. Different writing strategies like `storeImmediate`, `storeCyclic` and `storeAtShutdown` can be configured in the `NvBlock-Needs`.

▶ The Rte now supports memory mapping initialization strategy

For inter runnable variables, NvRam blocks, per instance memory, receive buffers and Smc buffers it is now possible to specify an initialization strategy mapped to a policy described by a SwAddrMethod corresponding to the variable data prototypes.

▶ The Rte now supports the external replacement invalidation for sender-receiver communication according to AUTOSAR 4.3.0.

For sender-receiver communication it is now possible on reception side to specify another variable data prototype to be used for replacement.

▶ The Rte now supports the feature to provide the activating event of an ExecutableEntity according to AUTOSAR 4.3.1.

It is now possible to configure an `ActivationReason` for an `RTE/BSW Event`. When configured, `RTE` shall pass an additional argument `activation` to the `ExecutableEntity`.

▶ The Rte now supports debounced activation of executables.

▶ The Rte now supports compound primitives

It is now possible to use compound primitives as long as their implementation is an array.

▶ The Rte now supports InitEvents triggering runnable entities via OS task according to AUTOSAR 4.3.1.

It is now possible to configure `InitEvents` triggering runnable entities via `OS task` for initialization purposes, i.e. for starting and restarting a partition.

▶ The Rte now supports Transport Protocol for Serialization for LdCom.

▶ The Rte now supports the implementation of a basic task for tasks shared by Rte and Bsw timing events.

It is now possible to enable a `OneScheduleTablePerPartition` option to implement a basic task for tasks shared by Rte and Bsw timing events.

▶ The Rte now supports mapping of external trigger events for category 1 executables to category 2 ISRs.

▶ The Rte now supports RTE Implementation Plugins (RIPS) for explicit/implicit sender receiver communication.

▶ The Rte now supports InitEvents triggering runnable entities via RteInitializationRunnableBatch according to AUTOSAR 4.3.1.

It is now possible to configure `InitEvents` triggering runnable entities for initialization purposes via `RteInitializationRunnableBatch`. i.e. for starting and restarting a partition.

▶ The Rte now supports the generation of human readable global variable names instead of cryptic ones

It is now possible to set the new configuration parameter `HumanReadableBufferNames` to true to let the Rte generator use a human readable naming approach for the generated global variables.

▶ The Rte now supports timeout monitoring for asynchronous client/server communication with non-blocking result calls.

▶ The Rte now supports partitioning for communication timeout in case of inter-partition inter-ecu communication

▶ The Rte now supports multiple mode user partitions.

It is now possible to configure a mode manager connected to multiple mode user which are mapped to different partitions which in turn are scheduled on different micro-controller cores.

▶ The Rte now supports intra-ECU sender-receiver communication with SubElementMappings to primitive DataPrototypes.

It is now possible to map a structure member on a sender side to a primitive data element on the receiver side.

▶ The Rte now supports the usage of LdCom without transformers.

▶ The Rte now supports mapping of timing events for category 1 executables to category 2 ISRs.

It is now possible to map a timing event or a BSW timing event to an ISR.

▶ The Rte now supports transmission acknowledgment for implicit sender receiver communication.

The information whether a value has been successfully passed to the communication infrastructure can be known by using the Rte_IFeedback API.

▶ The Rte now supports synchronous mode switching procedure over multiple partitions/cores.

It is now possible to have the mode switching procedure steps synchronized over multiple partitions/cores.

▶ The Rte now Supports mapping of variable data prototypes to parameter data prototypes.

► The Rte now supports the access to a reference to the VariableDataPrototype referenced by VariableAccess in the dataReadAccess role.

This can be done by using the Rte_IReadRef API.

► The Rte now supports retrieving partition context for Com on a per-PDU basis.

► The Rte now supports retrieving partition context for LdCom on a per-PDU basis.

► The Rte now supports graphical visualization for exclusive area optimization.

► The Rte now supports the improvement of start-up and initialization of NvBlockSwComponents. Restore default data if both ROM and an initialization callback are configured.

► The Rte now supports the access to a reference to the VariableDataPrototype defined with the implicitInterRunnableVariable role.

This can be done by using the Rte_IrvIWriteRef API.

► The Rte now supports the transmission preparation callback (Rte_COMCbkTxPrep_mn) for inter-partition inter-ECU communication.

► The Rte now supports the generation of review instructions to easily identify memory writes access for freedom from interference analyses (released).

The configuration parameter `GenerateRteFreedomFromInterferenceReviewInstructions` can be used to generate review instructions and an html report to easily detect where a write access occurs.

3.3.1.3. EB-specific enhancements

This chapter lists the enhancements provided by the module.

► On demand, you are able to add the OS objects required by the Rte automatically to the OS configuration using the Service Needs Assistant.

► On demand, the Service Needs Assistant automatically configures the required Com callbacks in the Com configuration.

► The Rte provides an additional configuration option *OSEK OS compatibility mode*. This option enables the usage of the Rte in connection with an operating system which is not compliant to AUTOSAR, but only compliant to OSEK OS.

► When applying data consistency mechanisms (Os resources, interrupt locking etc.), the Rte Generator considers whether the hardware can make an atomic access. In this case, no data consistency mechanism is applied.

► You are able to configure the data consistency mechanism which the Rte Generator shall apply if a data consistency mechanism is required. You can choose between interrupt locking and the usage of Os resources.

▶ You are able to configure the interrupt blocking function which the Rte Generator shall use. The following options are provided: `SuspendResumeAllInterrupts`, `DisableEnableAllInterrupts`, fast interrupt locking (EB-specific) or you can use a user-specific interrupt locking function.

▶ You can configure whether the Com callbacks are interruptible or not. If they are not interruptible, the Rte Generator does not need to lock interrupts at all in the Com callbacks.

▶ When allocating buffers for data elements, the Rte Generator shares buffers if possible to reduce the RAM consumption.

▶ The Rte Generator supports function elidation. If function elidation is enabled, several API functions are realized as macros instead of functions.

▶ The Rte Generator provides an option to directly read data from the Com module if possible. If this option is switched on, the Rte Generator does not allocate an additional receive buffer. This reduces the RAM consumption. You may disable this option if your application reads the data element more frequently than the mapped signal is updated by Com.

▶ The Rte supports additional trace hooks for the implicit `IRead` and `IWrite` API.

▶ You can use the component-specific memory mapping if you want to map the code of your software components to other memory sections than the default memory section `RTE_APPL_CODE`.

▶ In addition to the specified return values, the blocking Rte API calls return `RTE_E_SHUTDOWN_NOTIFICATION` if the Rte received a shutdown notification:

  ▶ `Rte_Receive`

  ▶ `Rte_Feedback`

  ▶ `Rte_Call` (synchronous invocation)

  ▶ `Rte_Result`

  This happens while the Rte waits for the wait point or synchronous server call point to be resolved.

▶ You can configure whether the Rte shall handle inter-ECU signal invalidation or not. If the Rte shall not handle inter-ECU invalidation, it is handled by the Com as defined in the specification.

▶ The Rte Generator now supports the partitioning of software components. Software components and their runnable entities can be mapped to different partitions. The Rte realizes the communication between software components mapped to different partitions using the Inter OS Application Communicator (IOC) which is part of the Os or the Shared Memory Communicator (SMC) which is part of the Rte.

  The partitioning support enables the usage of the Rte in multi-core and memory-protected systems.

▶ It is possible to have up to 256 clients per operation in a client/server communication.

  The Rte Generator allows to connect 256 clients with the same server, i.e. operation.

▶ The Rte Generator used different Os events for timing events, that were mapped to the same task, had the same period, but different mode disabling dependencies.

Now, the Rte Generator uses only one Os event for the timing events of a task, if they have the same period, but different mode disabling dependencies.

▶ The Rte supports the usage of partial records (signal degradation) for sender/receiver communication. This increases the flexibility of a sender-receiver interface by allowing new elements to be added to a record, while still being compatible. See Bugzilla issue http://www.autosar.org/bugzilla/show_bug.cgi?id=44863.

▶ It is possible to have multiple wait points that reference the same data send completed event or mode switched ack event. This implies that runnable entities mapped to different tasks can now call a blocking `Rte_SwitchAck` or `Rte_Feedback` at the same time.

▶ The Rte generator now supports `Rte_IsUpdated` for inter-partition and inter-partition/inter-ECU communication.

▶ The Rte provides an additional configuration parameter `SingleScheduleTablePartitionRef`. If this parameter is set and multiple partitions are used, only one schedule table will be generated for the referenced Os Application. Therefore a call to `Rte_Start`/`Rte_Stop()`/ `Rte_Restart()` in one of other partitions has no effect on the lifetime of the schedule table.

▶ The Rte provides an additional configuration parameter `DisablePartitionActiveChecks`. If set to true, the Rte will not fulfill requirements rte_sws_2538, rte_sws_2535, and rte_sws_2536 anymore. This means that the Rte does not check for each API function if the current partition is active. Ensure that no callbacks and no API functions are called before `Rte_Start()`/after `Rte_Stop()` has been executed. The effect of this optimization depends on the number of generated API functions and how often they are called by the application.

▶ The Rte generates further hook functions than specified by AUTOSAR 4.0.3:

At the end of each task, the hook function `Rte_Task_EndHook(OsTask)` is called.

Before each Bsw schedulable entity call, the hook function `Rte_Schedulable_<bsnp>[_<vi>_-<ai>]_Start()` is generated.

After each Bsw schedulable entity call, the hook function `Rte_Schedulable_<bsnp>[_<vi>_<ai>]_-Return()` is generated.

The header file `SchM_<bsnp>[_<vi>_<ai>]Hook.h` contains all hook functions accessible by that basic software module.

▶ The Rte supports combined provide and require ports as described by AUTOSAR 4.1.3:

The Rte generator now supports combined provide and require ports (i.e. `PRPortPrototype`s) by means of AUTOSAR 4.1. Therefore the following additional requirements are fullfilled by the Rte verifier and generator:

Requirement / constraint	Origin	Document Version
SWS_Rte_06030	Specification of RTE	3.5.0

Requirement / constraint	Origin	Document Version
constr_1200	Software Component Template	4.5.0
constr_1202	Software Component Template	4.5.0
constr_1203	Software Component Template	4.5.0
constr_1204	Software Component Template	4.5.0
constr_1205	Software Component Template	4.5.0

Table 3.3. Introduced requirements for PRPortPrototypes

NOTE

The Rte considers an `AssemblySwConnector` between two `PRPortPrototype`s as a directed connection where the direction is specified by means of the provider and requester role. If a bi-directional connector is needed, two `AssemblySwConnector`s for each direction must be specified.

► The Rte supports the concept of Data Transformation via IF-API according to AUTOSAR version 4.2.-2. A list of supported scenarios is documented in section *Supported features* of the EB tresos AutoCore Generic 8 RTE documentation.

► The Rte supports fire-and-forget asynchronous client/server calls as specified by RfC 70295 (planned for AUTOSAR version 4.3.0). It is not required anymore to define an AsynchronousServerCallResultPoint for each AsynchronousServerCallPoint. The Rte generates no Rte_Result API if the AsynchronousServer-CallResultPoint is omitted. This enables the client to initiate a subsequent asynchronous server call without the need to first fetch the result as soon as the server processed the last request.

► The Rte supports cooperative tasks which is a list of OsTasks that cannot interrupt each other regardless of their configured priorities, schedule, application assignment, and core assignment. This list will be taken into account by the Rte generator to optimize locks and to reduce the number of implicit buffers.

► The Rte supports the dirtyFlag mechanism as described by AUTOSAR 4.3.0.

The following additional requirements are fullfilled by the Rte generator: SWS_Rte_08080, SWS_-Rte_08081, SWS_Rte_08082, SWS_Rte_08083, SWS_Rte_08084, SWS_Rte_08085, SWS_Rte_08086, SWS_Rte_08087, SWS_Rte_08088, SWS_Rte_08089, SWS_Rte_08090.

► The Rte supports the InitEvents via OS task as described by AUTOSAR 4.3.1.

The following additional requirements are fullfilled by the Rte generator: SWS_Rte_06748, SWS_Rte_-06761, SWS_Rte_06762.

► The Rte supports the configuration parameters RteServerQueueLength and RteBswServerQueueLength as specified by RfC #79150. This enables projects to specify the server queue length for BSW module client-server communications and to overrule the queueLength attribute of the ServerComSpec for Rte client-server communications. The RteServerQueueLength resp. RteBswServerQueueLength can be configured in the event-to-task-mapping configuration for the particular OperationInvokedEvent which starts the server executable.

► The Rte supports the configuration parameters RtePeriod and RteBswPeriod as suggested by RfC #80049. This enables the time triggered activation of executable(s) assigned to an RteEvent/BswEvent and disables any event triggered activation. The RtePeriod resp. RteBswPeriod can be configured in the event-to-task-mapping configuration for the particular RteEvent/BswEvent which starts the executable. Rte currently supports the time triggered activation of the following types of event:

  ► OperationInvokedEvent

  ► DataReceivedEvent

  ► DataSendCompletedEvent

► The Rte supports including additional type header files for the ImplementationDataTypes which provide an `ADMIN-DATA` container as shown below:

If the TypeEmitter of an ImplementationDataType is available and set to a value different from RTE, the Rte will evaluate the AdminData of the ImplementationDataType and include the provided header file in the RTE Types Header File.

► The Rte supports value-based data filtering for primitive implementation data types on the sender side. The `DataFilter` algorithms `ALWAYS` and `MASKED-NEW-DIFFERS-MASKED-OLD` are only supported.

Since the AUTOSAR metamodel doesn't support to describe `DataFilter` on sender side at the moment, the following `AdminData` structure has to be used in the context of the `PPortPrototype`.

```
<P-PORT-PROTOTYPE>
  <SHORT-NAME>pPort1</SHORT-NAME>
  <ADMIN-DATA>
    <SDGS>
      <SDG GID="EB:DATA-FILTER">
        <SD GID="VARIABLE-DATA-PROTOTYPE">/interfaces/if1/de1</SD>
        <SD GID="DATA-FILTER-TYPE">MASKED-NEW-DIFFERS-MASKED-OLD</SD>
        <SD GID="MASK">0b11111111</SD>
      </SDG>
      <SDG GID="EB:DATA-FILTER">
        <SD GID="VARIABLE-DATA-PROTOTYPE">/interfaces/if1/de2</SD>
        <SD GID="DATA-FILTER-TYPE">MASKED-NEW-DIFFERS-MASKED-OLD</SD>
        <SD GID="MASK">0xffff</SD>
      </SDG>
    </SDGS>
  </ADMIN-DATA>
  <PROVIDED-INTERFACE-TREF DEST="SENDER-RECEIVER-INTERFACE">/interfaces/if1</PROVIDED-INTERFACE-
</P-PORT-PROTOTYPE>
```

3.3.1.4. Deviations

This chapter lists the deviations of the module from the AUTOSAR standard.

► The Rte Generator does not reject the input configuration if it contains software component related information, in case the Basic Software Scheduler Generation Phase is executed

Description:

The software component related information is ignored in case the Basic Software Scheduler Generation Phase is executed. Thus it is not necessary to have the software component related information removed from the input configuration.

Requirements:

rte_sws_7585

► Several elements of the Rte basic software module description are not exported by the Rte Generator

Requirements:

rte_sws_5179, rte_sws_5180, rte_sws_5182, rte_sws_5183, rte_sws_5188, rte_sws_5189, rte_sws_-5191, rte_sws_5192, rte_sws_5090, rte_sws_6725, rte_sws_7085

► The export of MC support description is not supported (reference to product description: ASCPD-57)

Requirements:

rte_sws_5121, rte_sws_5172, rte_sws_5136, rte_sws_5137, rte_sws_5138, rte_sws_5139, rte_sws_-5140, rte_sws_5141, rte_sws_5142, rte_sws_5143, rte_sws_5144, rte_sws_5152, rte_sws_5153, rte_-sws_5154, rte_sws_5155, rte_sws_5156, rte_sws_5157, rte_sws_5158, rte_sws_5159, rte_sws_5160, rte_sws_5161, rte_sws_5162, rte_sws_5168, rte_sws_5176, rte_sws_5174, rte_sws_5169, rte_sws_-5175

► Variant handling is not supported (reference to product description: ASCPD-56)

Requirements:

rte_sws_6610, rte_sws_6611, rte_sws_6612, rte_sws_6613, rte_sws_6543, rte_sws_6500, rte_sws_-6546, rte_sws_6501, rte_sws_6502, rte_sws_6503, rte_sws_6504, rte_sws_6505, rte_sws_6507, rte_-sws_6547, rte_sws_6548, rte_sws_6508, rte_sws_6509, rte_sws_6510, rte_sws_6511, rte_sws_6512, rte_sws_6549, rte_sws_6550, rte_sws_6517, rte_sws_6537, rte_sws_6601, rte_sws_6602, rte_sws_-6603, rte_sws_6604, rte_sws_6605, rte_sws_6606, rte_sws_6516, rte_sws_6514, rte_sws_6513, rte_-sws_6515, rte_sws_6518, rte_sws_6519, rte_sws_6520, rte_sws_6530, rte_sws_6541, rte_sws_6542, rte_sws_6535, rte_sws_6536, rte_sws_6532, rte_sws_6539, rte_sws_6540, rte_sws_6620, rte_sws_-6638, rte_sws_6529, rte_sws_6527, rte_sws_6528, rte_sws_6521, rte_sws_6522, rte_sws_6523, rte_-sws_6524, rte_sws_6525, rte_sws_6526, rte_sws_6531, rte_sws_5103, rte_sws_5104, rte_sws_6533, rte_sws_6544, rte_sws_6545, rte_sws_6534, rte_sws_3854, rte_sws_6633, rte_sws_6634, rte_sws_-7684

► Data conversion is not supported (reference to product description: ASCPD-56)

Requirements:

rte_sws_7038, rte_sws_3831, rte_sws_3833

► The ECU abstraction components are handled in the same way as application components

Description:

There is no reason why they should be handled differently. The Rte Generator does not make any differences between an application software component type and an ECU abstraction software component type. It is up to you to not map data elements of ECU abstraction components to signals.

Requirements:

rte_sws_2051

► The basic software scheduler feature is not fully implemented

Requirements:

rte_sws_7525, rte_sws_7582, rte_sws_7583, rte_sws_7564, rte_sws_7296, rte_sws_7261

► The sharing of mode machine instances between BSW modules and software components is not supported

Requirements:

rte_sws_7533, rte_sws_7534, rte_sws_7535, rte_sws_7582, rte_sws_7583, rte_sws_7564

The following requirements are partially affected by this deviation:

rte_sws_7055

► Monitoring of runnable execution time is not supported

Requirements:

rte_sws_7800, rte_sws_7801, rte_sws_7802, rte_sws_7803

► The user can currently not provide Os objects. The Rte allocates Os alarms and schedule tables automatically

Requirements:

rte_sws_7804, rte_sws_7805, rte_sws_7806, rte_sws_7179, rte_sws_7807, rte_sws_7180, rte_sws_-7808, rte_sws_7809, rte_sws_7181, rte_sws_5082

► Immediate restart of runnable entities is not supported

Requirements:

rte_sws_7061

► Buffer sharing for implicit inter runnable variables not supported

Requirements:

rte_sws_3582, rte_sws_7022

► Several configuration checks are not supported

Requirements:

rte_sws_7044, rte_sws_7662, rte_sws_8001, rte_sws_7640, rte_sws_2500, rte_sws_3790, rte_sws_-
7045, rte_sws_7670, rte_sws_4507, rte_sws_2557, rte_sws_3764, rte_sws_7610, rte_sws_2738, rte_-
sws_7026, rte_sws_5111

► The export of measurement information for inter-partition sender/receiver communication is not imple-
mented

Requirements:

rte_sws_7344

► Support for Measurement is not implemented (reference to product description: ASCPD-53)

Requirements:

rte_sws_3901, rte_sws_3975, rte_sws_3976, rte_sws_3977, rte_sws_7349, rte_sws_3978, rte_sws_-
5101, rte_sws_3980, rte_sws_5102, rte_sws_3979, rte_sws_5170, rte_sws_6726, rte_sws_6700, rte_-
sws_6701, rte_sws_6702, rte_sws_7097, constr_1017, constr_1018, constr_2035

► The flat instance descriptor is not supported

Requirements:

rte_sws_7029, rte_sws_7030, rte_sws_7035

The following requirements are partially affected by this deviation:

rte_sws_7186, rte_sws_7185, rte_sws_7034

► Unconnected ports with parameter interfaces are not supported

Requirements:

rte_sws_2749

► Specification of RTE 4.2.1, chapter 4.2.9.3.2 NVM Interfaces. The NvBlockSwComponentType can also
have ports used for NV data management and typed by Client-Server interfaces derived from the NVRAM
Manager standardized ones.

Requirements:

rte_sws_7398, rte_sws_7399

▶ Support for callback function Rte_NvMNotifyInitBlock. Rte_NvMNotifyInitBlock does not support the server referenced by the RoleBasePortAssignment with a NvMNotifyJobFinished role.

Requirements:

rte_sws_7630

The following requirements are partially affected by this deviation:

rte_sws_7682

▶ `NvBlockSwComponentTypes` are not completely supported (reference to product description: ASCPD-58). E.g. there is no support for client server interfaces, for inter-partition connections. Also some parameters of NvBlockDescriptor are not supported, for more details see also the limitations documentation.

Requirements:

rte_sws_7357, constr_2010, constr_2011, constr_2012, constr_2014, constr_2015

The following requirements are partially affected by this deviation:

rte_sws_1330, rte_sws_7663, rte_sws_1332, rte_sws_1346, rte_sws_1347, rte_sws_3817, rte_sws_-3816, rte_sws_7682, rte_sws_7632 1

▶ Local calibration parameters in NVRAM are not supported

Requirements:

rte_sws_3936, rte_sws_3935

▶ Immediate buffer update for implicit communication is not supported

Requirements:

rte_sws_7020, rte_sws_7021, rte_sws_7064, rte_sws_7068

▶ When separate buffers are used in implicit communication, the changes are not made available to runnables in the same preemption area

Requirements:

rte_sws_7041, rte_sws_7065

▶ Inter-partition timeout monitoring is not implemented (reference to product description: ASCPD-55)

Requirements:

rte_sws_2710

▶ The configuration parameter `RteUseComShadowSignalApi` is not supported. `Com_UpdateShadowSignal` is always used per default

Requirements:

The following requirements are partially affected by this deviation:

rte_sws_4526

► Dynamic length data types are not supported (reference to product description: ASCPD-54)

Requirements:

rte_sws_7813, rte_sws_7814, rte_sws_7811, rte_sws_7812

► In case of timeout, the client can make next call before server is finished, because the old request is discarded by the Rte

Description:

This deviation eased the implementation of asynchronous client server communication and is more convenient for the user.

Requirements:

rte_sws_3771

► The EB IOC API is reentrant and the Rte does not use IOC callbacks

Requirements:

rte_sws_2737, rte_sws_2736

► Usage of trusted functions in inter-partition communication is not supported

Requirements:

rte_sws_7606, rte_sws_2761

► Segmentation fault checks are not implemented. If the return value is needed to compile code, the user is responsible to define it

Requirements:

rte_sws_2752, rte_sws_2753, rte_sws_2757, rte_sws_2756, rte_sws_2754, rte_sws_2755, rte_sws_-8301, rte_sws_8302

► Range checks are not supported (reference to product description: ASCPD-54)

Requirements:

rte_sws_3839, rte_sws_3840, rte_sws_3841, rte_sws_3842, rte_sws_3843, rte_sws_3845, rte_sws_-3846, rte_sws_3847, rte_sws_3848, rte_sws_3849, rte_sws_8024, rte_sws_3861, rte_sws_8026, rte_-

sws_8039, rte_sws_8027, rte_sws_8040, rte_sws_8030, rte_sws_8031, rte_sws_8032, rte_sws_8033, rte_sws_8028, rte_sws_8041, rte_sws_8016, rte_sws_8025, rte_sws_8029, rte_sws_8042, rte_sws_-8034, rte_sws_8035, rte_sws_8036, rte_sws_8037, rte_sws_8038

The following requirements are partially affected by this deviation:

constr_1188

▶ The configuration parameter `RteToolChainSignificantCharacters` is not supported

Requirements:

rte_sws_7300

▶ Debugging support is only realized via VFB tracing

Requirements:

rte_sws_5094, rte_sws_5095, rte_sws_5096, rte_sws_5097, rte_sws_5098, rte_sws_5105

▶ External configuration switches for strict configuration checking are not supported

Description:

A custom parameter exists to consider unconnected require ports as an error.

Requirements:

rte_sws_5099, rte_sws_5148, rte_sws_7680, rte_sws_7642, rte_sws_7681, rte_sws_5150

▶ The symbol attribute of component types is not evaluated

Requirements:

rte_sws_6714, rte_sws_6715

The following requirements are partially affected by this deviation:

rte_sws_3837, rte_sws_7104, rte_sws_7110, rte_sws_7111, rte_sws_7114, rte_sws_7144, rte_sws_-7116, rte_sws_7109, rte_sws_7148, rte_sws_3793, rte_sws_3714, rte_sws_3731, rte_sws_1238, rte_-sws_1239, rte_sws_1248, rte_sws_1249, rte_sws_7190

▶ Unconnected mode switch ports are not supported

Requirements:

rte_sws_2639, rte_sws_2640

▶ In the current Rte implementation one partition must exist per core. Therefore the file structure differs from specification. For the partitioning Rte, there is one `Rte_OsAppl.c` file for each Os application. There are no core-specific C-files

Requirements:

rte_sws_1169, rte_sws_2711, rte_sws_2712, rte_sws_2713, rte_sws_7140, rte_sws_7616

► The component data structures types are defined in the Rte type header file instead of application header file because `Rte_Hook.h` requires the component data structure type definition

Requirements:

rte_sws_7132

► The port data structure types are defined in the types header file instead of application header file, because `Rte_Hook.h` requires the port data structure type definition

Requirements:

rte_sws_7137

► Infrastructure errors are not supported

Requirements:

rte_sws_7404, rte_sws_7405, rte_sws_7406, rte_sws_2593, rte_sws_2573

► The return value `RTE_E_IN_EXCLUSIVE_AREA` of API functions which are called when the task's call stack has entered or is running in an exclusive area is not supported

Description:

There is no efficient possibility to check if a blocking Rte function is called inside of an exclusive area. Furthermore, during the development process it is already possible to detect such errors, because the call of `WaitEvent()` during locked interrupts/resources results in an Os Error. See the AUTOSAR Bugzilla: http://www.autosar.org/bugzilla/show_bug.cgi?id=57797

Requirements:

rte_sws_2739, rte_sws_2740, rte_sws_2741, rte_sws_2743, rte_sws_2744, rte_sws_2745, rte_sws_-2746

The following requirements are partially affected by this deviation:

rte_sws_7673

► Invalidation for complex data elements is not supported (reference to product description: ASCPD-50)

Requirements:

rte_sws_5063, rte_sws_5064, rte_sws_5065, rte_sws_7639

► Restart behavior of mode provider/mode user partition is not defined

Description:

See the AUTOSAR Bugzilla: http://www.autosar.org/bugzilla/show_bug.cgi?id=50884

Requirements:

rte_sws_2731

► The attribute `PartitionCanBeRestarted` is not evaluated

Requirements:

rte_sws_7619, rte_sws_7336

► Client prefixes for VFB trace hook functions are not supported

Requirements:

rte_sws_5093, rte_sws_5092, rte_sws_5091

The following requirements are partially affected by this deviation:

rte_sws_1238, rte_sws_1239, rte_sws_1242, rte_sws_1243, rte_sws_1244, rte_sws_1245, rte_sws_-
1246, rte_sws_1247, rte_sws_1248, rte_sws_1249

► The revision label in ECU configuration is not supported

Requirements:

rte_sws_5184, rte_sws_5185

► Schedule points are not supported

Requirements:

rte_sws_5113, rte_sws_5114, rte_sws_7042, rte_sws_7043, rte_sws_5115, rte_sws_5116

► Mapping of modes and Os schedule tables is not supported

Requirements:

rte_sws_5146, rte_sws_2759, rte_sws_2760

► Partitioning support with the IOC as specified in AUTOSAR 4.0.3 is not supported

Requirements:

rte_sws_8400

► XML-content categorized as ICS is not evaluated

Requirements:

rte_sws_8305

▶ The `SchM_CData` API as specified in AUTOSAR 4.0.3 is not supported

Requirements:

rte_sws_7096, rte_sws_7093, rte_sws_7094, rte_sws_7095

▶ The Rte does not restart the time-out for received signals

Description:

The signal time-out mechanism is implemented in the Com module and doesn't require the Rte to perform special actions.

Requirements:

rte_sws_8004

▶ The MultiCore COM access via Rte `BswSchedulableEntities` as specified in AUTOSAR 4.0.3 is not supported

Requirements:

rte_sws_8306, rte_sws_8307, rte_sws_8308

▶ The port interface element mapping for elements of composite types as specified in AUTOSAR 4.0.3 is only supported for the mapping of implementation data type sub elements, and only for the mapping to primitive types

Requirements:

rte_sws_7092

The following requirements are partially affected by this deviation:

rte_sws_7091 1, rte_sws_7099 1

▶ The support for mixed elements of `CompuMethod` as specified in AUTOSAR 4.0.3 is not supported

Requirements:

rte_sws_3860

▶ The initialization and naming of `PerInstanceMemory` as specified in AUTOSAR 4.0.3 is not supported

Requirements:

rte_sws_8304

▶ The passing scheme for output parameters of pointer implementation data types is not supported

Requirements:

rte_sws_7083

► The inter-module check as specified in AUTOSAR 4.0.3 is not supported

Requirements:

rte_sws_7692

► The RTE Data Handle Types Header File as specified in AUTOSAR 4.0.3 is not supported

Requirements:

rte_sws_7924, rte_sws_7920, rte_sws_7921, rte_sws_7922, rte_sws_7923

The following requirements are partially affected by this deviation:

rte_sws_7136

► The generation of the types for `ArrayImplementationDataType` as specified in AUTOSAR 4.0.3 is not supported

Requirements:

rte_sws_6706, rte_sws_6707, rte_sws_6708

► The symbol attribute of implementation data types as specified in AUTOSAR 4.0.3 is not supported

Requirements:

rte_sws_6716, rte_sws_6717, rte_sws_6718, rte_sws_6719, rte_sws_6724

The following requirements are partially affected by this deviation:

rte_sws_6706, rte_sws_6707, rte_sws_6708

► The Rte does not support nameless parameters. If no runnable entity arguments are specified, the Rte uses the short names of client server operation arguments and automatically generated names for port defined argument values

Requirements:

rte_sws_6705

► The enhanced `SchM_Mode` API as specified in AUTOSAR 4.0.3 is not supported

Requirements:

rte_sws_7694, rte_sws_8507, rte_sws_8509, rte_sws_8510

▶ The Rte Generator does not support `SwDataDefProps` for `InstantiationDataDefProps`, `Access-Point`, `FlatInstanceDescriptor` and `McDataInstance`

Requirements:

The following requirements are partially affected by this deviation:

rte_sws_7196

▶ Several constraints of the BSW Module Description Template are not implemented

Requirements:

constr_4019, constr_4020, constr_4036, constr_4013, constr_4021, constr_4056, constr_4057, constr_4052, constr_4053, constr_4060, constr_4038, constr_4037, constr_4023, constr_4065, constr_4024, constr_4025, constr_4040, constr_4041, constr_4058, constr_4051, constr_4047, constr_4048, constr_4045, constr_4046, constr_4028, constr_4054, constr_4029, constr_4030, constr_4031, constr_4032, constr_4033, constr_4062, constr_4044, constr_4034, constr_4061, constr_4087

▶ Several constraints of the Software Component Template are not implemented

Description:

constr_2000 checks number of arguments (which can be ArgumentDataPrototypes or relatedPortDefinedArgumentValues) and PortAPIOption.errorHandling.

Requirements:

constr_1005, constr_1010, constr_1017, constr_1018, constr_1019, constr_1020, constr_1021, constr_1025, constr_1026, constr_1029, constr_1036, constr_1039, constr_1040, constr_1041, constr_1042, constr_1044, constr_1046, constr_1058, constr_1095, constr_1101, constr_1109, constr_1135, constr_1138, constr_1139, constr_2010, constr_2011, constr_2012, constr_2014, constr_2015, constr_2016, constr_2019, constr_2026, constr_2032, constr_2035, constr_2036, constr_2037, constr_2038, constr_2039, constr_2040, constr_2041, constr_2042, constr_2043, constr_2044, constr_2045, constr_2046, constr_2048, constr_2533, constr_2536, constr_4000, constr_4002, constr_4004, constr_4005, constr_4006, constr_4007, constr_4008, constr_4009, constr_4010, constr_1174, constr_2049, constr_1172, constr_1166, constr_1184, constr_1185, constr_1186, constr_1189, constr_1147, constr_1143, constr_2544, constr_1152, constr_1178, constr_1161, constr_2551, constr_2545, constr_1140, constr_2550, constr_2548, constr_2549, constr_1145, constr_1146, constr_2561, constr_1160, constr_2050, constr_2051, constr_1153, constr_1154, constr_1155, constr_1156, constr_1157, constr_1176, constr_1158, constr_1141, constr_1150, constr_2053, constr_1179, constr_1180, constr_1144, constr_1148, constr_1149, constr_1004, constr_2018, constr_2029, constr_2047

The following requirements are partially affected by this deviation:

rte_sws_3526, constr_2000 2

▶ The invalidation of a data element for multiple receivers with different `handleInvalid` values is not supported

Requirements:

rte_sws_7031, rte_sws_7032

▶ For intra-ECU and inter-partition communication, the Rte does not use the filter implementation of the COM layer but uses its own implementation. For inter-ECU communication the Rte relies on the COM filter implementation, but does not automatically configure the filter parameters in the COM configuration

Requirements:

rte_sws_5500

▶ The Rte does not consider a restart of the client partition during execution of the server

Requirements:

rte_sws_7346

▶ The Rte optimization mode is not supported

Requirements:

rte_sws_5053

▶ The `errorCode` of an application error (client/server interface) is allowed to be `0` if the `possibleError` is supposed to represent `E_OK`

Requirements:

The following requirements are partially affected by this deviation:

constr_1108

▶ DET segmentation fault checking is not supported

Description:

Os memory protection can be used instead.

Requirements:

rte_sws_7685

▶ If a mode machine instance is located in a different partition as its mode provider and non-blocking mode switch acknowledgement is enabled, the Rte will only return RTE_E_NO_DATA when the mode switch is actively being processed. If Rte_SwitchAck is called before the mode machine instance has been activated, RTE_E_TRANSMIT_ACK will be returned.

Description:

This deviation only applies to inter-partition, non-blocking mode acknowledgement.

Requirements:

The following requirements are partially affected by this deviation:

rte_sws_2727, rte_sws_2729

► Due to a typo in the Rte specification (ASR 4.0.x) a non-blocking Rte_Result API function is only generated if an AsynchronousServerCallReturnsEvent exists. This specification error has been fixed with ASR 4.1 so that the Rte_Result function is generated for an AsynchronousServerCallResultPoint no matter if there is an event or not.

Description:

The ASR-4.2.1 version of this requirement will be supported instead.

Requirements:

rte_sws_1296

► These requirements have been removed in AUTOSAR 4.2.2

Requirements:

rte_sws_5022

► These requirements have been removed in AUTOSAR 4.1.x or later

Requirements:

rte_sws_2652, rte_sws_2579, rte_sws_5066, rte_sws_5067, rte_sws_5055, rte_sws_6028, rte_sws_-5056, rte_sws_5057, rte_sws_5058, rte_sws_5059, rte_sws_5054

► The RTE Generator shall wrap each RunnableEntity's Entry Point Prototype in the Application Header File with the Memory Mapping and Compiler Abstraction macros. FUNC(<void|Std_ReturnType>, <c>_-<sadm>)

Description:

The handling of Compiler Abstraction macros is not supported

Requirements:

The following requirements are partially affected by this deviation:

rte_sws_7194

▶ The Rte properly performs an inter-partition C/S call even if another instance of the client runnable (i.e. the same client) is already calling the server on a different task in parallel. This behavior contradicts to [rte_sws_2658]. However, the result (i.e. return code and OUT and IN/OUT parameters) of each particular server call is contributed to the proper client. Please note: This behavior does not affect asynchronous inter-partition client server communication where RTE_E_LIMIT is immediately returned on an attempt to call the server a second time by the same client.

Description:

The Rte allows multiple inter-partition synchronous client server calls for the same client

Requirements:

The following requirements are partially affected by this deviation:

rte_sws_2658

▶ The minimumStartInterval is not supported for Executables started by ModeSwitchEvents/Mod-eSwitchedAckEvent, OperationInvokedEvents or only TimingEvents.

Requirements:

The following requirements are partially affected by this deviation:

rte_sws_7173, rte_sws_2697

▶ The RunnableEntity of an NvBlockDescriptor is activated by a DataReceivedEvent only if the attribute NvBlockNeeds.storeImmediate is set to true. It is not activated in case NvBlockNeeds.storeAtShutdown is set to true and NvBlockNeeds.storeImmediate is set to false.

Requirements:

The following requirements are partially affected by this deviation:

SWS_Rte_08088

▶ The Rte_LdComCbkStartOfReception_<sn> signature is generated according to ASR 4.0.3, to be compatible with the existing stack, while the module requirements are based on ASR SWS 4.2.2. Therefore the parameter "const PduInfoType* info" is missing from the signature.

Requirements:

The following requirements are partially affected by this deviation:

SWS_Rte_01396

▶ The Rte_LdComCbkTpRxIndication_<sn> signature is generated according to ASR 4.0.3, to be compatible with the existing stack, while the module requirements are based on ASR SWS 4.2.2. Therefore the parameter "PduIdType id" is missing from the signature and the type of the "result" parameter is "NotifResultType" instead of "Std_ReturnType".

Requirements:

The following requirements are partially affected by this deviation:

SWS_Rte_01403

▶ The Rte_LdComCbkTpTxConfirmation_<sn> signature is generated according to ASR 4.0.3, to be compatible with the existing stack, while the module requirements are based on ASR SWS 4.2.2. Therefore the parameter "PduIdType id" is missing from the signature and the type of the "result" parameter is "NotifResultType" instead of "Std_ReturnType".

Requirements:

The following requirements are partially affected by this deviation:

SWS_Rte_01407

▶ If a server is invoked asynchronously and no AsynchronousServerCallResultPoint resp. BswAsynchronousServerCallResultPoint exists then the RTE allows multiple outstanding asynchronous client-server invocations of the same client. The number of possible outstanding invocations is limited to the length of the server queue.

Requirements:

The following requirements are partially affected by this deviation:

rte_sws_3773, rte_sws_2658, rte_sws_1105, SWS_Rte_08742

▶ When a DataReceivedEvent references a RunnableEntity and a required VariableDataPrototype with "event" semantic (swImplPolicy equal to queued) and no WaitPoint references the DataReceivedEvent, the RunnableEntity will be activated even if the "event" has been discarded due to a full queue ([rte_-sws_2634]).

Requirements:

The following requirements are partially affected by this deviation:

rte_sws_1292

► The queued trigger feature for the inter-partition case deviates from the behavior described in SWS_Rte_-07089, because a dequeuing does not only happen at "the end of execution of all triggered ExecutableEntitys which are triggered by this trigger emitter" but for each task.

Requirements:

The following requirements are partially affected by this deviation:

SWS_Rte_07089

► If COM indicates a reception time-out (via RTE COM Rx time-out callback) the RTE shall raise an event of reception time-out to software components as data element outdated in case of data receiver error event was added.

Requirements:

The following requirements are partially affected by this deviation:

SWS_Rte_08062

► The inter partition mode communication deviates from the behaviour described in rte_sws_2724. This was deleted in ASR SWS 4.2.1. One ModeDeclarationGroupPrototype of a provide port can be connected to ModeDeclarationGroupPrototypes of require ports from more than one partition. A configuration in which the mode users of one mode machine instance are distributed over several partitions is supported.

Requirements:

rte_sws_2724

The following requirements are partially affected by this deviation:

rte_sws_2679

► A configuration with multiple mode user groups in a core is supported.

Requirements:

The following requirements are partially affected by this deviation:

rte_sws_2662, rte_sws_2663

► Timeout monitoring is not supported for BswModeSwitchAckRequest.

Requirements:

The following requirements are partially affected by this deviation:

rte_sws_7055, rte_sws_7056

► Compound Primitive Data Types are not supported.

Requirements:

The following requirements are partially affected by this deviation:

SWS_Rte_06765

► If a sender with status (IFeedback ApI) is connected to a receiver without status(IStatus API) , or a sender without status is connected to a Receiver with status and they are mapped to tasks that cannot interrupt each other the Data Handle with status will be shared (only one).

Requirements:

The following requirements are partially affected by this deviation:

rte_sws_2608

► API may only be used by the runnable that describes its usage

Description:

This is done on application code. This check is not supported

Requirements:

rte_sws_7649, SWS_Rte_CONSTR_09094, SWS_Rte_CONSTR_09092

► The trigger of DataWriteCompletedEvent after the task terminates or in the sending task and after the transmission acknowledgment from COM or LdCom is not supported

Requirements:

rte_sws_8045

The following requirements are partially affected by this deviation:

rte_sws_8017, rte_sws_8020, rte_sws_8022

► The Data Handle will be a pointer to a data element with extended status if and only if the runnable has write access and read access via a PRPortPrototype and status is enabled for both.

Requirements:

The following requirements are partially affected by this deviation:

SWS_Rte_06827

► The "I-PDU based time-out" is not supported.

Requirements:

The following requirements are partially affected by this deviation:

SWS_Rte_08103

▶ The status of data elements is never reseted on the receiver side to RTE_E_NEVER_RECEIVED when the receiver's partition is restarted.

Requirements:

rte_sws_7645

▶ Mapping constraint of scope MAPPING-SCOPE-ECU is not supported as the Ecu extract only generates the components for the selected Ecu only

Requirements:

The following requirements are partially affected by this deviation:

eb_custom_rte.Mapping_Constraints.TPS_SYST_01024,eb_custom_rte.Mapping_Constraints.TPS_-
SYST_01025,eb_custom_rte.Mapping_Constraints.TPS_SYST_01045,eb_custom_rte.-
Mapping_Constraints.TPS_SYST_01026

▶ Meta data for application is not supported.

Requirements:

SWS_Rte_03620

The following requirements are partially affected by this deviation:

SWS_Rte_04577

▶ The custom parameter `ComMainPartitionRef` is used instead of `ComMainRxPartitionRef` and `ComMainTxPartitionRef`.

Requirements:

The following requirements are partially affected by this deviation:

SWS_Rte_03001

3.3.1.5. Limitations

This chapter lists the limitations of the module. Refer to the module references chapter *Integration notes*, subsection *Integration requirements* for requirements on integrating this module.

▶ The value RTE_E_COM_STOPPED is not returned for inter-partition/inter-ECU communication

Description:

If the Com module is not able to successfully send a ComSignal or ComSignalGroup and the calling software component is located in another partition as the Com module, an Rte Api function will not return the value RTE_E_COM_STOPPED to the calling software component.

Rationale:

When the Com module is located in a different partition as the SWC, the ComSignal must be sent asynchronously via the configured BswTask. The Rte can therefore not access the result of the Com function execution before returning control to the software component.

▶ The Rte expects an IOC configuration and IOC API which is not fully compliant to AUTOSAR 4.0

Description:

The Rte expects an EB-specific IOC configuration. Moreover, the Rte expects the IOC to provide an `Ioc_-ReInit` function. Also the Rte expects that the IOC API is reentrant.

Rationale:

The AUTOSAR 4.0 specification of the IOC still has some gaps. The EB-specific solution allows more optimizations.

▶ When converting an AUTOSAR 3.1 system description to AUTOSAR 4.0, not all initial value types for parameters are supported

Description:

When converting an AUTOSAR 3.1 system description to AUTOSAR 4.0, initial values for parameters, that represent a nested record or array type, are not supported. Only primitive types, simple record, and array types are supported.

Rationale:

The converter cannot perform a deep copy on the value specification of the parameter's initial value. Thus, it is not supported that the initial value is a nested record or array, etc.

▶ When converting an AUTOSAR 3.1 system description to AUTOSAR 4.0, an opaque type will lose the byte size information

Description:

When converting an AUTOSAR 3.1 system description to AUTOSAR 4.0, an opaque type from AUTOSAR 3.1 will lose its byte size information.

Rationale:

The opaque type is converted to a primitive type. Thus the byte size information gets lost.

► If inter-partition mode management is used, it cannot be guaranteed that the initialization of the inter-partition mode users' partition is complete once its initialization routine returns

Description:

Because partitions execute independently, it may be possible that the initialization routine of the mode users' partition returns before that of the mode provider's partition. In this situation, the `OnEntry` runnables of the inter-partition mode users will not have been executed after the initialization of the partition has been completed. Also, the mode disabling dependencies of the initial mode would not be enabled.

Reliance on inter-partition modes for initialization should be avoided. If inter- partition modes are to be used for initialization, a check of the mode provider's partition state should be made before accessing any resources which require initialization.

Rationale:

Partitions execute, terminate, and start independently.

► Rte Enhanced Mode Api for the mode manager in case of inter-partition mode management

Description:

Rte Mode Api for the enhanced mode provider inter-partition mode management, will return the current mode for the previous and next mode in case all mode users have the same current mode.

► The data filter types `NEWISEQUAL`, `NEWISDIFFERENT`, `MASKEDNEWEQUALSMASKEDOLD`, `NEWISGREATER`, `NEWISLESSOREQUAL`, `NEWISLESS`, and `NEWISGREATEROREQUAL` for a require port typed by sender receiver interfaces are not supported

Description:

The EB tresos Studio supports AUTOSAR 4.0.3 where the mentioned data filter types are not supported any longer.

Rationale:

In AUTOSAR 4.0.3, the mentioned data filter types have been removed.

► Partial record support (signal degradation) is not available for inter-partition communication

Description:

Sender/receiver communication with the IOC or Shared memory communicator (Smc) will not function if the destination data element is a partial record.

► Segmentation fault checks are not supported

Description:

The segmentation fault check performed by the Rte in case of an inter-partition communication is not supported.

► Minimal offset of allocated Os schedule table expiry points is one microsecond

Description:

When allocating an Os schedule table, the offset of an expiry point must be divisible by one microsecond.

Rationale:

The expiry point names consist of the prefix `EP_` and the offset value in microseconds.

► Implementation data types of category function reference are not supported

Description:

Implementation data types or implementation data type elements of category `FUNCTION_REFERENCE` are not supported.

► Limited calibration support on S12X with Metroworks compiler

Description:

When a mixed memory model is used on the S12X, the following applies:

► `RTE_VAR` is mapped to far and `RTE_CONST` is mapped to near.

► `RTE_VAR` is mapped to near and `RTE_CONST` is mapped to far.

The following limitations apply when calibration is used:

1. `Rte_CData` and `Rte_CalPrm` API do not work with record and array types.

2. Online calibration using single pointered or double pointered methods is not possible.

Rationale:

1. For complex types, the APIs return pointers. This requires nested compiler abstraction. The Metroworks compiler cannot handle this correctly. Although nested compiler abstraction is forbidden by the AUTOSAR Compiler Abstraction specification (COMPILER058), nested compiler abstraction must be used because the RTE SWS specifies that the APIs shall return pointers. Here, the RTE SWS has to be changed. See Bugzilla #42952.

2. The pointers in the calibration reference table point to constants in ROM. For double pointered method, the calibration base pointer points to a calibration reference table in ROM. When `RTE_CONST` is

mapped to near, the pointers cannot be redirected to variables located in RAM which would have to be addressed far in this case.

▶ The activation offset of a timing event cannot be greater or equal to the period

Description:

The Rte does not support timing events with an activation offset greater or equal to the period. If you configure an activation offset greater or equal to the period anyway, then the offset is automatically shortened to be within the period by a modulo division (e.g. period is 100ms, original activation offset is 150ms and the resulting offset will be 50ms).

▶ If pointer to array type is used, linker type checking cannot be enabled

Description:

The function signature of API functions that are implemented by the Rte always uses pointer to array base type for array arguments. If pointer to array type is used, the array type in the function declaration of the software component header does not match with the array type of the Rte implementation. On object code level, both variants are equivalent. However a linker where type checking is enabled, may report an error.

▶ Limited support for NvBlockSwComponentType data consistency

Description:

The NvBlockSwComponentType currently only supports a simple interrupt blocking mechanism to assure data consitency. Ohter mechanisms, e.g. "Usage of OS resources", are not supported.

▶ Calibration access to NvBlockDescriptor romBlock is not supported

Description:

The NvBlockSwComponentType does not support calibration access for the NvBlockDescriptor romBlock.

▶ Limited support for NvBlockSwComponentType NvBlockDescriptor parameters

Description:

The NvBlockSwComponentType does not support the NvBlockDescriptor parameters, "constantValueMapping", "dataTypeMapping" and "instantiationDataDefProps".

▶ Limited support for PortInterfaceMapping

Description:

NvDataInterfaces currently support only Bitfield Textable PortInterfaceMapping.

▶ Limited support for SubElementMapping of VariableAndParameterInterfaceMapping without transformers

Description:

Currently the following limitations apply when using SubElementMappings without transformers:

Only mapping a composite element to primitive DataPrototype is supported

Only the mapping of struct elements is supported (no array elements)

Data conversion is not supported

Data invalidation is not supported

Mapping of variable data prototypes to parameter data prototypes is not supported

► Limited support for SubElementMapping of VariableAndParameterInterfaceMapping with transformers

Description:

Currently the following limitations apply when using SubElementMappings with transformers:

If optional structure elements are used, only SubElementMappings from one ImplementationDataType to another ImplementationDataType are supported

Mapping of variable data prototypes to parameter data prototypes is not supported

► Limited support for inter-ECU trigger communication

Description:

Currently the following limitations apply when using inter-ECU trigger communication:

Only supported with data transformation

Inter-partition communication is not supported (the mapped signal must be on the same partition as the SWC)

► The blocking Rte_Receive API is not supported for inter-partition communication

Description:

The Rte generator logs an error if all of the following conditions apply:

1. Sender and receiver are located in different partitions.

2. A blocking `Rte_Receive` API is configured.

► The Com_InvalidateSignalGroup API is not supported

Description:

The Rte does not use the `Com_InvalidateSignalGroup` API to invalidate a signal group. Instead, it always sends the signal group with the invalid value.

► The Com_InvalidateSignal API is not used in case of inter-partition inter-ECU signal invalidation

Description:

Regardless of the value of parameter `InterECUInvalidationHandledByRte` the Rte always sends the invalid value in case of inter-partition inter-ECU sender/receiver communication instead of using the `Com_InvalidateSignal` API.

▶ The operation invoked event of the server which is called by the `Rte_NvMNotifyJobFinished` callback shall not be mapped to a task

Description:

The `Rte_NvMNotifyJobFinished` callback supports only direct calls. Please note that the behavior of the Rte is undefined if this runnable calls any Rte API, because it's running in an interrupt context (see also SWS_Rte_03600).

▶ Mode disabling dependencies for direct call dirtyFlag runnables of a NvBlockSwComponent are not supported.

Description:

Mode disabling dependencies for dirtyFlag runnables of a NvBlockSwComponent are only supported if the DataReceivedEvent is mapped to a task.

▶ Data consistency is not guaranteed for sender-receiver invalidation external replacement

Description:

If a sender-receiver data element has been received invalidated and the attribute handleInvalid is set to externalReplacement the Rte doesn't guarantee data consistency for the external replacement value.

▶ Sender-receiver invalidation external replacement with ParameterDataPrototype not fully supported

Description:

Data element invalidation with the attribute handleInvalid is set to externalReplacement and replaceWith references a ParameterDataPrototype is not fully supported in case of inter-partition sender-receiver communication.

▶ Debounced activation of executables is not supported for certain events

Description:

The minimumStartInterval is ignored for executables started by ModeSwitchEvents, ModeSwitchedAck-Events, OperationInvokedEvents or only TimingEvents.

▶ Event order cannot be guaranteed for debounced activation of executables

Description:

The Rte cannot guarantee that a delayed event (i.e. the first event occurrence within the minimumStartInterval period) will finally lead to the start of the executable if another event for the same executable occurred in close succession to the expiration of the debounced alarm of the delayed event.

Rationale:

It depends on the internal scheduling of the Os which task of the executable will enter running state first when multiple events are set at once.

► Distribution of the Com onto multiple partitions

Description:

The following parameters cannot be configured separately for each Com partition:

- ► SendSignalQueueStrategy
- ► BswOsTaskPeriod

For client/server communication, the call and result signal must be mapped onto the same partition.

For client/server communication, an operation fan-in/out (multiple call/result signals mapped to the same operation) is not supported.

► Transport protocol for serialization

Description:

The non-AUTOSAR API LdCom_TpTransmit is used by the Rte.

► XfrmBufferLengthType UINT32 not supported for inter-partition communication via IOC

Description:

For inter-partition communication, IOC does not support an XfrmBufferLengthType value other than UINT16.

Rationale:

IOC does not provide an API for UINT32 variable length data.

► The following Signal fan-in/out scenario is not supported for inter-ECU sender/receiver communication

Description:

ISignal(1) to (Ld)Com (n)

► The following Signal fan-in/out scenarios are not supported for inter-ECU client/server communication

Description:

ISignal(1) to (Ld)Com (n)

SystemSignal(1) to ISignal (n)

Operation(n) to SystemSignal (1)

Server port(n) to SystemSignal (1)

► Asynchronous client/server communication with two clients mapped to Os Interrupt and OsTask is not supported

Description:

Asynchronous client/server communication with two clients where one client is mapped to an Os Interrupt and the other to an OsTask is not supported.

► Limited support for writing data to different NvRamBlocks

Description:

Writing data to different NvRamBlocks using the same RPortPrototype of a NvBlockSwComponentType is not supported.

► Limited support for Coherency Groups

Description:

Currently the following limitations apply when using Coherency Groups:

DirectReadFromCom is not supported

NvBlockSwComponentTypes are not supported

Grouping of accesses across execution conditions (e.g. different data received events or mode disabling dependencies) is not supported

For read/write accesses not in a Coherency Group, which share the same receive buffer with accesses in a Coherency Group, the same locking mechanisms are applied

Grouping of the status members for senders with transmition acknowledgement enabled is not supported

► Limited support for mapping events to ISRs

Description:

Currently the following limitations apply when mapping events to ISRs:

Only (BSW)TimingEvent and (BSW)ExternalTriggerOccurredEvent are supported.

It is allowed to map either timing or external trigger occurred events to an ISR but not the both event types.

► Limited support for mode switched acknowledgment event

Description:

Currently the following limitations apply when using mode switched acknowledgment event:

one mode switch may lead to more than one mode switched acknowledgment event in case of mode communication with multiple mode user partitions on several cores.

► Limited support for Det error reporting when multiple partitions, and Bsw Modules multiple instances with shared exclusive area are used

Description:

Currently the following limitations apply when multiple partitions are used. In this case the Det error reporting is not supported.

► Limited support for rule based value specification

Description:

Currently the following limitations apply when using RuleBasedValueSpecifications:

Compound Primitive Data Types are not supported.

RULE-ARGUMENTS of type VTF are not supported.

► Limited support for variable size array inter-ECU communication without transformers

Description:

Currently the following limitations apply when variable size array inter-ECU communication without transformers is used:

Data invalidation is not supported.

Multiple variable size arrays with different data types can not be received by the same TP LdCom PDU.

► Limited support for RIPS with parameter interface

Description:

Currently the following limitations apply when using RIPS with parameter interfaces:

Rte Online Calibration must be deactivated (RteCalibrationSupport = NONE)

RteRipsGlobalCopyInstantiationPolicy must be set to RTE_RIPS_INSTANTIATION_BY_PLUGIN

RTE_PTR2ARRAYTYPE_PASSING is not supported for `Rte_Prm` API

Only 1:n communication is supported

► Limited support for synchronous mode switching procedure over multi partitions/cores

Description:

Currently the following limitations apply when using synchronous mode switching procedure over multi partitions/cores:

Partition restarts are not considered. That means if a partition is not yet started or stopped then a mode switch during this time could lead to infinite waiting in transition of the active partitions. So we assume that all mode user partitions are either active or about to start so that the application will not get stuck in a mode transition.

► Limited support for mapping of variable data prototypes to parameter data prototypes

Description:

Currently the following limitations apply when mapping of variable data prototypes to parameter data prototypes:

PR port is not supported

Inter ECU is not supported

Only 1:n communication is supported

Transformers, SubElementMapping and DataConversion are not supported

Timeout monitoring, handleNeverReceived, enable update, handleOutOfRange, and invalidation are not applicable for this case

DataReceivedEvents/DataReceiveErrorEvents are not supported

RIPS is not supported

3.3.1.6. Open-source software

The software that is delivered with EB tresos AutoCore Generic can be classified into the following two categories:

► Software that is executed on the electronic control unit (ECU).
► Software that is used for the development infrastructure (configuration, generation, building) and thus executed on the development platform.

3.3.1.6.1. Open-source software in software executed on the ECU

No open-source software that runs on the ECU is delivered with Rte.

3.3.1.6.2. Open-source software in software used for the development infrastructure

The following list of open-source software that is used in development is delivered with Rte:

► Commons Math: The Apache Commons Mathematics Library
3.2.
https://commons.apache.org/proper/commons-math/

List of licenses:

► Apache License Version 2.0
commons-math.txt

List of copyrights:

► Copyright (C) The Apache Software Foundation.

4. ACG8 RTE user's guide

4.1. Overview of the Run-Time Environment (Rte)

The `EB tresos AutoCore Rte user's guide` describes the architecture, configuration and use of the `Run-Time Environment` (`Rte`). The user's guide will help you

▶   to understand the functionality of the `Rte`, see [Section 4.2, "Background information"](#),

▶   to configure the `Rte`, see [Section 4.3, "Configuring and generating the Rte"](#), and

▶   to generate the `Rte` module, see [Section 4.3.5, "Generating the Rte"](#).

You will find the Rte API in the document EB tresos AutoCore module references.

For the definition of individual terms, see the glossary, which is located at `$TRESOS_BASE/doc/1.0_In-stallation`.

4.1.1. Functionality in the context of AUTOSAR

The `Run-Time Environment` (`Rte`) is at the heart of the AUTOSAR architecture. The `AUTOSAR Rte` provides

▶   a *hardware-independent communication infrastructure* for application software (AUTOSAR software components) in an AUTOSAR system, and

▶   a *run-time environment* for the software component's runnable entities and

▶   a *run-time environment* for the basic software module's schedulable entities.

The communication infrastructure provided by the `Rte` enables communication between:

▶   AUTOSAR software components;

▶   AUTOSAR software components and specific basic software modules (AUTOSAR `Services` and `I/O Hardware Abstraction (IoHwAb)`);

The `Rte` is generated *for each individual ECU* to meet the specific needs of the application software on that ECU while keeping code and run-time overhead at a minimum.

In order to provide its functionality, the AUTOSAR `Rte` uses services of the `Os` module and optionally of the `Com` module. See [Section 4.1.2, "Dependencies on other AUTOSAR modules"](#) for details.

An application in an AUTOSAR system consists of AUTOSAR software components that communicate with each other via AUTOSAR interfaces, over the AUTOSAR virtual functional bus (*VFB*). The `Run-Time Environment` (`Rte`) provides the implementation of the VFB concepts.

This approach ensures that AUTOSAR software components and their communication channels are independent from the underlying hardware. Thus the `Rte` can be seen as an *abstraction layer* that hides the underlying basic software modules, hardware and communication infrastructure from the application. See Figure 4.1, "Overview of the AUTOSAR software layers" for an overview of the different layers. The `Rte` is located in the application abstraction layer (second from top).
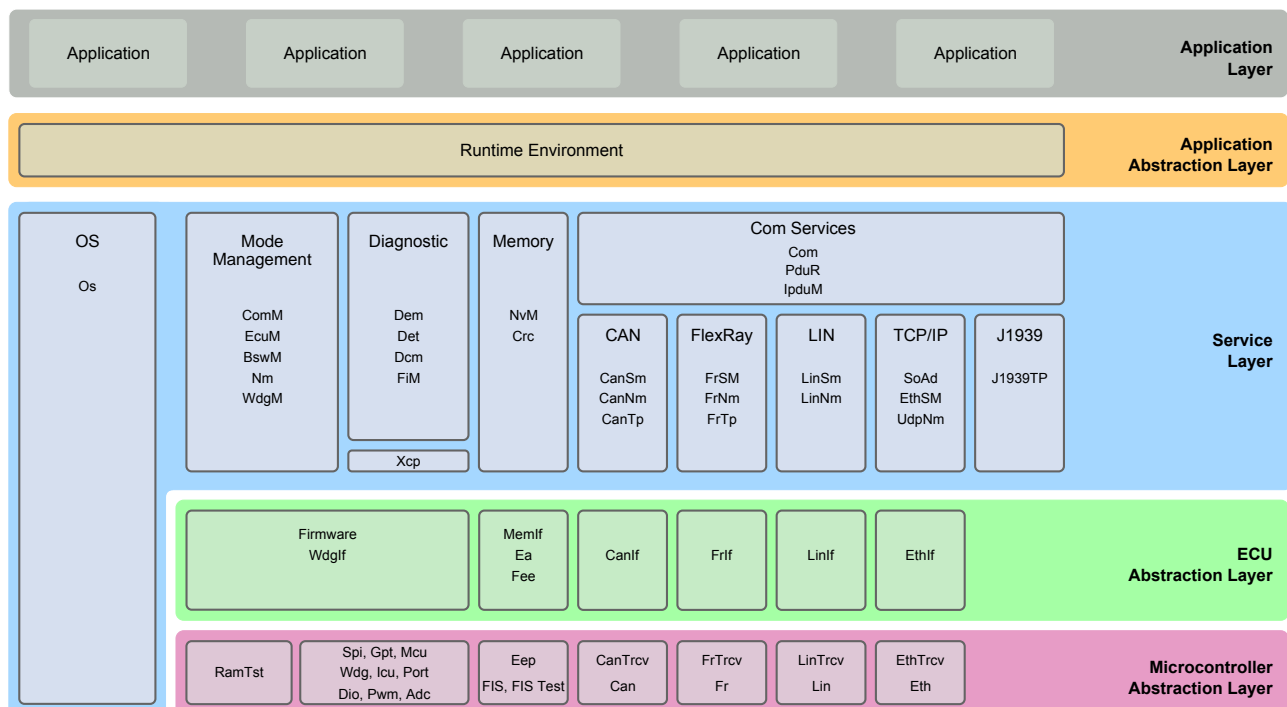
Figure 4.1. Overview of the AUTOSAR software layers

The main purpose of the `Rte` is to provide *communication and operating system functionality* for software components.

The module provides

► means to communicate through ports (sender/receiver or client/server communication) and

► it provides the task context necessary to execute the runnable entities of a software component.

The AUTOSAR 4.0 Rte specification merged the Bsw Scheduler module into the Rte module. The Bsw Scheduler is now apart of the Rte, but has its own interface. The Rte module now

► provides the task context necessary to execute the schedulable entities (i.e. main functions) of basic software modules and

► implements the Bsw Scheduler API functions

For a detailed description of the `Rte`'s functionality, see [1]

4.1.2. Dependencies on other AUTOSAR modules

The `Rte` implementation uses AUTOSAR `Com`, AUTOSAR `Base`, and AUTOSAR `Os` to provide its functionality.

4.1.2.1. Os

Depending on the input configuration, the generated `Rte` uses the following objects of the `Os` module and the objects' respective API calls :

► `Os` tasks

► `Os` alarms

► `Os` counters

► `Os` events

► `Os` resources

► `Os` spinlocks

► `Os` schedule tables

► `Os` Ioc channels

4.1.2.2. Base

Provides basic services to CPU-specific functionality like `TS_MemCpy`.

4.1.2.3. Com

The `Rte` uses AUTOSAR `Com` to

► send,

► receive, and

► invalidate

signals and signal groups for inter ECU communication. For this it invokes the `Com` API calls

```
extern uint8 Com_SendSignal
```

```
(Com_SignalIdType SignalId, Com_ApplicationDataRefType SignalDataPtr);
extern uint8 Com_ReceiveSignal
(Com_SignalIdType SignalId, Com_ApplicationDataRefType SignalDataPtr);
extern void Com_UpdateShadowSignal
(Com_SignalIdType SignalId, Com_ApplicationDataRefType SignalDataPtr);
extern uint8 Com_SendSignalGroup
(Com_SignalGroupIdType SignalGroupId);
extern uint8 Com_ReceiveSignalGroup
(Com_SignalGroupIdType SignalGroupId);
extern void Com_ReceiveShadowSignal
(Com_SignalIdType SignalId, Com_ApplicationDataRefType SignalDataPtr);
extern uint8 Com_InvalidateSignal (Com_SignalIdType SignalId);
```

The `Rte` provides callback functions that are invoked by AUTOSAR `Com` to notify the `Rte` of

► data reception,

► transmission acknowledgements,

► transmission preparation,

► signal invalidation or

► timeouts.

The header file *Rte_Cbk.h* contains the prototypes of the callback functions, provided by the `Rte`.

The callbacks generated by the `Rte` are:

```
Rte_COMCbk_<sn>
Rte_COMCbkInv_<sn>
Rte_COMCbkRxTOut_<sn>
Rte_COMCbkTxTOut_<sn>
Rte_COMCbkTAck_<sn>
Rte_COMCbkTErr_<sn>

Rte_COMCbk_<sg>
Rte_COMCbkRxTOut_<sg>
Rte_COMCbkTxTOut_<sg>
Rte_COMCbkTAck_<sg>
Rte_COMCbkTErr_<sg>

Rte_COMCbkTxPrep_<mn>
```

whereas <sn> is the name of a `Com` signal, <sg> is the name of a `Com` signal group and <mn> is the name of the Com_MainFunctionTx instance.

For more detailed information about dependencies of the `Rte` module, see [1].

4.2. Background information

The following chapters will provide you with an understanding of the knowledge necessary to configure and generate the `Rte` module for an ECU. If you are familiar with the concepts of the `Rte`, proceed to Section 4.3, "Configuring and generating the Rte" for step-by-step instructions on how to configure and generate the `Rte` for your ECU(s).

Note that the `Rte` API is provided in a separate document: EB tresos AutoCore module references.

4.2.1. Sender-receiver communication

In sender-receiver communication data is transmitted from a sender to a receiver either *locally* (*intra-ECU communication*) or *between different ECUs* (*inter-ECU communication*).

The data element prototypes to be transmitted over one sender-receiver connection are collected in a sender-receiver interface. The sending software component has a *provide port* for the sender-receiver interface, the receiving software component has a *require port* for the same interface.

For *intra-ECU communication* both ports must be connected by an assembly connector.

For *inter-ECU communication* the data element prototypes of both ports must be mapped to the `Com` signals or system signals that are used for transmission over the bus.

Depending on the semantics of the data element prototype the data is either unqueued or queued.

The *virtual functional bus* (*VFB*) distinguishes two ways of initiating transmission/reception of data in a sender-receiver connection:

▶ *explicit transmission/reception* (Section 4.2.1.1, "Explicit transmission/reception")

▶ *implicit transmission/reception* (Section 4.2.1.2, "Implicit sender-receiver communication")

4.2.1.1. Explicit transmission/reception

Explicit transmission/reception is available for both:

1. unqueued data element prototypes

2. queued data element prototypes

The term *explicit* indicates that the sending/receiving software component explicitly requests transmission/reception of new data by calling the appropriate `Rte` API function.

The `Rte` API functions for explicit transmission/reception actively initiate transmission or reception of the data at the time they are called.

The `Rte` API functions for explicit *transmission* (`Rte_Write()`/`Rte_Send()`) are always non-blocking, i.e. they return immediately after transmission has been initiated.

Depending on the configuration, the `Rte` API functions for explicit *reception* (`Rte_Read()`, `Rte_Receive()`) can be:

▶ blocking[1]: If the `Rte` API function for the reception of data is blocking, it waits for new data to arrive or for a time-out to expire.

▶ non-blocking: If the `Rte` function is non-blocking, it returns immediately:

    ▶ with the last received data (in case of unqueued data element prototypes), or

    ▶ with the first value stored in the reception queue, or

    ▶ with an error, indicating that no new data was available (in case of queued data element prototypes).

See [1], Chapter 4.3.1 for further details.

4.2.1.2. Implicit sender-receiver communication

Implicit transmission/reception is only available for unqueued data element prototypes.

The term *implicit* indicates that no transmission/reception request is initiated while a runnable entity is being executed. Instead, the `Rte` glue code generated initiates the sending and receiving at the beginning and at the end of the task on which the runnable entity is mapped.

While the runnable entity is being executed all read and write accesses of the runnable entity to the data element prototype are redirected to a runnable entity-specific copy of the data element prototype. Therefore no run-time overhead is necessary for ensuring data consistency during access of the copy while the runnable entity is being executed.

For *implicit reception* (`Rte_IRead()`) the `Rte` glue code reads the last received value and stores it in the runnable entity-specific copy before the runnable entity is activated. The `Rte` guarantees that the content of the runnable entity-specific copy does not change while the runnable entity is executed.

---

[1]This is only available for queued data element prototypes.

For *implicit transmission* (`Rte_IWrite()`) the `Rte` glue code initiates sending only at the end of the task on which the runnable entity is mapped. The value to be sent is the last value that was written to the runnable entity-specific copy while the runnable entity was executed.

The status information (success or failure) of implicit transmission or reception *is not available as a return value* of the implicit read and write access `Rte` API functions. To access the status information, use the (separate) `Rte_IStatus()` and `Rte_IFeedback()` `Rte` API functions.

TIP   **Optimizing implicit communication**

It is possible to prevent the Rte from copying the runnable entity-specific buffer to the global visible receive buffer for `Rte_IWrite` or vice-versa for `Rte_IRead` to save runtime and memory.

To achieve this, the following conditions must be fulfilled:

- ► the task priority of all connected readers and writers must be the same

- ► no inter-partition communication is configured

- ► no communication timeout is configured

- ► handle never received status is disabled

- ► invalidation is not used or the invalidation policy is set to `NONE`

- ► update flag is disabled

- ► all readers have the same initial value

- ► no data filter is configured or the data filter is set to `ALWAYS`

In this case `Rte_IWrite` and `Rte_IRead()` will directly write/read to/from the runnable entity-specific buffer. If an explicit writer exists that is connected to an implicit receiver, `Rte_-Write` will also directly write into the runnable entity-specific buffer.

4.2.1.3. Transmission acknowledgment

NOTE   **Transmission acknowledgment informs whether the values have been sent successfully**

The transmission acknowledgment mechanism does not provide any information whether a sent value has been received successfully or not. But it only informs whether sent values have been transmitted successfully.

The `Rte` provides a mechanism for the sending component in a sender-receiver communication to access transmission acknowledgment (`Rte_Feedback()`, `Rte_IFeedback()`), i.e. the information whether a value has been successfully passed to the communication infrastructure or whether a time-out has expired. Transmission acknowledgment is available for explicit and implicit sender-receiver communication. Access to the

transmission acknowledgment is either blocking or non-blocking for explicit communication and non-blocking for implicit communication. See [1], Chapter 4.3.1 for details.

4.2.1.4. Data element invalidation

The `Rte` offers a mechanism to invalidate data element prototypes.

A common use case for data element invalidation is for instance if a sensor delivers invalid data. In this case, the sensor software component is able to inform other application components that no valid data is available by using the data element invalidation feature.

For the sending component, `Rte` API functions are provided for both,

► explicit transmission and

► implicit transmission

to invalidate the *data element prototype*.

For *explicit* transmission (`Rte_Invalidate()`) the explicit invalidation API call - like the explicit transmission API call - actively initiates invalidation of the data element prototype at the time the API call is invoked.

WARNING	**Data loss occurs for implicit transmission/invalidation**
⚠	For implicit transmission/invalidation an invalidation API call can be overwritten by the transmission API call that was called last while the runnable entity was executed. This only applies if both API calls belong to the same data element.
	That means, if an `Rte_IInvalidate()` API call is followed by an `Rte_IWrite()` API call, data is transmitted and no invalidation occurs.
	If an `Rte_IWrite()` call is followed by an `Rte_IInvalidate()` call, invalidation is initiated after the runnable entity terminates.
	In order to avoid data loss ensure that an `Rte_IInvalidate()` API call is not followed by an `Rte_IWrite()` call for the same data element.

For *implicit* transmission (`Rte_IInvalidate()`) the implicit invalidation API call ensures that invalidation is initiated by the `Rte` glue code after the runnable entity has terminated.

For the receiving component, the `Rte` provides a notification mechanism for invalidated data. If the receiving component uses explicit reception, notification about the reception of an invalid value is handled via the return value of the respective `Rte` API function. If implicit reception is used, the invalidation information can be accessed by using `Rte_IStatus()`.

See [1], Chapter 4.3.1 for details.

In addition, when the option **Inter-ECU Invalidation handled by Rte** is enabled, the signal invalidation for inter-ECU communication is handled by the `Rte` as well. For details on how to configure the inter-ECU signal invalidation, see Section 4.3.4.4.5, "Configuring the inter-ECU signal invalidation handled by Rte parameter".

If the option **Inter-ECU Invalidation handled by Rte** is disabled, then the signal invalidation for inter-ECU communication is handled by the `Com` as described in its specification [2].

4.2.1.5. Dirty flag mechanism

The Rte Generator provides a functionality called *dirty flag mechanism* in order to write updated NV data of NVRAM blocks to NV memory. This mechanism interacts directly with the NvM module when an `Rte_Write()` API is invoked by an *atomic software component* which uses a `PortPrototype` typed by an `NvDataInterface`.

For the mapping between the `Rte` configuration and the NvM module, the Generic Editor in the **NVRAM Allocation** tab is used. The reference of the `NvBlockDescriptor` from the `Rte` is mapped to the reference of the `NvMBlockDescriptor` from the NvM module. This mapping determines the value of the block ID.

If the `NvBlockDescriptor` has the attribute supportDirtyFlag assigned, the Rte Generator ignores the `PortApiOption` with `PortDefinedArgumentValue` applied to a `PPortPrototype` of a `NvBlockSwComponentType`. Instead, the Rte Generator uses the block ID value.

4.2.1.5.1. Behavior of the dirty flag mechanism

Depending on the writing strategy of the related `NvBlockDescriptor`, the behavior of the dirty flag mechanism can be described by the following use cases:

- ► Explicit write store at shutdown
- ► Explicit write store immediate
- ► Explicit write store cyclic
- ► Implicit write store at shutdown
- ► Implicit write store immediate
- ► Implicit write store cyclic

4.2.1.5.1.1. Explicit write store at shutdown

After the data accessed by the `Rte_Write()` function is written back to the RAM blocks, the Rte Generator marks the associated RAM blocks as CHANGED by calling the `NvM_SetRamBlockStatus()` function of the NvM module with the BlockChanged parameter set to true. The related RAM blocks are stored during shutdown by the NvM module which calls the `Rte_GetMirror()` callbacks.

Conditions:

▶ The attribute `NvBlockDescriptor.dirtyFlagSupport` is set to true.

▶ The attribute `NvBlockDescriptor.NvBlockNeeds.storeAtShutdown` is set to true.

▶ The `Rte_Write()` API is invoked by an *atomic software component* using a `PortPrototype` with a `NvDataInterface`.

4.2.1.5.1.2. Explicit write store immediate

After the data accessed by the `Rte_Write()` function is written back to the RAM blocks, the Rte Generator immediately writes the RAM blocks to NV memory by calling the `NvM_WriteBlock()` of the NvM module. The call is made in the context of the `NvBlockDescriptor`'s `RunnableEntity`.

Conditions:

▶ The attribute `NvBlockDescriptor.dirtyFlagSupport` is set to true.

▶ The attribute `NvBlockDescriptor.NvBlockNeeds.storeImmediate` is set to true.

▶ The `Rte_Write()` API is invoked by an *atomic software component* using a `PortPrototype` with a `NvDataInterface`.

▶ A `DataReceivedEvent` activates the `RunnableEntity` of the `NvBlockDescriptor`.

4.2.1.5.1.3. Explicit write store cyclic

After the data accessed by the `Rte_Write()` function is written back to the RAM blocks, in the next cycle, the Rte Generator writes the associated RAM blocks to the NV memory by calling the `NvM_WriteBlock()` function of the NvM module. The call is made in the context of the `NvBlockDescriptor`'s `RunnableEntity`.

Conditions:

▶ The attribute `NvBlockDescriptor.dirtyFlagSupport` is set to true.

▶ The attribute `NvBlockDescriptor.NvBlockNeeds.storeCyclic` is set to true.

▶ The `Rte_Write()` API is invoked by an *atomic software component* using a `PortPrototype` with a `NvDataInterface`.

▶ A `TimingEvent` referenced by `NvBlockDescriptor` activates the `RunnableEntity` of the `NvBlock-Descriptor`.

4.2.1.5.1.4. Implicit write store at shutdown

After the data accessed by the `Rte_IWrite()`/`Rte_IWriteRef()` function is written back from the preemption area buffer to the RAM blocks, the Rte Generator marks the associated RAM blocks as CHANGED by

calling the `NvM_SetRamBlockStatus` function of the NvM module with the BlockChanged parameter set to true. The related RAM blocks are stored during shutdown by the NvM module which calls the `Rte_GetMirror()` callbacks.

Conditions:

▶ The attribute `NvBlockDescriptor.dirtyFlagSupport` is set to true.

▶ The attribute `NvBlockDescriptor.NvBlockNeeds.storeAtShutdown` is set to true.

▶ The `Rte_IWrite()`/`Rte_IWriteRef()` API are invoked by aen *atomic software component* using a `PortPrototype` with a `NvDataInterface`.

4.2.1.5.1.5. Implicit write store immediate

After the data accessed by the `Rte_IWrite()`/`Rte_IWriteRef()` functions is written back from the preemption area buffer to the RAM blocks, the Rte Generator writes the RAM blocks to NV memory by calling the `NvM_SetRamBlockStatus()` function of the NvM module. The call is made in the context of the `NvBlockDescriptor`'s `RunnableEntity`.

Conditions:

▶ The attribute `NvBlockDescriptor.dirtyFlagSupport` is set to true.

▶ The attribute `NvBlockDescriptor.NvBlockNeeds.storeImmediate` is set to true.

▶ The `Rte_IWrite()`/`Rte_IWriteRef()` API are invoked by aen *atomic software component* using a `PortPrototype` with a `NvDataInterface`.

▶ A `DataReceivedEvent` activates the `RunnableEntity` of the `NvBlockDescriptor`.

4.2.1.5.1.6. Implicit write store cyclic

After the data accessed by the `Rte_IWrite()`/`Rte_IWriteRef()` function is written back from the preemption area buffer to the RAM blocks, in the next cycle, the Rte Generator writes the associated RAM blocks to the NV memory by calling the `NvM_WriteBlock()` function of the NvM module. The call is made in the context of the `NvBlockDescriptor`'s `RunnableEntity`.

Conditions:

▶ The attribute `NvBlockDescriptor.dirtyFlagSupport` is set to true.

▶ The attribute `NvBlockDescriptor.NvBlockNeeds.storeCyclic` is set to true.

▶ The `Rte_IWrite()`/`Rte_IWriteRef()` API is invoked by an *atomic software component* using a `PortPrototype` with a `NvDataInterface`.

► A `TimingEvent` referenced by `NvBlockDescriptor` activates the `RunnableEntity` of the `NvBlock-Descriptor`.

4.2.2. Inter runnable variables

Inter runnable variables offer a sender-receiver-alike communication mechanism for runnable entities within one *atomic software component*. An inter runnable variable is typed and specifies the way it can be accessed (implicit or explicit). The `Rte` ensures data consistency for accesses to the inter runnable variables.

If a runnable entity accesses an implicit inter runnable variable (`Rte_IrvIRead()`, `Rte_IrvIWrite()`), then the `Rte` provides access to a runnable entity-specific copy of the inter runnable variable.

If a runnable entity accesses an explicit inter runnable variable (`Rte_IrvRead()`, `Rte_IrvWrite()`), then the `Rte` provides access to the shared buffer that represents the inter runnable variable. The access is protected against preemption by other runnable entities that have access to the same inter runnable variable.

See [1], Chapter 4.3.3 for details.

4.2.3. Client/server communication

Client/server communication operations, implemented by a server, are called by a client either

► locally (intra-ECU communication) or

► between different ECUs (inter-ECU communication).

The *operation prototypes*, available in one client/server connection, are aggregated in a client/server interface.

The *server software component* has a *provide port* for the *client/server interface*, the *client software component* has a *require port* for the same interface.

When an *operation prototype* is called, the operation prototype's *argument prototype*s are passed on after the server computation has terminated.

► If they are *invocation argument prototype*s, they are passed on from the client software component to the server software component.

► If they are *result argument prototype*s, they are passed on from the server software component to the client software component, after the server computation has terminated.

► If *application errors* are specified for the operation prototype, the error code is also passed on from the server software component to the client software component after the server computation is done.

The `Rte` stores the incoming calls in a queue, which is provided by a server software component for each operation prototype.

The `Rte` distinguishes two ways of calling a server:

▶ synchronous call, see Section 4.2.3.1, "Synchronous client/server communication"

▶ asynchronous call, see Section 4.2.3.2, "Asynchronous client/server communication"

4.2.3.1. Synchronous client/server communication

If a *client software component* calls an *operation prototype* synchronously (`Rte_Call()`), the `Rte` API call generated for the call blocks

▶ until the server execution has finished and the results are available, or

▶ until an error occurs.

A special case of *synchronous call* is the *direct call*. In a direct call the `Rte_Call()` is implemented as a macro that directly maps the `Rte` API call to the runnable entity that implements the server. If an operation prototype is directly called, the call is not stored in the call queue and the `Rte` does not perform any time-out monitoring. See Section 4.2.17.1.2.1, "No task mapping necessary" for the conditions to directly call an operation prototype. For details, see [1], Chapter 4.3.2 for details.

4.2.3.2. Asynchronous client/server communication

If a client software component calls an operation prototype asynchronously, two `Rte` API calls are generated: one to call the server (`Rte_Call()`) and one to retrieve the result or possible errors (`Rte_Result()`).

The `Rte` API call to call the server does not block, while the `Rte` API call that retrieves the result can be blocking or non-blocking, depending on the configuration. See [1], Chapter 4.3.2 for details.

4.2.3.3. Inter-Ecu client/server communication

If a client software component is mapped to a different Ecu as the server software component, inter-Ecu communication will be implemented by the `Rte`s. When the `Rte_Call()` function is executed by the client, the client's `Rte` will serialize the client's IN and INOUT arguments using the `SomeIp` module into a byte array and send the operation request with the SystemSignal configured within the corresponding ClientServerToSignalMapping.

On the server's side, the server's `Rte` will unpack the request signal that contains the IN and INOUT arguments and enqueue a request in the server's operation request queue. Once the operation request is processed by the server, the server's `Rte` will pack the INOUT and OUT results as well as the server's operation status into

a byte array using the `SomeIp` module and send the result back to the client using the result signal configured within the ClientServerToSignalMapping.

Once the client's `Rte` receives the result signal, it will unpack the byte array using the `SomeIp` module, and signal the client that the operation result is ready.

For both the client and server, the use of SystemSignals to send and receive the IN, INOUT and OUT parameters is not visible. The system design must be aware though that a time-out value will be required for a blocking client in order to avoid an indefinite wait when a signal is not received successfully.

For more information on configuring the ClientServerToSignalMappings within the System Description, see Section 4.2.8, "Data transformation chains".

4.2.3.4. BSW client/server communication

BSW client/server communication mainly differs from SWC client/server communication in the following ways:

► There are no *provide port*/*require port* entities. Instead, `RequiredClientServerEntry`/`Provided-ClientServerEntry` are used.

► Instead of `RunnableEntities`, `BswSchedulableEntity` on the client side and `CalledEntity` on the server side are used.

► The connection between a `RequiredClientServerEntry` and a `ProvidedClientServerEntry` is established by references in a `RteBswRequiredClientServerConnection` in the module configuration. Since there are no ports, there are no assembly connectors either.

For more details, see [6], chapters 6.6 and 7.9.2.5.

4.2.4. Partitioning support

The `Rte` supports mapping of software components to different partitions of an application. This partitioning support is an extension of the EB tresos AutoCore `Rte`.

An application partition can be located on different cores or can use different memory sections of the same ECU. Thus, partitioning enables the usage of the `Rte` in a multicore or a memory protected system.

A partition itself is represented by an `Os` application. For each `Os` application, a separate `Rte` is generated. The `Rte` offers possibilities to communicate between software components on different partitions:

► the Inter `Os` Application Communicator (`Ioc`)

► the Shared Memory Communicator (`Smc`)

► a mix between the `Ioc` and `Smc`

The basic software (BSW) must also be assigned to a single partition. The `Rte` delegates the BSW accesses of the application software components to the BSW partition.

In safety relevant projects, partitioning is a prerequisite to ensure the freedom of interference among application software components and also between application software components and the basic software.

WARNING	**To use the `Rte` in a safety relevant project, you need additional validation processes**
⚠	Additional validation processes are required to use the `Rte` in a safety relevant project since the `Rte` itself is developed according to quality management processes. To find out what this validation process consists of, see the **Maintenance and Support Annex** document. You receive this document together with the product quote supplied by EB.

NOTE	**Partitioning differs in some part from the AUTOSAR 4.0 specification**
ⓘ	The partitioning support is based on the AUTOSAR 4.0 specification but has some vendor-specific extensions. Thus, it is not fully AUTOSAR 4.0 compliant, e.g. the ECU configuration and the required `Ioc` ECU configuration and `Ioc` API differ from the original AUTOSAR 4.0 specification.

4.2.4.1. Communication between partitions

The `Rte` behaves differently depending on the selected approach for the communication between software components which are located in different partitions, i.e. inter-partition communication.

If the `Ioc` is selected for inter-partition commmunication the `Rte` will delegate the communication between software components which are located in different partitions directly to the `Ioc`.

In case the `Smc` is selected for inter-partition communication the `Rte` will use a shared memory approach to allow the communication between software components which are located in different partitions.

If `Mixed` is selected for inter-partition communication the `Rte` will use a shared memory approach for intra-core and the `Os Ioc` for inter-core communication.

In order to abstract from the underlying inter-partition communication mechanism the `Rte` uses inter-partition channels and channel groups which represent a wrapper for the underlying inter-partition communication approach.

Thus, independent of the chosen inter-partition communication mechanism the `Rte` offers the same API to the application.

Therefore the `Rte` generates an additional file called `Rte_Partitioning.h`. This file contains the mapping of the abstract inter-partition channels and channel groups to the corresponding `Ioc` or `Smc` inter-partition channels and channel groups.

NOTE	**It is possible to combine intra- and inter-partition communication**
	Intra- and inter-partition communication can be combined, i.e. it is possible to connect a sender to two receivers if one receiver is located in the same partition as the sender and the other receiver mapped to a different partition. A server can be connected to clients which are located in different partitions.

4.2.4.1.1. Inter-partition communication with the `Ioc`

The `Ioc` module is part of the `Os`.

In order to allow communication between partitions the `Ioc` provides configurable channels and channel groups that can be used to exchange data between `Os` applications.

The **Service Needs Calculator** automatically allocates the required channels and channel groups in the `Os` configuration.

The `Rte` associates the inter-partition channels and channel groups with the corresponding `Ioc` channels and channel groups.

4.2.4.1.2. Inter-partition communication with the Smc

The `Smc` is not a module but a mechanism provided by the `Rte` for inter-partition communication.

If you select the `Smc` for inter-partition communication, the `Rte` applies a shared memory concept.

In the case that two partitions send data to a third partition, the following concepts are realized by the `Rte`:

▶   In order to allow the partitions of the senders to write the data, an auxiliary partition which holds the data is applied.

▶   The partitions of the senders have write access to the auxiliary partition.

▶   The auxiliary partition represents a shared partition with regard to the partitions that provide the same data.

▶   The partition where the data is received can read the data from the shared partition.

If the `Smc` is used for inter-partition communication the `Rte` itself implements the functionality for the inter-partition channels and channel groups.

Hence the **Service Needs Calculator** will not allocate channels or channel groups in the `Os` configuration.

When using the `Smc` the `Rte` generates additional files:

▶   `Rte_Smc.h` - this header file includes the `Smc` header files

▶ `Rte_Smc_Data_<partition>.c` - this source file contains the buffers and queues which are used by the inter-partition channels and channel groups that are provided to the partition with the partition name by the `Smc`.

WARNING	**Using the `Smc` in a multi-core environment**
⚠	The `Smc` supports data consistency mechanisms for inter-core communication (spinlocks). However, it does not support hardware-specific cache handling. To verify that the `Smc` can be used in a multi-core environment, check that a shared memory approach between different cores is applicable and that your hardware either supports cache coherence or that caches are disabled.

4.2.4.2. Sender/receiver communication

For sender-receiver communication, the following communication models exist in conjunction with partitioning:

▶ Intra-ECU/intra-partition communication

▶ Intra-ECU/inter-partition communication

▶ Inter-ECU/intra-partition communication

▶ Inter-ECU/inter-partition communication

The `Rte` uses the corresponding inter-partition channel for inter-partition sender-receiver communication.

4.2.4.2.1. Intra-ECU communication

Intra-ECU/intra-partition communication behaves as intra-ECU communication with disabled partitioning support.

Intra-ECU/inter-partition communication requires inter-partition channels. For the inter-partition communication, the `Rte` uses the corresponding inter-partition channel to facilitate the communication between partitions. Sender/receiver intra-ECU/inter-partition communication can be either queued or unqueued.

In case of queued intra-ECU/inter-partition communication, the inter-partition channel provides a queue and handles the queuing and dequeuing.

4.2.4.2.2. Inter-ECU communication

Inter-ECU communication requires the `Com`. When partitioning is enabled, both have to be mapped to partitions: software components as well as the BSW and therefore the `Com` module, too.

In case of inter-ECU/intra-partition communication, the software component is mapped to the same partition as the BSW. Therefore the `Rte` can directly delegate inter-ECU communication to the `Com` module.

Inter-ECU/inter-partition communication requires inter-partition channels since the sending or receiving software component is mapped to a different partition than the BSW.

Two inter-partition channels are used by the `Rte` for inter-partition communication between the software component's partition and the partition to which the BSW is mapped.

The first inter-partition channel is used to transmit the data to be sent externally by the `Com`. The second inter-partition channel is used to transmit the signal or signal group ID, which corresponds to the data to be sent externally.

4.2.4.3. Client/server communication

For client/server communication, the following communication models exist in conjunction with partitioning:

► Intra-ECU/intra-partition communication

► Intra-ECU/inter-partition communication

The `Rte` delegates inter-partition client/server communication to inter-partition channel groups.

4.2.4.3.1. Intra-ECU communication

Intra-ECU/intra-partition communication behaves as intra-ECU communication with disabled partitioning support.

Intra-ECU/inter-partition communication requires two inter-partition channel groups, one for transferring the request from the client to the server, and one for transferring the result from the server back to the client.

Each inter-partition channel group contains group channels for each parameter used in the corresponding client/server communication.

4.2.4.4. Mode switch communication

For mode switch communication, the following communication models exist in conjunction with partitioning:

► Intra-ECU/intra-partition communication

► Intra-ECU/inter-partition/inter-core communication

For inter-partition mode management the mode providers of a mode machine instance have to reside in the same partition.

All mode users of a mode machine instance, which use inter-partition mode management, have to reside in the same partition.

A different set of IP channels is used to facilitate the inter-partition mode management. This depends on the number of mode switch events and mode disabling dependencies.

4.2.5. Partitioning of BSW modules

Up until AUTOSAR 4.1.1, AUTOSAR made the assumption that all BSW modules run inside a single `OsApplication`. This constraint simplifies the specification as well as the implementation of the AUTOSAR BSW stack considerably. The move to partition the BSW stack over multiple `OsApplications` was caused by the necessity to exploit possible gains of multi-core platforms as well as provide freedom of interference between BSW modules.

This section provides some background information regarding the partitioning of BSW modules over multiple partitions as well as required changes to their BSW module descriptions and `Rte` configuration.

4.2.5.1. Multiple instantiation and BSW distribution

The AUTOSAR meta model provides two methods to enable BSW modules to run in multiple partitions:

► BSW distribution

► Multiple instantiation

With the BSW distribution method, a BSW module is not bound to a specific partition. The `BswEvent` to task mappings determine which partitions the BSW module shall run inside, i.e. based on the partitions that the tasks are mapped to. A BSW module may define one or more `BswDistinguishedPartitions`, which are in effect virtual partitions that are mapped to `OsApplications` after the basic software module description is created. The BSW modules can optionally limit the use of artifacts defined within its basic software module description to these `BswDistinguishedPartitions` via a context limitation. The BSW Scheduler (`SchM`) uses this information to determine if inter-partition/inter-core communication exists and adapts the provided `SchM` API functions accordingly.

The advantage of the BSW distribution method is that for the user, only one instance of a BSW module exists. The BSW module must determine at run-time which partition it is currently running in and perform the corresponding behavior. The disadvantage of this method is that the BSW module description may be more difficult to create since context limitations may need to be defined for basic software module description artifacts.

Example: When a basic software main function shall run in three partitions, one `BswTimingEvent` per partition shall be defined within the basic software module description. Each `BswTimingEvent` must be mapped to a task which belongs to the target partition. To determine which partition the main function runs in the `Os` API, `GetApplicationID(void)` is used.

```
void Module_MainFunction(void)
```

```
{
  switch(GetApplicationID())
  {
    case Partition1:
    /* Do actions for partition 1 here.*/
    break;
    case Partition2:
    /* Do actions for partition 2 here.*/
    break;
}
```

The second method of BSW partitioning specified by AUTOSAR is the multiple instantiation method. Here the BSW module description provides one `BswImplementation` per instance. Each instance is explicitly mapped to a partition within the `Rte` ECU configuration. For the Bsw Scheduler, each `BswImplementation` is in effect a separate BSW module. The provided `SchM` API functions is not shared between the instances and the called `BswModuleEntry` instances also have unique names.

The advantage of this method is that there is a clear separation of a BSW module into instances that are assigned to separate partitions:

```
void Module_1_Partition1_MainFunction(void)
{
  /* Do actions for partition 1 here.*/
}



void Module_1_Partition2_MainFunction(void)
{
  /* Do actions for partition 2 here.*/
}
```

NOTE	**Important Note**
(i)	As of ACG-7.6, only the multiple instantiation method is supported by the Bsw Scheduler.

4.2.5.2. Impact on the Bsw module description

In this section the required changes to a BSW module's basic software module description is described.

The BSW module description template document defines the three-layer approach to the basic software module description. The first layer is the top-level `BswModuleDescription`. Each AUTOSAR BSW module shall create exactly one `BswModuleDescription` element. The second layer is the `BswInternalBehavior`. The `BswModuleDescription` defines, among other things, one or more `BswInternalBehaviors`. Within the `BswInternalBehavior`, the entry points into a BSW module instance, the `BswSchedulableEntities`, are defined as well as the events that trigger them. The third layer of the basic software module description is the `BswImplementation`. Each `BswImplementation` represents a separate instance of a BSW module. A `BswImplementation` must reference a single `BswInternalBehavior`. These layers are typically combined in the following ways:

▶ One `BswModuleDescription`, one `BswInternalBehavior`, and one `BswImplementation`. This is the architecture for BSW modules that have one instance.

▶ One `BswModuleDescription`, one `BswInternalBehavior`, and two `BswImplementations`. Two instances of a BSW module have the exact same behavior, e.g. the left-door, right-door architecture.

▶ One `BswModuleDescription`, two `BswInternalBehaviors`, and two `BswImplementations`. Two instances of a BSW module have different behavior.

In order to ensure that the generated `SchM` API functions have unique names, the `BswImplementations` must define the `VendorId` and `VendorApiInfix` attributes. When the `VendorId` is set to `1` and the `VendorApiInfix` is set to `VariantA` the following artifacts contain a infix `_1_VariantA`:

C artifact	Without infix	With infix
Bsw Module Interlink Header File	`SchM_Mod.h`	`SchM_1_VariantA_Mod.h`
SchM API function	`SchM_Read_Mod_Port()`	`SchM_Read_Mod_1_VariantA_Port()`
BswModuleEntry	`Mod_MainFunction()`	`Mod_1_VariantA_MainFunction()`

4.2.5.2.1. Shared configuration artifacts

The `BswModuleDescription` may contain some artifacts that may be shared between instances:

▶ Public API:

    ▶ `ProvidedEntrys`

    ▶ `RequiredEntrys`

The referenced `BswModuleEntrys` may be provided by multiple instances of a BSW module. When the `VendorId` and `VendorApiInfix` are defined by a `BswImplementation`, these function names contain the infix `_{VendorId}_{VendorApiInfix}` to ensure that there are no duplicate implementations of the same function. If an instance of a BSW module does not provide a `BswModuleEntry`, the corresponding function must not be defined.

► Communication interfaces:

   ► `CalledEntrys`

   ► `ProvidedDatas`

   ► `RequiredDatas`

   ► `ReleasedTriggers`

   ► `RequiredTriggers`

These artifacts are referenced by `BswSchedulableEntities`, which are defined within the individual `BswInternalBehaviors`. The Bsw Scheduler (`SchM`) provides the corresponding API function to access these interfaces when the instance's `BswInternalBehavior` references a communication interface. No `SchM` API function is generated for an instance if the communication interface is not referenced. This behavior ensures that no unneeded API functions are generated.

4.2.5.2.2. Instance-specific artifacts

Artifacts defined within the `BswInternalBehavior` belong only to the instances of a BSW module where their `BswImplementation` references that `BswInternalBehavior`. As a result, the sharing of `ExclusiveAreas` is not possible between instances of a BSW module.

4.2.5.2.3. Modeling shared code

If multiple instances of a BSW module are required, but commonly used code shall be shared between the instances, a shared `BswImplementation` must be created. For this `BswImplementation`, no `VendorApiInfix` must be defined. Also, if this shared code shall also be called by the `Bsw Scheduler`, the associated `BswEvent` may only be called in the context of a single `OsApplication`. Commonly, this `OsApplication` is assigned to the ECU's master core. Alternatively, this shared `BswInternalBehavior` may have no `BswEvent`s defined within it. In this case the functions provided by it are basically library functions.

As a general rule, shared code shall not modify shared data. This may lead to memory protection errors or in case of multi-core platforms, cache coherency issues. Communication between BSW module instances shall be done via the `Bsw Scheduler` only.

4.2.5.3. Rte Editor configuration

After you import the `BswImplementation` instances, you can see the instances of a BSW module within the **Implementation Selection** tab's **Basic software module instances** table. The BSW module instance name is a default name used within the `Rte` ECU configuration. The BSW module description path is the same for

each instance of a BSW module. The BSW module implementation corresponds to the path to the instance's `BswImplementation`.

The assignment of BSW module instances to `OsApplications` is done within the **Partitioning** tab. The group **Bsw partitioning** contains a table with the columns **Bsw module instance name** and **Os application**. The names in the first column correspond to the BSW module instance names found in the **Implementation Selection** tab. The **Os application** column contains drop-down list boxes for the desired `OsApplication`. Each BSW module instance must be assigned to an `OsApplication` when partitioning is enabled.

When partitioning is enabled, the `BswEvent` instances must be mapped to tasks that are assigned to the `OsApplication` that the BSW module instance is mapped to. Mapping a BSW event to a task that is assigned to another `OsApplication` leads to a configuration error.

On multi-core systems, `ExclusiveAreas` used by shared code that can run on multiple cores (i.e. defined within a shared `BswImplementation`) must use spinlocks for their implementation mechanism. This behavior can be configured within the **Exclusive Areas** area tab.

Other configuration steps, such as the mapping of provided and required mode groups are not done differently for multiple instances of a BSW module.

4.2.6. Indirect API

The `Rte` provides an alternative way to call port-related API functions, i.e. all API functions that are related to

► sender-receiver,

► client/server or

► mode switch communication.

Instead of directly calling the port-oriented `Rte` API functions, the indirect API offers a means to call the `Rte` API functions indirectly through a *port handle*. This mechanism is useful if several ports of the same usage type (require or provide) have the same interface and the same operation is going to be performed for all these ports. See [1], Section 5.2.3 for details.

To support the indirect API, the `Rte` provides `Rte` API functions to

► access an array of all port handles belonging to ports of a given interface type and provide/require usage (`Rte_Ports()`),

► determine the number of ports in such a port array (`Rte_NPorts()`),

► indirectly access a single port (`Rte_Port()`).

4.2.7. Exclusive areas

Exclusive areas are a data consistency mechanism that the `Rte` provides for executable entities, i.e. for the runnable entities of a software component (SWC) and for basic software module entities of a basic software module (BSWM).

If executable entities have simultaneous access to shared resources, e.g. runnable entities have access to *per instance memory*, a data consistency mechanism may be needed. An executable entity that runs within an *exclusive area* cannot be preempted by other executable entities, which have access to the same exclusive area.

The `Rte` distinguishes two ways of using exclusive areas:

► Implicit access:

   The entire execution of the executable entity is protected by the exclusive area. The `Rte` glue code enters the exclusive area before the executable entity is activated and leaves the exclusive area after the executable entity has terminated.

► Explicit access:

   Executable entities explicitly enter and leave exclusive areas by calling the appropriate `Rte` API functions in the case of SWCs or by calling the appropriate `BSW Scheduler` API functions in the case of BSWMs.

The `Rte` provides different implementation mechanisms for exclusive areas. For more information, see [Section 4.2.17.2.5, "Data consistency for exclusive areas"](#)

NOTE **Exclusive areas cannot be shared between runnable entities and basic software module entities**

The `Rte` cannot provide the access to one specific exclusive area to both a runnable entity and a basic software module entity.

Either only runnable entities or only basic software module entities have access to a specific *exclusive area*.

4.2.7.1. Exclusive areas for software components

The `Rte` provides exclusive areas for the software components.

For explicitly accessing an exclusive area, the `Rte` provides following API functions:

► `Rte_Enter()` for entering the exclusive area
► `Rte_Exit()` for exiting the exclusive area

WARNING	**Blocked runnable entities cause errors of the operating system**
⚠	If a category 2 runnable entity has implicit access to an exclusive area and then gets blocked, the runnable entity causes errors of the operating system.
	In order to avoid errors of the operating system, ensure that category 2 runnable entities do not have implicit access to exclusive areas.
	See Section 4.2.17.1.1, "Executable entity categories" for details on runnable entities' categories.

4.2.7.2. Exclusive areas for basic software modules

The BSW scheduler of the `Rte` provides exclusive areas for the basic software modules.

For explicitly accessing an exclusive area, the BSW scheduler of the `Rte` provides following API functions:

▶ `SchM_Enter()` for entering the exclusive area

▶ `SchM_Exit()` for exiting the exclusive area

WARNING	**The basic software might not run correctly on the ECU**
⚠	If you set the implementation mechanism of an exclusive area, that is associated with a basic software module entity, to **All Interrupt Blocking**, the basic software will always run correctly on the ECU. If you set that implementation mechanism to any other value, the basic software might not run correctly on your ECU.

NOTE	**`Os` generation fails if `Os` resources are used within category 1 interrupts**
ⓘ	If category 1 interrupts are configured in the `Os` module, keep in mind, that the basic software modules use the BSW scheduler of the `Rte` (`SchM`) to protect their critical sections. As the calling context would be within the category 1 interrupt, resources shall not be used in this case. For details see also [3] chapter *Permitted calling context*.

Further information about the exclusive areas that can be used for the basic software modules is provided in EB tresos AutoCore Generic documentation.

To configure the implementation mechanism of an exclusive area, see Section 4.3.4.8, "Configuring exclusive areas".

4.2.8. Data transformation chains

The AUTOSAR release 4.1.2 introduced a new mechanism for transmitting sender-receiver and client-server data between ECUs: data transformation chains. When the `Rte` is configured with the associated data trans-

formation chain, the `Rte` transforms software component data with one or more transformers. For more information on data transformation chains, see the chapter *Data Transformation* of document [4] as well as document [5].

AUTOSAR defines different types of data transformation. If you want to learn more about the concept of data transformation as supported by EB, see chapter *Data transformation* of the EB tresos AutoCore Generic 8 documentation.

Data transformation chains work similar to Unix pipelines; the `Rte` uses the output of one transformer as the input to the next transformer. When a transformed signal is received on another ECU, the original transformation chain is reversed by the `Rte`, so that the original data can be passed onto the receiving software component.

The actual use of transformers is not visible to the software component that sends or receives the data; the same `Rte` API functions `Rte_Read`, `Rte_Write`, and `Rte_Call` are used as before. However, if the error status of a data transformation chain is required by a software component, you can configure a transformation error argument for the Rte API function within the software component description's `PortApiOption`. The transformation errors are grouped into soft and hard errors. Soft errors are basically warnings given during transformation. If a hard error occurs during transformation though, the `Rte` does not send a signal since the resulting data is not valid.

4.2.8.1. Specification within the system description

In the following paragraphs the specification of data transformers within the system description is described.

`SenderReceiverToSignalMapping`

A `SenderReceiverToSignalMapping`, like other types of data mappings, maps a specific variable data prototype to a `SystemSignal` used to send the data. If a connected port is located on another ECU, i.e. the port's software component is mapped to a different ECU, the `Rte` automatically routes the data over the associated `SystemSignal`.

A `SenderReceiveToSignalMapping` can now be also used to transmit complex data as well as primitive data. The use of data transformation chains simplifies the system description considerably since complex data subelement mappings are no longer required to be defined.

`ClientServerToSignalMapping`

For client-server communication a `ClientServerToSignalMapping` is used. This data mapping maps a client-server operation of a port to a *call* `SystemSignal` and a *result* `SystemSignal`. Again, the `ClientServerToSignalMapping` simplifies the configuration of inter-ECU client-server communication because the operation arguments do not need to be mapped to individual `SystemSignals`, which must then mapped to a `SystemSignalGroup`.

Inter-ECU client-server communication is only supported in combination with data transformation chains.

`SystemSignals` and `ISignals`

As previously mentioned, a `ClientServerToSignalMapping` or `SenderReceiverToSignalMapping` reference `SystemSignals` to be used to transmit the software component data. The low-level description of a signal is specified by an `ISignal`, which then references the high-level `SystemSignal` that it shall be associated to. So, when a variable data prototype which is to be sent by the `Rte` is mapped to a `SystemSignal`, the `Rte` sends all `ISignals` that reference that `SystemSignal`. If an `ISignal` also references a data transformation chain, the `Rte` uses this data transformation chain to serialize and transform the software component data before it sends the `ISignal`.

`DataTransformationChain`

A `DataTransformationChain` defines an ordered list of transformers that are to be used to serialize the software component data, i.e. flatten the data into a byte array. The `Rte` uses the data transformation chain data to determine which transformers shall be used, and in which order. The actual implementations of the transformers is determined by the corresponding transformer generators which have been added to the EB tresos Studio project.

Transformer

A transformer defines a standardized protocol for transforming data. In addition, the transformer definition describes the buffer properties required to transform the data.

4.2.8.2. Specification of the ECU configurations

The description of a transformer is defined within the system description. To generate the implementations though, the corresponding transformer generator must be configured. For each transformer function which is used by the `Rte`, one `XfrmImplementationMapping` must be defined in the transformer's ECU configuration. The implementation mapping maps the function provided by the transformer generator (i.e. the `BswModuleEntry`) with the `SystemSignal` that is sent or received by the `Rte`. For more information on how to configure transformer generators, see AUTOSAR document [5].

4.2.9. Per instance memory

Per instance memory sections are instance-specific, typed blocks of memory. All runnable entities of a software component's instance can access the *per instance memory* (`Rte_Pim()`). However no data consistency is guaranteed for accesses to the per instance memory. For that purpose *exclusive areas* (Section 4.2.7, "Exclusive areas") are used, for example.

4.2.10. Mode management

The `Rte` and the `BSW Scheduler` support mode-dependent execution of runnable entities/schedulable entities within software components/basic software modules. Changes in the current mode are transmitted to the

software component or basic software module via a sender-receiver-like communication mechanism with a dedicated `Rte` API call (`Rte_Switch()`) for software components or to basic software modules with the API call (`SchM_Switch()`).

The `Rte` and the `BSW Scheduler` offers two ways of reacting to changes in the current mode:

▶ *mode switch event*s

▶ *mode disabling dependencies*

Furthermore, the `Rte`/`BSW Scheduler` offers an API call to request the currently active mode (`Rte_Mode()`/ `SchM_Mode()`). Additionally, the `Rte` provides the enhanced mode API that provides the current, previous and next mode.

Mode switch events can trigger executable entities

▶ when leaving the previous mode,

▶ when entering the new mode, or

▶ on the transition between two modes.

Mode disabling dependencies can disable `Rte`/BSW events while certain modes are active. This means that the runnable entities/schedulable entities that would normally be triggered by these `Rte`/BSW events are not triggered while the respective mode is active and that wait points are not resolved while the mode is active.

The `Rte` distinguishes between two mode switch procedures:

▶ Synchronous mode switching procedure

▶ Asynchronous mode switching procedure

4.2.10.1. Synchronous Mode Switch

If no mode switch events are configured, the mode transition is done inside the `Rte_Switch` API .

4.2.10.2. Asynchronous Mode Switch

The RTE generator supports invocation of runnable entities via direct function call if all of the following conditions are fulfilled:

▶ Asynchronous mode switch behavior is configured.

▶ The runnable entities do not define a minimum start distance.

▶ The mode manager and mode user are in the same partition.

▶ The events should either be mapped to no task or all of them must be mapped to the task of the `Rte_-Switch` API.

See [1], Chapter 4.4 for details on mode management.

4.2.11. Triggers

AUTOSAR 4.0 supports the execution of basic software (BSW) module entities and software components (SWC) by the definition of internal and external trigger events. These trigger mechanisms can be used to decompose the execution sequence of a BSW module (BSWM) entity or a SWC into multiple units. Each with its own priority. When working in a high-priority task or interrupt, low-priority operations can be executed asynchronously. As a result, the task queue is free for more important work.

The `Rte` distinguishes between external and internal triggers. An external trigger can be used to activate a BSW schedulable entity/SWC runnable entity of the same or a different BSWM instance/SWC instance. An internal trigger can be used to activate a BSW schedulable entity/SWC runnable entity of the same BSWM instance/SWC instance.

Triggers can activate a BSW schedulable entity/SWC runnable entity either synchronously (by a direct function call) or asynchronously (by activating an Os task).

4.2.11.1. External Triggering

The external triggering can be used to request the activation of a BSW schedulable entity/SWC runnable entity by a BSW schedulable entity/SWC runnable entity of another BSWM instance/SWC instance or the same BSWM instance/SWC instance. The call of the `SchM_Trigger()`/`Rte_Trigger()` API associated to a released trigger activates the BSW schedulable entities/SWC runnable entities of the connected required trigger.

4.2.11.2. Internal Triggering

The internal triggering can be used to request the activation of a BSW schedulable entity/ SWC runnable entity by a BSW schedulable entity/SWC runnable entity of the same BSWM instance/SWC instance. The call of the `SchM_ActMainFunction()` API/`Rte_IrTrigger()` activates the BSW/SWC *internal trigger occurred events* which reference the associated activation point. Activation points of other BSWM instances/SWC instances must not be referenced.

4.2.11.3. Synchronous Activation

The RTE generator supports invocation of triggered BSW schedulable entities/SWC runnable entities by direct function call, if all of the following conditions are fulfilled:

► the triggered schedulable entities/runnable entities do not define a minimum start distance

► For BSW triggers, no BSW trigger direct implementation is defined in the affected basic software module description

► the triggered schedulable entities/runnable entities are not mapped to an execution context with more permissions than the caller.

The supported combinations of execution contexts are listed in detail in table 4.5. in [1].

Note: Typically the feature of inter BSW module entity/SWC triggering is used to decouple the execution context of BSW entities. However, the performance of this decoupling mechanism highly depends on the applied scheduling concept. For example, a direct function call may block several other tasks if the call is made from a task with a high priority.

4.2.11.4. Asynchronous Activation

BSWM/SWC trigger sources have two options to activate BSW schedulables/SWC runnables.

The first option is that the BSWM/SWC *trigger source* calls the `SchM_Trigger()` API/`Rte_Trigger()` API. The call of the `SchM_Trigger()` API/ `Rte_Trigger()` activates all BSW schedulable entities/SWC runnable entities that are triggered by *external trigger occurred events* which are 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.

The second option is that for the BSW modules, the BSWM *trigger source* directly takes care about the activation of the particular `Os` task to which the *external trigger occurred events* of the triggered BSW schedulable entities are mapped. In this case the *trigger source* has to define a basic software trigger direct implementation. The name of the `Os` tasks used is annotated by the task attribute. If a basic software trigger direct implementation is defined, no `SchM_Trigger()` API is generated by the `Rte` generator.

4.2.11.5. Inter-partition Queued Triggering

The BSWM/SWC trigger sources can activate BSW schedulable entities/SWC runnable entities that are in different memory partitions than the trigger source. This is called inter-partition triggering.

NOTE

ⓘ

Configure a maximum activation count for basic tasks with lower priority

For inter-partition queued triggering, every time the trigger API is called, the `Os` task activations are queued if the `Os` task type is basic and has a lower priority than the trigger source `Os` task, so the `Os` configuration should have a suitable maximum activation count.

4.2.12. Calibration

The `Rte` supports the allocation of *calibration parameter*s and provides an API (application programming interface) to access them. Calibration parameters can be defined within one atomic software component or within a *parameter software component*.

Calibration parameters defined for an atomic software component can only be used within the atomic software component. Several instances of the same atomic software component type can either share the same instance of a calibration parameter or use their own instances. The `Rte` provides the API `Rte_CData()` to access the calibration parameters.

A parameter software component has ports with a parameter interface. The parameter interface defines calibration parameters which are provided to other atomic software components. Ports of atomic software components can be connected to the parameter components via assembly connectors. The `Rte` provides the `Rte_-Prm()` API to read the calibration parameters.

The `Rte` also supports the online calibration process. During the online calibration process, the calibration parameters are changed invisibly for the application. The following methods are supported:

1. Double pointered

2. Single pointered

3. Initialized RAM

1. *Double pointered method*: When the double pointered method is used, the Rte Generator allocates a calibration reference table (array of void pointers) in ROM. The calibration reference table has pointers to groups of calibration parameters also located in ROM. A calibration parameter group is struct and contains all calibration parameters of a component that you have assigned to the same memory section. The memory section assignment is determined by evaluating the `SwAddrMethod` reference in the `SwDataDef-Props` attributes of the calibration parameter in the software component description. A base reference (pointer to array) located in the RAM points to the reference table. The `Rte` API accesses the calibration parameters through double pointer indirection. Double indirection in this case means that the `Rte` first dereferences the base pointer to the calibration base pointer and then dereferences the pointer to the calibration reference table. The label of the base reference is `CONSTP2CONST(void, AUTOMATIC, RTE_-CONST) (*RteParameterBase)[i]`, where `i` is the number of calibration parameter groups within the reference. This label is exported and is made available to other software on the ECU. During the calibration process, you may allocate another calibration reference table in RAM. You may change the entries of the table so that they point to the modified calibration parameters in RAM. Finally you can switch content of the base reference to the new calibration reference table so that the calibration parameters referenced by the new calibration reference table are used.

2. *Single pointered method*: For the single pointered method, there is one calibration reference table in RAM (array of void pointers). The table references groups of calibration parameters, which are located in ROM. The groups are allocated in the same way as in the double pointered method. The `Rte` API accesses the calibration parameters indirectly via this table. The label of the reference table is `P2CONST(void, AUTOMATIC, RTE_VAR) RteParameterRefTab[i]`, where `i` is the number of calibration parameter

groups. This label is exported and is available for other software on the ECU. When the calibration process is carried out, you can allocate new calibration parameters in RAM and you can change the corresponding entry in the calibration reference table to setup a redirection to the new calibration parameter.

3. Initialized RAM: In case of initialized RAM method, the Rte Generator allocates a variable in RAM for each calibration parameter. The `Rte` API directly accesses this variable without any indirection. During the calibration process, you can change each calibration parameter individually by directly modifying the value of the RAM variable.

   If calibration support is switched off, the Rte Generator allocates all calibration parameters in ROM. The `Rte` API accesses them without any indirection.

4.2.12.1. Calibration parameters in NVRAM

In case of online calibration support is switched off, the `Rte` supports the mapping of calibration parameters to NVRAM blocks. In order to map a calibration parameter to an NVRAM block, the calibration parameter

▶ must be defined within an atomic software component, and

▶ has to be a per instance calibration parameter.

Moreover, per instance memory has to be defined which is used as a RAM mirror for the NVRAM block. The data type used for the per instance memory must be the same as for the per instance calibration parameter.

The `Rte` then allocates a ROM constant which contains the default value for the calibration parameter and a RAM variable for the per instance memory which is used by the NVRAM manager as mirror. The application can then access the calibration parameter using the `Rte_CData()` API.

4.2.13. Array passing scheme

In AUTOSAR 4.0, all input parameters that are an *array implementation data type* are passed as a *pointer to the array base* type. In AUTOSAR 3.x two different array passing schemes exist for `Rte` API functions:

▶ *pointer to the array base* type

▶ *pointer to the array* type

To easily integrate AUTOSAR 3.x software components into an AUTOSAR 4.0 `Rte`, the `Rte` supports both array passing schemes. On a component type level, it is possible to call `Rte` API functions with an array passing scheme that differs i.e. from a component A to a component B.

NOTE	The `Rte` passes arrays or strings as pointer to the array type
(i)	Internally, the `Rte` passes arrays or strings as pointer to the array type.

The array passing scheme can be configured for a component in its source file using the macro `RTE_-PTR2ARRAYTYPE_PASSING`. If the macro `RTE_PTR2ARRAYTYPE_PASSING` is defined in a component's source file, arrays are passed to `Rte` API functions as *pointer to the array* type.

If for a component arrays shall be passed to `Rte` API functions as pointer to the array type, it is necessary that the macro `RTE_PTR2ARRAYTYPE_PASSING` is defined in the component's source file before the inclusion of the component's header file.

If no macro is defined, the `Rte` API is AUTOSAR 4.0 compliant which means that *pointer to the array base* type is used.

4.2.14. Initialization and finalization

WARNING	The `Rte` ignores incoming transmissions via the `Com` module
⚠	No `Rte` functionality is available before the `Rte_Start()` function is called. If `Rte_-Start()` has not been called, the `Rte` ignores incoming transmissions via the `Com` module. If the application depends on ignored incoming `Com` data, the application is not initialized correctly.
	To avoid that the `Rte` ignores all incoming transmissions via the `Com` module, call `Rte_-Start()` before the `Com` module starts receiving transmissions.

The `Rte` provides dedicated API functions for initializing and finalizing the `Rte` and the `BSW Scheduler`, namely `Rte_Start()`, `Rte_Stop()`, `SchM_Init()`, `SchM_Deinit()`, `Rte_Init_<InitContainer>()` and `Rte_StartTiming()`. The *ECU state manager* calls these functions if it is present on the ECU. See EB tresos AutoCore Generic Mode Management documentation for details about the ECU state manager.

If no ECU state manager is available, the integrator must provide a specific init task. If an init task is used, its code has to include the `Rte_Main.h` header file, which contains the prototypes of the `Rte_Start()`, `Rte_-Stop()`, `SchM_Init()`, `SchM_Deinit()`, `Rte_Init_<InitContainer>()` and `Rte_StartTiming()` API functions.

4.2.15. Activation of executable entities

Executable entities are a generalization of runnable entities (which are activated by Rte events) and schedulable entities (which are activated by BSW events). Activating executable entities is the second main feature of

---

the `Rte` besides providing the communication infrastructure for software components. The `Rte` provides the task context for the execution of executable entities. It is during ECU configuration of the `Rte` that you configure which executable entity is going to be executed in which task, see Section 4.3.4, "Configuring the Rte".

NOTE	**Runnable entities and schedulable entities have to be mapped to a task**
(i)	All executable entities known to the `Rte` have to be mapped to a task. This includes runnable entities of AUTOSAR services that use `Rte` API functions. See Section 4.2.17.1.2.1, "No task mapping necessary" for exceptions. If an executable entity is not mapped to a task but a mapping is required, the Rte Generator will report an error.

4.2.16. VFB (virtual functional bus) tracing

The `Rte` and `BSW Scheduler` offer a tracing mechanism to the software components and basic software modules, the *VFB tracing*. The VFB tracing distinguishes between specified events and offers a series of hook functions that are invoked whenever a specified event occurs. Specified events include:

▶ start and return of `Rte` API functions

▶ interaction with the `Com` module, in particular:

    ▶ sending of signals,

    ▶ receiving of signal, and

    ▶ calling `Com` callback functions

▶ interaction with the `Os`, in particular:

    ▶ task activation, termination and dispatch,

    ▶ setting,

    ▶ waiting for and

    ▶ receiving `Os` events

    ▶ task chain start and end (see Section 4.2.22, "Task Chains" for more information)

▶ calling and finishing of runnable entities and basic software schedulable entities

It is your task as user to implement the trace hook functions. See [1], Chapter 5.10 for details on the VFB tracing mechanism. Note that the trace book functions for the basic software scheduler and for the task termination are not specified by AUTOSAR but are EB-specific.

4.2.17. `Rte` implementation-specific details

The following chapters provide you with implementation-specific details of the EB tresos AutoCore `Rte`. Here some of the basic concepts are explained, which are necessary to understand the chapters that follow.

If you decide to proceed to [Section 4.2.18, "Generation modes"](#) immediately, you may want to make a note of the concepts explained here so that you may refer back to these basic ideas later on.

4.2.17.1. Executable entity to task mappings

4.2.17.1.1. Executable entity categories

The *category* of an executable entity is not configurable but is implicitly determined by the *type of used access* and *call points* or the *type of events which activate it*.

The EB tresos AutoCore `Rte` assigns the categories under the following conditions:

Category 1A	The executable entity is of category 1A if
	▶ it is activated by a `Rte`/BSW timing event
	or it makes use of
	▶ implicit data read access
	▶ implicit data write access
	▶ implicit exclusive areas
	▶ implicit inter runnable variables
Category 1B	The executable entity is of category 1B if
	▶ it is activated by a `Rte`/BSW timing event and
	▶ has explicit usage of at least one exclusive area
	or it has at least one
	▶ data receive point without wait point
	▶ data send point without wait point
	▶ explicit usage of exclusive area
	▶ explicit access to inter runnable variable
	▶ synchronous server call point without time-out
	▶ asynchronous server call point without wait point and without time-out
Category 2	The runnable entity is of category 2 if it has at least one
	▶ directly called executable of category 2

►   synchronous inter-ECU call

►   synchronous inter-partition-inter-core call

►   synchronous inter-partition call where at least one client runnable is mapped to a non-preemptable task

►   synchronous intra\inter-partition call where the server has equal or lower priority than the client

►   synchronous non-directly called server which is cyclically triggered

►   synchronous non-direct-call of a server of category 2

►   synchronous server call point with time-out

►   asynchronous server call point with wait point or time-out

►   data receive point with wait point

►   data send point with wait point

4.2.17.1.2. Task mapping scenarios

This chapter summarizes the constraints of possible task mappings.

There are three cases for which no task mapping is necessary, see Section 4.2.17.1.2.1, "No task mapping necessary".

For all other cases, the EB tresos AutoCore `Rte` defines five different mapping scenarios:

►   Scenario A1, see Section 4.2.17.1.2.2, "Scenario A1"

►   Scenario A2, see Section 4.2.17.1.2.3, "Scenario A2"

►   Scenario B1, see Section 4.2.17.1.2.4, "Scenario B1"

►   Scenario B2, see Section 4.2.17.1.2.5, "Scenario B2"

NOTE	`Rte` **and BSW events are mapped to tasks**
	In the `Rte` ECU configuration, it is not the executable entities which are mapped to tasks, but the `Rte` and BSW events. Thus, if an executable entity is triggered by multiple events, each event must be mapped to a task individually. If the executable entity can be invoked concurrently, the triggering events may also be mapped to different tasks.

If you want to map an executable entity to a task, you need to define an `Rte` or BSW event that triggers the executable entity. It is not possible to map an executable entity to a task, if the executable entity is not triggered by an event.

In this document the expression *an executable entity is mapped to a task* will appear several times. What is actually meant is that the `Rte` or BSW events, which trigger the executable entity, are mapped to a task.

4.2.17.1.2.1. No task mapping necessary

NOTE	**Executable entities can be triggered by interrupt service routines**
	An executable entity, which is not triggered by an `Rte` or BSW event, might be triggered by an interrupt service routine if the executable entity belongs to:

► a service component module, or

► the `I/O Hardware Abstraction`, or

► a complex driver module.

If one of these cases applies, the `Rte` is not responsible for triggering the executable entity but the executable entity must still be available in the system model of the project. If the executable entity is not in the system model, the Rte Generator will not generate or compile successfully. The executable entity description must be imported because the `Rte` might have to generate API functions for the executable entity.

An executable entity will be available in the system model, if you import the description files that contain the definition of that executable entity.

In the following cases, an executable entity does not need to be mapped to a task. They are:

1. executable entities which are not triggered by an `Rte` or BSW events,

2. server runnable entities that implement unconnected operation prototypes,

3. intra-partition direct call server runnable entities,

4. inter-partition/inter-core direct server executable entities with `RteEventToTaskMapping.RteAllowInterPartitionDirectCall` or `RteBswEventToTaskMapping.RteBswAllowInterPartitionDirectCall` vendor-specific flag set to `TRUE`.

---

5.   runnable entities mapped to schedulable entities,

6.   direct call triggered schedulable entities.

In cases 1. and 2., the executable entities are never invoked by the `Rte`, therefore they do not have to be mapped to a task.

In case 3., a server runnable entity does *not* have to be mapped to a task, if it implements only servers that can be directly called. A server runnable entity can be directly called if:

► the server is called synchronously,

   i.e. the connected client component has a synchronous server call point, referring to an operation prototype which is implemented by the server runnable entity,

► and the server runnable entity has the `CanBeInvokedConcurrently` flag set to `TRUE` or `CanBeInvokedConcurrently` is set to `FALSE` and all client runnable entities which call the server are mapped to the same task or tasks with the same priority

► and the server runnable entity does not have:

   ► wait points,

   ► asynchronous server call points,

   ► data write access,

   ► data read access,

   ► read or write access to implicit inter runnable variables, or

   ► implicit access to exclusive areas.

In all other cases, the server runnable entity must be mapped to a task, i.e. a context switch is enforced.

In case 4., the server executable entity is directly called. If the `RteEventToTaskMapping.RteAllowInterPartitionDirectCall` or `RteBswEventToTaskMapping.RteBswAllowInterPartitionDirectCall` vendor-specific flag is set to `FALSE`, the inter-partition client-server calls will require a task mapping of the server's OperationInvokedEvent.

In case 5., a runnable entity does not need to be mapped to a task if a `SwcBswMapping` exists where the runnable entity is mapped to a basic software schedulable entity. Such a mapping is reasonable if the schedule entity needs to call `Rte` API functions that can only be configured for runnable entities. If such a mapping exists and a task mapping is required for the runnable entity (e.g. because it acts as a client), the task mapping is derived from the task mapping of the schedulable entity.

In case 5., a task mapping is not required if the conditions of rte_sws_7214 are fulfilled (see [1])

4.2.17.1.2.2. Scenario A1

Task mapping scenario A1 is applied if:

▶ one or more events are mapped to a task, and

▶ all executable entities triggered by the events mapped to the task are category 1 executable entities, and

▶ all events are timing events with the same period, and

▶ all timing events are either `Rte` or BSW timing events, and

▶ the task type is `BASIC` or is not defined.

In this case, a task body is generated, which just contains the consecutive calls of the executable entity functions. The basic task is activated when the `Rte` or BSW event occurs.
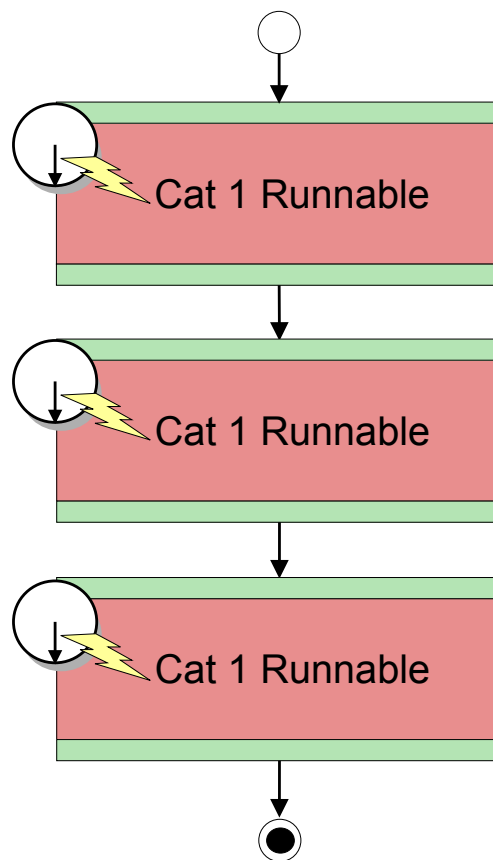
Figure 4.2. Task mapping scenario A1 example

4.2.17.1.2.3. Scenario A2

Task mapping scenario A2 is applied if:

▶ two or more events are mapped to a task, and

▶ all executable entities triggered by the events mapped to the task are category 1 executable entities, and

- ► all events are timing events with different periods, and

- ► all timing events are either `Rte` or BSW timing events, and

- ► the task type is `BASIC` or is not defined.

Here the task body contains a counter, which counts the number of task activations. The executable entity functions are conditionally executed based on the counter value. The basic task is activated when the `Rte` or BSW event occurs.
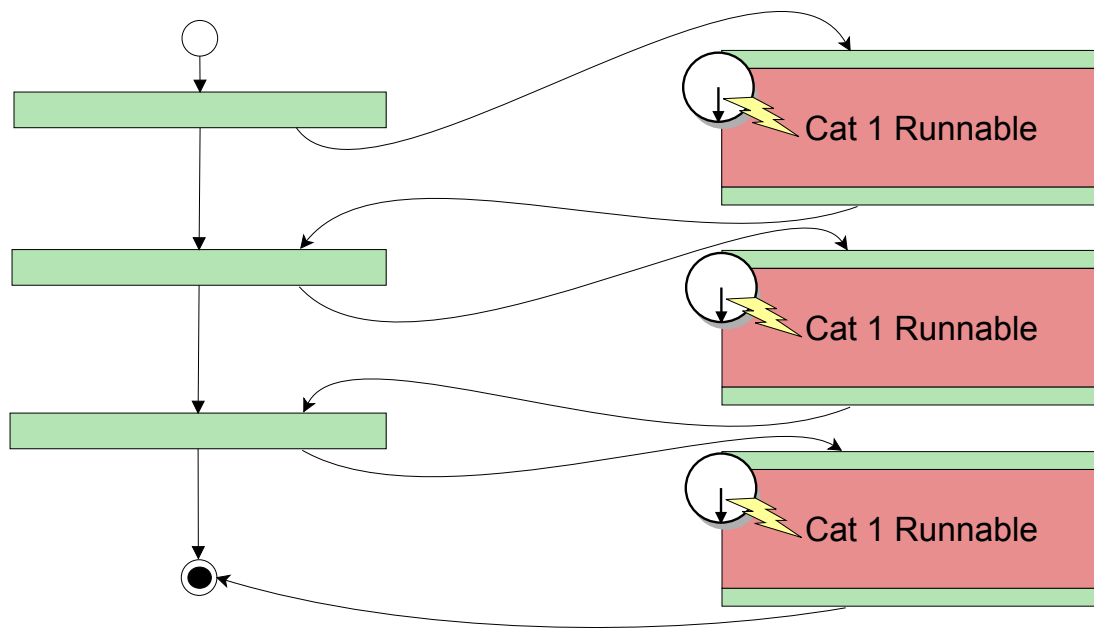
Figure 4.3. Task mapping scenario A2 example

4.2.17.1.2.4. Scenario B1

Task mapping scenario B1 is applied if:

- ► one non-timing event is mapped to the task which is not a background event, or

- ► only background events are mapped to the task,

and

- ► no timing events are mapped to the task, and

- ► all executable entities that are triggered by the events and mapped to the task are category 1 executable entities, and

- ► the task type is `BASIC` or is not defined.

The task body generated is the same as in scenario A1. The task is also activated only when an event occurs.
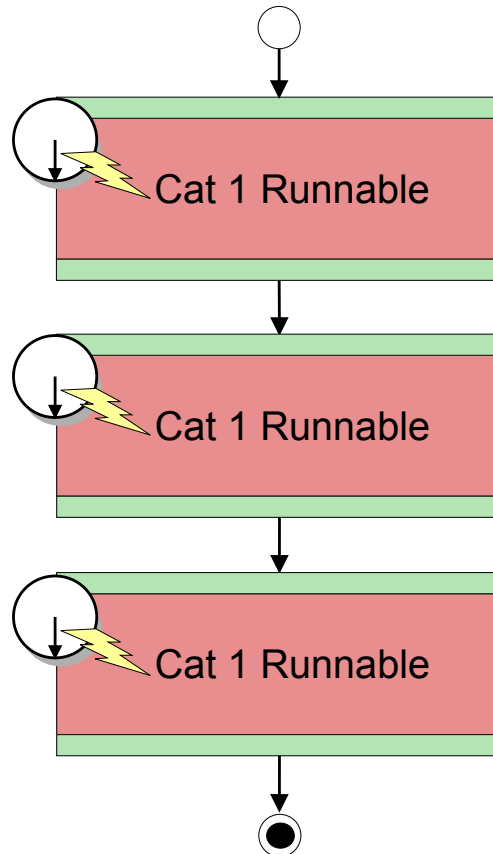
Figure 4.4. Task mapping scenario B1 example

4.2.17.1.2.5. Scenario B2

WARNING	**Blocked runnable entities cause wrong timing behavior**
⚠	If multiple runnable entities are mapped to the same task and at least one of these runnable entities is of category 2, the Rte Generator reports a warning. If you ignore this warning, the category 2 runnable entity blocks the execution of the other executable entities and thus influences the overall timing behavior.
	To avoid wrong timing behavior, map category 2 runnable entities to a separate extended task.

Task mapping scenario B2 is applied if:

▶ the task type is `EXTENDED` or,

▶ one or more events are mapped to a task, and the executable entities that are triggered by the events and mapped to the task are category 1 or 2 executable entities.

The generated task is an EXTENDED task, i.e. the task body contains an `Os` WaitEvent call; when an `Rte` event occurs, an `Os` event is set to run the task.
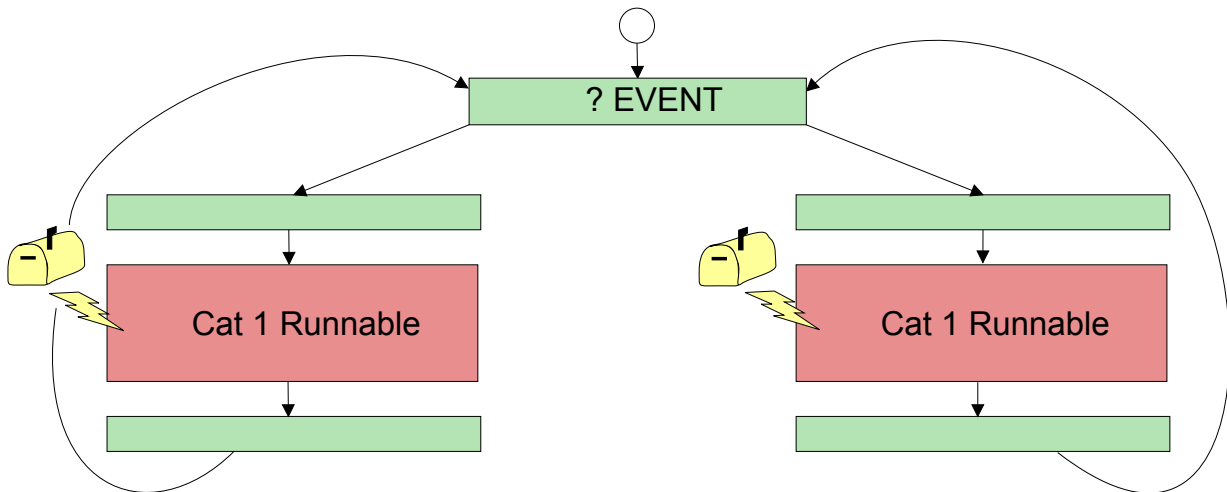
Figure 4.5. Task mapping scenario B2 example

4.2.17.1.3. Task mapping constraints for mode management

For mode switch events or `Rte`/BSW events involved in the mode management consider the following constraints regarding the task mapping:

▶ For asynchronous mode switch, `on-entry, on-transition, on-exit ExecutableEntity's` of the same `partition local mode user group` must be mapped to the same task.

▶ The `PositionInTask` of on exit mode switch events must be smaller than the `PositionInTask` of on entry mode switch events.

▶ For synchronous mode switch, `on-exit ExecutableEntity's` must not be mapped after `on-entry ExecutableEntity's`. This is valid for all the ExecutableEntity's associated with the same mode machine instance.

▶ For synchronous mode switch, `on-transition ExecutableEntity's` must not be mapped after `on-entry ExecutableEntity's`. This is valid for all the ExecutableEntity's associated with the same mode machine instance.

▶ For synchronous mode switch, `on-exit ExecutableEntity's` must not be mapped after `on-transition ExecutableEntity's`. This is valid for all the ExecutableEntity's associated with the same mode machine instance.

▶ For asynchronous mode switch, `on-exit ExecutableEntity's` must not be mapped after `on-entry ExecutableEntity's`. This is valid for all the ExecutableEntity's of the same software component or Basic Software Module for a mode machine instance.

▶ For asynchronous mode switch, `on-transition ExecutableEntity's` must not be mapped after `on-entry ExecutableEntity's`. This is valid for all the ExecutableEntity's of the same software component or Basic Software Module for a mode machine instance.

▶ For asynchronous mode switch, `on-exit ExecutableEntity's` must not be mapped after `on-transition ExecutableEntity's`. This is valid for all the ExecutableEntity's of the same software component or Basic Software Module for a mode machine instance.

Consider the following constraints regarding the task mapping when you assign the task priorities:

▶ For synchronous mode switch, the priority of the tasks on which events with mode disabling dependencies are mapped must be equal or higher than the priority of the task on which the mode switch events are mapped. This is valid for all events associated with the same `partition local mode user group`.

▶ For asynchronous mode switch, the priority of the tasks on which events with mode disabling dependencies are mapped can be lower than the priority of the task on which the mode switch events are mapped. This is valid for all events associated with the same `partition local mode user group`.

4.2.17.1.4. Task mapping constraints for client/server communication

For client/server communication, consider the following constraints regarding the task mapping when you assign the task priorities:

▶ A *synchronous* server call point *with* time-out monitoring requires a higher priority of the *client* runnable entity's task than of the *server* runnable's task. The server task must be preemptable.

▶ For *synchronous* server call point *without* time-out monitoring it is recommended that the *client* runnable's tasks have lower priority than the priority of the *server* runnable's task. Otherwise the client explicitly waits for the server to finish. A wait point and thus an extended task is required. The client task must be preemptable.

▶ An *asynchronous* server call point with time-out monitoring requires a higher priority of the *client* runnable's task that collects the result, than of the *server* runnable's task. The task of the server runnable must be preemptable.

4.2.17.1.5. Inter-partition intra-core synchronous client/server calls

The inter-partition intra-core synchronous client/server calls rely on task priorities. The `Rte` supports the mapping of the client to a basic task instead of an extended task under the following circumstances:

▶ The call is executed on the same core.

▶ The server is not of category 2.

▶ The server has a higher priority than the client.

▶ No timeout is configured.

As a result, the resources and runtime are saved.

4.2.17.2. Data consistency mechanisms

Under certain conditions, the Rte Generator automatically applies data consistency mechanisms to prevent data corruption when it accesses internal data structures in preemptive tasks with different priorities.

In particular, data consistency has to be guaranteed when:

▶ accessing sender-receiver buffers and queues,

▶ accessing activated exclusive areas,

▶ accessing inter runnable variables,

▶ calling `Com_SendSignal()`/`Com_SendSignal Group()`.

4.2.17.2.1. Atomic access

WARNING	Uncontrolled behavior of the software components
⚠	These base type property settings are highly depend on the microcontroller target, the derivative, the compiler and the compiler settings. Thus the default properties may only be used for the default

    ▶ target,

    ▶ derivative,

    ▶ compiler and

    ▶ compiler settings.

If you use the default base type properties for a target, derivative, compiler or compiler settings that differ from the default, the software components behave in an uncontrolled way.

The same applies, if you change the default base type properties for the default target, derivative, compiler and compiler settings.

To avoid uncontrolled behavior of the software component, check and adapt the default properties accordingly if you want to use the `Rte` in a different environment (different derivative, different compiler, different compiler settings etc.).

When applying data consistency mechanisms, the Rte Generator considers whether the microcontroller is able to make an *atomic access* to a variable on assembly level.

The information about the capability of atomic accesses is read from the `Base` plugin [2].

The properties are preconfigured by EB and can be found in a derivate-specific file `<TARGET>_<DERI-VATE>.properties` located in `$(TRESOS_BASE)/plugins/Platforms_TS_TxxDxMxIxR0/resources`.

For each AUTOSAR base type, the file contains a property whether atomic access is possible or not. If the following file property is true, atomic access to the respective base type is possible and therefore the Rte Generator does not apply any data consistency mechanism.

> **Example 4.1. Example of an atomic access configuration for base types**

```
Basetypes.boolean.AtomicAccess:true
Basetypes.sint8.AtomicAccess:true
Basetypes.sint16.AtomicAccess:true
Basetypes.sint32.AtomicAccess:true
Basetypes.uint8.AtomicAccess:true
Basetypes.uint16.AtomicAccess:true
Basetypes.uint32.AtomicAccess:true
Basetypes.float32.AtomicAccess:true
Basetypes.float64.AtomicAccess:false
```

WARNING   **Atomic access in the context of data structures**

The Rte uses structure types for complex data buffers, for example to hold status and value information for a data element. Atomic access to structure elements is compiler dependent and therefore cannot be guaranteed by the Rte.

If this is the case, you should set the atomic access attributes to `false` even if atomic access for single variables of a certain base type is possible.

4.2.17.2.2. Default data consistency mechanism

The *default data consistency mechanism* is applied for

▶   receive buffers, and

▶   queues, and

▶   inter runnable variables.

---

[2]This plugin is not defined in AUTOSAR but is EB-specific.

You may configure this mechanism during ECU configuration. For details see Section 4.3.4, "Configuring the Rte".

Possible values for the default data consistency mechanism are:

► usage of `Os` resources,

► interrupt blocking.

If the option is set to *Os resources*, the `Rte` uses a special `Os` resource for each variable, which has to be protected.

If the option is set to *interrupt blocking*, the configured interrupt blocking function (see Section 4.2.17.2.3, "Interrupt blocking function") is used.

4.2.17.2.3. Interrupt blocking function

You may configure the way in which interrupt blocking is realized during ECU configuration (see Section 4.3.4.4.3, "Configuring the interrupt blocking function"). The following options are available:

► `SuspendAllInterrupts()` and `ResumeAllInterrupts()`:

These functions are standard AUTOSAR `Os` API functions. They support nested interrupt locking and memory protection but are quite slow on some architectures. Use this mechanism if you require memory protection support and if you configure runnable entities which run in an exclusive area with interrupt blocking. Otherwise problems regarding nested blocking of interrupts might occur.

► `DisableAllInterrupts()` and `EnableAllInterrupts()`:

These functions are also standard AUTOSAR `Os` API functions. On some architectures, they might be faster than `Suspend/ResumeAllInterrupts()`, but nested interrupt locking is not supported. Memory protection is supported. Use `DisableEnableAllInterrupts()` if you require memory protection support.

► Usage of EB-specific functions from the EB `Base` module:

The EB-specific macros are faster than the standard AUTOSAR `Os` API functions, but they support neither nesting nor memory protection. Choose this option to reduce the runtime overhead caused by the `Os` interrupt blocking functions.

► Usage of user-defined functions/macros:

If you want to use your own functions you must define the macros `Rte_UserDefinedIntLock` and `Rte_UserDefinedIntUnlock` in the `Rte_UserDefinedIntLock.h` header file. The `Rte` generated then includes the header file that you provided. Select this option if you use the `Rte` with an non-AUTOSAR `Os`-compliant operating system or with an architecture not officially supported.

4.2.17.2.4. Data consistency for receive buffers and queues

In general, if you use *internal communication* only, the access to internal receive buffers or queues is protected by the default data consistency mechanism. You may configure the default data consistency mechanism during ECU configuration (see Section 4.2.17.2.2, "Default data consistency mechanism").

In *external communication*, interrupt blocking is always used, i.e. the buffer or queue is accessed from a `Com` callback, which might run in interrupt context. Within the callback itself, you may avoid interrupt blocking if the callback is not interruptible. You may configure this during ECU configuration (see Section 4.3.4.4.4, "Configuring the Com Callback not interruptible parameter").

A data consistency mechanism is not applied if

► The tasks have the same priority. This concerns those tasks on which the runnable entities are mapped that access the buffer/queue.

► The tasks are non-preemptive. This concerns those tasks on which the runnable entities are mapped that access the buffer/queue.

► The access to the buffer is atomic on assembly level.

4.2.17.2.5. Data consistency for exclusive areas

For *exclusive area*s, you may configure the data consistency mechanism used. The following options are available:

1. *All Interrupt Blocking*: All interrupts are blocked

2. *Os Resource*: A standard Os resource is used for protecting the exclusive area

3. *Os Interrupt Blocking*: Os interrupts are blocked

4. *Cooperative Runnable Placement*: An internal resource from the Os is used for the tasks, where the executable entities that run cooperatively are mapped to

5. *EB Fast Lock*: All interrupts are locked via an assembly instruction

6. *Disable Exclusive Area*: No interrupts are locked

7. *Os Spinlock*: An Os spinlock is used for protecting the exclusive area

8. *User Callout*: A callout defined by the user is used for protecting the exclusive areas

   ► User callout enter: The user defines the name of the function. The `Rte` calls this function to enter the exclusive area if the implementation mechanism is set to *User Callout*. The user is responsible for the implementation of this function.

   ► User callout exit: The user defines the name of the function. The `Rte` calls this function to leave the exclusive area if the implementation mechanism is set to *User Callout*. The user is responsible for the implementation of this function.

► The `Rte` generates the header file `Rte_UserDefinedExclusiveArea.h` that contains external declarations of the user-defined enter callouts and exit callouts. The user is responsible to implement these callouts.

4.2.17.2.5.1. Data consistency for exclusive areas of software components

The `Rte` optimizes the implementation of exclusive areas which belong to a software component.

The mechanisms *All Interrupt Blocking*, *User Callout* and *Os Resource* are only applied when data corruption could occur. This means that the mechanisms are not applied if:

► the tasks on which the runnable entities are mapped have the same priority; this concerns the runnable entities which run in or can enter the exclusive area,

► the tasks on which the runnable entities are mapped are non-preemptive; this concerns the runnable entities which run in or can enter the exclusive area,

For option 3, the Rte Generator blocks all Os-relevant interrupts.

For option 4, the Rte Generator generates corresponding glue code so that cooperative runnable entities cannot interrupt each other while they can still be interrupted by other non-cooperative runnable entities.

The options *EB Fast Lock*, *Disable Exclusive Area*, and *Os Spinlock* are not available for exclusive areas that belong to software components.

The mechanism *Os Resource* is overruled with the mechanism *All Interrupt Blocking* if `the Rte` cannot determine the task context of the runnable(s) which use the exclusive area. The `Rte` respects the user input for all other mechanisms.

4.2.17.2.5.2. Data consistency for exclusive areas of basic software modules

The implementation mechanism of exclusive areas, which belong to a basic software module, i.e. they are provided by the BSW scheduler of the `Rte`, will never be optimized.

For exclusive areas, which belong to a basic software module, all implementation mechanisms including *EB Fast Lock*, *Disable Exclusive Area*, and *Os Spinlock* are available.

4.2.17.2.6. Data consistency for inter runnable variables

The default data consistency mechanism is used to protect *inter runnable variables*; configure this mechanism during ECU configuration. (see Section 4.2.17.2.2, "Default data consistency mechanism").

A data consistency mechanism is not applied if:

► the tasks on which the runnable entities are mapped have the same priority; this concerns runnable entities which access the inter runnable variable,

► the tasks on which the runnable entities are mapped are non-preemptive; this concerns the runnable entities which access the inter runnable variable,

► the access to the variable is atomic on assembly level.

4.2.17.2.7. Data consistency for Com signals and signal groups

Since the `Com` API functions are not re-entrant for the same signal, the `Rte` uses `Os` resources if multiple *data element prototype*s are mapped to the same `Com` signal/signal group, and a data consistency issue might occur. See chapter **Signals and signal groups** in the EB tresos AutoCore Generic COM Services documentation.

4.2.17.2.8. Lock-free queues

Due to the introduction of thread fences, locks are omitted for queue accesses under the following circumstances:

► On the reader side, if all readers cannot interrupt each other.

► On the writer side, if all writers cannot interrupt each other.

The lock-free queue handling is based on the separated write access of queue.tail and queue.head:

► Addition to the queue modifies the queue.tail only.

► Subtraction from the queue modifies the queue.head only.

► The access to the queue itself is locked by comparisons of tail and head.

NOTE ⓘ	The thread fence APIs are provided by the Os and used by the Atomics library. If another Os is used, one that does not support this functionality, then the user has to provide the hardware specific implementation of the thread fence.
	This optimization cannot be disabled.

4.2.17.3. Receive buffer allocation, sharing and initialization

4.2.17.3.1. Receive buffer allocation

The Rte Generator allocates a receive buffer for each unqueued data element prototype instance when

► the require port is connected to at least one provide port on the same ECU (internal communication) or the data element prototype is mapped to at least one `Com` signal/signal group and direct read from `Com` is not possible or disabled,

► and at least one runnable entity has a data read access or data receive point defined for the data element prototype.

Direct read from `Com` is an optimization to reduce RAM consumption. If the communication is external, the `Com` module buffers the signal values anyway, so the `Rte` does not need to allocate an additional buffer and can directly read the data from the `Com` buffers.

You may configure whether the Rte Generator applies this optimization or not. To configure this (in the `Rte` ECU configuration), see [Section 4.3.4.4.6, "Configuring the directly read from `Com` buffer option"](#).

Even if this optimization is enabled, there are conditions under which a separate receive buffer has to be allocated for the data element. These conditions are:

► the require data element is queued,

► or the require data element is connected to multiple provide data elements (mixed intra- and inter-ECU connections), or

► the require data element is mapped to more than one `Com` signal (multiple inter-ECU connections), or

► data element invalidation is active for the require data element, or

► communication timeout is defined for the data element.

4.2.17.3.2. Receive buffer sharing

To reduce the RAM consumption, the Rte Generator optimizes the allocation of receive buffers so that multiple data element prototypes can share the same buffer.

Two or more data element prototype instances of a require port share the same receive buffer when they are

► connected to the same provide data element prototype instances

► mapped to the same `Com` signals/signal groups.

4.2.17.3.3. Receive buffer initialization

When an initial value is defined in the `Com` specification of a data element, the receive buffer is initialized with this value.

When an initial value is only specified on the receiver side but not on the sender side, the initial value from the receiver side is used.

When an initial value is only specified on the sender side but not on the receiver side, the initial value from the sender side is used.

When an initial value is specified both on the sender and receiver side, the initial value from the receiver side has priority. When the init values of the sender and receiver side do not match, the Rte Generator reports a warning.

When an initial is neither specified on sender nor on receiver side, 0 is used as default and the Rte Generator reports a warning.

When multiple require data element prototypes are connected to the same provide data element and for at least one require data element an initial value is defined, the same initial value must also be defined for all other require data element prototypes. If you define different initial values for the require data elements, only one of them is used for all data element prototypes. It is not defined, which one is used. If only one of the two initial values is used, the Rte Generator reports a warning.

When a data element is mapped to a `Com` signal, the initial value specified in the `Com` configuration is used. When an initial value is specified in the `Com` specification of the data element and this initial values does not equal the initial value configured for the mapped `Com` signal, the Rte Generator reports a warning. Note that the initial value from the `Com` specification can be automatically pushed to the `Com` ECU configuration.

4.2.17.4. Function Elision

Function Elision is a vendor-specific optimization of the EB tresos AutoCore `Rte`. If *function elision* is enabled via the parameter in the ECU configuration, some parts of the API will be realized as macros instead of functions under certain conditions. For instructions how to enable function elision, see Section 4.3.4.4.1, "Configuring the function elision option".

In general, function elision is supported for

▶ `Rte_Read()`

▶ `Rte_DRead()`

▶ `Rte_Write()`

▶ `Rte_IsUpdated()`

▶ `Rte_IrvRead()`

▶ `Rte_IrvWrite()`

▶ `Rte_Enter()`

▶ `Rte_Exit()`

▶ `Rte_Mode()`

▶ `Rte_CData()`

▶ `Rte_Prm()`

▶ `Rte_Call()`

▶ API for unconnected ports

4.2.17.4.1. Conditions

Function elision is only applied if it is explicitly enabled in the ECU configuration and if the software component description fulfills the following conditions:

▶ The software component is available as source code, i.e. the code type attribute in the implementation is set to `SRC`.

▶ The software component does not support multiple instantiation, i.e. the corresponding attribute in the internal behavior of the component is set to `false`.

▶ The indirect API of the port of the software component is disabled, i.e. the corresponding attribute in the port API options of the port is set to `false`.

▶ The enable take address option of the port of the software component is disabled, i.e. the corresponding attribute in the port API options of the port is set to `false`.

4.2.17.5. Implicit communication

NOTE	**Runnable entities that access the same data element prototype share a buffer**
ⓘ	A buffer is shared for runnable entities that access the same data element prototype; this concerns runnable entities which are mapped to tasks with the same priority or to non-preemptive tasks.
	This optimization has no influence on the semantics of implicit communication.

For data *read* access, the data is copied for a runnable entity at the beginning of a task. The `Rte` allocates special buffers, which contain the data copies. These buffers have task-wide scope, i.e. all runnable entities that have data read access to the same data element prototype access the same buffer.

For data *write* access, the runnable entities also modify a special buffer with task-wide scope, i.e. all runnable entities, which are mapped to the same task and have data write access to the same data element prototype, access the same buffer. The content of the buffer is written back right before the task is ended.

The buffers might also be shared, if:

▶ a runnable entity has data write access to a data element prototype, and

▶ another runnable entity has data read access to a connected data element prototype, and

▶ both runnable entities are mapped to the same task.

The conditions are:

▶ the communication is 1:1, m:1 or 1:n (sharing for m:n is not possible!),

▶ the settings for data element invalidation and communication time-out is the same for all sender and receiver data element prototypes.

These optimizations reduce the memory overhead caused by implicit communication; however they influence the communication semantics:

▶ An implicit write access to a data element prototype *is not visible* to receiver runnable entities mapped on a different task other than the sender runnable entity until the sender's task has terminated.

▶ An implicit write access to a data element prototype *is visible* to receiver runnable entities mapped on the same task during the same execution of the task when the buffer sharing is possible.

▶ An implicit write access to a data element prototype *is not visible* to receiver runnable entities mapped on the same task until the next task execution when buffer sharing is not possible.

4.2.18. Generation modes

The `Rte` supports several generation modes: *contract generation phase*, *RTE only generation mode*, *BSW Scheduler only generation mode* and *Full generation mode*.

4.2.18.1. Contract generation phase

In the contract phase, the *application header files* and the *AUTOSAR types header file* are generated. Furthermore, for each component a template source code file is generated, which contains exemplary implementations of the component's runnable entities.

In the *contract phase*, the Rte Generator offers an optional component template source code file merge (component template merge). If this component template merge is enabled and the *contract phase* is being executed, the Rte Generator tries to merge each component's template source code file with the newly generated template source code file on a runnable entity level. Therefore, each runnable entity in the available template source code file is surrounded by tags:

▶ The tag `BEGIN_RUNNABLE` denotes the beginning of a runnable entity within a component template source code file. It contains the name of the runnable entity it is associated with and the parameter `MODIFIED`.

▶ The tag `END_RUNNABLE` denotes the end of a runnable entity and contains the name of the runnable entity it is associated with.

By means of the `MODIFIED` parameter it is possible to define for each runnable entity within a component template source code file, whether it should be merged into the newly generated component template source code file. It can either have the value `FALSE` or `TRUE`:

► The default setting for the `MODIFIED` parameter of a runnable entity's `BEGIN_RUNNABLE` tag is `MODIFIED=FALSE`. The source code of a runnable entity, for which `MODIFIED=FALSE`, is not merged into the newly generated template source code file.

► If the `MODIFIED` parameter of a runnable entity is set to the value `TRUE`, then its source code is merged into the newly generated template source code file. The source code of a runnable entity is represented by the code between the `BEGIN_RUNNABLE` and `END_RUNNABLE` tags, which are associated with that runnable entity.

The following runnable entity implementation exemplifies the use of the `BEGIN_RUNNABLE` and `END_-RUNNABLE` tag, as well as the `MODIFIED` parameter in the `BEGIN_RUNNABLE` tag. Since the `MODIFIED` parameter has the value `TRUE`, the runnable entity's implementation is merged when the *contract phase* is being run again.

```
/* TAG BEGIN_RUNNABLE runnable MODIFIED=TRUE */

FUNC(void, RTE_CODE) runnable(void)
{
    Std_ReturnType status = RTE_E_NOT_OK;
    UInt32 data;

    status = Rte_Write_port_de(&data);
}

/* TAG END_RUNNABLE runnable */
```

If the component template merge is enabled and the *contract phase* is being executed, the Rte Generator processes each existing software component, thus each existing component template source code file, sequentially:

► First, the Rte Generator parses the existing component template source code file of the currently processed software component for `BEGIN_RUNNABLE` tags. If there is no such file, the Rte Generator skips the merge for the current software component.

► If the Rte Generator finds a `BEGIN_RUNNABLE` tag within the existing component template source code file, the Rte Generator retrieves the value of the `MODIFIED` parameter.

► Furthermore, if `MODIFIED=TRUE`, the Rte Generator buffers the content between the `BEGIN_RUNNABLE` and its corresponding `END_RUNNABLE` tag in a buffer.

► The buffer, into which the content is saved in, is associated with the runnable entity, which is identified by the name you provided in the `BEGIN_RUNNABLE` tag.

► If the file end of the currently processed component template source code file is reached, the Rte Generator starts the new generation of the component template source code file for the currently processed software component.

► Before newly generating the code for a runnable entity, the Rte Generator checks, if code has been buffered and the `MODIFIED` parameter has been set to `TRUE` for this runnable entity. If this is the case, the Rte Generator writes the buffer content into the newly generated component template source code file. Otherwise, the Rte Generator writes the runnable entity definition including exemplary function calls for the available API functions into the newly generated component template source code file.

► Should there be a buffer associated with a runnable entity that does not exist any more and for which the `MODIFIED` parameter is set to `TRUE`, the Rte Generator writes a corresponding preprocessor error to the newly generated component template source code file.

If you want to keep your implementation of the runnable entity when you execute the *contract phase* again, then set the `MODIFIED` parameter to the value `TRUE` in the `BEGIN_RUNNABLE` tag of the runnable entity. Your implementation is then merged with the newly generated template.

With the application header file and the AUTOSAR types header file it is possible to compile an atomic software component's code and deliver the software component as an object-code component.

The application header file depends on certain standard headers from the EB `Base` module. Make sure that the `include` folder of the EB `Base` module in your tresos installation is contained in the default include search paths of the used compiler.

The contract phase does not depend on the ECU configuration and therefore the contract phase does not depend on a EB tresos Studio configuration *project.*

If you use different Rte Generators for the two generation phases, they must operate in *compatibility mode* .

For details on how to generate the application header files, see Section 4.3.5.1, "Generating the application header files only"

4.2.18.2. RTE only generation mode

In the RTE only mode, the `Rte` is generated without `BSW Scheduler` artifacts. The generated artifacts include the

► application header files and

► AUTOSAR type header files,

► the `VFB tracing` header file,

► the `Rte` configuration header file, and

► the C-code of the generated `Rte` itself.

Atomic software components that are delivered as source-code components are compiled against the application header files that are generated in the Rte only generation mode.

If you compiled atomic software components that are delivered as object-code against application header files generated in compatibility mode, the Rte Generator would have to use compatibility mode for the generation mode as well.

The object code of all software components (object-code and source-code components) is linked to the code of all basic software modules and to the code of the Rte to build the ECU software. See Section 4.3.5.2, "Generating the Rte source code" for instructions on how to generate the Rte.

4.2.18.3. BSW Scheduler only generation mode

In the BSW Scheduler generation mode, only the BSW Scheduler artifacts are generated by the Rte module. The generated artifacts include the

► module interlink header files

► the module interlink type header files and

► the C-code of the Rte with only the required task bodies and BSW Scheduler API functions

BSW modules that are delivered as source-code are compiled against the module interlink header files that are generated in the BSW Scheduler only generation phase.

4.2.18.4. Full generation mode

In the Full generation mode, all artifacts of the AUTOSAR 4.0 Rte are generated.

4.2.18.5. Files generated

The Rte generates the following files:

$(OUTPUT_PATH)/include/Rte_Type.h	AUTOSAR types header file
$(OUTPUT_PATH)/include/Rte_Main.h	Rte lifecycle header file
$(OUTPUT_PATH)/include/Rte_Intern.h	Rte internal header file
$(OUTPUT_PATH)/include/Rte_Hook.h	VFB tracing header file
$(OUTPUT_PATH)/include/Rte_Cfg.h	Rte configuration header file
$(OUTPUT_PATH)/include/Rte_Cbk.h	Rte Com callback prototypes header file
$(OUTPUT_PATH)/include/ Rte_<Component>.h	application header file for component <Component>

`$(OUTPUT_PATH)/src/Rte.c`	`Rte` source file
`$(OUTPUT_PATH)/src_appl/*.c`	Template files for software components
`$(OUTPUT_PATH)/include/ SchM_<Module>.h`	Module Interlink header file
`$(OUTPUT_PATH)/include/ SchM_<Module>_Type.h`	Module Interlink types header file

4.2.19. AUTOSAR 3.2 wrapper

EB tresos Studio supports the import of AUTOSAR 3.2 conformant software component descriptions into an AUTOSAR 4.0 environment. The software component descriptions are converted by EB tresos Studio to the AUTOSAR 4.0 schema so that they can be used by the `Rte` as if AUTOSAR 4.0 software components had been imported.

After the automatic conversion to AUTOSAR 4.0, the `Rte Editor` can be used to create an AUTOSAR 4.-0 conformant `Rte` configuration. Note that it is not possible to automatically convert the configuration from an existing AUTOSAR 3.2 project. In this case, it is necessary to reconfigure the project manually.

To reuse existing implementations of AUTOSAR 3.2 conformant software components, the configuration parameter `Generate AUTOSAR 3.2 wrapper` must be enabled before the `Rte` generation is triggered. If this parameter is enabled, the `Rte` generator creates a wrapper that maps the function signatures specified in AUTOSAR 3.2 to the API of AUTOSAR 4.0. With the wrapper it is possible to use existing source code of AUTOSAR 3.2 conformant software components in an AUTOSAR 4.0 environment. However, the wrapper has some limitations that the user has to consider. The following section describes some guidelines that need to be fulfilled to reuse AUTOSAR 3.2 software components.

4.2.19.1. Guidelines for using AUTOSAR 3.2 software components in an AUTOSAR 4.0 environment

▶ The software components must not be available in object code because the wrapper uses macros that require the source code of the software components.

▶ Software components must use pointer to `const` for complex input parameters in server runnable entities. The reason is that in AUTOSAR 3.2 it is unspecified whether pointer to `const` or pointer to `var` shall be used. If no pointer to `const` for complex input parameters is used, it may lead to compiler warnings or errors.

▶ When online calibration is used, the integration code must be adapted because the names of the calibration reference table and base pointer have been changed from `RteCalprmRefTab` to `RteParameterRefTab` and from `RteCalprmBase` to `RteParameterBase` respectively.

▶ The return values of API functions are not mapped by the compatibility wrapper. This means that the implementation of software components shall not rely on the return values of the API functions `Rte_-Read()`, `Rte_IStatus()`, `Rte_Feedback()` or `Rte_Call()` (asynchronous) for unconnected ports.

In AUTOSAR 3.2, these API functions return `RTE_E_OK` in case of unconnected ports while in AUTOSAR 4.0, the return value is `RTE_E_UNCONNECTED`. For all other API functions, it is recommended to do the error handling as shown below to ensure that the behavior of the software component is the same in an AUTOSAR 3.2 and AUTOSAR 4.0 environment:

```
if ( Rte_xxx(..) != RTE_E_OK ) {
    /* handle error */
}
```

Note that AUTOSAR 4.0 specifies additional return values for several error conditions: `RTE_E_IN_EX-CLUSIVE_AREA`, `RTE_E_SEG_FAULT` and `RTE_E_UNCONNECTED`.

▶ A software component shall only use enumeration constants of data types that are actually used by the software component. In AUTOSAR 3.2, enumeration constants were generated in the global `Rte_Type.-h` header file. In AUTOSAR 4.0, they are generated in the application's types header file. This file only contains the enumeration constants of the data types which are used by the software component.

▶ A software component shall not call `Rte_Feedback()` before calling `Rte_Write()`, `Rte_Send()` or `Rte_Switch()` because the initial return value of `Rte_Feedback()` has been changed from `RTE_E_-NO_DATA` to `RTE_E_TRANSMIT_ACK`.

▶ A runnable entity shall not call `Rte_IrvIRead()` multiple times in case this same runnable entity calls `Rte_IrvIWrite()` between the `Rte_IrvIRead()` calls. Instead the implicit inter runnable variable shall be read once into a local copy and the runnable shall operate on the local copy instead of using multiple `Rte_IrvIRead()` calls.

Code like:

```
foo(Rte_IrvIRead_x());
[..]
Rte_IrvIWrite_x(6);
[..]
bar(Rte_IrvIRead_x());
```

shall be replaced by:

```
x = Rte_IrvIRead_x()
foo(x);
[..]
Rte_IrvIWrite_x(6);
[..]
bar(x);
```

The update behavior on implicit inter runnable variables with shared buffers differs between AUTOSAR 3.2 and 4.0

▶ Multiple runnable entities that are mapped to different tasks and that share a common data element with implicit communication where at least one of these runnable entities performs a write access to the common data element shall not be placed in exclusive areas using "cooperative scheduling" as the exclusive

area implementation mechanism. Since AUTOSAR 4.0 permits buffer sharing for this scenario while AU-TOSAR 3.2 does not, the values might be different in those scenarios depending on whether and when a preemption actually occurs.

► The exclusive area implementation mechanism `NON_PREEMPTIVE_TASK` shall not be used as AUTOSAR 4.0 does not support the exclusive area implementation mechanism `NON_PREEMPTIVE_TASK` anymore (as a configuration option). Instead `COOPERATIVE_RUNNABLE_PLACEMENT` shall be chosen as the exclusive area implementation mechanism and the runnable entities shall be mapped to non-preemptive tasks. The same behavior can be achieved by simply mapping the runnable entities to non-preemptive tasks.

► The file `MemMap.h` shall be extended by mappings for `<c>_START_SEC_CODE` and `<c>_STOP_SEC_-CODE`, where <c> is the short name of the software component type. `<c>_START_SEC_CODE` and `<c>_-STOP_SEC_CODE` shall be mapped to the same regions as `RTE_START_SEC_APPL_CODE` and `RTE_-STOP_SEC_APPL_CODE`.

► Sender/Receiver interfaces with both mode declaration group prototypes and data element prototypes shall not be used. Sender/Receiver interfaces which have both mode declaration group prototypes and data element prototypes are not supported in AUTOSAR 4.0

4.2.20. Data conversion

Since AUTOSAR 4.0, the `Rte` officially supports data conversion. This feature can be divided into three central parts:

1. Conversion that uses the `CompuMethod` in the context of an `ApplicationDataType` used by a `DataPrototype`

2. Conversion that uses the `BaseType` and `networkRepresentation` either of an `ISignal` or `ComSpec`

3. Conversion of constant values, i.e. `invalidValue` and `initValue` that use an `ApplicationValueSpecification`

The EB tresos AutoCore Generic 8 RTE supports the following parts of the data conversion feature:

► Linear conversion of data in unqueued intra-partition sender-receiver communication for `DataPrototype` elements typed by an `ApplicationPrimitiveDataType`

► Linear conversion of data in unqueued intra-partition sender-receiver communication for `DataPrototype` elements typed by a primitive `ImplementationDataType`

► Linear conversion of data in unqueued intra-partition sender-receiver communication from `DataPrototype` element typed by a primitive `ImplementationDataType` to a `DataPrototype` element typed by an `ApplicationPrimitiveDataType`

► Text table data representation (enumeration) to another text table data representation (enumeration) conversion of data in unqueued intra-partition sender-receiver communication for `DataPrototype` elements typed by a primitive `ImplementationDataType`

► Text table data representation (enumeration) to another text table data representation (enumeration) conversion of data in unqueued intra-partition sender-receiver communication for `DataPrototype` elements typed by an `ApplicationPrimitiveDataType`

► Text table data representation (enumeration) of `DataPrototype` element typed by an `Application-PrimitiveDataType` to another text table data representation (enumeration) of an `DataPrototype` element typed by a primitive `ImplementationDataType` conversion of data in unqueued intra-partition sender-receiver communication

► Mixed linear-scaled and text table data representation to another mixed linear-scaled and text table data representation conversion of data in unqueued intra-partition sender-receiver communication for `Dat-aPrototype` elements typed by an `ApplicationPrimitiveDataType`

► Mixed linear-scaled and text table data representation to another mixed linear-scaled and text table data representation conversion of data in unqueued intra-partition sender-receiver communication for `Dat-aPrototype` elements typed by a primitive `ImplementationDataType`

► Definition of constants in the context of an `ApplicationDataType` that use a `NumericalValueSpec-ification` without applying a constant mapping

The following chapters provide an overview of the implemented features and describe the potential risk for the application.

4.2.20.1. Linear conversion in sender-receiver communication

The `Rte` performs a linear conversion if n connected senders and receivers have compatible type semantics in terms of the applied `CompuMethod` elements. If the type semantics are identical, no conversion is performed. The conversion formula is derived from the applied `CompuMethod` elements. Since the data processed by the `Rte` is always an internal representation, it is not necessary to convert between the *physical* and *internal* representation. The conversion formula between several *internal* values is calculated by putting the `CompuMethod` of the sender in relation to the `CompuMethod` defined by the receiver.

The conversion according to the coefficients provided by a `CompuMethod` is performed by setting the `Com-puMethod` of the receiver in relation to the `CompuMethod` of the sender. The following example is taken from the `Rte` specification:

► Affected `ImplementationDataType`: `uint8` for both sender and receiver

► `CompuMethod` 1: `factor1 = 1/4, offset1 = 0`

► `CompuMethod` 2: `factor2 = 1/2, offset2 = 1/2`

► Resulting conversion formula: `factor1 * x + offset1 = factor2 * y + offset2`

`y = (uint8)( (1 * x)/2 - 1)`

```
y(0) = (uint8)( (1 * 0)/2 - 1) = (uint8)-1 = implementation specific
y(1) = (uint8)( (1 * 1)/2 - 1) = (uint8)-1 = implementation specific
y(2) = (uint8)( (1 * 2)/2 - 1) = 0
y(3) = (uint8)( (1 * 3)/2 - 1) = 0
y(4) = (uint8)( (1 * 4)/2 - 1) = 1
y(5) = (uint8)( (1 * 5)/2 - 1) = 1
y(6) = (uint8)( (1 * 6)/2 - 1) = 2
```

WARNING

The risk of introducing undefined or unspecified behavior is quite high if you use data conversion. Use this feature only if you know the effects on the generated code.

The `Rte` does not perform range checks during run-time. Without any range check the behavior may be unspecified or undefined. It is highly recommended to perform such checks on application level before you send any data via the `Rte` if data conversion is configured for this signal path.

4.2.21. BSW Module Description

4.2.21.1. Content of the BSW Module Description

The `Rte` provides a BSW Module Description in generation phase which contains the following information:

► BSW Module Entry elements for the lifecycle API

► All common memory sections that are used by the generated Rte

► The `McSupport` element containing `McDataInstance` elements for measurement and calibration support

4.2.21.2. Measurement and calibration support using `McSupport`

The basic idea behind the export of measurement and calibration information is to have an intermediate format which can be used as a basis to generate an A2L-file for the `Rte` code. The generator provides entities for `VariableDataPrototype` and `ParameterDataPrototype` elements for which the attribute `swCalibrationAccess` is configured to `readOnly` or `readWrite`. The resulting measurement entities, that are either `McDataInstance` or `McParameterInstance` elements, represent measurable objects within the generated code. A `McDataInstance` (and `McParameterInstance`) and its subelements reflect thereby the structure

of the actual data object. Although the origin of each `McDataInstance` can be traced back into the hierarchical model via the referenced flat-map entry, each instance is self-contained. Self-contained in the sense that it contains all the data type information necessary to be easily processed by external tools later on.

The following paragraphs provide an overview of `McDataInstance` elements for different data types. The examples are also applicable for `McParameterInstance` elements.

4.2.21.2.1. `McDataInstance` for primitive types

Assume, you have a connected require port with a sender-receiver interface which receives the non-queued data element `deInteger1` of the primitive type uint32. If measurement is enabled and the `SW-CALIBRATION-ACCESS` of the data element is set to read-only, the generated `MC-DATA-INSTANCE` could look like this:

```
<MC-DATA-INSTANCE>
  <SHORT-NAME>MEASUREMENT_PRIMITIVE</SHORT-NAME>
  <CATEGORY>VALUE</CATEGORY>
  <FLAT-MAP-ENTRY-REF DEST="FLAT-INSTANCE-DESCRIPTOR">
/EcuExtract/FlatMap/deInteger1</FLAT-MAP-ENTRY-REF>
  <RESULTING-PROPERTIES>
    <SW-DATA-DEF-PROPS-VARIANTS>
      <SW-DATA-DEF-PROPS-CONDITIONAL>
        <BASE-TYPE-REF DEST="SW-BASE-TYPE">
/AUTOSAR_Platform/BaseTypes/uint32</BASE-TYPE-REF>
        <SW-CALIBRATION-ACCESS>READ-ONLY</SW-CALIBRATION-ACCESS>
        <DATA-CONSTR-REF DEST="DATA-CONSTR">
/AUTOSAR_Platform/DataConstrs/uint32</DATA-CONSTR-REF>
      </SW-DATA-DEF-PROPS-CONDITIONAL>
    </SW-DATA-DEF-PROPS-VARIANTS>
  </RESULTING-PROPERTIES>
  <SYMBOL>Rte_SwcBDS.Rte_ReceiveBuffer_1.value</SYMBOL>
</MC-DATA-INSTANCE>
```

NOTE	A `McDataInstance` always shows the base type
(i)	The resulting properties always show the base type instead of an implementation or application data type. This is different to the original definition of the data element in the sender-receiver interface.

NOTE	The symbol targets to a global C-identifier
(i)	The `SYMBOL` attribute shows a C-symbol with external declaration which reflects the received value, i.e. the value that is returned by `Rte_Read`.

WARNING	BSW Module Description not compliant to AUTOSAR 4.0 XML schema
(!)	The measurement data may be nested in structures like the buffer data structure as shown in the listing above. Therefore the value of the symbol attribute may represent a C-style path to the structure element separated by dots. Unfortunately, the AUTOSAR 4.0 XML schema forbids dots in the symbol attribute and so the exported `Rte_Bswmd.arxml` file may not be schema-compliant anymore. The AUTOSAR 4.1 schema was adapted accordingly, though.

4.2.21.2.2. `McDataInstance` for array types

The `McDataInstance` of a complex data type additionally contains subelements. If you enable measurement for a data element of the uint8-based array type `TestString`, the resulting output could look like in the following example:

```
<MC-DATA-INSTANCE>
  <SHORT-NAME>deTestString_2</SHORT-NAME>
  <CATEGORY>ARRAY</CATEGORY>
  <ARRAY-SIZE>8</ARRAY-SIZE>
  <FLAT-MAP-ENTRY-REF DEST="FLAT-INSTANCE-DESCRIPTOR">
/EcuExtract/FlatMap/deTestString_2</FLAT-MAP-ENTRY-REF>
  <RESULTING-PROPERTIES>
    <SW-DATA-DEF-PROPS-VARIANTS>
      <SW-DATA-DEF-PROPS-CONDITIONAL>
        <SW-CALIBRATION-ACCESS>READ-ONLY</SW-CALIBRATION-ACCESS>
      </SW-DATA-DEF-PROPS-CONDITIONAL>
    </SW-DATA-DEF-PROPS-VARIANTS>
  </RESULTING-PROPERTIES>
  <SUB-ELEMENTS>
    <MC-DATA-INSTANCE>
      <SHORT-NAME>TestString</SHORT-NAME>
      <CATEGORY>VALUE</CATEGORY>
      <RESULTING-PROPERTIES>
        <SW-DATA-DEF-PROPS-VARIANTS>
          <SW-DATA-DEF-PROPS-CONDITIONAL>
```

```
            <BASE-TYPE-REF DEST="SW-BASE-TYPE">
/AUTOSAR_Platform/BaseTypes/uint8</BASE-TYPE-REF>
            <SW-CALIBRATION-ACCESS>READ-ONLY</SW-CALIBRATION-ACCESS>
            <DATA-CONSTR-REF DEST="DATA-CONSTR">
/AUTOSAR_Platform/DataConstrs/uint8</DATA-CONSTR-REF>
          </SW-DATA-DEF-PROPS-CONDITIONAL>
        </SW-DATA-DEF-PROPS-VARIANTS>
      </RESULTING-PROPERTIES>
    </MC-DATA-INSTANCE>
  </SUB-ELEMENTS>
  <SYMBOL>Rte_SwcBDS.Rte_ReceiveBuffer_0.value</SYMBOL>
</MC-DATA-INSTANCE>
```

NOTE   **Arrays always contain one subelement that shows the array's base type**

Similar to the definition of an array data type, the `McDataInstance` for an array always
contains exactly one subelement that represents the base type of the array regardless of
the actual size of the array. This size is defined by the attribute `ARRAY-SIZE`.

4.2.21.2.3. `McDataInstance` for structure types

Finally if your measured data element is a structure data type with three elements (`a` of type uint32, `b` of type
uint32, `c` of the `TestString` data type shown above), the following snippet is a valid example for the exported
`McDataInstance`:

```
<MC-DATA-INSTANCE>
  <SHORT-NAME>deTestRecord_2</SHORT-NAME>
  <CATEGORY>STRUCTURE</CATEGORY>
  <FLAT-MAP-ENTRY-REF DEST="FLAT-INSTANCE-DESCRIPTOR">
/EcuExtract/FlatMap/deTestRecord_2</FLAT-MAP-ENTRY-REF>
  <RESULTING-PROPERTIES>
    <SW-DATA-DEF-PROPS-VARIANTS>
      <SW-DATA-DEF-PROPS-CONDITIONAL>
        <SW-CALIBRATION-ACCESS>READ-ONLY</SW-CALIBRATION-ACCESS>
      </SW-DATA-DEF-PROPS-CONDITIONAL>
    </SW-DATA-DEF-PROPS-VARIANTS>
  </RESULTING-PROPERTIES>
  <SUB-ELEMENTS>
    <MC-DATA-INSTANCE>
      <SHORT-NAME>a</SHORT-NAME>
      <CATEGORY>VALUE</CATEGORY>
      <RESULTING-PROPERTIES>
        <SW-DATA-DEF-PROPS-VARIANTS>
          <SW-DATA-DEF-PROPS-CONDITIONAL>
            <BASE-TYPE-REF DEST="SW-BASE-TYPE">
```

```
/AUTOSAR_Platform/BaseTypes/uint32</BASE-TYPE-REF>
          <SW-CALIBRATION-ACCESS>READ-ONLY</SW-CALIBRATION-ACCESS>
        </SW-DATA-DEF-PROPS-CONDITIONAL>
      </SW-DATA-DEF-PROPS-VARIANTS>
    </RESULTING-PROPERTIES>
    <SYMBOL>a</SYMBOL>
  </MC-DATA-INSTANCE>
  <MC-DATA-INSTANCE>
    <SHORT-NAME>b</SHORT-NAME>
    <CATEGORY>VALUE</CATEGORY>
    <RESULTING-PROPERTIES>
      <SW-DATA-DEF-PROPS-VARIANTS>
        <SW-DATA-DEF-PROPS-CONDITIONAL>
          <BASE-TYPE-REF DEST="SW-BASE-TYPE">
/AUTOSAR_Platform/BaseTypes/uint32</BASE-TYPE-REF>
          <SW-CALIBRATION-ACCESS>READ-ONLY</SW-CALIBRATION-ACCESS>
        </SW-DATA-DEF-PROPS-CONDITIONAL>
      </SW-DATA-DEF-PROPS-VARIANTS>
    </RESULTING-PROPERTIES>
    <SYMBOL>b</SYMBOL>
  </MC-DATA-INSTANCE>
  <MC-DATA-INSTANCE>
    <SHORT-NAME>c</SHORT-NAME>
    <CATEGORY>ARRAY</CATEGORY>
    <ARRAY-SIZE>8</ARRAY-SIZE>
    <RESULTING-PROPERTIES>
      <SW-DATA-DEF-PROPS-VARIANTS>
        <SW-DATA-DEF-PROPS-CONDITIONAL>
          <SW-CALIBRATION-ACCESS>READ-ONLY</SW-CALIBRATION-ACCESS>
        </SW-DATA-DEF-PROPS-CONDITIONAL>
      </SW-DATA-DEF-PROPS-VARIANTS>
    </RESULTING-PROPERTIES>
    <SUB-ELEMENTS>
      <MC-DATA-INSTANCE>
        <SHORT-NAME>TestString</SHORT-NAME>
        <CATEGORY>VALUE</CATEGORY>
        <RESULTING-PROPERTIES>
          <SW-DATA-DEF-PROPS-VARIANTS>
            <SW-DATA-DEF-PROPS-CONDITIONAL>
              <BASE-TYPE-REF DEST="SW-BASE-TYPE">
/AUTOSAR_Platform/BaseTypes/uint8</BASE-TYPE-REF>
              <SW-CALIBRATION-ACCESS>READ-ONLY</SW-CALIBRATION-ACCESS>
              <DATA-CONSTR-REF DEST="DATA-CONSTR">
/AUTOSAR_Platform/DataConstrs/uint8</DATA-CONSTR-REF>
            </SW-DATA-DEF-PROPS-CONDITIONAL>
          </SW-DATA-DEF-PROPS-VARIANTS>
        </RESULTING-PROPERTIES>
```

```
      </MC-DATA-INSTANCE>
    </SUB-ELEMENTS>
    <SYMBOL>c</SYMBOL>
  </MC-DATA-INSTANCE>
  </SUB-ELEMENTS>
  <SYMBOL>Rte_SwcBDS.Rte_ReceiveBuffer_0.value</SYMBOL>
</MC-DATA-INSTANCE>
```

NOTE   **Relative symbol attribute for subelements**

The C-symbols of the particular subelements are relative to the symbol of the whole structure. To query a single element, you can concatenate the symbols to a new structure path (e.g. `Rte_SwcBDS.Rte_ReceiveBuffer_0.value.b`).

4.2.21.2.4. Specifying the measurement ID by providing a system flat map entry

To enable measurement of a certain data protoype (e.g. a data element of a sender-receiver interface), it is sufficient to do the following:

► Set the general configuration switch `RteMeasurementSupport` to true.

► Set the attribute `SW-CALIBRATION-ACCESS` of the variable data prototype or the referenced data type to `READ-ONLY` or `READ-WRITE`.

In this case the generated `MC-DATA-INSTANCE` will be exported with an arbitrary short name. In most cases these generated short names are not desired since external tools (e.g. an A2L-converter) need to identify the entry by a well-known measurement identifier in order to extract the type and symbol information. Therefore your input model should contain a system flat map with a flat instance descriptor for each measured data prototype. The short name of the exported `MC-DATA-INSTANCE` will then be equal to the short name of the flat instance descriptor referencing the measured object.

The following snippet shows a system flat map entry that references the data element `de1` received by a certain sender-receiver require port:

```
<FLAT-MAP>
  <SHORT-NAME>systemFlatMap</SHORT-NAME>
  <INSTANCES>
    <FLAT-INSTANCE-DESCRIPTOR>
      <SHORT-NAME>MEASUREMENT_RECEIVER_PORT_2</SHORT-NAME>
      <UPSTREAM-REFERENCE-IREF>
        <CONTEXT-ELEMENT-REF DEST="ROOT-SW-COMPOSITION-PROTOTYPE">
/System/system1/rootSwPrototype</CONTEXT-ELEMENT-REF>
        <CONTEXT-ELEMENT-REF DEST="SW-COMPONENT-PROTOTYPE">
/rte_test/TestTopLevelComposition/ReceiverComponent_P1</CONTEXT-ELEMENT-REF>
```

```
      <CONTEXT-ELEMENT-REF DEST="R-PORT-PROTOTYPE">
/components/ReceiverComponent/ReceiverPort2</CONTEXT-ELEMENT-REF>
        <TARGET-REF DEST="VARIABLE-DATA-PROTOTYPE">
/interfaces/if1/de1</TARGET-REF>
      </UPSTREAM-REFERENCE-IREF>
    </FLAT-INSTANCE-DESCRIPTOR>
  </INSTANCES>
</FLAT-MAP>
```

NOTE　　　　**No measurement of queued sender-receiver communication**

(i)　　　　　Measurement of a queued sender-receiver data element is not supported by AUTOSAR.

4.2.22. Task Chains

When timing events are used, they are usually allocated by the `Rte` generator as expiry points in the schedule table. If one expiry point activates multiple tasks, the order of these activations can only be specified by using different task priorities. However, this has the drawback that certain optimizations e.g. for exclusive areas can't be applied by the `Rte generator` anymore. To still be able to have equal task priorities and a specified activation order, the concept of task chains has been introduced.

With task chains it is possible to specify the order of tasks by defining a successor task. At the end of a task, the `Rte` calls `ChainTask()` to activate the configured successor task.

Some constraints must be fulfilled to be able to configure a task chain:

▶ tasks must be basic tasks

▶ only timing events must be mapped to this tasks

▶ a task cannot have more than one successor/predecessor task

▶ the task chain must not contain a cycle

▶ task periods (calculated from the mapped timing events) must be compatible meaning that the first task must be a divisor of all periods and offsets of all successor tasks

The Rte will only allocate the expiry point for the first task of the task chain in the `Os` schedule table.

If the chained tasks have different periods, the Rte generator will ensure that each task is activated at the right point in time.

If the initial tasks has an offset, this offset will apply to the entire task chain.

4.2.22.1. Example

The following tasks with the specified properties shall be configured for a task chain in the given order:

▶ T1, period: 5ms, offset: 0ms

▶ T2, period: 10ms, offset: 5ms

▶ T3, period: 15ms, offset: 0ms

▶ T4, period: 10ms, offset: 0ms

The length of the allocated schedule table is calculated as the least common divisor of all periods and offsets. For the example above, the schedule table length will be 30ms. The Rte will allocate an expiry point in the schedule table for the first task every 5ms.

The following effective task chains will be generated:

▶ 0ms: T1 # T3 # T4

▶ 5ms: T1 # T2

▶ 10ms: T1 # T4

▶ 15ms: T1 # T2 # T3

▶ 20ms: T1 # T4

▶ 25ms: T1 # T2

The Rte uses an internal counter to ensure, that the effective task chains are executed as mentioned above.

NOTE   **Configuring task chains**

It is currently only possible to configure task chains in the `Generic Rte editor`. For that, open the **Task Chains** tab that contains a list of successor/predecessor task pairs. The above mentioned example could be configured with the following pairs (T1, T2), (T2, T3), (T3, T4).

TIP   **Bsw tasks can be part of the task chain**

If the Bsw task is configured to be triggered periodically, it can be part of the task chain, too.

4.2.23. Cooperative tasks

As described in chapter Section 4.2.17.2, "Data consistency mechanisms", the Rte ensures that internal buffers are protected against concurrent access. The Rte takes several means like task priorities, interruptibility of a task or whether the data can be written atomically to determine if a data consistency mechanism is required.

However, there are cases where those conditions are not fulfilled although you can ensure, e.g. with a defined scheduling, that the tasks cannot preempt each other. Especially in a multicore environment, the `Rte` cannot make any assumptions about the interruptibility of tasks which are mapped to different cores and have the same priority. For those cases, you can define a `Cooperative Task Group`.

A `Cooperative Task Group` is a group of tasks that cannot interrupt each other, regardless of their configured priorities, schedule, application assignment, and core assignment. This list is taken into account by the Rte Generator to optimize locks and to reduce the number of implicit buffers.

Multiple groups can be specified and each task can be part of many groups.

WARNING	**Ensure that the tasks of one group cannot interrupt each other**
⚠	The `Rte` cannot guarantee that tasks of a group never interrupt each other, nor can it detect a wrong configuration of such a group. You are responsible to verify the correctness of this information. Otherwise data corruption cannot be excluded.

4.2.24. Service port mapping

AUTOSAR application software components communicate via AUTOSAR ports and interfaces with the AUTOSAR services, e.g. `Ecu State Manager` or `NVRAM Manager`. Therefore the services have to provide a service software component description.

EB tresos Studio generates the service software component descriptions from the service module configuration. In most cases, the component type and the internal behavior of the service component description depends on the configuration of the service module. For instance, the number of ports of the `NvM` service software component depends on the number of configured `NvMBlockDescriptor`s.

The AUTOSAR interfaces normally do not depend on the configuration, i.e. they are static. An exception is for instance the `Dcm`, which requires a separate interface for each port.

The workflow for integrating your application software components with AUTOSAR services is as follows:

1. During the development of your application, i.e. while you create the software component description of your application components, you add service ports to your application software components using a system design tool or a software component editor. These service ports implement the standardized AUTOSAR service interfaces, i.e. you need the service interfaces at this point.

   There are several possibilities how you get the service interfaces:

   a. Your system design tool/software component editor provides the interfaces, e.g. in an enclosed library.

   b. You create the interfaces on your own using the system design tool/software component editor based on the AUTOSAR service software specifications.

c. You generate the service interfaces as described in Section 4.2.24.2, "Generating the service component interfaces" and import them into your software component editor / system design tool.

Approaches 1. and 2. are the most convenient ones if the service interfaces are static, i.e. do not depend on the configuration of the service.

Approach 3. is the best when the service interfaces depends on the module configuration of the service. If you choose this option to generate the service interfaces, proceed to Section 4.2.24.2, "Generating the service component interfaces".

2. After you have finished the design of your application software components, you export the application software component description or complete system description and import it into EB tresos Studio. This step is described in Section 4.3.1, "Importing system descriptions".

3. Finally, you have to generate the service component descriptions and map the application ports to service ports in EB tresos Studio with the **Connection Editor**.

WARNING	**You cannot use the Rte Editor any more to connect application and service ports**
⚠	The functionality to connect application and service ports is now available in the **Connection Editor**.

4.2.24.1. Preconditions for generating the service components and interfaces

Before you can generate the service components or service component interfaces, do the following:

▶ Add the required service modules (e.g. the `Det`) to your project.

▶ Configure the service modules, e.g. for the `Det`, add an entry to the **Det Accessing Software Components** list

▶ Enable the option **Enable Rte Usage** of the service modules.

► Enable the generation of the service modules in the **Module Configurations** dialog.
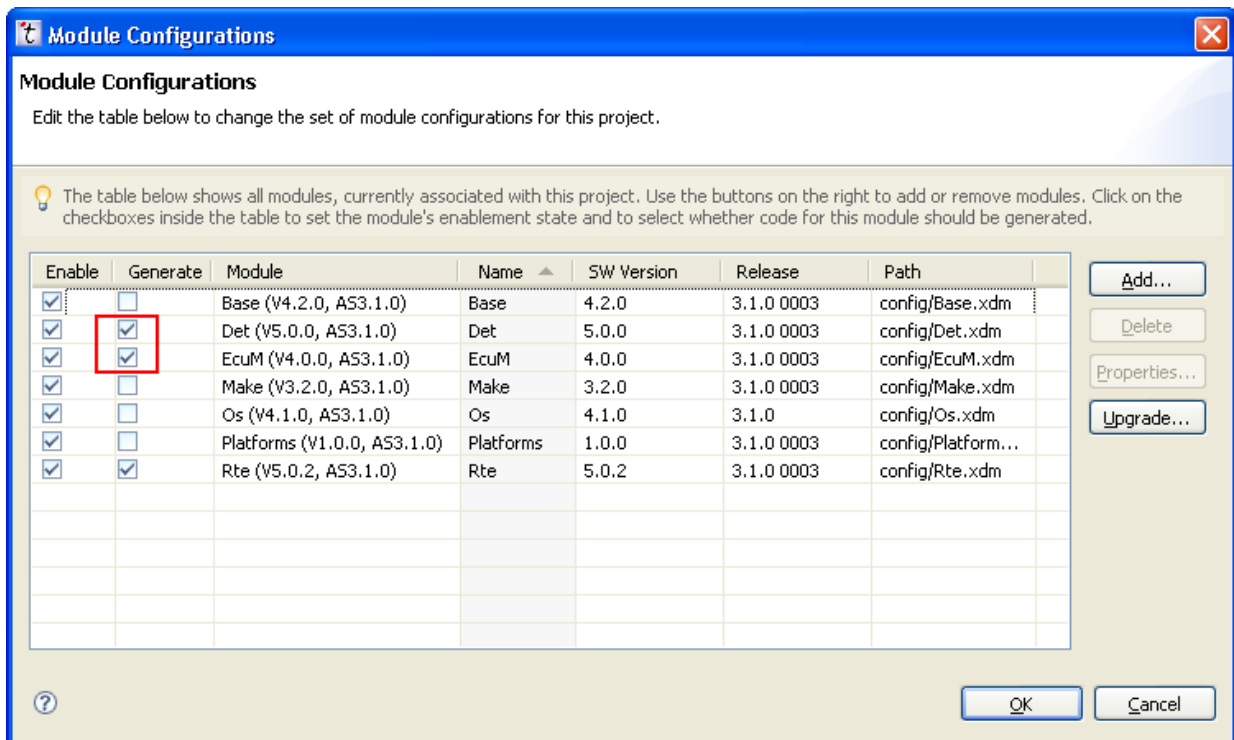
► Check that the service module configuration does not contain any errors, i.e. no errors are shown in the **Problems View** for the service modules you want to use. In case of errors, the generation of the service components or interfaces will fail with an error message.

4.2.24.2. Generating the service component interfaces

TIP	**Skip this step if your software component editor contains AUTOSAR interfaces**
	This step is not required if you already have the standardized AUTOSAR interfaces available in you software component editor/system design tool.

The service component descriptions including the interfaces are generated by the generators of the service modules. To generate them, proceed as follows:

► Make sure that the preconditions given in Section 4.2.24.1, "Preconditions for generating the service components and interfaces" are fulfilled for the service modules you want to use.

► To generate the service component description, select **Build Project** in the context menu of your configuration, then select **generate_swcd**.
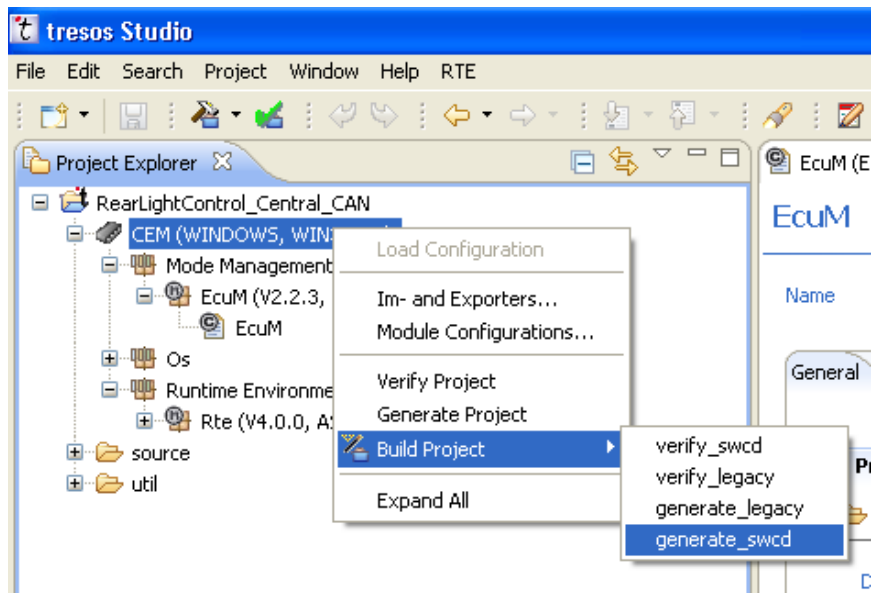
In the `output/swcd`, you will then find the following files for each service:

▶ `<Service>_swc_interface.arxml`

▶ `<Service>_swc_internal.arxml`

The first contains the service interfaces, the latter the component type, internal behavior and implementation of the service component.

Finally you can import the generated service component interfaces located in `<Service>_swc_interface.arxml` in your system design tool or software component editor. Then the service interfaces are available and you can assign them to the service ports of your application.

TIP **EB tresos Studio instantiates the service components automatically**

In most cases, you do not need to import the files `<Service>_swc_internal.arxml` into you software component editor/system design tool. The instantiation of the service components is handled by the **Connection Editor**.

4.3. Configuring and generating the Rte

The following chapters provide you with step-by-step instructions to configure and generate `Rte` module in the full generation mode.

Proceed through the instructions below step-by-step in the order presented for the best possible results.

The instructions are Rte-specific and do not go into details as far as the EB tresos Studio graphical user interface (GUI) is concerned. Therefore, if you need further assistance with the EB tresos Studio GUI and opening or

closing a project, importing data, etc. consult the `EB tresos Studio documentation`, chapter `EB tresos Studio user's guide`.

4.3.1. Importing system descriptions

The Rte Generator requires three kinds of information for the `Rte` to be generated:

► the ECU configuration,

► the description of the system and the software components for which the `Rte` is to be generated, and

► the *Ecu Extract* from the imported system.

When you add the `Rte` module to a EB tresos Studio project, a *System Description Importer* is automatically added to the project. After importing the system description, the Ecu Extract Creator wizard must be run to create the Ecu Extract.

For details how to add, edit or delete modules to a project, consult the `EB tresos Studio documentation`, chapter `Using the GUI`, section `Creating a new project`, `Adding, deleting, and setting up module configurations`.

For details how to import a system description, consult the `EB tresos Studio documentation`, chapter `Importing and Exporting`, section `Importing AUTOSAR system descriptions`.

4.3.1.1. Avoiding conflicts in interface compatibility and names

When importing software component and/or basic software module descriptions, the `Rte` module checks the compatibility of the interfaces of connected ports, data types and computation methods. The compatibility checks adhere to the rules specified in [4].

Since the short name of an element must only be unique within an AR-PACKAGE, you need to perform some additional checks when importing the software component descriptions.

► Component types:

Make sure that the short names of component types are globally unique since the name of the *component type* is used as C identifier in Rte data structures.

 ► If the importer finds component types with the same short name, the import fails with an error message. Adapt your software component descriptions accordingly.

► Data types:

Make sure that there are not two data types with the same name. The short names of data types are directly used as name for the C type.

> ► If the importer finds two data types with the same short name, it checks if the definition is compatible. If they are compatible, only one data type is imported.

> ► If they are not compatible, the import fails with an error. Fix the error in your software component descriptions.

If you are going to import AUTOSAR elements with the same short name path (i.e. they have the same short name and are located in the same package), the importer will ignore duplicate elements and report a warning. The import might fail if elements with the same short name path are not compatible.

4.3.2. Using the Service Needs Calculator

The `Service Needs Calculator` helps you resolve configuration dependencies between different EB tresos AutoCore modules and software components. The `Service Needs Calculator` is a contract between modules that provides a service, e.g. for the service provider and the modules that use or request these services. For example, the Rte can generate the schedule table by itself and save the generated schedule table as an ECU configuration. The `Service Needs Calculator` can also resolve dependencies that may not have a direct reference in the service requester.

Always run the `Service Needs Calculator` after the following situations:

► Software component descriptions were updated.

► Basic software module descriptions were updated.

► Basic system descriptions were updated.

► Software component partition mapping was changed.

► Scheduling, e.g. event to task mapping, was changed.

► NvRam allocation was changed.

► BSW module communication was changed.

The `Service Needs Calculator` allocates the following Os objects and their relevant properties. The following table lists the dependencies for `OSSERVICES`:

Service Needs	Functionality
Os events	Scheduling
Os alarms	Scheduling
Os schedule table	Scheduling
Os spinlocks	Data consistency
Os resources	Data consistency, exclusive areas

Service Needs	Functionality
IOC channels	Inter-partition communication

Table 4.1. OSSERVICES

These are the dependencies for `COMSERVICES`:

Service Needs	Functionality
Com signals	Inter-ECU communication
Com signal groups	Inter-ECU communication

Table 4.2. COMSERVICES

These are the dependencies for `LDCOMSERVICES`:

Service Needs	Functionality
LdCom PDUs	Inter-ECU communication

Table 4.3. LDCOMSERVICES

These are the dependencies for `XFRMSERVICES`:

Service Needs	Functionality
Xfrm mapping	Data transformation

Table 4.4. XFRMSERVICES

4.3.3. BSW module configuration

The configuration of any BSW module added to a project should take place before the Rte configuration. If a BSW module offers services to software components, ensure that the check box **Enable Rte usage** is selected for that module.

If the `Dem` module is available, the `Service Needs Calculator` should be run after the `Dem` module is configured.

4.3.4. Configuring the Rte

To configure an ECU, you first need to configure the `Rte` according to the following steps:

1. Configure the general parameters (Section 4.3.4.1, "Configuring general parameters").

2. Select an implementation for each software component (Section 4.3.4.2, "Selecting an implementation for each software component instance").

3. Map the software components to partitions (Section 4.3.4.3, "Configuring the partitioning support").

4. Configure the optimization parameters (Section 4.3.4.4, "Configuring optimization options").

5. Map the executable entities to Os tasks (Section 4.3.4.5, "Mapping executable entities to Os tasks").

6. Map the executable entities to Os ISRs (Section 4.3.4.6, "Mapping executable entities to Os ISRs").

7. Map the data element prototypes to Com signals/signal groups or system signals/system signal groups (Section 4.3.4.7, "Data mapping").

8. Select an implementation mechanism for exclusive areas (Section 4.3.4.8, "Configuring exclusive areas").

9. Configure the measurement and calibration options (Section 4.3.4.9, "Configuring the measurement and calibration support").

10. Configure the NVRAM allocation (Section 4.3.4.10, "Configuring the NVRAM allocation").

11. Configure the VFB tracing (Section 4.3.4.11, "Configuring the VFB tracing").

12. Configure BSW Trigger connections(Section 4.3.4.12, "BSW Trigger Connections").

13. Configure BSW Mode mapping (Section 4.3.4.13, "BSW Mode Mapping").

The following sections detail the configuration using the **Rte Editor** provided by the EB tresos AutoCore.

The **Rte Editor** is a specific editor for the Rte module, which allows the configuration of the Rte module in a more convenient way than the generic editor. Of course, you can also use the generic editor to configure the Rte. In the following sections, the configuration process using the **Rte Editor** is explained.

NOTE

The Rte Editor modifies the configuration only when you close the Rte Editor
The **Rte Editor** does not modify the ECU configuration until you close the **Rte Editor**. When closing the **Rte Editor** you are asked whether your changes to the configuration shall be applied or not.

While the **Rte Editor** is opened, you are not able to import files, generate code or edit other configurations.

To open the **Rte Editor**, double-click on the corresponding label below the Rte module configuration in the **Project Explorer**.

RearLightControl
　　CEM (WINDOWS, WIN32X86)
　　　　EB
　　　　Error Handler
　　　　Mode Management
　　　　Os
　　　　Runtime Environment
　　　　　　Rte (V5.0.2, AS3.1.0)
　　　　　　　　Rte
　　　　　　　　　　Generic Editor
　　　　　　　　　　Rte Editor
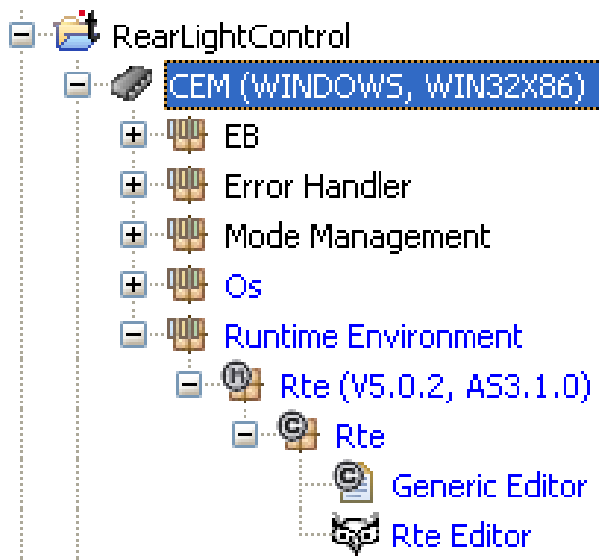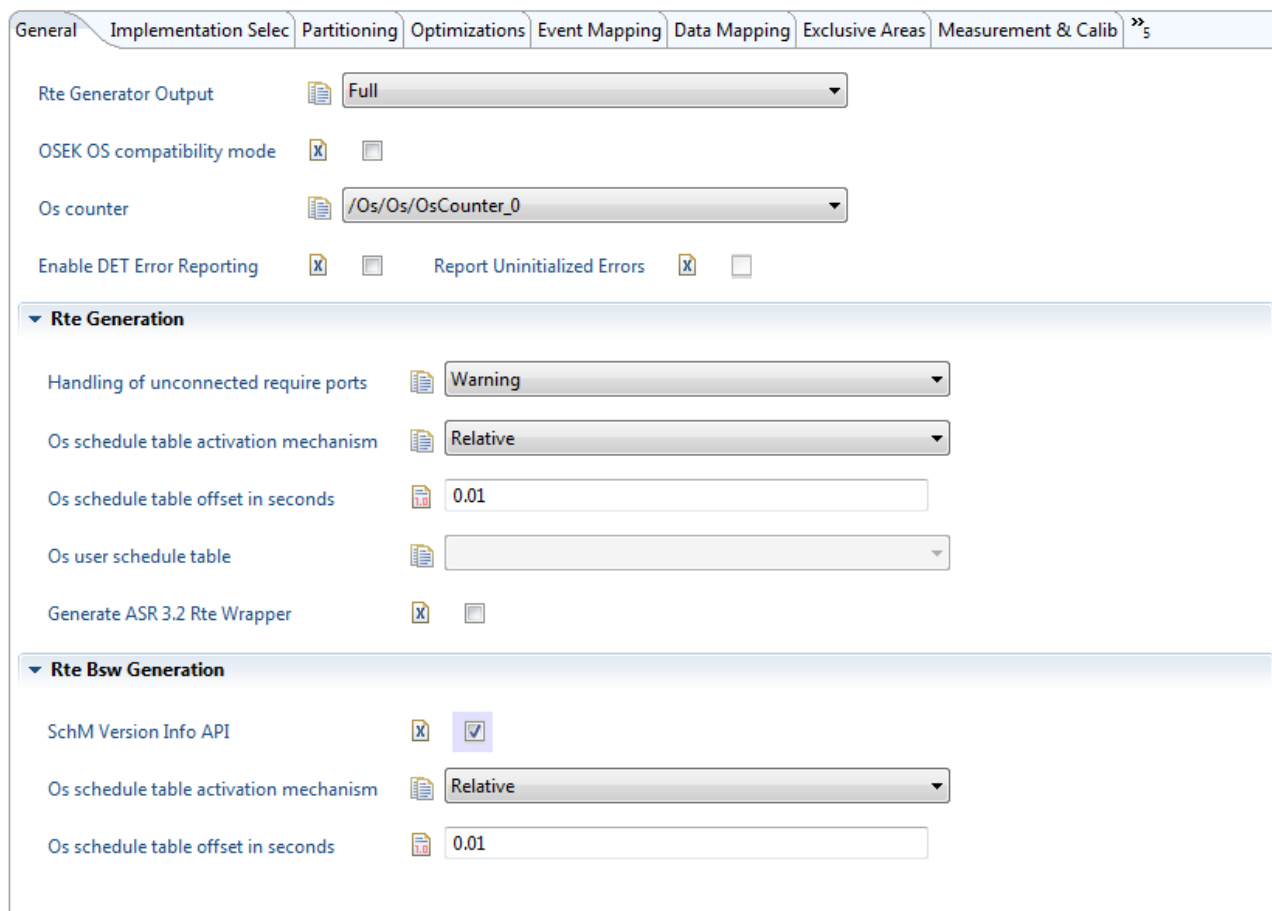
TIP	**Getting to know the EB tresos Studio GUI**
	For general information about the EB tresos Studio GUI and how to configure a module using the generic editor, consult the `EB tresos Studio documentation`, chapter `The GUI main window`, `The Editor view` and the chapter `Editing parameters of a module configuration`.

4.3.4.1. Configuring general parameters

In the **Rte Editor** view, select the **General** tab:

Each of the following chapters shows you what each menu item of this screen stands for and how to configure it.

4.3.4.1.1. Rte generator output

The Rte generation mode is configured by the drop-down list **Rte generator output**. The possible options are

▶ **Rte only** The `Rte` will be generated without the `BSW Scheduler` artifacts.

▶ **BSW Scheduler only** Only the `BSW Scheduler` artifacts will be generated. This mode can be used for testing and delivering basic software modules.

▶ **Full** Both the `Rte` and `BSW Scheduler` artifacts will be generated.

4.3.4.1.2. Configuring the OSEK OS compatibility mode

In the OSEK OS compatibility mode, the Rte Generator does not use AUTOSAR `Os`-specific functions such as `schedule tables`. Setting the generator to Rte, OSEK compatibility mode enables using the `Rte` generated with an OSEK-compliant operating system. To generate the `Rte` with an OSEK-compliant operating system:

▶ Locate the **General** tab, section **Rte Generation**, check box **OSEK OS compatibility mode**.

▶ Check the **OSEK OS compatibility mode**.

▶ Configure the initial offset that is used when starting each alarm by entering a value for the parameter **Os schedule table alarm offset.**

The `Rte` is now set to generate with an OSEK-compliant operating system.

NOTE	**The OSEK OS compatibility mode and the compatibility mode of the Rte Generator might be confused**
ⓘ	The *OSEK OS compatibility mode* must not be confused with the compatibility mode of the Rte Generator. The latter only indicates that the `Rte` generated complies with the requirements of the compatibility mode. The compatibility mode ensures compatibility of Rte Generators by different vendors between contract phase and generation phase.

4.3.4.1.3. Configuring the Os counter

The `Rte` requires an `Os` counter (either software or hardware) to start alarms for time-out monitoring and to set up a schedule table for triggering time-triggered runnable entities. Therefore the `Rte` ECU configuration refers to an `Os` counter. The drop-down list provides a list of all counters configured in the `Os` configuration.

▶ Select the drop-down list of the **Os counter** menu item.

▶ Choose a counter from the list.

The `Os` counter is now configured.

NOTE	**Hardware counters with a resolution smaller than 1 ns are not supported**
ⓘ	Currently it is not possible to use hardware counters that require a resolution smaller than 1 ns. Use a software counter instead.

4.3.4.1.4. Configuring DET Error Reporting

If the `Rte` shall report development errors to the `DET` module, the checkbox **Enable DET Error Reporting** must be enabled.

The following types of errors are reported to the `DET` module:

▶ `RTE_E_DET_ILLEGAL_INVOCATION` - reported when a runnable calls an API for that it is not configured

▶ `RTE_E_DET_WAIT_IN_EXCLUSIVE_AREA` - reported when the application enters a wait state by calling a blocking `Rte` API function although an exclusive area that was entered with `Rte_Enter()` was not left with a `Rte_Exit()` call

▶ `RTE_E_DET_ILLEGAL_NESTED_EXCLUSIVE_AREA` - reported when the application does not exit exclusive areas in the reverse order they were entered

Additionally, if the checkbox **Report Uninitialized Errors** is enabled, the Rte will also report the `RTE_E_DET_-UNINIT` error. This error will be reported when an API has been called before the partition was started or after it has been stopped.

NOTE	**Impact on runtime and memory**
(i)	Note that the `DET` error reporting feature may significantly increase runtime and memory consumption of the generated `Rte`.

4.3.4.1.5. Rte Generation

The section **Rte Generation** contains configuration elements pertaining to the core Rte.

4.3.4.1.5.1. Configuring the handling of unconnected require ports

Unconnected require ports are considered an invalid configuration. Nevertheless, the `EB tresos AutoCore Rte` supports unconnected require ports.

Thus you have a choice on how to treat unconnected require ports.

In the **Handling of unconnected require ports** menu item line, select the message you would like the generator to produce. Your options are:

▶ `Warning` (default):

The generator reports a warning for each unconnected require port, but still generates code.

▶ `Error`:

The generator reports an error for each unconnected require port, code is not generated.

▶ `Ignore`:

The generator ignores all unconnected require ports, no error or warning is reported.

4.3.4.1.5.2. Configuring the Os schedule table activation

You may activate the `Os` schedule table for Rte timing events in three different ways. The recommended activation depends on how the `Os` schedule table activation mechanism is configured. Select one of the following activation mechanisms from the **Os schedule table activation mechanism** drop-down list:

Choose entry:	Configures the activation mechanism
Absolute	To make the API function `Rte_Start()` start the schedule table with an *absolute* offset value, select **Absolute**. Starting with an absolute offset value means the schedule table starts running when the counter value equals the offset value. If you select **Absolute**, configure the schedule table offset with the parameter **Os schedule table alarm offset**. Specify the value in nanoseconds.
Relative	To make the API function `Rte_Start()` start the schedule table with a *relative* offset value, configure **Relative** as activation mechanism. Starting the schedule table with a relative offset value means the schedule table starts running when the counter value equals the current counter value plus offset value. If you select **Relative**, configure the schedule table offset with the parameter **Os schedule table alarm offset**. Specify the value in nanoseconds.
Next	If you select **Next**, the API function `Rte_Start()` uses the `Os` API `NextScheduleTable()` to start the `Rte` schedule table. Therefore you also have to configure a **Os user schedule table**. The `Os` then stops the execution of the user schedule table after a complete run and starts the `Rte` schedule table. Moreover, the **Rte Editor** merges the expiry points and actions of the user schedule table to the Rte schedule table so that the actions of the user schedule table are still triggered after the Rte is started. This is helpful if you want to synchronize your system with e.g. a FlexRay bus. When you configure **Next** as schedule table activation mechanism and select the schedule table of the `BSW Scheduler` as user schedule table, you only have to synchronize one schedule table in your system. You may configure the adjustable expiry points for both, the `BSW Scheduler` schedule table and for the `Rte` schedule table.

Table 4.5. Selection criteria for the Os schedule table activation mechanism

4.3.4.1.5.3. Configuring the Os schedule table offset

The Os schedule table offset value is a floating point value that directs the Os how many seconds to wait before processing the first expiry point of the schedule table.

4.3.4.1.5.4. Configuring the user Os schedule table

The user Os schedule table must be configured when the schedule table activation mechanism **Next** has been configured. The `Rte` schedule table will run after the user Os schedule table has been executed.

4.3.4.1.5.5. Configuring the AUTOSAR 3.2 wrapper

When the check box **Generate AUTOSAR 3.2 wrapper** is selected, the `Rte` will generate code that wraps the AUTOSAR 4.0 API and will provide it with the AUTOSAR 3.2 signature.

You will find details about the AUTOSAR 3.2 wrapper in [Section 4.2.19, "AUTOSAR 3.2 wrapper"](#).

4.3.4.1.6. Rte BSW generation

4.3.4.1.6.1. Configuring the BSW Scheduler Version API

When the checkbox **SchM Version API** is selected, the `BSW Scheduler` interface for programmatically accessing the `BSW Scheduler` version information will be generated.

4.3.4.1.6.2. Configuring the BSW Scheduler Os schedule table

You may activate the `Os` schedule table for BSW timing events in two different ways. The recommended activation depends on how the `Os` schedule table activation mechanism is configured. Select one of the following activation mechanisms from the **Os schedule table activation mechanism** drop-down list:

Choose entry:	Configures the activation mechanism
Absolute	To direct the API function `SchM_Init()` start the schedule table with an *absolute* offset value, select **Absolute**. Starting with an absolute offset value means the schedule table starts running when the counter value equals the offset value. If you select **Absolute**, configure the schedule table offset with the parameter **Os schedule table alarm offset**. Specify the value in nanoseconds.
Relative	To direct the API function `SchM_Init()` start the schedule table with a *relative* offset value, configure **Relative** as activation mechanism. Starting the schedule table with a relative offset value means the schedule table starts running when the counter value equals the current counter value plus offset value. If you select **Relative**, configure the schedule table offset with the parameter **Os schedule table alarm offset**. Specify the value in nanoseconds.

Table 4.6. Selection criteria for the Os schedule table activation mechanism

4.3.4.1.6.3. Configuring the BSW Scheduler Os schedule table offset

The Os schedule table offset value is a floating point value that directs the Os how many seconds to wait before processing the first expiry point of the `BSW Scheduler` schedule table.

4.3.4.1.6.4. Configuring the BSW Scheduler exclusive area legacy support

The `AUTOSAR` 3.1 `BSW Scheduler` API for exclusive areas is not compatible to the `AUTOSAR` 4.0 API. If **BSW Scheduler exclusive area support** is enabled, C function-like macros will be generated for each basic software module, which map the legacy API to the `AUTOSAR` 4.0 API.

4.3.4.2. Selecting an implementation for each software component instance

For each software component instance, select an implementation on the **Implementation Selection** tab. The basic software modules listed are there for informational purposes, they must not be configured.

In the column **Implementation**, select one of the available implementations from the drop-down list.

If there is only one implementation available for each component, the **Rte Editor** already has automatically chosen this implementation as default.

Moreover, you are optionally able to configure a specific instance name for each software component instance in the **Instance name** column. The instance name is used in names of internal data structures (e.g. component data structure instance) of the generated `Rte`.

You may want to change it to be able to better identify certain data structures during debugging or calibration.

If you don't configure the instance name, the **Rte Editor** provides reasonable default names for each software component instance.

If you want to remove a software component instance, e.g. because it is not needed in your application, select this software component instance in the table and click on the **Remove selected software component**

instance(s) button ( ).

You can also remove more than one software component instance at a time, by selecting multiple software component instances in the table and clicking on the **Remove selected software component instance(s)** button.

When removing a software component instance, be aware of the following:

▶ The target component prototype, which is referenced by the selected software component instance, is removed.

▶ If several software component instances reference the same target prototype, all these software component instances are removed.

▶ The selected software component instance references a target component prototype. The component type of this target component prototype must be instantiated by another target component prototype. If the component type of this target component prototype is not instantiated by another target component prototype, this component type is removed also.

The same applies to the internal behavior and the implementation that correspond to the component type: if they are not instantiated, they are removed.

▶ All runnable entities of the removed software component instance(s) are removed. Thus the mapping of those runnable entities to tasks is removed as well.

▶ The mapping of signals to data elements that belong to a port of the software component instance(s) are removed as well.

▶ All service connectors are removed that connect application ports of the software component instance with ports of a service component.

▶ If you want to remove the internal behavior(s) that references the component type of the removed target component prototype as well, the following is also deleted: corresponding exclusive areas and NvRam allocations, as well as the calibration support information.

4.3.4.3. Configuring the partitioning support

On the **Partitioning** tab you can enable the partitioning support. If the partitioning support is enabled, you can configure the partitioning options for the BSW and map each software component separately to a partition. Moreover, you are able to select the `Os` counter to be used for each partition.

NOTE

Partitioning support is an extension of the EB tresos AutoCore Rte. If you haven't purchased a license for this extension, all configuration parameters on this tab will be disabled and you cannot configure the partitioning support.

In order to enable the partitioning support, you have to select either **Ioc**, **SharedMemory** or **Mixed** as the inter-partition communication approach from the drop-down list **Partitioning support**. Then the configuration options for the partitioning support are enabled and accessible.

By selecting **Ioc** for **Partitioning support** the Rte will use the Ioc module for inter-partition communication.

By selecting **SharedMemory** for **Partitioning support** the `Rte` will use `Smc` for inter-partition communication.

By selecting **Mixed** for **Partitioning support** the `Rte` will use `Smc` for inter-partition-intra-core communication and `Ioc` for inter-partition-inter-core communication.

NOTE	Partitions are represented by `Os` applications, which you must first configure in the `Os` Editor before you can map software components and the BSW to these `Os` applications.

4.3.4.3.1. Configuring empty Rte_Start/Rte_Stop stubs

When the checkbox **Generate empty Rte_Start/Rte_Stop stubs** is enabled, the `Rte` will generate empty `Rte_Start` and `Rte_Stop` stubs in `Rte_Main.c`.

This is required in case that `Rte_Start`/`Rte_Stop` is called from a non-trusted partition. They initialize/de-initialize all other Rte partitions by calling `Rte_Start`/ `Rte_Stop` on the other partitions. However, if at least one partition is trusted and `Rte_Start`/`Rte_Stop` needs access to a memory location, an exception is thrown by the `Os`. This can be prevented by generating empty `Rte_Start` and `Rte_Stop` stubs.

It is then up to the user to call `Rte_Start` and `Rte_Stop` for each partition. It might be possible to define a task for each trusted partition that calls these API functions. These tasks must then be activated on startup.

4.3.4.3.2. Configuring single schedule table for all partitions

Generally, if multiple partitions are used and each partition contains periodically triggered runnables, an `Os` schedule table is allocated by the `Rte` for each partition.

If `Rte_Start` is called, the corresponding schedule table for this partition is started. However, the schedule tables are not synchronized across the partitions so that expiry points with the same offset are not guaranteed to be executed at the same time.

To simultaneously activate runnables of different partitions with the same corresponding offset in the schedule table, the user can enable the checkbox **Generate single Rte schedule table for all partitions**. In this case you have to configure an Os application in **Single schedule table Os Application**, which starts the global schedule table if `Rte_Start()` has been called.

4.3.4.3.3. Configuring the BSW-related options

You have to map the BSW to a partition. Therefore select the `Os` application from the **BSW Os application** drop-down list in which the BSW is running.

If software components that use inter-ECU communication are mapped to a different partition than the BSW partition, the Rte requires a BSW task. This BSW task is used to call the Com API to for inter-ECU/inter-partition communication.

NOTE

The Rte uses the BSW task exclusively for inter-ECU/inter-partition communication. Do not map any runnable entities to this task.

▶ First change to the Os Editor and add a new task to the Os configuration.

▶ Then enable the checkbox **BSW Os task required** and select the task from the **BSW Os task** drop-down list.

▶ Finally define queue lengths for the send signal and send signal groups by entering a valid number in the **BSW send signal queue length** and **BSW send signal group queue length**. These queues are used by Rte to buffer the inter-ECU communication requests from software components mapped to a partition other than the BSW partition.

TIP

The value for the queue lengths depend on the priority of the BSW task and of the frequency with which software component instances mapped on partitions other than the BSW partition perform inter-ECU communication. To reduce the required queue lengths, it is recommended to assign a high priority to the BSW task.

However, note that a high priority for the BSW task also leads to many context switches between the sender task and the BSW task, because every inter-partition Rte_Write() call will set an event for the BSW task and thus set it to the running state. To minimize these context switches, the BSW task can be configured with a certain period by enabling the checkbox **Trigger BSW Os task periodically** and configuring a period in the field **BSW Os Task period in seconds**. In this case, the Rte will not set an event for each inter-partition Rte_Write() call but will trigger it periodically. Ensure that the queue length is sufficient so that no data loss occurs.

4.3.4.3.3.1. Multiple Com instances on different partitions

The Rte generator supports multiple Com instances that are mapped to different partitions. For that, the Rte generator evaluates the vendor specific ComMainFunctionRef reference from the ComIPdu container and determines the partition of that main function from the mapping of the related BSW module to the OsApplication. For each Com instance, a dedicated BswImplementation and InternalBehavior must exist in the Com BSWMD file. Each BswImplementation must be referenced in the Rte configuration, and the timing events for the BswSchedulableEntitys of the main functions must be mapped to tasks via the RteBswEventToTaskMapping configuration. See Section 4.2.5.3, "Rte Editor configuration" for details.

In contrast to the **BSW Os application** and the **BSW Os task** that are used if only one instance of the `Com` is present, multiple partitions and tasks might be required for a distributed `Com`.

▶ Using `Com` tasks:

A configuration container `RteGeneration/ComTaskConfiguration` must exist for each Com partition where an inter-partition signal/signal group is sent. In that container, a `ComTaskOsTaskRef` must be specified, which is a reference to an Os task that is used to call the `Com` API in the context of the corresponding Com partition.

Similar to the **BSW Os task**, a signal/signal group length can be specified by using the `ComTaskSendSignalQueueLength/ComTaskSendSignalGroupQueueLength` parameter of that container. No such configuration is required for the receive signals/signal groups, because they are handled via callbacks.

▶ Using `Com` transmission preparation callbacks:

By setting the configuration parameter `RteGeneration/SendSignalQueueStrategy` to "OnePerMainFunction", this will enable the ComPreparationNotification feature. The Rte will generate a transmission preparation callback (Rte_COMCbkTxPrep_mn) for each Com_MainFunctionTx that is referenced by a `ComIPdu` in the vendor specific `ComMainFunctionRef` reference and which handles inter-partition send signals/signal groups. By enabling this feature, individual `Com` tasks are no longer needed since the transmission preparation callbacks are called by their corresponding Com_MainFunctionTx prior to sending the signals/signal groups.

Make sure that each Com_MainFunctionTx for which the Rte generates a transmission preparation callback has the configuration parameter `ComPreparationNotification` set to `Rte_COMCbkTxPrep_<mn>` where <mn> is the name of the particular Com_MainFunctionTx.

NOTE Even with multiple `Com` instances, a BSW Os task might be required e.g. to redirect `Det` errors to the partition of the `Det` module. However, if the BSW and Com task points to the same Os task the values from the `ComTaskConfiguration` container will be used.

4.3.4.3.4. Software component partition mapping

All available software component instances are listed in the table **Software component instance partitioning**. The first column **Software component instance** holds the name of the corresponding software component instance.

To map a software component instance to a partition, select an `Os` application in the second column named `Os application`.

NOTE	Multiple software component instances can be mapped to the same partition, but all software component instances of the same software component type must be mapped to the same partition.

4.3.4.3.5. Os counter partition assignment

For each partition, you have to specify which `Os` counter shall be used for the partition.

First,

► change to the `Os` Editor,

► add an `Os` counter for each `Os` application on which software components are mapped,

► and assign the `Os` counter to the `Os` application.

Then configure the `Rte` so that it uses this specific `Os` counter for each `Os` application. In the table **Os counter partition assignment** all available `Os` applications are listed where the first column holds the name of the corresponding `Os` application.

The second column **Os counter** holds a drop-down list where you have to choose the `Os` counter that shall be used for the related `Os` application.

In the last column **Nanoseconds per tick** you have to state the nanoseconds per tick of the selected `Os` counter for the corresponding `Os` application.

NOTE	When partitioning support is enabled, the Rte Generator ignores the parameters **Os counter nanoseconds per tick Os counter** on the **General** tab.

4.3.4.4. Configuring optimization options

The following sections explain the options on the **Optimizations** tab.

4.3.4.4.1. Configuring the function elision option

Select the **Function elision** check box to implement the following as API macros rather than as functions:

▶ Rte_Read(),

▶ Rte_DRead(),

▶ Rte_Write(),

▶ Rte_IsUpdated(),

▶ Rte_IrvRead(),

▶ Rte_IrvWrite(),

▶ Rte_Enter(),

▶ Rte_Exit(),

▶ Rte_Mode(),

▶ Rte_CData(),

▶ Rte_Prm() , and

▶ Rte_Call()

If you check **Function elision**, the API, which is associated with an unconnected port, is also implemented as a macro. For conditions in which you can apply function elision, see Section 4.2.17.4, "Function Elision".

4.3.4.4.2. Configuring the default data consistency mechanism

The `default data consistency mechanism` parameter defines the default data consistency mechanism for protecting accesses to receive buffers/queues and inter runnable variables.

To configure this parameter in the **Default data consistency mechanism** menu line, select one of the following options:

▶ `OsResource` (default):

Os resources are used to protect the access.

▶ `Interrupt Blocking`:

Interrupts will be blocked while the buffer/queue/variable is accessed.

For more details, see Section 4.2.17.2, "Data consistency mechanisms".

4.3.4.4.3. Configuring the interrupt blocking function

The **Interrupt blocking function** specifies the function or macro that is used to block interrupts.

Select one of the following options:

▶ `SuspendResumeAllInterrupts()` (default):

Uses the AUTOSAR `Os` functions `SuspendAllInterrupts()` and `ResumeAllInterrupts()`.

▶ `DisableEnableAllInterrupts()`:

Uses the AUTOSAR `Os` functions `DisableAllInterrupts()` and `EnableAllInterrupts()`.

▶ `TS_IntDisableEnable()`:

Uses EB-specific functions from the EB `Base` module.

▶ `Rte_UserDefinedIntLockUnlock()`:

Uses user-defined functions/macros, which you have to declare in the `Rte_UserDefinedIntLock.h` header file.

Find out about advantages and drawbacks of the above options in chapter Section 4.2.17.2, "Data consistency mechanisms".

4.3.4.4.4. Configuring the Com Callback not interruptible parameter

Select the **Com callback is not interruptible** check box to define that the `Com` callback functions are not interruptible. If they are not interruptible (e.g. they run in non-interruptible *ISR*s or tasks), the `Rte` does not apply any data consistency mechanism within `Com` callbacks.

If the `Com` callback functions are configured as interruptible, i.e. **Com callback is not interruptible** is unchecked, then the `Rte` may lock interrupts within a `Com` callback.

4.3.4.4.5. Configuring the inter-ECU signal invalidation handled by Rte parameter

Select the **Inter-ECU signal invalidation handled by Rte** check box to define whether the signal invalidation for inter-ECU communication shall be handled by the `Rte` or by the `Com` module.

▶ When the option is enabled, the signal invalidation for inter-ECU communication is handled by the `Rte`. In this case, on sender side, the invalid value is read from the `Com` configuration. The invalid value in the software component description (attached to the data type) will be ignored. On receiver side, the invalid value is read from the `Com` configuration, too. When handle invalid replace is configured for the receiver, the initial value is always read from the software component description. A warning will be reported if the initial value of the `Com` signal does not match the initial value specified in the software component description.

▶ If this option is disabled, then the signal invalidation for inter-ECU communication is handled by the `Com` according to the AUTOSAR `Com` specification [2].

4.3.4.4.6. Configuring the directly read from `Com` buffer option

Select the **Directly read data from Com if possible** check box to define that the generated `Rte` always directly fetches the data from `Com` for data elements, which are mapped to `Com` signals. In this case, the `Rte` does not allocate an additional receive buffer.

You can use this option to reduce the RAM usage of the `Rte`. By default, this option is enabled.

You may disable this option if your application reads the data element more frequently than the mapped signal is updated by `Com`. This increases the runtime performance because the `Rte` calls the `Com_ReceiveSignal` API function less frequently then.

Under certain circumstances, direct read from `Com` is not possible, see Section 4.2.17.3.1, "Receive buffer allocation" for details.

4.3.4.4.7. Disable partition active checks

Select **Disable partition active checks** to avoid that the Rte generates code to check whether the partition is active at the beginning of certain API functions. If these checks are disabled, the runtime and ROM consumption of the generated code can be improved.

WARNING	**After disabling the partition active checks you must ensure that nothing is triggered before `Rte_Start()` or after `Rte_Stop()` is called**
⚠	The partition active check is implemented to fulfill the requirements `rte_sws_2538`, `rte_-sws_2535` and `rte_sws_2536` of the `Rte` specification (see [1]). If this check is not generated, it can happen that buffers are modified or runnables are started even if `Rte_Start()` has not been called before. If `Rte_Stop()` is called, it can still happen that runnables that are in execution can trigger further runnables e.g. by calling `Rte_Call()`, the `Rte` will activate the server runnable anyway even if the partition is already stopped. For the `Com` callbacks, it might be possible that incoming data triggers runnables or modify buffers even if the `Rte` was not started or has been already stopped. The user must ensure that no further runnables are triggered or that such activations doesn't harm anything.

4.3.4.4.8. Configuring the Os spinlock lock method

If the Os supports the `OsSpinlockLockMethod`, select **Os supports OsSpinlockLockMethod** to configure every spinlock in the `Os` with the `OsSpinlockLockMethod` set to LOCK_ALL_INTERRUPTS. Otherwise, the Rte generates interrupt locks for every spinlock.

4.3.4.4.9. Configuring the Os spinlock allocation strategy

The parameter `SpinlockAllocationStrategy` is used to control how many spinlocks are generated to protect the inter-partition channel communication, the options are:

▶ `OnePerChannel (default):`

   To generate a single spinlock for every channel.

▶ `OnePerECU:`

   To generate a single global spinlock that can be used for all channels.

▶ `OnePerCoreGroup:`

   To generate a single spinlock for every unique combination of the communicating cores. The communication direction is ignored.

4.3.4.5. Mapping executable entities to `Os` tasks

Select the **Event Mapping** tab.

Now, you can map each executable entity of atomic software component instances and basic software modules to an `Os` task. Direct-call server runnable entities must not be mapped to tasks, see Section 4.2.17.1.2.1, "No task mapping necessary". See Section 4.2.10.2, "Asynchronous Mode Switch" for the mapping conditions of

direct call asynchronous mode switch. Strictly speaking, each `Rte` or BSW event that triggers an executable entity is mapped to a task separately. Events that trigger an executable entity can be mapped to different tasks if the runnable or schedulable entity can be invoked concurrently.

To map an `InitEvent` to a `RteInitializationRunnableBatch`, follow these steps:

Step 1
Create a container in the generic editor first.

Step 2
Select this container in the **Rte Initialization Runnable Batch** column.

Step 3
Ensure, that this event is also mapped to a task or the **Rte Initialization Runnable Batch** is mapped to a task by using the **Rte generic editor**, additionally to the Rte specification.

The reason for the event task mapping is, that the Rte needs to know the task context in that the `RteInitializationRunnableBatch` is executed. The events without a **Rte Initialization Runnable Batch** are generated in the task body. The other events are called in the generated `Rte_Init_<InitContainer>()` function.

The **Unmapped RTE and BSW events** table in section **Unmapped RTE and BSW events** of the **Event Mapping** tab contains the executable entities that are not mapped to a task yet. Executable entities, which are triggered by multiple `Rte` or BSW events, will be displayed several times in the table. Map each of these events separately.

To display only executable entities for which a task mapping is required in the **Unmapped RTE and BSW events** table, check the **Display RTE and BSW events with required mapping only** check box.

The **Task** drop-down list in section **Mapped RTE and BSW events** provides a list of all tasks that are available in the Os configuration.

Select a task from the **Task** drop-down list.

The **Mapped RTE and BSW events** table displays the executable entities that are mapped to this task.

The table columns provide some additional information about the events and runnable entities:

Position	The position within the task or the Rte initialization runnable batch; mapped runnable entities table only.
Event	The short name of the Rte or BSW event that triggers the executable entity.
Executable entity	The path of the software component instance or basic software module to which the executable entity belongs.
Category	The category of the executable entity.
Required	Shows if the executable entity must to be mapped to a task. Some runnable entities such as direct-call server runnable entities or runnable entities which are not triggered by an Rte event, do not need to be mapped to a task. Nevertheless, they *may* be mapped to a task.
Period	If the executable entity is time-triggered, this column shows the period of the timing event which triggers the executable entity.
Offset	If the executable entity is time-triggered, the activation offset of the executable entity can be configured with this parameter.
RteInitializationRunnableBatch	The range of containers which should be defined in Generic Editor.
Mode machine instance description	If the event is a mode switch event this column shows a description of the mode machine instance. For SWC mode switch events it shows the name of the corresponding mode declaration group prototype together with the names of the software component prototype and that of the provided port. For BSW mode switch events it shows the qualified path of the mode declaration group reference.

WARNING	**The category of a runnable entity may not be displayed accurately in the Rte Editor**
⚠	The Rte Editor only checks certain elements of the system description (e.g. the existence of wait points, server call points, timeouts etc.) to determine the category of a runnable entity.

However, the Rte configuration is not considered at this time, because it can only be assumed to be in a complete state at Rte code generation. Therefore, the category displayed in the Rte Editor may be incorrect.

For example, if a runnable entity uses synchronous inter-partition client/server communication, the Rte Editor does not display the category correctly. In this case, the Rte Editor shows the runnable entity to be of category 1 although it is really of category 2.

Note that this does not cause incorrect Rte generation. The Rte will process the Rte configuration during generation and will correctly categorize the runnable entity.

To sort the event mapping tables by columns:

▶ Click on the table header.

The table is sorted alphabetically along the respective column's content.

To configure the visible columns of the tables:

▶

Click on the **Edit visibility of columns** button (🗔) .

The visibility of the table columns can be switched on and off individually.

To map an executable entity, perform the following four steps:

1. Before configuring the task mapping in the **Rte Editor**, add the required tasks in the `Os` configuration. First, open the editor for the `Os` configuration, then change to the **OsTask** tab and add a new container for the task to the list.

2. Configure the task parameters (priority etc.) in the newly added task container in the `Os` configuration editor.

3. In the **Rte Editor**, select a task from the **Event Mapping** tab to which you want to map executable entities. Select the **Task** drop-down list box and select one of the tasks listed.
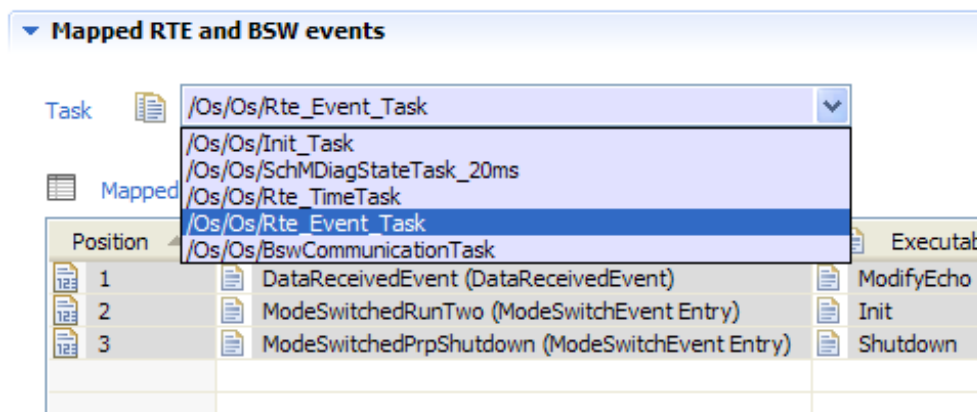
4. Select an executable entity and click the **Map the executable entity selected above to the task selected below** button ( ).

The executable entity is now removed from the **Unmapped RTE and BSW events** table and appears in the **Mapped RTE and BSW events** table.

TIP	Selecting multiple executable entities at once
	To select more than one unmapped event at once, press and hold **Shift** key for consecutive entries or **Ctrl** key for non-consecutive entries, then select the executable entities. You can then map them all at once.

To remove an executable entity from a task:

► Select the executable entity in the **Mapped RTE and BSW events** table.

► Select the **Unmap the executable entity selected below** button ( ). The executable entity now appears in the **Unmapped RTE and BSW events** table again.

To remove all executable entity mappings from the selected task:

► Select the **Unmap all runnable entities that are mapped to the task selected below** button ( ).

All executable entities now appear in the **Unmapped RTE and BSW events** table.

To change the position of a mapped executable entity within the task:

► Select the executable entity and use the **up** and **down** arrow buttons ( ).

TIP	**Moving multiple mapped events at once**
	To move multiple mapped events at once, select more than one event and push the arrow buttons.
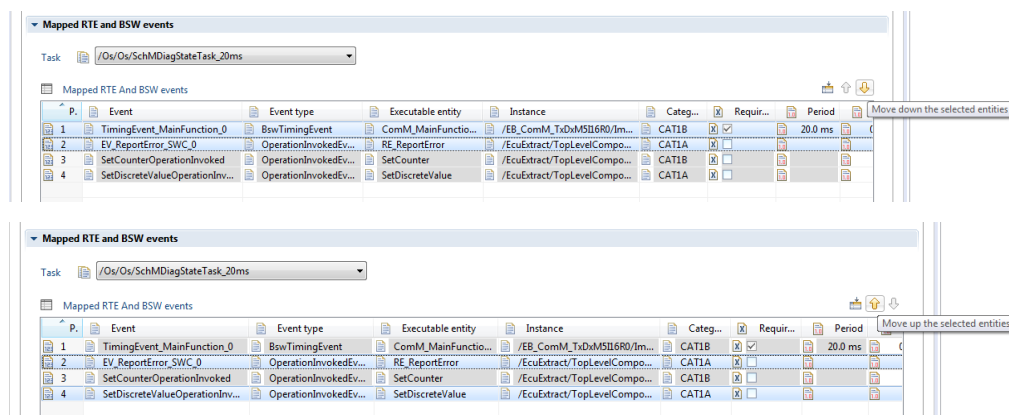
▶ Another way to change the position of an event is to edit the Position cell. The position entered has to be a number bigger than zero and less than the total number of mapped events.
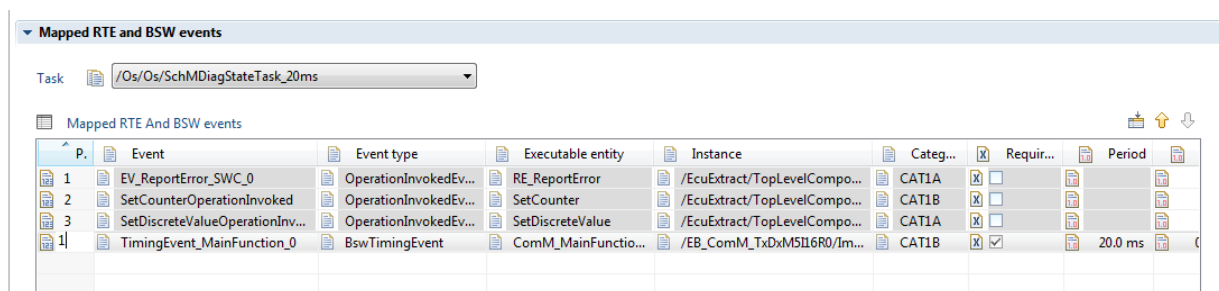
The events will rearrange as follows:
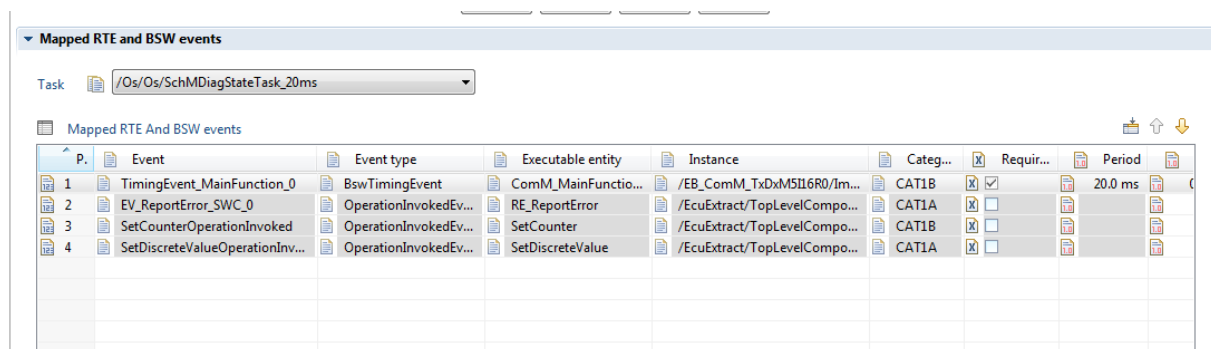
To automatically map time-triggered BSW main functions to default Os tasks:

▶ Select the **Auto-map all BSW main functions that are triggered by BSW timing events** button
( ).

The selection of the **Unmapped RTE and BSW events** and **Mapped RTE and BSW events** tables is not relevant for the action and can be ignored. The automatic mapping directly creates Os tasks with a default stack size and task priority and then maps the BSW main functions to these tasks. A result dialog shows the configuration changes at the end of the process.

NOTE	**Why are not all BSW main functions automatically configured?**
	The automatic mapping functionality only processes BSW main functions that are triggered by BSW timing events. This information requires a valid basic software module description from each module. Additionally, each basic software module must provide information about the default Os task to which its main function shall be mapped. This default task mapping is specified by the module vendor and cannot be changed by the user.

TIP	**Changing the mapping of individual BSW main functions**
	You can manually map BSW main functions to Os tasks other than the default task. Running the automatic mapping feature afterwards will not change the mapping of BSW main functions that are already mapped.

To configure the activation offset of an executable entity:

▶ Select the executable entity in either of the tables **Unmapped RTE and BSW events** or **Mapped RTE and BSW events**.

▶ Click in the **Offset** column and enter the offset in milliseconds.

For each executable entity that requires a mapping but none exists yet, the **Rte Editor** will report a warning. To directly navigate to the executable entities for which a task mapping is missing:

▶ Locate the **warnings detected** at the head of the **Rte Editor** view.

▶ Click on the **warnings detected** link.

▶ Select the warning of the list.
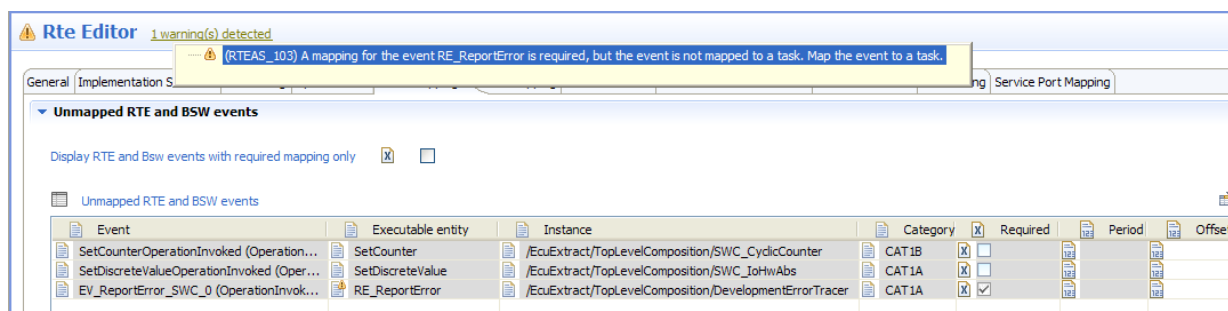
You will be directed to the executable entity for which a task mapping is missing.

4.3.4.6. Mapping executable entities to `Os` ISRs

The Rte generator supports the construction of ISR bodies. The necessary input information, e.g. the mapping of `RunnableEntities` and `BswSchedulableEntities` to ISRs must be provided by the ECU configuration description and can only be configured in the generic editor.

The Rte supports the activation of executables in an interrupt context under the following circumstances:

▶ An `ExternalTriggerOccurredEvent`/`BswExternalTriggerOccurredEvent` is mapped to an ISR (via a `RteEventToIsrMapping`).

▶ The referenced require port/required trigger is unconnected.

OR

▶ A `TimingEvent`/`BswTimingEvent` is mapped to an ISR (via a `RteEventToIsrMapping`).

▶ The timing event period equals or is a multiple of the `OsIsrPeriod`.

NOTE **Activation of the ISR is not in the scope of the Rte**

ⓘ Note that the activation of the ISR is not in the scope of the Rte, therefore the related require port/required trigger must be unconnected and no other trigger source may exist.

The Rte Generator rejects configurations if a timing event with an activation offset greater than zero is mapped to a category 2 ISR. The ISRs activation cannot be managed by the RTE generator. You must either set the offset value to zero or map the timing event to a task. For task mapping the RTE generates the schedule table entries with the right offset and period.

The Rte Generator rejects configurations where an event that activates a category 2 `ExecutableEntity` is mapped to an ISR or when both timing and external trigger events are mapped to an ISR. `ExecutableEntities` of category 2 and mapping both timing and external trigger events to an ISR require a wait point (waiting for events) but an ISR is not allowed to enter a wait state.

4.3.4.7. Data mapping

For inter-ECU communication, you must map data element prototypes to `Com` signals/signal groups or system signals/signal groups.

TIP   **No data mapping is required for intra-ECU communication**

For intra-ECU communication *no data mapping* is required. Here the assembly connectors in the software component description files connect port prototypes of software component instances on the same ECU.

4.3.4.7.1. Background information

Mapping data elements to system signals / system signal groups is the AUTOSAR compliant-approach.

WARNING   **You cannot use the Rte Editor any more to map system signals or system signal groups**

The functionality to map system signals or system signal groups to data elements is now available in the **System Signal Mapping Editor**. The mapping of `Com` signals or signal groups is not affected and can still be configured in the **Rte Editor**.

For system signals/signal group mappings it is recommended to either import them directly e.g. via a `Sender-ReceiverToSignalMapping` or by using the **System Signal Mapping Editor**. If these mappings are available, the generator resolves the system signals/system signal groups into `Com` signals/`Com` signal groups by evaluating the `SystemTemplateSystemSignalRef` and `SystemTemplateSignalGroupRef` parameters of the `Com` configuration.

A system signal may be referenced by multiple `Com` signals if there are multiple instances of the system signal in the system description. In this case, you only define one mapping to the system signal. The Rte Generator will then generate mappings for all `Com` signals that refer to the instances of the system signal.

In contrast to `Com` signal / signal group mappings, you may define system signal / signal group mappings also for ports of compositions. The Rte Generator will then automatically generate mappings for all data elements of ports of atomic software components that are connected to the port of the composition via delegation connectors.

Directly mapping data elements to `Com` signals/signal groups is an EB tresos Studio-specific enhancement. Mappings to `Com` signals / signal groups are read from vendor-specific parameters of the ECU configuration. When the **Rte Editor** is closed, the mappings are written to the ECU configuration.

You may directly map data elements to `Com` signals / signal groups

►   because you do not have a complete system description in your project

---

► because not all signals (e.g. debug signals) are defined in your system description

► due to backward compatibility with older EB tresos releases that did not support mappings to system signals.

TIP　**You may mix data mappings to Com signals and system signals**

It is possible to mix data mappings to `Com` signals and system signals in the same project, even for the same data element. To be fully AUTOSAR-compliant, it is recommended to use system signal mappings.

► Map a data element prototype, which is a primitive type, to a single `Com` signal.

► Map a complex data element prototype (`ImplementationDataType` of category `ARRAY` or `STRUCTURE` or an `ApplicationArrayDataType`/`ApplicationRecordDataType`) to a `Com` signal group.

► Map each primitive element of the complex type to a separate `Com` signal that belongs to the signal group.

4.3.4.7.2. Mapping sender/receiver data elements to signals

Select the **Data Mapping** tab.
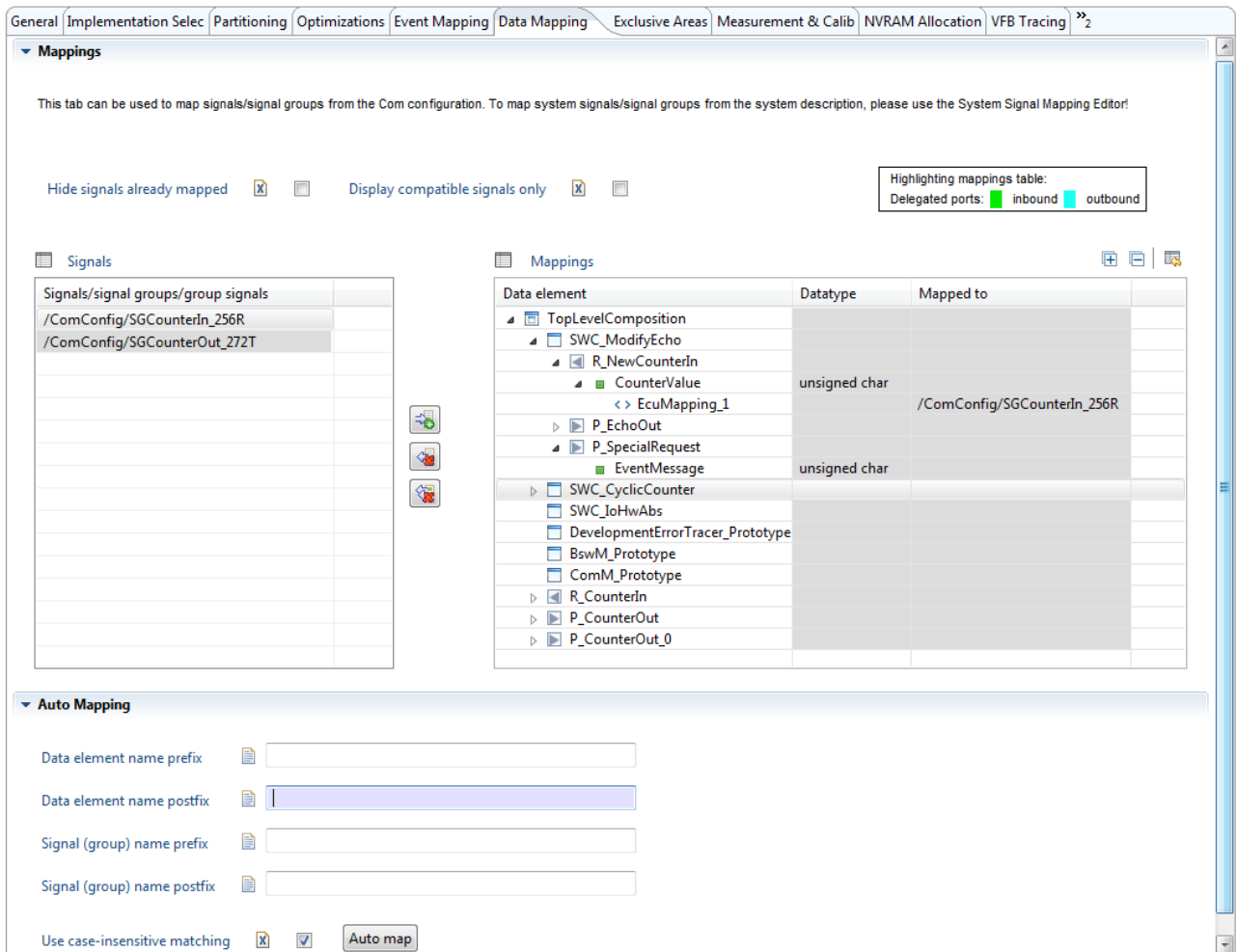
The **Data Mapping** tab section **Mappings** consists of two parts: the **Signals** list and the **Mappings** tree table.

The **Signals** table displays all available signals and signal groups of the `Com` configuration. To hide signals/signal groups that are already mapped to a data element, check the check box **Hide signals already mapped.**

The **Mappings** tree view displays all *compositions* and *atomic software component*s with their *ports*, *data elements*s and *data mapping*s. Connected data element prototypes are either internally connected via assembly connectors or externally via data mappings; in the figure below they are represented by green bullets, whereas unconnected data element prototypes are shown as grey bullets.

If a port of a composition is connected to one or more ports of atomic software components via delegation connectors, the **Rte Editor** displays this by highlighting the ports of the connected atomic software components when you select the port of the composition (see figure below).

The other way round, when you select a port of an atomic software component, which is delegated to a port of a composition, the **Rte Editor** highlights the port of the connected composition (see figure below).

To expand or collapse an element of the **Mappings** tree:

►   Select a tree item.

►

To expand the tree view, press the **Expand the selected elements** button ( ).

►

To collapse the tree view, press the **Collapse the selected elements** button ( ).

To add a data mapping:

►   Select a data element prototype from the **Mappings** tree view.

►   Select a signal from the **Signals** list.

►

Click on the **Create mapping** button ( ).

A data element prototype can be mapped to multiple signals or signal groups.

To remove an existing mapping:

►   Select the mapping.

►

Select the **Delete data mapping** button ( ).

To remove all existing mappings:

▶

   Select the **Delete all data mappings** button ( ).

If you select a data element prototype and you have checked the option **Display compatible signals only**, the **Rte Editor** displays only those signals in the **Signals** list which have a compatible data type, i.e. for which you can actually add a mapping.

To deselect all elements of the **Mappings** table:

▶

   Select the **Remove row selection** button ( ).

The **Rte Editor** also provides an auto-mapping mechanism based on signal and data element names in the section **Auto Mapping**.

NOTE	**Automatic mapping for signal groups is not supported.**
ⓘ	Unlike automatically signal mapping, automatic mapping of signal groups is not supported. Thus you have to map signal groups manually.

To use the auto-mapping function:

▶   Configure the data element/signal name pre- and postfixes.

▶   Click on the **Auto map** button.

The auto-mapping algorithm first removes the configured pre- and post fixes from the names of the data elements and signals. Then it tries to find a matching signal with compatible data type for each data element.

The auto-mapping algorithm ignores a data element if it is already connected (neither internally nor externally).

If you enable the option **Use case-insensitive matching**, the auto-mapping algorithm ignores the case of the data element names, signal (group) names, pre- and postfixes.

In case that a port of an atomic software component is delegated to a port of a composition, the auto-map algorithm adds the mapping to the port of the composition if a matching signal could be found.

You may also use regular expressions for the pre- and postfixes. Example: you have the following signals:

▶ `SIG22_de1_abc`

▶ `SIG44_de2_efg`

▶ `SIG55_de3_xyz`

and want to map it to the data elements `de1`, `de2` and `de3`. Then you can use the following pre- and postfixes:

▶ Signal Name Prefix: `SIG[0-9]+_`

▶ Signal Name Postfix: `_[a-z]*`

4.3.4.8. Configuring exclusive areas

Select the **Exclusive Areas** tab.

This screen displays a list of all available *exclusive area*s.

Specify the implementation mechanism for both exclusive areas belonging to software components and exclusive areas belonging to basic software modules. You have the following options:

▶ **All Interrupt Blocking**

▶ **Os Resource**

▶ **Os Interrupt Blocking**

▶ **Cooperative Runnable Placement**

▶ **EB Fast Lock (only for basic software module exclusive areas)**

▶ **Disable Exclusive Area (only for basic software module exclusive areas)**

▶ **Spinlock (only for basic software module exclusive areas on multi-core architectures)**

▶ **User Callout**

Exclusive area	Implementation mechanism	User callout enter	User callout exit
/components/AtomicComponentA/InternalBehaviorA/ea1	User Callout	Ea1_UserCalloutEnter	Ea1_UserCalloutExit
/components/AtomicComponentA/InternalBehaviorA/ea2	User Callout	Ea2_UserCalloutEnter	Ea2_UserCalloutExit
/components/AtomicComponentA/InternalBehaviorA/ea3	User Callout	Ea2_UserCalloutEnter	Ea3_UserCalloutExit
/components/AtomicComponentA/InternalBehaviorA/ea4	Os Resource		
/components/AtomicComponentA/InternalBehaviorA/ea5	All Interrupt Blocking		

Tabs: General | Implementation Selec | Partitioning | Optimizations | Event Mapping | Data Mapping | Exclusive Areas | Measurement & Calib | NVRAM Allocation | »4

Exclusive areas

For more information about exclusive areas and the configuration options, see Section 4.2.7, "Exclusive areas" and Section 4.2.17.2.5, "Data consistency for exclusive areas".

4.3.4.9. Configuring the measurement and calibration support

Select the **Measurement & Calibration** tab.
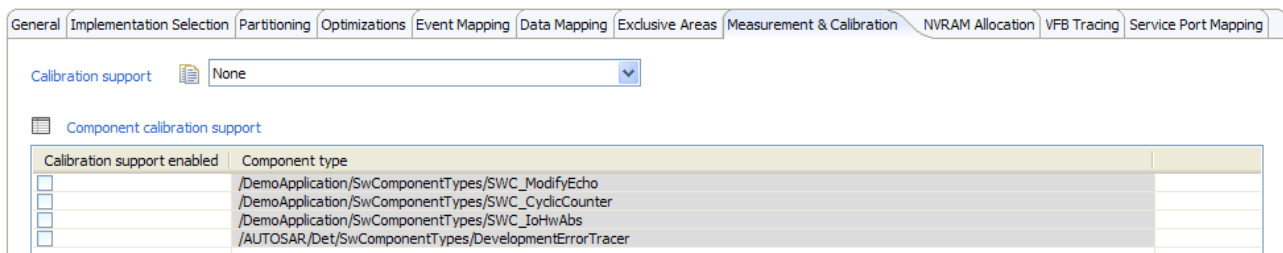
To globally enable calibration, select a calibration method from **Calibration support**.

You will find details about the supported calibration methods in Section 4.2.12, "Calibration".

You can globally disable calibration support by selecting **None**. Then the Rte Generator will allocate all calibration parameters in ROM or NVRAM, i.e. the `Rte_CData()` and `Rte_Prm()` API will always access variables in ROM or NVRAM. You cannot perform an online calibration during runtime.

If you have enabled calibration support, you also have to explicitly enable the calibration support for each component of your application individually by checking the corresponding box in the **Component calibration support** table.

If calibration support is disabled for a component, the Rte Generator will allocate all calibration parameters of that component in ROM or NVRAM, you cannot change them during runtime.

4.3.4.10. Configuring the NVRAM allocation

Select the **NVRAM Allocation** tab.

The **NVRAM Allocation** table displays all `SwcServiceDependency` which are defined in the `ServiceNeeds` section of your software component description.

4.3.4.10.1. Mapping per instance memory to an `NvM` block

To map per instance memory ('C' typed or AUTOSAR typed) to a `NvM` block:

► make sure that there is a `SwcServiceDependency` in your software component description which references the per-instance memory.

▶ Add an `NvMBlockDescriptor` to the `NvM` configuration on which you want to map the per instance memory.

▶ In the **NVMRAM Allocation** table of the **Rte Editor**, select the `NvM` block from the **NvM block reference** drop-down list for the corresponding `SwcServiceDependency`.

▶ Configure a **RAM block location symbol** for the per instance memory.

The Rte Generator then allocates a variable for the per instance memory and uses the name specified in **RAM block location symbol** for the name of the variable.

When you close the **Rte Editor**, the **Rte Editor** writes the address of the configured RAM block location symbol to the `NvmRamBlockDataAddress` parameter of the referenced `NvMBlockDescriptor`. The `NvM` then uses the per instance memory variable provided by the `Rte` as permanent RAM mirror for the NVRAM block.

| General | Implementation Selec | Partitioning | Optimizations | Event Mapping | Data Mapping | Exclusive Areas | Measurement & Calib | NVRAM Allocation | VFB Tracing | Bsw Trigger Connecti | Bsw Mode Mapping | »₁ |

NVRAM allocation

Software Component Instance N...	SWC Service Dependency	NvM Block Reference	ROM Block Location Symbol	RAM Block Location Sy...
AtomicComponentA_I1	/components/AtomicComponentA/InternalBehaviorA/SwcNvBlockNeed0	/test/NvN	NVROM_CalprmUInt32_Inst1	NVRAM_PIMUInt32_Inst1
AtomicComponentA_I1	/components/AtomicComponentA/InternalBehaviorA/SwcNvBlockNeed1	/test/NvN	NVROM_CalprmSimpleRecord_Inst1	NVRAM_PIMSimpleRec...
AtomicComponentA_I1	/components/AtomicComponentA/InternalBehaviorA/SwcNvBlockNeed2	/test/NvN	NVROM_CalprmSimpleArray_Inst1	NVRAM_PIMSimpleArra...
AtomicComponentA_I1	/components/AtomicComponentA/InternalBehaviorA/SwcNvBlockNeed3	/test/NvN	NVROM_CalprmUInt8_Inst1	NVRAM_PIMUInt8_Inst1
AtomicComponentA_I2	/components/AtomicComponentA/InternalBehaviorA/SwcNvBlockNeed0	/test/NvN	NVROM_CalprmUInt32_Inst2	NVRAM_PIMUInt32_Inst2
AtomicComponentA_I2	/components/AtomicComponentA/InternalBehaviorA/SwcNvBlockNeed1	/test/NvN	NVROM_CalprmSimpleRecord_Inst2	NVRAM_PIMSimpleRec...
AtomicComponentA_I2	/components/AtomicComponentA/InternalBehaviorA/SwcNvBlockNeed2	/test/NvN	NVROM_CalprmSimplaArray_Inst2	NVRAM_PIMSimpleArra...

4.3.4.10.2. Mapping per instance calibration parameter to an `NvM` block

Mapping a per instance calibration parameter to an `NvM` block is similar to mapping a per instance memory to a `NvM` block:

▶ make sure that there is a `SwcServiceDependency` in your software component description which references the per instance calibration parameter and a corresponding per instance memory.

▶ Add a `NvMBlockDescriptor` to the `NvM` configuration on which you want to map the per instance calibration parameter.

▶ In the **NVRAM Allocation** table of the **Rte Editor**, select the `NvM` block from the **NvM block reference** drop-down list for the corresponding `SwcServiceDependency`.

▶ Configure a **ROM block location symbol** for the per instance calibration parameter and a **RAM block location symbol** for the per instance memory.

The Rte Generator then allocates a constant for the calibration parameter, using the name specified in **ROM block location symbol** for the name of the variable. Furthermore, the Rte Generator allocates a variable for the per instance memory, using the name specified in **RAM block location symbol** for the name of the variable.

When you close the **Rte Editor**, the **Rte Editor** writes the address of the configured ROM block location symbol to the `NvmRomBlockDataAddress` and the configured RAM block location symbol to the `NvmRamBlockDataAddress` parameter of the referenced `NvMBlockDescriptor`. The `NvM` then uses the per instance

memory variable provided by the Rte as permanent RAM mirror for the per instance calibration parameter, which is mapped to the NVRAM block. Moreover, the `NvM` uses the calibration parameter constant as default value for the NVRAM block.

NOTE	**Calibration parameters can only be mapped to NVRAM for offline calibration**
ⓘ	Mapping calibration parameters to NVRAM is only possible for offline calibration i.e. if the online calibration support is set to **None** (see Section 4.3.4.9, "Configuring the measurement and calibration support").
	If the online calibration support is set to a value different from **None**, the Rte Generator will ignore the mappings of calibration parameters to NVRAM. Moreover each calibration parameter will then have the value that was initially configured.

4.3.4.11. Configuring the VFB tracing

Select the **VFB Tracing** tab.

The `Rte` allows to enable or disable each VFB trace hook function individually.

To globally enable the VFB tracing for each VFB trace hook function:

► Select the **Enable VFB tracing** check box.

If you do not want to enable VFB tracing for any of the functions listed, simply unselect the respective check box beside the function.
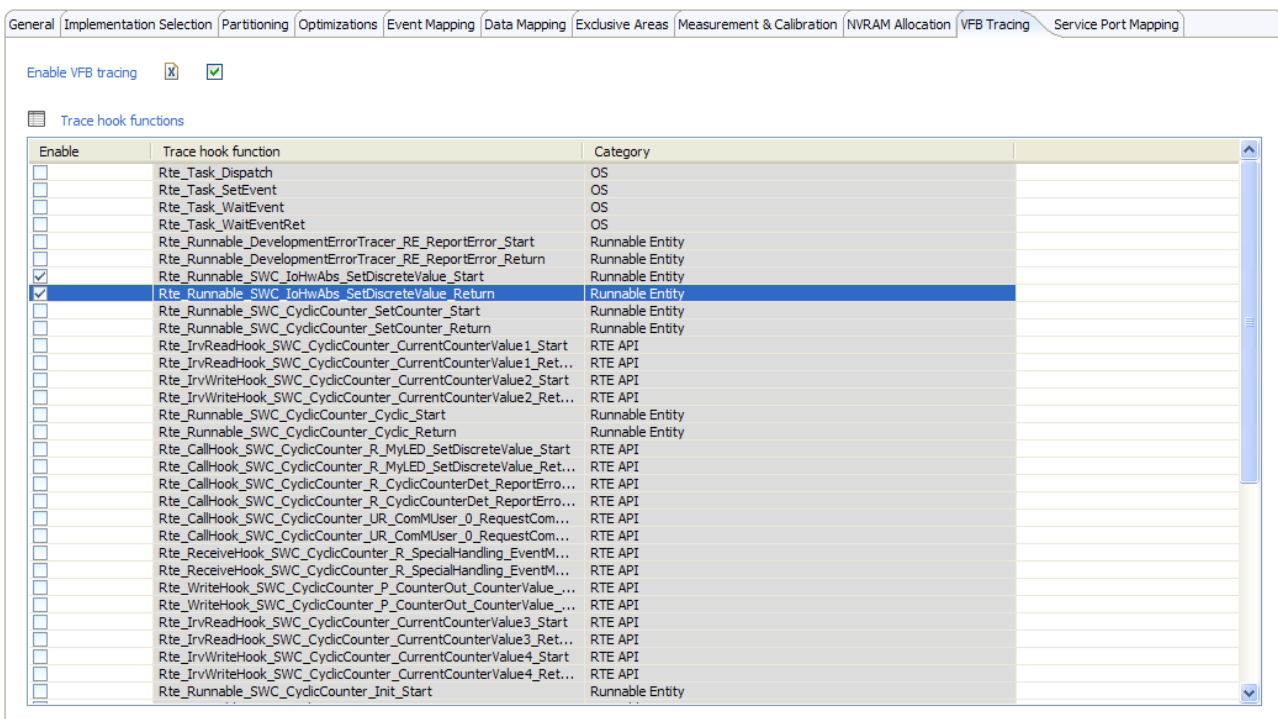
4.3.4.12. BSW Trigger Connections

Select the **BSW Trigger Connections** tab.

The left table **Released Triggers** shows the declared triggers which are used to trigger events across BSW modules.

The right table **Required Triggers** shows the required triggers which are used to react on triggers.

The released trigger from the left table can be mapped to the required triggers on the right table. It is possible to connect one released trigger to multiple required triggers (1:n connection).

4.3.4.13. BSW Mode Mapping

Select the **BSW Mode Mapping** tab.

The left table **Provided Mode Groups** shows the declared mode declaration groups which are used to set mode disabling dependencies or to trigger mode switch events across BSW modules.

The right table **Required Mode Groups** shows the required mode groups which are used to react on mode switches.

The provided mode group from the left table can be mapped to the required mode group on the right table. It is possible to connect one provided mode group to multiple required mode groups (1:n connection).

NOTE	**Connecting incompatible mode groups causes an error message**
(i)	You can only connect mode groups which are compatible. Moreover, you can only connect a provided mode group to a required mode group. If you try to connect incompatible mode groups, the Rte Editor reports an error.

4.3.4.14. BSW Required Sender Receiver Connections

Select the **BSW Required SR Connections** tab.

The left table **Provided variable data prototypes** shows the variable data prototypes provided by BSW modules as declared by the BSW module descriptions.

The right table **Required variable data prototypes** shows the variable data prototypes required by the BSW modules as declared by the BSW module descriptions.

The provided variable data prototype from the left table can be mapped to the required variable data prototype on the right table. It is possible to connect one provided variable data prototype to multiple required variable data prototypes (1:n connection).

NOTE	**Connecting incompatible variable data prototype causes an error message**
ⓘ	You can only connect variable data prototypes which are compatible. Moreover, you can only connect a provided variable data prototype to a required variable data prototype. If you try to connect incompatible variable data prototypes, the Rte Editor reports an error.

4.3.4.15. Rte triggers

Select the **Rte Trigger Config** tab.

The table shows the external triggers which are referenced by an external triggering point and the internal triggers defined in a runnable. You can configure the trigger queue lengths in the **Queue Length** column.

NOTE	**Entering negative or non-integer values results in an error**
(i)	You can only enter positive integer values or zero. If you try to enter negative or non-integer values, the Rte Editor reports an error.

4.3.4.16. Rte initialization runnable batch

Select the **Rte Initialization Runnable Batch** tab.

The table shows the already added initialization runnable batches.

The task reference for the **Init Event** can either be defined by the **RteInitializationRunnableBatch** or in the **event to task mapping** editor.

The task reference for the **Mode Switch Event** can only be defined by the **RteInitializationRunnableBatch**.

4.3.5. Generating the Rte

The `Rte` supports several generation modes: *contract phase*, *Rte only*, `BSW Scheduler only` and `Full`. In the contract phase, only the application header files are generated. In the `Rte only`, `BSW Scheduler only` and `Full` modes, the generation of the executable RTE source code is carried out.

4.3.5.1. Generating the application header files only

To generate the application header files:
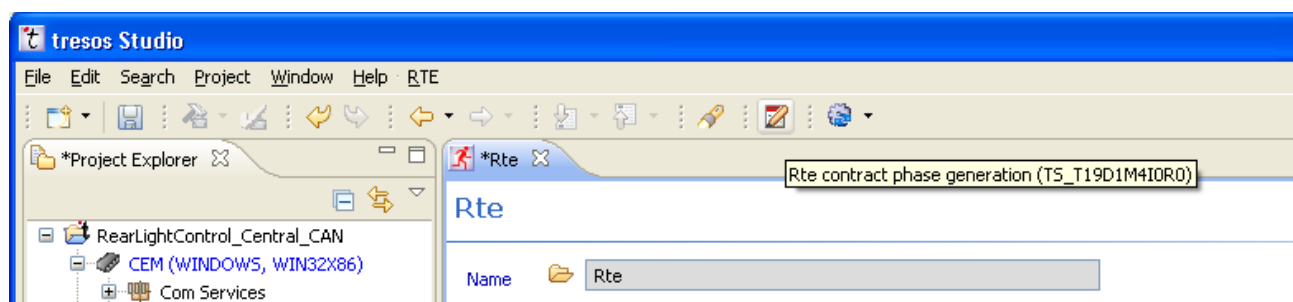
► Select the button **Rte Contract Phase generation** in the tool bar.

   Alternatively, select **Contract Phase** from the **Rte** menu.

   A pop-up window opens up.
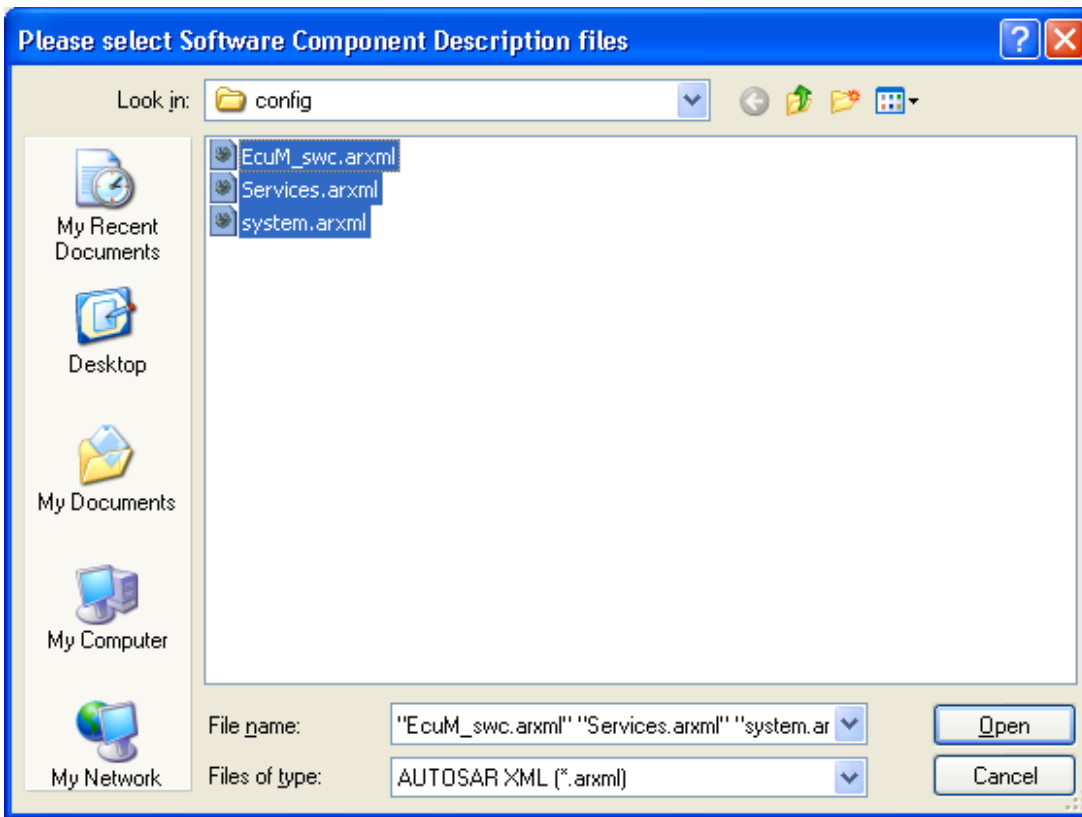
From the pop-up window:

► Select all software component description files which you need to use to generate the application header files.

► Select **Open**.

This opens the **Contract Phase Generation** window.
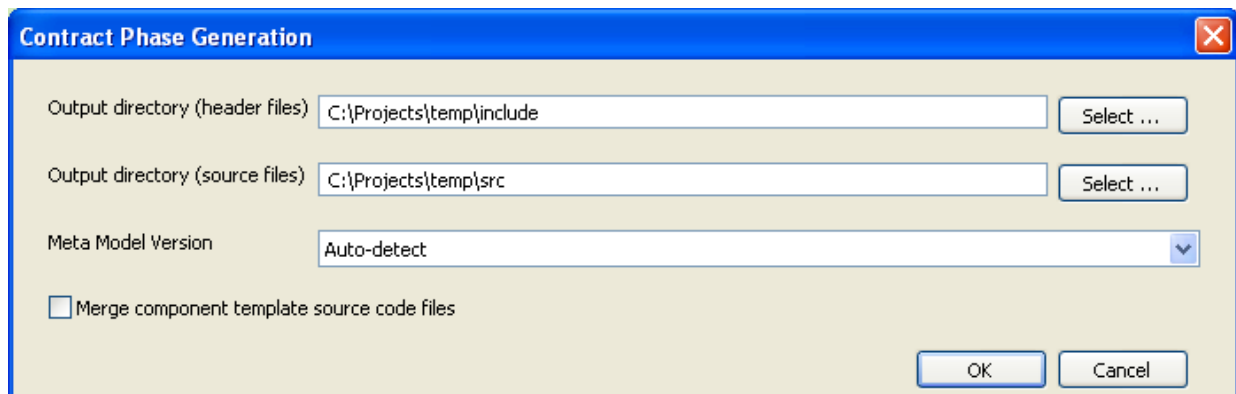
► Specify the directory to which to write the generated header files.

► Specify the directory to which to write the generated source files.

► Select the **Meta Model Version** or use **Auto-detect**.

►   Select whether the **Merge of component template source code files** shall be enabled.

►   Select **OK** to start the contract phase.

Application header files are generated for all atomic software component types described in the selected files.

Moreover, templates for the implementation of the software components are generated. The template files contain skeletons for the runnable entities and example calls for all available API functions.

Possible errors and warnings are displayed in the **Error Log** view.

4.3.5.2. Generating the `Rte` source code

In the `Rte` phase generation, the `Rte` and `BSW Scheduler` source code is generated. If only the core Rte is needed, or only the BSW Scheduler is needed change the parameter **Rte Generator Output** to `Rte Only` or `BSW Scheduler Only`.

Once the generation mode is configured, start the `Rte` phase generation:

►   Either select the **Build Project** menu item, provided by the EB tresos Studio GUI. For details, see the `EB tresos Studio user's guide - Using the GUI - Generating a project.`

►   or generate the `Rte` on the command line with the following command:

```
make generate
```

Both procedures generate the necessary configuration files. The files which are generated are described in Section 4.2.18.5, "Files generated" in detail.

4.3.6. Working without a GUI-configured project in the legacy mode command line

In addition to the project-based command line interface described in the `EB tresos Studio user's guide`, chapter `Using the command line`, EB tresos Studio also supports a legacy mode which directly operates on the configuration files and does not rely on a project configured in the GUI.

For a complete introduction to the legacy mode, consult the `EB tresos Studio user's guide`, chapter `Using the command line`, subchapter `Using the legacy mode command line`.

4.3.6.1. Contract phase in legacy mode

To generate the contract phase on the command line, the following additional parameters must be passed on as arguments:

`contract [<file.arxml>@sysd:<[4.0.3]>]+`	Specifies the .arxml files which contain the software component descriptions for which the contract phase is to be generated. Must specify the version of the XML schema on which the files are based.
`[-Dcontract.merge=[true|false]]`	Optional, specifies whether the component template code files are to be merged (true) or overwritten (false).
`[-Dcontract.components=[swcName1[;swcName2[;swcName3[;...]]]]]`	Optional, specifies for which software components the contract phase shall be performed. If the parameter is not defined the contract phase will be performed for all software components from the input files.

Example:

```
$TRESOS_BASE/bin/tresos_cmd.bat -Dbase.path=C:/EB/tresos
-Dcontract.merge=true
-Dcontract.components=AtomicComponent1;AtomicComponent2
-DEcuResourceModuleIds=Base_TS_TxDxM5I0R0;Rte_TS_TxDxM6I1R0;Make_TS_TxDxM4I0R0;
Platforms_TS_T19D1M2I0R0
legacy make
contract file1.arxml@sysd:4.0.3 file2.arxml@sysd:4.0.3
-o C:/ContractPhaseOutput
-g Rte_TS_TxDxM6I1R0
```

4.3.6.2. Rte phase in legacy mode

To use the legacy mode for the Rte, the following additional parameters must be passed on as arguments:

`[-DtopLevelComposition=/path/to/TopLevelComposition]`	Optional, specifies the top level composition if the input files are software component description
`[-DecuInstance=/path/to/EcuInstance]`	Optional, specifies the ECU instance if the input files are a system description
`[-Dsystem=/path/to/System]`	Specifies the system which contains the system signal, software component to ECU mappings and the root software composition
`[-Drte.configureOs=[ecuc]]`	Optional, specifies the `Os` configuration mode

Example:

```
$TRESOS_BASE/bin/tresos_cmd.bat
-Dtarget=<TARGET> -Dderivate=<DERIVATE>
-Drte.configureOs=ecuc
-Dsystem=/System/system1
-DEcuResourceModuleIds=Base_TS_TxDxM5I0R0;Rte_TS_TxDxM6I1R0;Make_TS_TxDxM4I0R0;
Platforms_TS_T19D1M2I0R0;MemMap_TS_TxDxM1I0R0;Ts5Atl_TS_TxDxM2I0R0;
Os_TS_T19D1M4I4R0_AS40
-Decuid=LegacyEcu
legacy make generate
file1.arxml@sysd file2.arxml@sysd fileN.arxml@sysd
C:/Project/config/Rte.xdm
C:/Project/config/Com.xdm
C:/Project/config/Os.xdm
-g Rte_TS_TxDxM6I1R0
-o C:/Project/output/generated -u
```

4.3.7. Exporting the internal `Rte` model

The `Rte` generator exports the `Rte` internal model when the configuration parameter `RteDataModelExport` is enabled. The model is exported to `output/doc/RteDataModel.xml` and the file contains the internal representation of the `Rte` generator for the *configuration* project.

4.3.7.1. Visualize particular views of the `Rte` model

The exported model can be used to generate a graphical representation of particular elements and their relationships with the goal to provide a view that shows all relevant information which are needed to understand the `Rte` generator *decisions* (e.g. for certain *optimizations*).

EB provides a set of *python* scripts to visualize certain aspects of the Rte model. The scripts can be found in `$(TRESOS_BASE)/plugins/Rte_TS_TxDxMxIxR0/graph`.

The folder `$(TRESOS_BASE)/plugins/Rte_TS_TxDxMxIxR0/graph` contains the *rte_graphs.py* script which can be used to generate a certain diagram. The subfolder `rtegraph` contains the framework to generate a graph and the subfolder `scripts` contains a script per diagram. To add a new diagram, a new *python* file needs to be added here(similar to a plugin system).

Executing the *rte_graphs.py* script shows all the available options.

```
python rte_graphs.py
```

To generate all the diagrams(one diagram per script inside the subfolder `scripts`).

```
python rte_graphs.py -i <RteModel.xml> -o <output folder>
```

To generate a particular diagram(diagram for one of the script inside the subfolder `scripts`).

```
python rte_graphs.py -i <RteModel.xml> -o <output folder> -g <diagram>
```

Example:

```
python rte_graphs.py -i <RteModel.xml> -o <output folder> -g <exclusive_areas_graph>
```

The scripts provided by EB and their purpose are briefly explained in each of the following chapters.

WARNING	Disclaimer: All diagrams are just examples and do not claim to be complete or correct. It is only a support to better understand the internal data model of the generator and the decisions made during the code generation. Only the generated code is qualified.

4.3.7.1.1. Exclusive Area Graph

This diagram shows all exclusive areas and their accessing runnables up to the task context in which they are running. If a runnable is directly called by another runnable, the calling chain is included, too. Exclusive areas

that are colored green, are optimized meaning that even if a lock is configured, the Rte decided that it is not necessary. Otherwise, they are colored red.

With this diagram, you can get a better understanding of the decisions made by the Rte generator. You can figure out why a certain exclusive area is optimized or even not. By looking at the tasks at the end of the chain, you can see the decision whether they can preempt each other or not. In case they preempt each other, a lock is necessary.
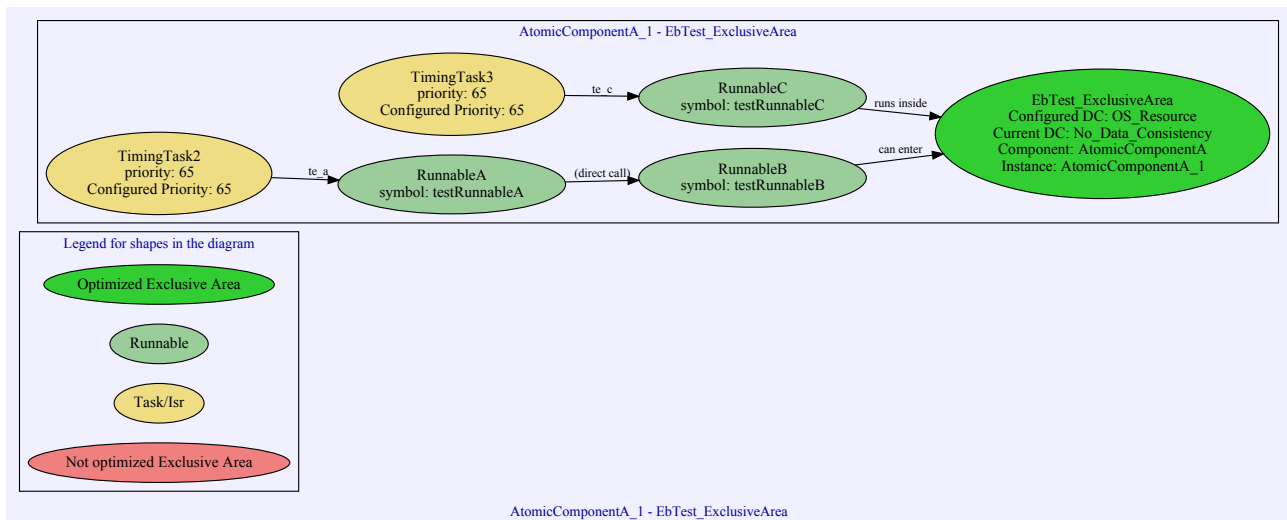
Figure 4.6. Exclusive Area Graph with optimization
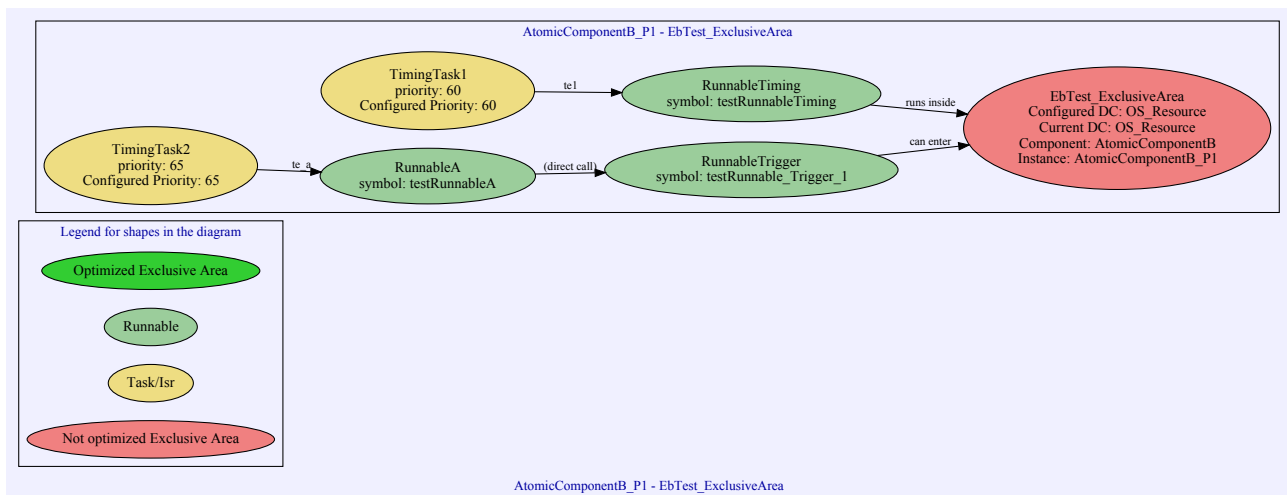
Figure 4.7. Exclusive Area Graph without optimization

4.3.7.1.2. Task Chain Graph

The diagram shows the scheduling of the tasks mapped to a particular task chain and gives an overview of the timing in which each of the tasks are executed.

With this diagram, you can get a better understanding of the scheduling of the tasks in a task chain.
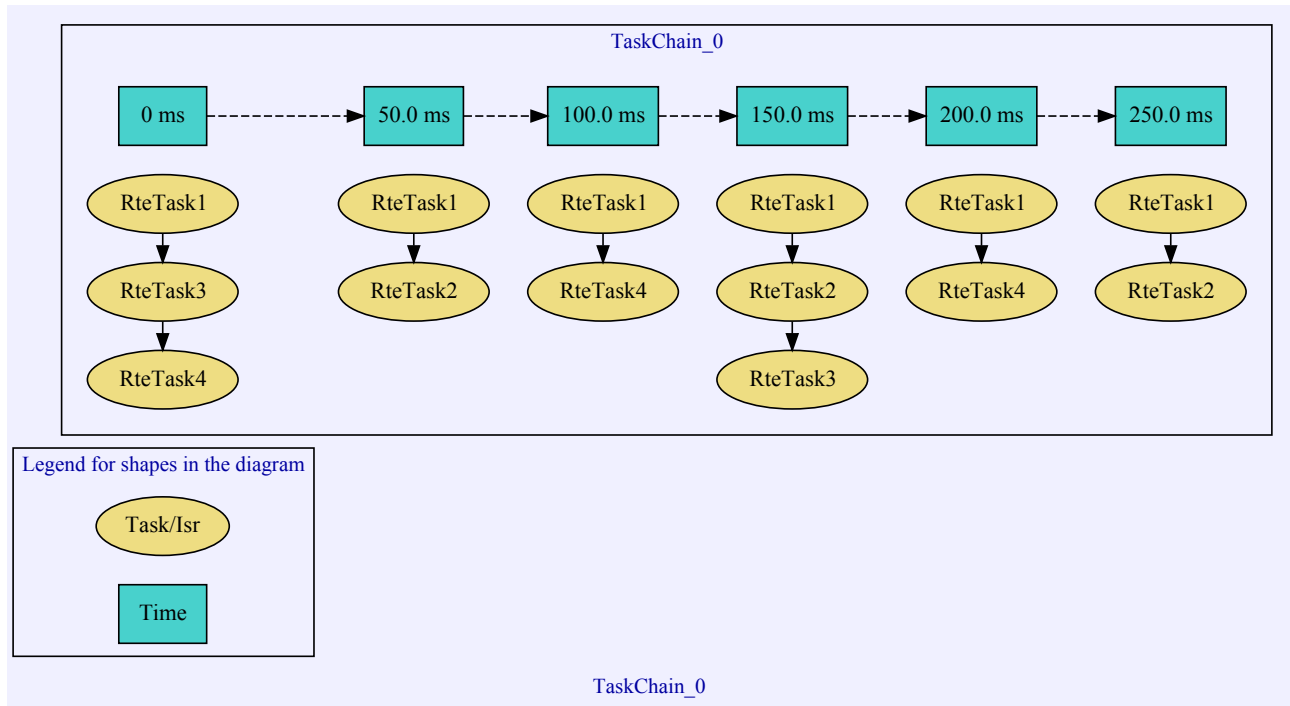
Figure 4.8. Task Chain Graph

4.3.7.1.3. IP Channel Data Consistency Graph

For every inter-partition shared memory buffer (IP Channel), all producers on the left-hand side and all consumers on the right-hand side are shown in the diagram. For every core, the existing consumers/producers are grouped. The calculated data consistency mechanism (core-global/core-local) is also shown.

With this diagram, you can get a better understanding of the Rte generators decision why certain Rte APIs that access that buffer use a certain locking mechanism.
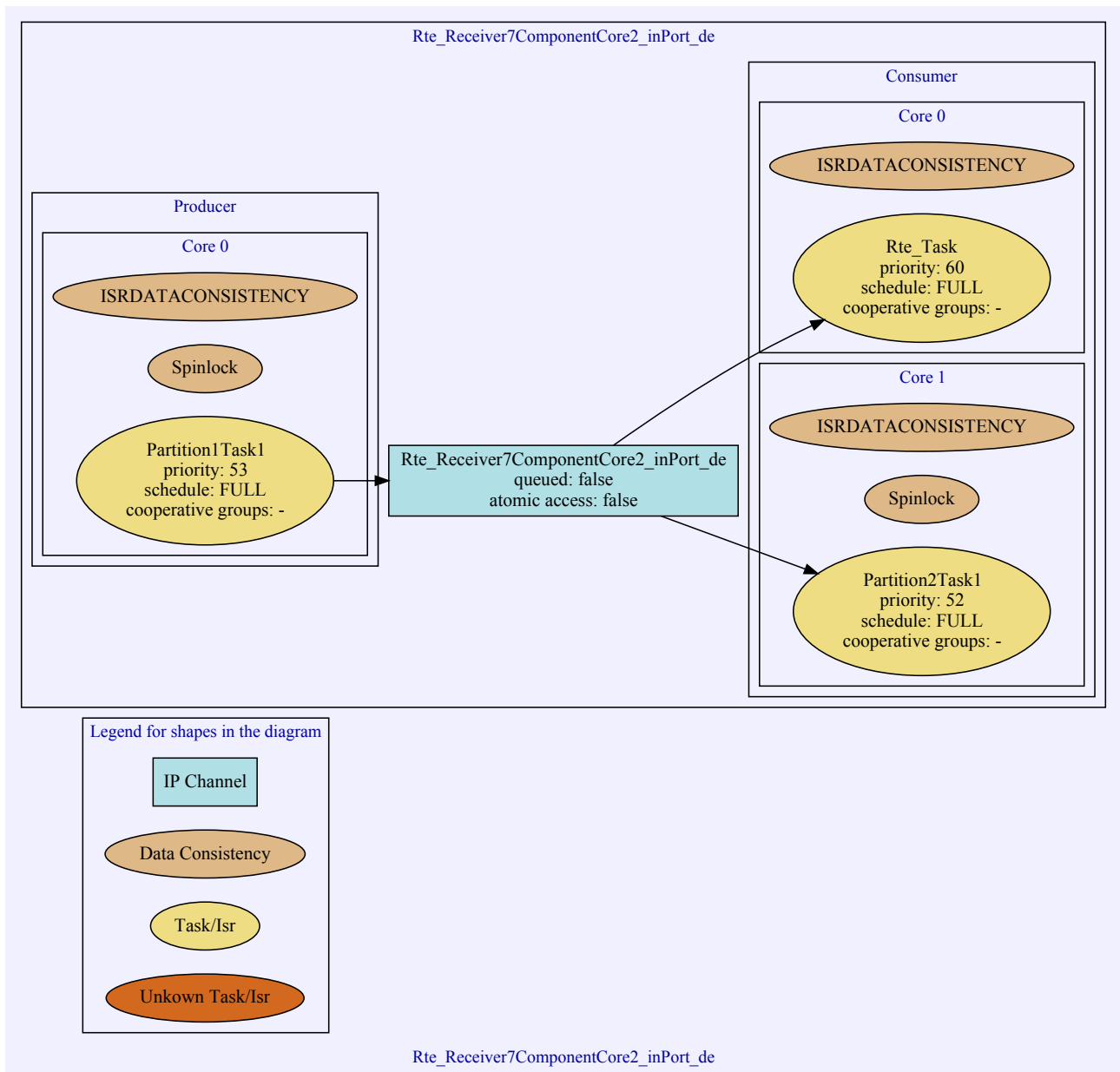
Figure 4.9. IP Channel Data Consistency Graph

4.3.7.1.4. SWC Graph

The diagram shows events, runnables, and APIs of the software components and their connection to other software components via ports and data/operation/ trigger/mode instances.

With this diagram, you can get an overview about the Rte relevant insights of a software component as well as the connected ports and port elements.
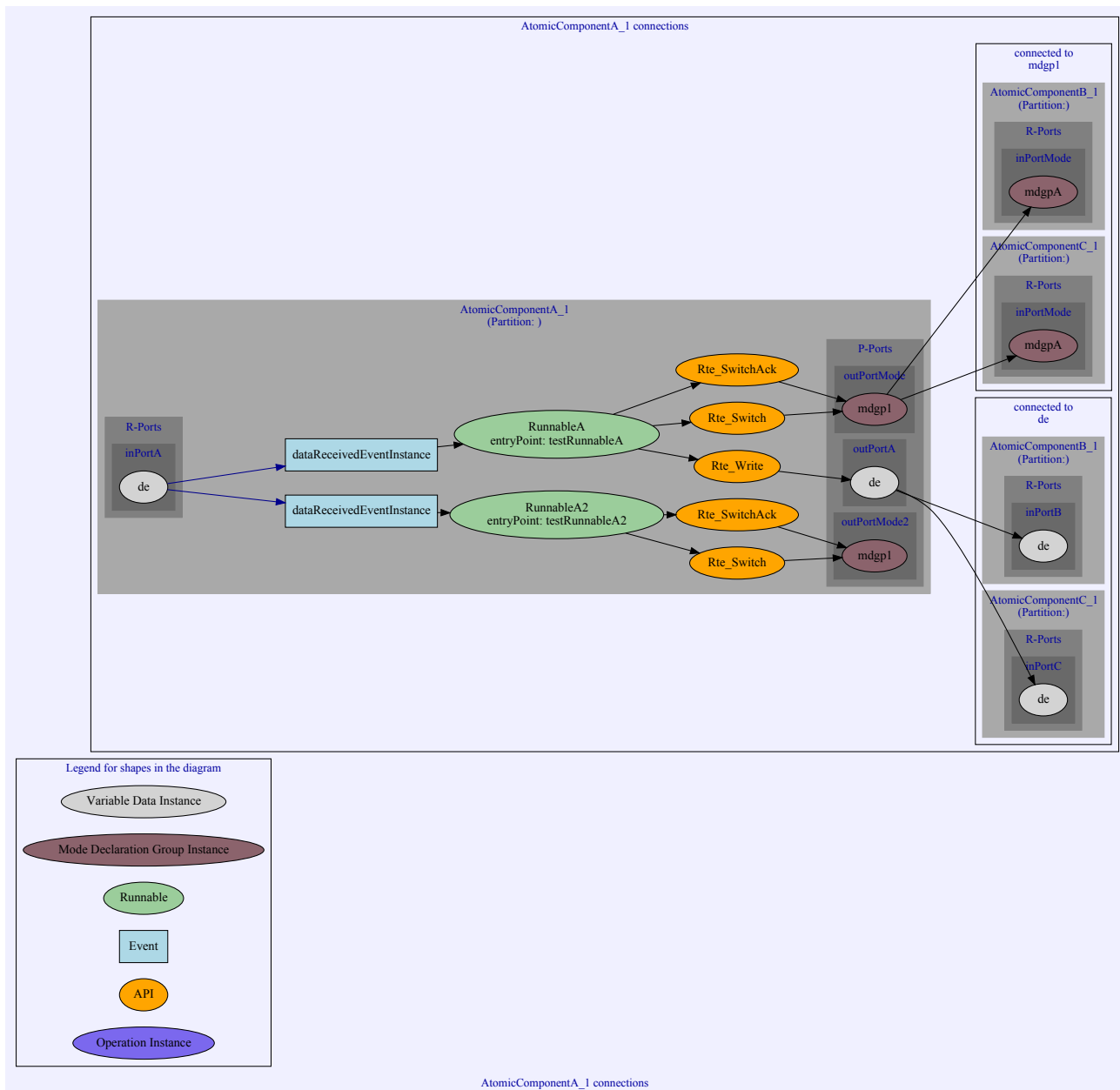
Figure 4.10. SWC Graph

4.3.7.1.5. Tasks Graph

The diagram shows the generated tasks and all runnables, that are running in that task which is either directly mapped, i.e. via an event-to-task mapping, or directly called from another runnable. This means, via direct calling chain. In addition, it shows if a runnable needs an explicit or implicit waitpoint and is thus of category 2. An explicit waitpoint is a waitpoint that is explicitly configured in the model, e.g. to configure a blocking API whereas an implicit waitpoint is shown if the Rte implementation requires a WaitEvent() call. If a runnable has such a waitpoint it is colored red, otherwise green.

With this diagram, you can get a better understanding on how a task needs to be an extended task.If at least one runnable which is running in that task needs a wait point, then it is an interrupt of category 2, and thus the whole task needs a wait state, too which is the characteristic of an extended task. By looking at the diagram, it is possible to easily identify such runnables and to know which one caused the task to be extended.

Please note: Many more conditions need to be checked whether a task is basic or extended. This diagram mainly focuses on the calling chains and possible wait points.
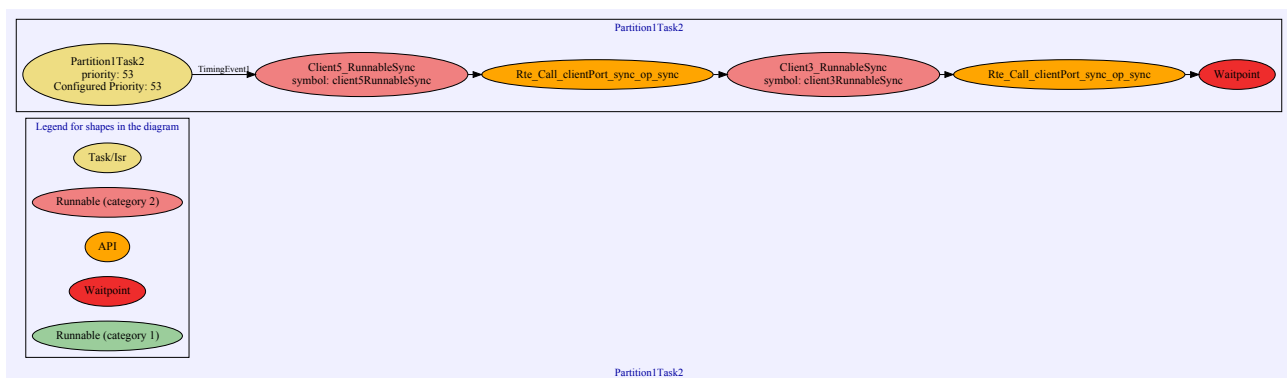
Figure 4.11. Tasks Graph with waitpoint
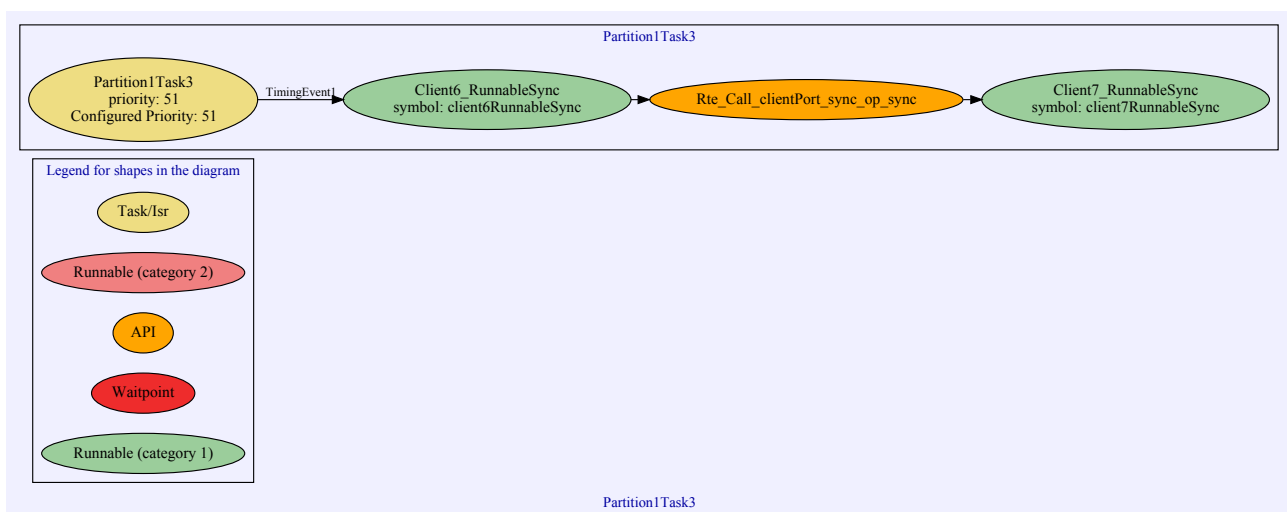
Figure 4.12. Tasks Graph without waitpoint

4.3.7.1.6. Runnable Context Graph

The diagram shows the context for every runnable which is either one or multiple tasks or other runnables in between (direct calling chains). If a runnable is mapped to a BSW entity, this mapping is shown too.

With this diagram, you can get a better understanding of the context in which each runnable can run.
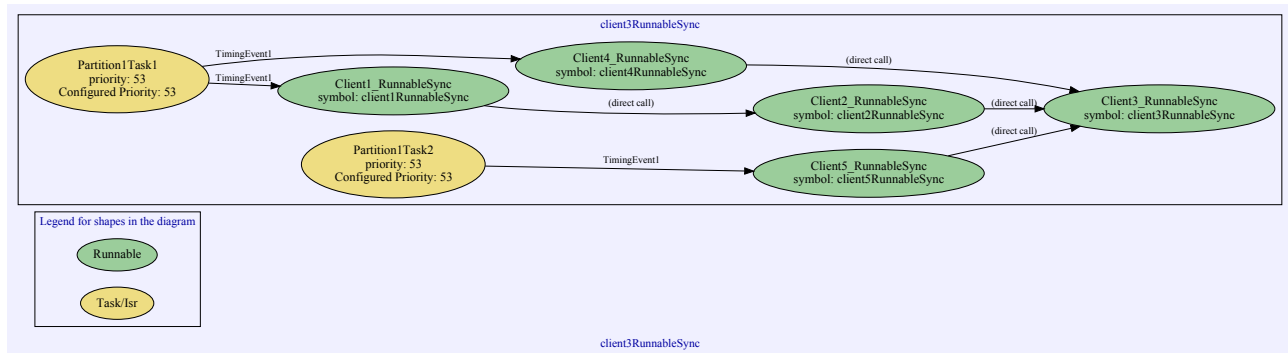
Figure 4.13. Runnable Context Graph

4.3.7.1.7. SR Buffers Graph

The diagram shows all S/R connections between senders and receivers and their related buffers. Each cluster/diagram shows one connection where a connection is defined as a group of RVariableDataInstances (instances of a require variable data prototypes) that have the same data source (same PVariableDataInstances and same receive signals/signal groups) or all PVariableDataInstances (instances of a provide variable data prototypes) if they do not have any receivers. The diagram includes the affected sender and receiver runnables and their context in which they are running as well as the S/R APIs, callbacks, and signals. It shows the data flow from the sender runnables to the implicit or global buffers and back to the connected receivers. Please note that some feature are not supported or not correctly shown in this diagram. For example transformers or receivers that need to identify the sender by a sender ID.

With this diagram, you can get a better understanding on how the Rte allocates the implicit and global buffers. It shows the relevant information to make a decision how many buffers are needed and if buffer sharing is possible between implicit and explicit senders/receivers.

For the SR Buffer Graph with sharing not possible for one receiver : it cannot be shared since the tasks have different priorities.
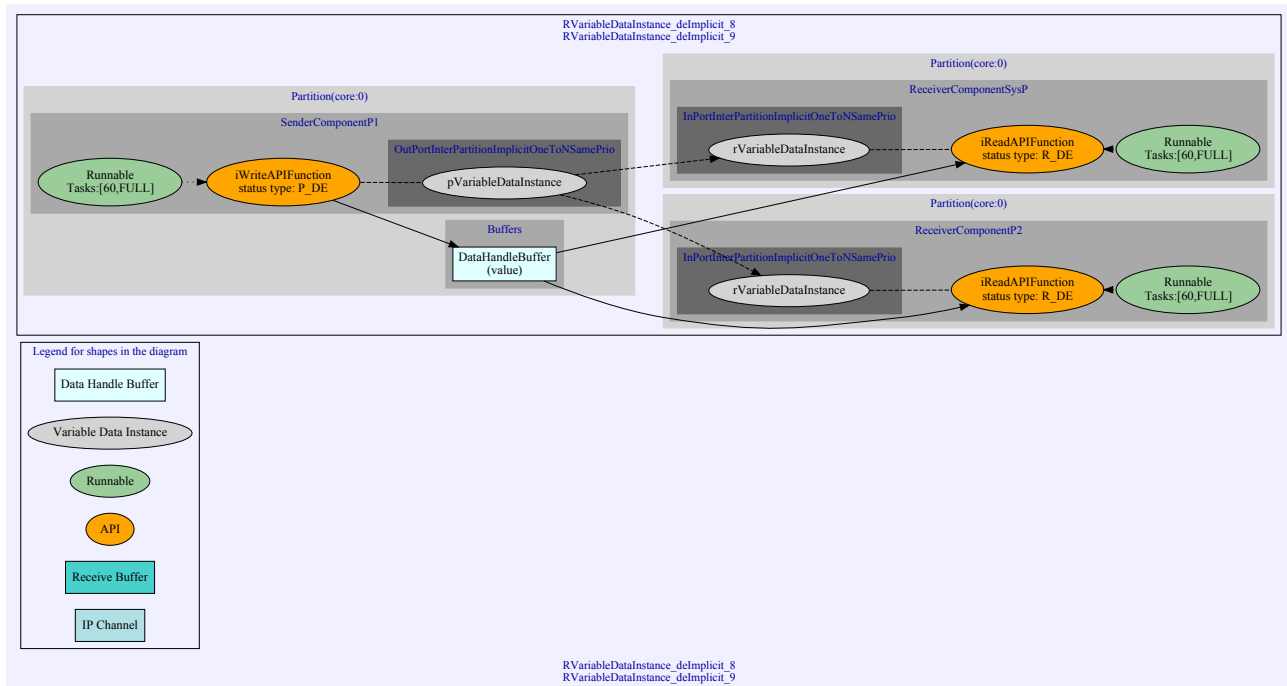
Figure 4.14. SR Buffers Graph with sharing possible
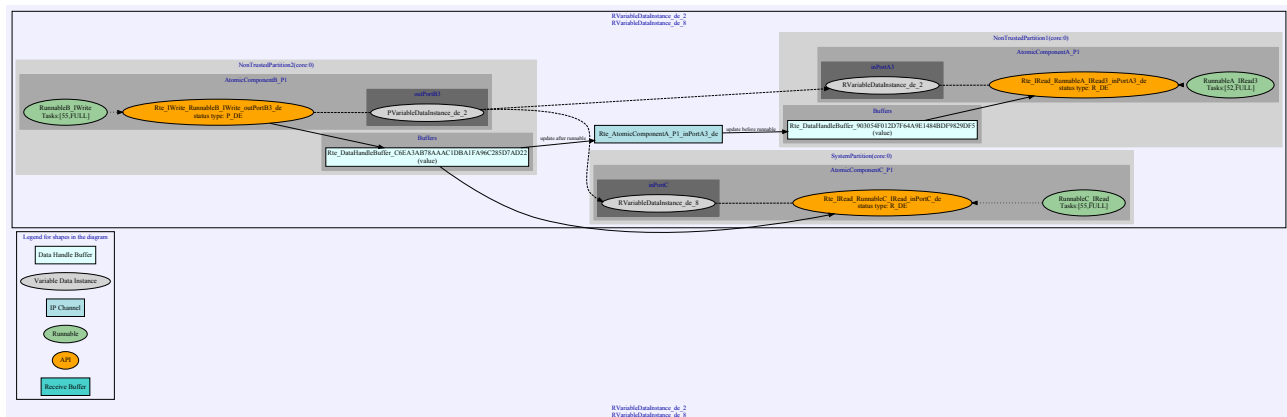
Figure 4.15. SR Buffers Graph with sharing not possible for one receiver

Executing the python scripts

Step 1

Download and install `python` and `graphviz` and add them to the `path` environment variable.

Step 2

Install the python modules `lxml` and `graphviz` using the pip command.

Step 3

The provided scripts are tested with the versions `python` 3.5 and `graphviz` 2.38.0.

4.4. Third party license

This product includes third party components that require the following notices. For respective license terms, see the EB tresos Studio subfolder /licenses

► This product includes Commons Math: The Apache Commons Mathematics Library 3.2. (http://commons.apache.org/proper/commons-math/)

Copyright (C) The Apache Software Foundation. All rights reserved.

5. ACG8 RTE module references

5.1. Overview

This chapter provides module references for the ACG8 RTE product modules. These include a detailed description of all configuration parameters. Furthermore this chapter lists the application programming interface with all data types, constants and functions.

The content of the sections is sorted alphabetically according the EB tresos AutoCore Generic module names.

For further information on the functional behavior of these modules, refer to the chapter ACG8 RTE user's guide.

5.1.1. Notation in EB module references

EB notation may differ from the AUTOSAR standard notation in the software specification documents (SWS). This section describes the notation of *default value* and *range* fields in the EB module references.

5.1.1.1. Default value of configuration parameters

If there is no default value specified for a parameter, the default value field is omitted to prevent ambiguity with parameters that have `--` as default values.

Example: The parameter `BswMCompuConstText` of the `BswM` module of EB tresos AutoCore Generic 8 Mode Management has no default value field, therefore it is omitted.

5.1.1.2. Range information of configuration parameters

The range of a configuration parameter contains an upper and a lower boundary. However, in special cases the range of allowed values can be computed by means of an XPath function that is evaluated at configuration time. An XPath function can either be a standard `xpath:<function>()` or a custom `cxpath:<function>()` function. The range of a configuration parameter may be computed based on other configuration parameters that are referenced from the XPath function. For more information on custom XPath functions, see section *Custom XPath Functions API* of the EB tresos Studio developer's guide.

Example: The parameter `BswMCompuConstText` of the `BswM` module of EB tresos AutoCore Generic 8 Mode Management has the custom XPath function `cxpath:getCompuMethodsVT()` in the range field which provides the allowed values.

5.2. Rte

5.2.1. Configuration parameters

Containers included		
Container name	**Multiplicity**	**Description**
CommonPublishedInformation	1..1	**Label:** Common Published Information Common container, aggregated by all modules. It contains published information about vendor and versions.
PublishedInformation	1..1	**Label:** EB Published Information Additional published parameters not covered by Common-PublishedInformation container.
RteBswGeneral	1..1	**Label:** BSW General General configuration parameters of the BSW scheduler section.
RteBswModuleInstance	0..n	**Label:** BSW Module Instance Represents one instance of a BSW module configured on one ECU.
RteGeneration	1..1	**Label:** Rte Generation This container holds the parameters for the configuration of the Rte generation.
RteImplicitCommunication	0..n	**Label:** Implicit Communication Configuration of the implicit communication behavior to be generated.
RteInitializationBehavior	0..n	**Label:** Initialization Behavior Specifies the initialization strategy for variables allocated by the Rte with the purpose to implement VariableDataPrototypes. The container defines a set of RteSectionInitializationPolicys and one RteInitializationStrategy which is applicable for this set.
RteInitializationRunnableBatch	0..n	**Label:** Rte Initialization Runnable Batch This container corresponds to an Rte_Init_<shortName of this container> function invoking the mapped RunnableEntities.

Containers included		
RteOsInteraction	1..n	**Label:** Os Interaction *The functionality related to this parameter is not supported by the current implementation.* Interaction of the Rte with the Os.
RtePostBuildVariantConfiguration	1..n	**Label:** Post Build Variant Configuration *The functionality related to this parameter is not supported by the current implementation.* Specifies the PostbuildVariantSets for each of the PostBuild configurations of the Rte. The shortName of this container defines the name of the RtePostBuildVariant.
RteRips	0..1	**Label:** Rips Configuration This container provides the configuration of the Rte Implementation Plug-In support by RTE.
RteSwComponentInstance	0..n	**Label:** Software Component Instance 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..n	**Label:** Software Component Type Representation of one SwComponentType for the base of all configuration parameter which are affecting the whole type and not a specific instance.

Parameters included	
Parameter name	**Multiplicity**
IMPLEMENTATION_CONFIG_VARIANT	1..1

Parameter Name	IMPLEMENTATION_CONFIG_VARIANT
Label	Configuration Variant
Multiplicity	1..1
Type	ENUMERATION
Default value	VariantPreCompile
Range	VariantPostBuild

	VariantPreCompile	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile

5.2.1.1. CommonPublishedInformation

Parameters included	
Parameter name	**Multiplicity**
ArMajorVersion	1..1
ArMinorVersion	1..1
ArPatchVersion	1..1
SwMajorVersion	1..1
SwMinorVersion	1..1
SwPatchVersion	1..1
ModuleId	1..1
VendorId	1..1
Release	1..1

Parameter Name	ArMajorVersion
Label	AUTOSAR Major Version
Description	Major version number of AUTOSAR specification on which the appropriate implementation is based on.
Multiplicity	1..1
Type	INTEGER_LABEL
Default value	3
Configuration class	**PublishedInformation:**
Origin	Elektrobit Automotive GmbH

Parameter Name	ArMinorVersion
Label	AUTOSAR Minor Version
Description	Minor version number of AUTOSAR specification on which the appropriate implementation is based on.

Multiplicity	1..1
Type	INTEGER_LABEL
Default value	2
Configuration class	**PublishedInformation:**
Origin	Elektrobit Automotive GmbH

Parameter Name	ArPatchVersion
Label	AUTOSAR Patch Version
Description	Patch level version number of AUTOSAR specification on which the appropriate implementation is based on.
Multiplicity	1..1
Type	INTEGER_LABEL
Default value	0
Configuration class	**PublishedInformation:**
Origin	Elektrobit Automotive GmbH

Parameter Name	SwMajorVersion
Label	Software Major Version
Description	Major version number of the vendor specific implementation of the module.
Multiplicity	1..1
Type	INTEGER_LABEL
Default value	6
Configuration class	**PublishedInformation:**
Origin	Elektrobit Automotive GmbH

Parameter Name	SwMinorVersion
Label	Software Minor Version
Description	Minor version number of the vendor specific implementation of the module. The numbering is vendor specific.
Multiplicity	1..1
Type	INTEGER_LABEL
Default value	6
Configuration class	**PublishedInformation:**

Origin	Elektrobit Automotive GmbH

Parameter Name	SwPatchVersion
Label	Software Patch Version
Description	Patch level version number of the vendor specific implementation of the module. The numbering is vendor specific.
Multiplicity	1..1
Type	INTEGER_LABEL
Default value	3
Configuration class	PublishedInformation:
Origin	Elektrobit Automotive GmbH

Parameter Name	ModuleId
Label	Numeric Module ID
Description	Module ID of this module from Module List
Multiplicity	1..1
Type	INTEGER_LABEL
Default value	2
Configuration class	PublishedInformation:
Origin	Elektrobit Automotive GmbH

Parameter Name	VendorId
Label	Vendor ID
Description	Vendor ID of the dedicated implementation of this module according to the AUTOSAR vendor list
Multiplicity	1..1
Type	INTEGER_LABEL
Default value	1
Configuration class	PublishedInformation:
Origin	Elektrobit Automotive GmbH

Parameter Name	Release
Label	Release Information
Multiplicity	1..1

Type	STRING_LABEL
Default value	
Configuration class	**PublishedInformation:**
Origin	Elektrobit Automotive GmbH

5.2.1.2. PublishedInformation

Parameters included	
Parameter name	**Multiplicity**
PbcfgMSupport	1..1

Parameter Name	PbcfgMSupport
Label	PbcfgM support
Description	Specifies whether or not the Rte can use the PbcfgM module for post-build support.
Multiplicity	1..1
Type	BOOLEAN
Default value	false
Configuration class	**PublishedInformation:**
Origin	Elektrobit Automotive GmbH

5.2.1.3. RteBswGeneral

Parameters included	
Parameter name	**Multiplicity**
RteSchMOsScheduleTableActivationMechanism	1..1
RteSchMOsScheduleTableOffset	1..1
RteSchMVersionInfoApi	1..1
RteUseComShadowSignalApi	1..1

Parameter Name	RteSchMOsScheduleTableActivationMechanism
Label	Os Schedule Table Activation Mechanism

Description	Switch between the two available mechanisms for activating the SchM schedule table.
	► **RELATIVE**: Starts the Rte schedule table with a relative offset.
	► **ABSOLUTE**: Starts the Rte schedule table with an absolute offset.
Multiplicity	1..1
Type	ENUMERATION
Default value	RELATIVE
Range	RELATIVE
	ABSOLUTE
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH

Parameter Name	RteSchMOsScheduleTableOffset	
Label	Os Schedule Table Alarm Offset (s)	
Description	The offset in seconds used for the schedule table start, in case the schedule table activation mechanism is set to **RELATIVE** or **ABSOLUTE**.	
Multiplicity	1..1	
Type	FLOAT	
Default value	0.01	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	RteSchMVersionInfoApi	
Label	Enable Version Info API	
Description	Enables the generation of the SchM_GetVersionInfo() API.	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	false	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile

Origin	AUTOSAR_ECUC

Parameter Name	RteUseComShadowSignalApi	
Description	This parameter defines whether the ComShadowSignalAPIs ((Com_UpdateShadowSignal, Com_InvalidateShadowSignal, Com_ReceiveShadowSignal) are used or not.	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	false	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

5.2.1.4. RteBswModuleInstance

Containers included		
Container name	**Multiplicity**	**Description**
RteBswEventToTaskMapping	0..n	**Label:** BSW Event to Task Mapping Maps a BswSchedulableEntity onto one OsTask based on the activating BswEvent.
RteBswEventToIsrMapping	0..n	**Label:** BSW Event to Isr Mapping Maps a BswSchedulableEntity onto one ISR based on the activating BswExternalTriggerOccurredEvent or BswTimingEvent.
RteBswExclusiveAreaImpl	0..n	**Label:** Exclusive Area Implementation Represents one ExclusiveArea of one BswImplementation. Used to specify the implementation means of this ExclusiveArea.
RteBswExternalTriggerConfig	0..n	Defines the configuration of Inter Basic Software Module Entity Triggering.
RteBswInternalTriggerConfig	0..n	Defines the configuration of internal Basic Software Module Entity Triggering.
RteBswRequiredModeGroup-Connection	0..n	**Label:** Required Mode Group Connection

Containers included

		Defines the connection between one requiredModeGroup of this BSW module instance and one providedModeGroup instance.
RteBswRequiredSender-ReceiverConnection	0..n	Defines the connection between one requiredData and one providedData of a BswModuleDescription. This container shall be provided on the receiver side of the connection.
RteBswRequiredClien-tServerConnection	0..n	Defines the connection between one requiredClientServer-Entry and one providedClientServerEntry of a BswModuleDescription. This container shall be provided on the client side of the connection.
RteBswRequiredTriggerCon-nection	0..n	**Label:** Required Trigger Connection Defines the connection between one requiredTrigger of this BSW module instance and one releasedTrigger instance.

Parameters included

Parameter name	Multiplicity
RteBswImplementationRef	1..1
RteBswModuleConfigurationRef	0..1
RteMappedToOsApplicationRef	0..1
ASR31SchMExclusiveAreaAPISupport	1..1
ASR31SchMExclusiveAreaAPIUseInstanceParameter	1..1

Parameter Name	RteBswImplementationRef	
Label	Implementation	
Description	Reference to the BswImplementation for which the BSW scheduler is configured.	
Multiplicity	1..1	
Type	FOREIGN-REFERENCE	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	RteBswModuleConfigurationRef
Label	Module Configuration

Description	Reference to the ECU configuration values provided for this BswImplementation.
Multiplicity	0..1
Type	FOREIGN-REFERENCE
Configuration class	**PreCompile:**	VariantPostBuild
	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC

Parameter Name	**RteMappedToOsApplicationRef**
Label	Mapped to Os Application
Description	A reference to an Os Application to which the Bsw Module Instance shall be mapped.
Multiplicity	0..1
Type	REFERENCE
Configuration class	**PreCompile:**	VariantPostBuild
	PreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH

Parameter Name	**ASR31SchMExclusiveAreaAPISupport**
Label	Generate the ASR-3.1 SchM Exclusive Area API
Description	When enabled, the ASR-3.1 SchM Exclusive Area API wrapper will be generated. This API wrapper will map the legacy ASR-3.1 SchM Exclusive Area API to the ASR-4.0 one. This configuration option only functions in combination when the BSW module instance is provided as source code.
Multiplicity	1..1
Type	BOOLEAN
Default value	false
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH

Parameter Name	**ASR31SchMExclusiveAreaAPIUseInstanceParameter**
Label	Include the instance parameter within ASR 3.1 SchM Exclusive Area API
Description	When enabled, the parameter "instance" will be included in the ASR-3.1 SchM Exclusive Area API parameter list. The inclusion of the instance parameter was optional in the ASR-3.1 SchM specification. This configuration option is need-

	ed only when the containing ASR-3.1 BSW module instance code passes its instance handle to the SchM API.	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	false	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

5.2.1.5. RteBswEventToTaskMapping

Parameters included	
Parameter name	**Multiplicity**
[RteBswActivationOffset](#)	0..1
[RteBswPeriod](#)	0..1
[RteBswImmediateRestart](#)	1..1
[RteBswPositionInTask](#)	0..1
[RteBswServerQueueLength](#)	0..1
[RteOsSchedulePoint](#)	0..1
[RteBswEventRef](#)	1..1
[RteBswMappedToTaskRef](#)	0..1
[RteBswAllowInterPartitionDirectCall](#)	1..1
[RteBswUsedOsAlarmRef](#)	0..1
[RteBswUsedOsEventRef](#)	0..1
[RteBswUsedOsSchTblExpiryPointRef](#)	0..1
[RteRipsFillRoutineRef](#)	0..1
[RteRipsFlushRoutineRef](#)	0..1

Parameter Name	**RteBswActivationOffset**
Label	Activation Offset (s)
Description	Activation offset in seconds.
Multiplicity	0..1
Type	FLOAT

Default value	0.0	
Configuration class	**PreCompile:**	VariantPostBuild
	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	**RteBswPeriod**	
Label	Period (s) for OperationInvokedEvents	
Description	Period in seconds for OperationInvokedEvents. Enabling this parameter disables the event based triggering of any runnable assigned to this event.	
Multiplicity	0..1	
Type	FLOAT	
Default value	0.1	
Range	>0	
Configuration class	**PreCompile:**	VariantPostBuild
	PreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	**RteBswImmediateRestart**	
Label	Immediate Restart	
Description	*The functionality related to this parameter is not supported by the current implementation.* 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..1	
Type	BOOLEAN	
Default value	false	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	**RteBswPositionInTask**

Label	Position in Task
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.
Multiplicity	0..1
Type	INTEGER
Default value	0
Range	<=65535
	>=0
Configuration class	**PreCompile:**	VariantPostBuild
	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC

Parameter Name	**RteBswServerQueueLength**
Label	Server Queue Length
Description	Specifies the length of the queue for the server call serialization.
Multiplicity	0..1
Type	INTEGER
Default value	1
Range	<=65535
	>=0
Configuration class	**PreCompile:**	VariantPostBuild
	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC

Parameter Name	**RteOsSchedulePoint**
Label	Os Schedule Point
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	ENUMERATION	
Range	CONDITIONAL	
	NONE	
	UNCONDITIONAL	
Configuration class	**PreCompile:**	VariantPostBuild
	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	**RteBswEventRef**	
Label	BSW Event	
Description	Reference to the BswEvent which is pointing to the BswSchedulableEntity being mapped. This allows a fine grained mapping of BswSchedulableEntites based on the activating BswEvent.	
Multiplicity	1..1	
Type	FOREIGN-REFERENCE	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	**RteBswMappedToTaskRef**	
Label	Mapped to Task	
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	
Configuration class	**PreCompile:**	VariantPostBuild
	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	**RteBswAllowInterPartitionDirectCall**
Label	Allow inter partition direct call

Description	When RteBswMappedToTaskRef is empty and RteBswAllowInterPartitionDirect-Call is set to true the inter-partition direct-call is allowed.	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	false	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	RteBswUsedOsAlarmRef	
Label	Used Os Alarm	
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	
Configuration class	**PreCompile:**	VariantPostBuild
	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	RteBswUsedOsEventRef	
Label	Used Os Event	
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	
Configuration class	**PreCompile:**	VariantPostBuild
	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	RteBswUsedOsSchTblExpiryPointRef	
Label	Used Os Schedule Table Expiry Point	
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	
Configuration class	**PreCompile:**	VariantPostBuild
	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	**RteRipsFillRoutineRef**	
Label	Rips Fill Routine	
Description	Reference to a Buffer-Fill Routine implemented by an RTE Implementation Plug-In.	
Multiplicity	0..1	
Type	URI-REFERENCE	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
	VariantPostBuild:	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	**RteRipsFlushRoutineRef**	
Label	Rips Flush Routine	
Description	Reference to a Buffer-Flush Routine implemented by an RTE Implementation Plug-In.	
Multiplicity	0..1	
Type	URI-REFERENCE	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
	VariantPostBuild:	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

5.2.1.6. RteBswEventToIsrMapping

Parameters included	
Parameter name	**Multiplicity**

Parameters included	
RteBswPositionInIsr	0..1
RteBswEventRef	1..1
RteBswMappedToIsrRef	0..1

Parameter Name	RteBswPositionInIsr	
Label	Position in ISR	
Description	Each BswSchedulableEntity activation mapped to an ISR has a specific position within the ISR execution. For event driver activation this is the order of evaluation which actual BswSchedulableEntity has to be executed.	
Multiplicity	0..1	
Type	INTEGER	
Default value	0	
Range	<=65535	
	>=0	
Configuration class	**PreCompile:**	VariantPostBuild
	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	RteBswEventRef	
Label	BSW Event	
Description	Reference to the BswExternalTriggerOccurredEvent or BswTimingEvent which is pointing to the BswSchedulableEntity being mapped. This allows a fine grained mapping of BswSchedulableEntity based on the activating BswEvent.	
Multiplicity	1..1	
Type	FOREIGN-REFERENCE	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	RteBswMappedToIsrRef
Label	Mapped to Isr
Description	Reference to the ISR the BswSchedulableEntity activated by the RteBswEventRef is mapped to.

Multiplicity	0..1	
Type	REFERENCE	
Configuration class	**PreCompile:**	VariantPostBuild
	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

5.2.1.7. RteBswExclusiveAreaImpl

Parameters included	
Parameter name	**Multiplicity**
RteExclusiveAreaImplMechanism	1..1
RteBswExclusiveAreaRef	1..1
ExclusiveAreaUserCalloutEnter	0..1
ExclusiveAreaUserCalloutExit	0..1
RteBswExclusiveAreaOsResourceRef	0..1
RteBswExclusiveAreaResponsibleRipsPluginRef	0..1

Parameter Name	**RteExclusiveAreaImplMechanism**	
Label	Exclusive Area Implementation Mechanism	
Description	To be used implementation mechanism for the specified ExclusiveArea.	
Multiplicity	1..1	
Type	ENUMERATION	
Range	ALL_INTERRUPT_BLOCKING	
	COOPERATIVE_RUNNABLE_PLACEMENT	
	OS_INTERRUPT_BLOCKING	
	OS_RESOURCE	
	EB_FAST_LOCK	
	NO_LOCK	
	OS_SPINLOCK	
	USER_CALLOUT	
	RTE_PLUGIN	
Configuration class	**VariantPostBuild:**	VariantPostBuild

	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	**RteBswExclusiveAreaRef**	
Label	Exclusive Area	
Description	Reference to the ExclusiveArea for which the implementation mechanism shall be specified.	
Multiplicity	1..1	
Type	FOREIGN-REFERENCE	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	**ExclusiveAreaUserCalloutEnter**	
Label	Exclusive Area UserCalloutEnter	
Description	The name of the function that is called by the Rte to enter the exclusive area if RteExclusiveAreaImplMechanism is set to USER_CALLOUT.	
Multiplicity	0..1	
Type	STRING	
Configuration class	**PreCompile:**	VariantPostBuild
	PreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	**ExclusiveAreaUserCalloutExit**	
Label	Exclusive Area UserCalloutExit	
Description	The name of the function that is called by the Rte to leave the exclusive area if RteExclusiveAreaImplMechanism is set to USER_CALLOUT.	
Multiplicity	0..1	
Type	STRING	
Configuration class	**PreCompile:**	VariantPostBuild
	PreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	**RteBswExclusiveAreaOsResourceRef**

Label	Os Resource
Description	Optional reference to an OsResource in case RteExclusiveAreaImplMechanism is configured to OS_RESOURCE for this ExclusiveArea.
Multiplicity	0..1
Type	REFERENCE
Configuration class	**PreCompile:**	VariantPostBuild
	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC

Parameter Name	**RteBswExclusiveAreaResponsibleRipsPluginRef**	
Label	Rips Plugin	
Description	Reference to destinationUri [RteRipsUriDefSet/RteRipsPlugin]	
Multiplicity	0..1	
Type	URI-REFERENCE	
Configuration class	**PreCompile:**	VariantPostBuild
	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

5.2.1.8. RteBswExternalTriggerConfig

Parameters included	
Parameter name	**Multiplicity**
RteBswTriggerSourceQueueLength	1..1
RteBswTriggerSourceRef	1..1

Parameter Name	**RteBswTriggerSourceQueueLength**
Description	Length of trigger queue on the trigger source side.
Multiplicity	1..1
Type	INTEGER
Default value	0
Range	<=4294967295
	>=0

Configuration class	VariantPostBuild:	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	RteBswTriggerSourceRef	
Description	Reference to a Trigger instance in the role releasedTrigger of the related BSW Module instance.	
Multiplicity	1..1	
Type	FOREIGN-REFERENCE	
Configuration class	VariantPostBuild:	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

5.2.1.9. RteBswInternalTriggerConfig

Parameters included	
Parameter name	Multiplicity
RteBswTriggerSourceQueueLength	1..1
RteBswTriggerSourceRef	1..1

Parameter Name	RteBswTriggerSourceQueueLength	
Description	Length of trigger queue on the trigger source side.	
Multiplicity	1..1	
Type	INTEGER	
Default value	0	
Range	<=4294967295	
	>=0	
Configuration class	VariantPostBuild:	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	RteBswTriggerSourceRef

Description	Reference to a BswInternalTriggeringPoint of the related BSW Module instance.	
Multiplicity	1..1	
Type	FOREIGN-REFERENCE	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

5.2.1.10. RteBswRequiredModeGroupConnection

Parameters included	
Parameter name	**Multiplicity**
RteBswProvidedModeGroupRef	1..1
RteBswRequiredModeGroupRef	1..1
RteBswProvidedModeGrpModInstRef	1..1

Parameter Name	**RteBswProvidedModeGroupRef**	
Label	Provided Mode Group	
Description	References the providedModeGroupPrototype to which this requiredModeGroup shall be connected.	
Multiplicity	1..1	
Type	FOREIGN-REFERENCE	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	**RteBswRequiredModeGroupRef**	
Label	Required Mode Group	
Description	References requiredModeGroupPrototype which shall be connected to the providedModeGroupPrototype.	
Multiplicity	1..1	
Type	FOREIGN-REFERENCE	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile

Origin	AUTOSAR_ECUC

Parameter Name	RteBswProvidedModeGrpModInstRef	
Label	Provided Mode Group Module Instance	
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..1	
Type	REFERENCE	
Configuration class	VariantPostBuild:	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

5.2.1.11. RteBswRequiredSenderReceiverConnection

Parameters included	
Parameter name	Multiplicity
RteBswProvidedVariableDataPrototypeRef	1..1
RteBswRequiredVariableDataPrototypeRef	1..1
RteBswProvidedDataModInstRef	1..1

Parameter Name	RteBswProvidedVariableDataPrototypeRef	
Description	Reference the providedData for this connection.	
Multiplicity	1..1	
Type	FOREIGN-REFERENCE	
Configuration class	VariantPostBuild:	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	RteBswRequiredVariableDataPrototypeRef
Description	Reference the requiredData for this connection.
Multiplicity	1..1
Type	FOREIGN-REFERENCE

Configuration class	VariantPostBuild:	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	RteBswProvidedDataModInstRef	
Description	Reference to the RteBswModuleInstance configuration container which identifies the instance of the BSW Module.	
Multiplicity	1..1	
Type	REFERENCE	
Configuration class	VariantPostBuild:	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

5.2.1.12. RteBswRequiredClientServerConnection

Parameters included	
Parameter name	**Multiplicity**
RteBswProvidedClientServerEntryRef	1..1
RteBswRequiredClientServerEntryRef	1..1
RteBswProvidedClientServerEntryModInstRef	1..1

Parameter Name	RteBswProvidedClientServerEntryRef	
Description	Reference the providedClientServerEntry for this connection.	
Multiplicity	1..1	
Type	FOREIGN-REFERENCE	
Configuration class	VariantPostBuild:	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	RteBswRequiredClientServerEntryRef	
Description	Reference the requiredClientServerEntry for this connection.	
Multiplicity	1..1	
Type	FOREIGN-REFERENCE	

Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	**RteBswProvidedClientServerEntryModInstRef**	
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..1	
Type	REFERENCE	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

5.2.1.13. RteBswRequiredTriggerConnection

Parameters included	
Parameter name	**Multiplicity**
[RteBswReleasedTriggerRef](#)	1..1
[RteBswRequiredTriggerRef](#)	1..1
[RteBswReleasedTriggerModInstRef](#)	1..1

Parameter Name	**RteBswReleasedTriggerRef**	
Label	Released Trigger	
Description	References the releasedTrigger to which this requiredTrigger shall be connected.	
Multiplicity	1..1	
Type	FOREIGN-REFERENCE	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	**RteBswRequiredTriggerRef**	

Label	Required Trigger
Description	References one requiredTrigger which shall be connected to the releasedTrigger.
Multiplicity	1..1
Type	FOREIGN-REFERENCE
Configuration class	**VariantPostBuild:** VariantPostBuild
	VariantPreCompile: VariantPreCompile
Origin	AUTOSAR_ECUC

Parameter Name	**RteBswReleasedTriggerModInstRef**
Label	Released Trigger Module Instance
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..1
Type	REFERENCE
Configuration class	**VariantPostBuild:** VariantPostBuild
	VariantPreCompile: VariantPreCompile
Origin	AUTOSAR_ECUC

5.2.1.14. RteGeneration

Containers included		
Container name	**Multiplicity**	**Description**
ComTaskConfiguration	0..n	**Label:** Com/LdCom Task Configuration
BswConfiguration	1..1	**Label:** BSW Configuration
OsCounterAssignments	0..1	**Label:** Assignment of Os Counters to Partitions
CooperativeTasks	0..n	**Label:** Cooperative Task Groups Defines cooperative task groups. Each entry represents a list of OsTasks that cannot interrupt each other. This list will be taken into account by the Rte generator to optimize locks and to reduce the number of implicit buffers if the InterPartitionCommunication value is 'Disabled' or 'SharedMemory'.
TaskChain	0..n	**Label:** Task Chain

Containers included		
		Specifies a set of OsTasks that shall be executed in a specific order. Once one task completes the next task in the chain will be scheduled.

Parameters included	
Parameter name	**Multiplicity**
ASR32RteWrapper	1..1
GenerateTimestamp	1..1
OverrideXfBufferComputation	1..1
BswmdOutputDirectory	1..1
ComCbkNotInterruptable	1..1
OptimizeCdsGeneration	1..1
DataConsistencyMechanism	1..1
DirectReadFromCom	1..1
DisableInvalidationDataConsistency	1..1
DisablePartitionActiveChecks	1..1
FunctionElision	1..1
GenerateEmptyRteStartStopStubs	1..1
InterECUInvalidationHandledByRte	1..1
OsSupportsSpinlockLockMethod	1..1
RespectConfiguredTaskType	1..1
InterPartitionCommunication	1..1
SpinlockAllocationStrategy	0..1
OneSendSignalQueuePerCore	1..1
SendSignalQueueStrategy	1..1
OneScheduleTablePerPartition	1..1
InterruptBlockingFunction	1..1
OsCounterRef	1..1
OSEKCompatibilityMode	1..1
OsScheduleTableActivationMechanism	1..1
OsScheduleTableMaxExpiryPoints	1..1
OsScheduleTableOffset	1..1
OsUserScheduleTableRef	1..1

Parameters included	
RteCalibrationSupport	1..1
RteCodeVendorId	1..1
RteDataModelExport	0..1
RteDevErrorDetect	1..1
HumanReadableBufferNames	1..1
RteDevErrorDetectUninit	1..1
RteGenerationMode	1..1
RteGeneratorOutput	1..1
RteIocInteractionReturnValue	1..1
RteMeasurementSupport	1..1
RteOptimizationMode	1..1
RteToolChainSignificantCharacters	0..1
RteValueRangeCheckEnabled	1..1
RteVfbTraceClientPrefix	0..n
RteVfbTraceEnabled	1..1
RteVfbTraceFunction	0..n
SingleScheduleTablePartitionRef	0..1
UnconnectedRequirePorts	1..1
GenerateRteFreedomFromInterferenceReviewInstructions	0..1
ExecuteTransformersDespiteLocalDataUnqueued	0..1
ExecuteTransformersDespiteLocalDataQueued	0..1

Parameter Name	ASR32RteWrapper	
Label	Generate ASR 3.2 Rte Wrapper	
Description	Enables or disables the generation of the AUTOSAR 3.2 Rte Wrapper. The AUTOSAR 3.2 Rte Wrapper provides an AUTOSAR 3.2 compliant Rte API based on the AUTOSAR 4.0 Rte implementation. The AUTOSAR 3.2 Rte Wrapper is directly implemented in the Application Header file.	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	false	
Configuration class	**VariantPostBuild:**	VariantPostBuild

	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	GenerateTimestamp	
Label	Generate timestamp	
Description	Defines if a timestamp shall be generated in RTE generated files (true) or not (false).	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	true	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	OverrideXfBufferComputation	
Label	Override transformer Buffer Computation	
Description	Replaces the bufferComputation of each transformer in a chain by the length of the Com container of the ISignal(Group) which references the transformer chain. In case of LdCom the PduLength defined by the referenced EcuC Pdu is used.	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	false	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	BswmdOutputDirectory	
Label	Bswmd Configuration Output File Directory	
Description	Defines the directory into which Rte_Bswmd.arxml will be generated.	
Multiplicity	1..1	
Type	STRING	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile

Origin	Elektrobit Automotive GmbH	

Parameter Name	**ComCbkNotInterruptable**	
Label	Com Callbacks are not interruptable	
Description	Defines that Com callback functions are not interruptible if set to true. In this case (e.g. they run in non-interruptible ISRs or tasks), the Rte does not apply any data consistency mechanism within Com callbacks.	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	false	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	**OptimizeCdsGeneration**	
Label	Optimize CDS generation	
Description	If set to true, the Rte does not create a Component Data Structure entry for data handle buffers, inter runnable variables and dirtyFlag variables, if implicit communication is used, the SWC is not delivered as object code and multiple instantation is not configured. In this case these variables are accessed directly.	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	false	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	**DataConsistencyMechanism**	
Label	Data Consistency Realization Mechanism	
Description	Defines the default data consistency mechanism. Possible options are: usage of Os resources (default) or interrupt blocking. The data consistency mechanism is applied to receive buffers/queues and inter runnable variables if data corruption might occur.	
Multiplicity	1..1	
Type	ENUMERATION	
Default value	OsResource	

Range	OsResource	
	InterruptBlocking	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	**DirectReadFromCom**	
Label	Direct Read from Com	
Description	If enabled, the Rte may do optimizations to save some RAM by directly reading values from Com instead of buffering them. In some scenarios where the software components poll Com values at a frequency higher than the update frequency, this optimization may cause an unwanted increase of runtime. If the I-PDU group is re-started after Rte startup, the Rte will read the initial values provided by Com module. If disabled, the Rte will always buffer values from Com. This may cause an increase of memory consumption. If the I-PDU group is re-started after Rte startup, the Rte will keep the last received value and will not read the initial value provided by Com module.	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	true	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	**DisableInvalidationDataConsistency**	
Label	Disable interrupt locks for data invalidation	
Description	If set to true, the Rte will not generate interrupt locks for data invalidation when reading/writing the Rte status and the data value. This only applies to intra-partition (intra-ECU and inter-ECU) communication and primitive data types.	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	false	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	DisablePartitionActiveChecks	
Label	Disable partition active checks for API functions and callbacks	
Description	If set to true, the Rte will not fulfill requirements rte_sws_2538, rte_sws_2535 and rte_sws_2536 anymore. This means that the Rte does not check for each API function if the current partition is active. You have to ensure that no call-backs and no API functions are called before Rte_Start()/after Rte_Stop() has been executed. The effect of this optimization depends on the number of generated API functions and how often they are called by the application.	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	false	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	FunctionElision	
Label	Function Elision	
Description	Defines if the function elision shall be enabled (true) or disabled (false). If function elision is enabled, macros will be used instead of functions where possible.	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	false	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	GenerateEmptyRteStartStopStubs
Label	Generate empty life-cycle API stubs
Description	When true the Rte will generate empty life-cycle API stubs for:

> ► Rte_Start

> ► Rte_Stop

> ► SchM_Deinit

> ► SchM_Init

> ► Rte_StartTiming

	This will be required when the EcuM is operating from within a non-trusted partition and calling a trusted partition (multiple partitions with memory protection enabled). It is under the responsibility of an integrator to implement and call the <OsApplication> specific functions.
Multiplicity	1..1
Type	BOOLEAN
Default value	false
Configuration class	**VariantPostBuild:** VariantPostBuild
	VariantPreCompile: VariantPreCompile
Origin	Elektrobit Automotive GmbH

Parameter Name	**InterECUInvalidationHandledByRte**
Label	Inter-ECU Signal Invalidation handled by Rte
Description	Defines if the inter-ECU signal invalidation is handled by the Rte or not. If it is not handled by the Rte then it is handled by the Com according to the Autosar specification.
Multiplicity	1..1
Type	BOOLEAN
Default value	false
Configuration class	**VariantPostBuild:** VariantPostBuild
	VariantPreCompile: VariantPreCompile
Origin	Elektrobit Automotive GmbH

Parameter Name	**OsSupportsSpinlockLockMethod**
Label	Os supports OsSpinlockLockMethod
Description	Defines if OsSpinlockLockMethod is supported by the Os. If it is supported then the Rte configures the OsSpinlockLockMethod to LOCK_ALL_INTERRUPTS. Otherwise, the Rte generates interrupt locks for every spinlock.
Multiplicity	1..1
Type	BOOLEAN
Default value	false
Configuration class	**VariantPostBuild:** VariantPostBuild
	VariantPreCompile: VariantPreCompile

Origin	Elektrobit Automotive GmbH

Parameter Name	**RespectConfiguredTaskType**	
Label	Respect configured task type	
Description	If set to true, the Rte respects the configured OsTaskType and reports an error if the calculated task mapping scenario requires an EXTENDED task although the OsTaskType is set to BASIC. Otherwise, the Rte ignores the configured OsTaskType and might generate an EXTENDED task even if OsTaskType is set to BASIC.	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	false	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	**InterPartitionCommunication**	
Label	InterPartitionCommunication	
Description	Defines the inter-partition communication method to be used.	
Multiplicity	1..1	
Type	ENUMERATION	
Default value	Disabled	
Range	Disabled	
	Ioc	
	SharedMemory	
	Mixed	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	**SpinlockAllocationStrategy**
Label	SpinlockAllocationStrategy
Description	Defines the strategy how the Rte shall allocate Spinlocks for the SMC.
Multiplicity	0..1
Type	ENUMERATION

Default value	OnePerChannel	
Range	OnePerChannel	
	OnePerECU	
	OnePerCoreGroup	
Configuration class	**PreCompile:**	VariantPostBuild
	PreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	**OneSendSignalQueuePerCore**	
Label	Generate one SendSignal queue for each core	
Description	When true the Rte will generate a separate SendSignal queue for each core. This option is deprecated and replaced by 'SendSignalQueueStrategy'.	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	false	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	**SendSignalQueueStrategy**
Label	SendSignalQueueStrategy
Description	Defines the strategy how the Rte will generate a separate SendSignal queue. Rte will generate separate SendSignal queues in case that the sending SWC is on the non-BSW partition, with the following strategies: ► **Global**: current and default strategy which is only one separate sendSignal queue. ► **OnePerCore**: one queue per core. ► **OnePerPartition**: one queue per partition. ► **OnePerMainFunction**: one queue per Partition and per associated Com MainFunctionTx. This option will enable the transmission preparation (ComPreparationNotification) feature.
Multiplicity	1..1
Type	ENUMERATION
Range	Global

	OnePerCore	
	OnePerPartition	
	OnePerMainFunction	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	**OneScheduleTablePerPartition**	
Label	Generate one schedule table per partition	
Description	When true the Rte will generate one shared schedule table for Rte and Bsw timing events for each partition.	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	false	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	**InterruptBlockingFunction**
Label	Interrupt Blocking Function
Description	Defines the functions which shall be used to block interrupts Possible options are: ► SuspendResumeAllInterrupts (default): uses the standard Autosar Os functions SuspendAllInterrupts and ResumeAllInterrupts ► DisableEnableAllInterrupts: uses the standard Autosar Os functions DisableAllInterrupts and EnableAllInterrupts ► TS_IntDisableEnable: EB-specific functions from the EB Base module. ► Rte_UserDefinedIntLockUnlock: user-defined functions/macros which have to be declared in a header file called Rte_UserDefinedIntLock.h
Multiplicity	1..1
Type	ENUMERATION
Default value	SuspendResumeAllInterrupts
Range	SuspendResumeAllInterrupts
	DisableEnableAllInterrupts

	TS_IntDisableEnable	
	Rte_UserDefinedIntLockUnlock	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	**OsCounterRef**	
Label	Os Counter	
Description	A reference to an OsCounter (HW or SW) used by the Rte. The Rte Generator requires the counter to setup alarms and schedule tables.	
Multiplicity	1..1	
Type	REFERENCE	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	**OSEKCompatibilityMode**	
Label	OSEK OS Compatibility Mode	
Description	Enables or disables the OSEK OS compatibility mode. In this mode, the Rte can be used with an OSEK OS compliant operating system.	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	false	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	**OsScheduleTableActivationMechanism**
Label	Os Schedule Table Activation Mechanism
Description	Switch between the three available mechanisms for activating the Rte schedule table. ► **RELATIVE**: Starts the Rte schedule table with a relative offset. ► **ABSOLUTE**: Starts the Rte schedule table with an absolute offset.

	▶ **NEXT**: Merges the selected user schedule table with the Rte schedule table and starts it. after the user schedule table has been processed.	
Multiplicity	1..1	
Type	ENUMERATION	
Default value	RELATIVE	
Range	RELATIVE	
	ABSOLUTE	
	NEXT	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	OsScheduleTableMaxExpiryPoints	
Label	Max number of expiry points	
Description	The maximum number of Os schedule table expiry points. The Rte shows a warning if the actual number of expiry points exceeds this value.	
Multiplicity	1..1	
Type	INTEGER	
Default value	5000	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	OsScheduleTableOffset	
Label	Os Schedule Table Alarm Offset (s)	
Description	The offset in seconds used for the schedule table start, in case the schedule table activation mechanism is set to **RELATIVE** or **ABSOLUTE**.	
Multiplicity	1..1	
Type	FLOAT	
Default value	0.01	
Configuration class	**PreCompile:**	VariantPostBuild
	PreCompile:	VariantPreCompile

Origin	Elektrobit Automotive GmbH

Parameter Name	OsUserScheduleTableRef	
Label	Os User Schedule Table	
Description	A reference to the OsScheduleTable representing the user schedule table. The schedule table selected here will be merged with the Rte default schedule table if **NEXT** is selected as schedule table activation mechanism.	
Multiplicity	1..1	
Type	REFERENCE	
Configuration class	**PreCompile:**	VariantPostBuild
	PreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	RteCalibrationSupport	
Label	Rte Calibration Support	
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..1	
Type	ENUMERATION	
Default value	NONE	
Range	DOUBLE_POINTERED	
	INITIALIZED_RAM	
	NONE	
	SINGLE_POINTERED	
Configuration class	**VariantPreCompile:**	VariantPreCompile
	VariantPostBuild:	VariantPostBuild
Origin	AUTOSAR_ECUC	

Parameter Name	RteCodeVendorId
Label	Code Vendor Id
Description	*The functionality related to this parameter is not supported by the current implementation.* Holds the vendor ID of the generated Rte code.

Multiplicity	1..1	
Type	INTEGER	
Default value	0	
Range	<=65535	
	>=0	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	**RteDataModelExport**	
Label	Rte Data Model Export	
Description	If set to true, then the Rte exports the Rte data model during code generation. The exported data model is located in the doc subfolder of the specified generation path. The Rte data model can be used by other modules as input.	
Multiplicity	0..1	
Type	BOOLEAN	
Default value	false	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	**RteDevErrorDetect**	
Label	Enable Development Error Detection	
Description	The Rte shall log development errors to the Det module.	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	false	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	**HumanReadableBufferNames**
Label	Generate Human Readable Buffer Names

Description	The Rte shall use human readable names for the generated global variables.	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	false	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	**RteDevErrorDetectUninit**	
Label	Enable Development Error Detection Uninit	
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..1	
Type	BOOLEAN	
Default value	false	
Configuration class	**PreCompile:**	VariantPostBuild
	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	**RteGenerationMode**	
Label	Rte Generation Mode	
Description	*The functionality related to this parameter is not supported by the current implementation.* Switch between the two available generation modes of the Rte Generator.	
Multiplicity	1..1	
Type	ENUMERATION	
Default value	COMPATIBILITY_MODE	
Range	COMPATIBILITY_MODE	
	VENDOR_MODE	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	**RteGeneratorOutput**	

Label	Rte Generation Output
Description	The Rte Generator supports three different generation modes with regard to the Generation Phases specified by AUTOSAR: ► RTE_ONLY: Only the Rte is generated (RTE Generation Phase) ► BSW_SCHEDULER_ONLY: Only the BSW scheduler is generated (Basic Software Scheduler Generation Phase) ► FULL: Both the Rte and the BSW scheduler are generated (EB-specific enhancement)
Multiplicity	1..1
Type	ENUMERATION
Default value	RTE_ONLY
Range	RTE_ONLY BSW_SCHEDULER_ONLY FULL

Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	**RtelocInteractionReturnValue**
Description	*The functionality related to this parameter is not supported by the current implementation.* Defines whether the return value of RTE APIs is based on RTE-IOC interaction or RTE-COM interaction.
Multiplicity	1..1
Type	ENUMERATION
Default value	RTE_IOC
Range	RTE_COM RTE_IOC

Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	**RteMeasurementSupport**

Label	Rte Measurement Support
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..1
Type	BOOLEAN
Default value	false
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC

Parameter Name	**RteOptimizationMode**
Label	Rte Optimization Mode
Description	Switch between the two available optimization modes of the Rte Generator.
	MEMORY optimization will use TS_MemBZero for complex variables that are zero-initialized and will not generate init variables for them.
Multiplicity	1..1
Type	ENUMERATION
Default value	RUNTIME
Range	MEMORY
	RUNTIME
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC

Parameter Name	**RteToolChainSignificantCharacters**
Label	Toolchain Significant Characters
Description	*The functionality related to this parameter is not supported by the current implementation.*
	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	INTEGER
Default value	31

Range	<=65535	
	>=0	
Configuration class	**PreCompile:**	VariantPostBuild
	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	RteValueRangeCheckEnabled	
Label	Value Range Check	
Description	*The functionality related to this parameter is not supported by the current implementation.* If set to true the Rte Generator shall enable the value range checking for the specified VariableDataPrototypes.	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	false	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	RteVfbTraceClientPrefix	
Label	VFB Trace Client Prefix	
Description	*The functionality related to this parameter is not supported by the current implementation.* 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..n	
Type	LINKER-SYMBOL	
Configuration class	**PreCompile:**	VariantPostBuild
	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	RteVfbTraceEnabled

Label	VFB Tracing
Description	The Rte Generator shall globally enable VFB tracing when RteVfbTrace is set to "true".
Multiplicity	1..1
Type	BOOLEAN
Default value	false
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC

Parameter Name	**RteVfbTraceFunction**	
Label	VFB Trace Function	
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..n	
Type	FUNCTION-NAME	
Configuration class	**PreCompile:**	VariantPostBuild
	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	**SingleScheduleTablePartitionRef**	
Label	Generate single Rte schedule table for all partitions in Os Application	
Description	A reference to an Os Application that owns the single schedule table. In the generated Rte code, this is the partition which will start the schedule table for all partitions by calling Rte_Start(). Please note a call to Rte_Stop()/Rte_RestartPartition() on a partition that does not own the schedule table has no effect on the lifetime of the schedule table. A schedule table can only be stopped by the partition that owns the schedule table.	
Multiplicity	0..1	
Type	REFERENCE	
Configuration class	**PreCompile:**	VariantPostBuild
	PreCompile:	VariantPreCompile

Origin	Elektrobit Automotive GmbH

Parameter Name	UnconnectedRequirePorts	
Label	Handling of unconnected Require Ports	
Description	Defines how to handle unconnected require ports. Possible options are: report a warning (default), report an error (AUTOSAR-compliant) or ignore unconnected require ports.	
Multiplicity	1..1	
Type	ENUMERATION	
Default value	Warning	
Range	Warning	
	Error	
	Ignore	
Configuration class	VariantPostBuild:	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	GenerateRteFreedomFromInterferenceReviewInstructions	
Label	Generate Freedom From Interference Review Instructions	
Description	If set to true, the Rte generator adds review instructions to the generated C code to easily identify where memory write access occurs. These instructions can be used to check that the marked expression does not overwrite memory of other software units (freedom from interference). For expressions with memory write access that do not harm (e.g. a simple assignment), a justification is added. In addition, an html report is generated in the generated/doc/rte_review_report folder which uses a colored markup to easily recognize the start and end of the commented expressions.	
Multiplicity	0..1	
Type	BOOLEAN	
Default value	false	
Configuration class	PreCompile:	VariantPostBuild
	PreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	ExecuteTransformersDespiteLocalDataUnqueued
Label	Execute the transformer chain despite local data (unqueued)

Description	If set to true, the Rte will always execute the transformer chain for an unqueued required sender-receiver data element if ExecuteDespiteDataUnavailability is set to true. In case that input data is not available (e.g. before first COM reception), the input to all the transformers in the chain is NULL_PTR with a dataLength equal to 0. If set to false, the transformer chain is not executed before the first COM reception or if data from a local sender is received. Set it to false (default value) if backwards-compatibility to previous Rte versions is required.	
Multiplicity	0..1	
Type	BOOLEAN	
Default value	false	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	ExecuteTransformersDespiteLocalDataQueued	
Label	Execute the transformer chain despite local data (queued)	
Description	If set to true, the Rte will always execute the transformer chain for a queued required sender-receiver data element if ExecuteDespiteDataUnavailability is set to true. In case that input data is not available (e.g. in case of an empty queue), the input to all the transformers in the chain is NULL_PTR with a dataLength equal to 0. If set to false, the transformer chain is not executed if data from a local sender is received. Set it to false (default value) if backwards-compatibility to previous Rte versions is required.	
Multiplicity	0..1	
Type	BOOLEAN	
Default value	false	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

5.2.1.15. ComTaskConfiguration

Parameters included	
Parameter name	**Multiplicity**
ComTaskOsTaskRef	0..1

Parameters included	
ComTaskSendSignalQueueLength	1..1
ComTaskSendSignalGroupQueueLength	1..1

Parameter Name	ComTaskOsTaskRef	
Label	Com Os Task	
Description	A reference to an OsTask which is used by the Rte to make calls to the Com/Ld-Com for inter-partition communication.	
Multiplicity	0..1	
Type	REFERENCE	
Configuration class	PreCompile:	VariantPostBuild
	PreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	ComTaskSendSignalQueueLength	
Label	Send Signal Queue Length used for Com/LdCom	
Description	The length of the send signal request queue of the task used for Com/LdCom.	
Multiplicity	1..1	
Type	INTEGER	
Default value	1	
Range	<=65535	
	>=1	
Configuration class	PreCompile:	VariantPostBuild
	PreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	ComTaskSendSignalGroupQueueLength
Label	Send Signal Group Queue Length used for Com
Description	The length of the send signal group request queue of the Com task. (considered only for the Com signal groups)
Multiplicity	1..1
Type	INTEGER
Default value	1
Range	<=65535

	>=1	
Configuration class	PreCompile:	VariantPostBuild
	PreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

5.2.1.16. BswConfiguration

Parameters included	
Parameter name	Multiplicity
BswOsApplicationRef	0..1
BswOsTaskPeriod	0..1
BswOsTaskRef	0..1
BswSendSignalQueueLength	1..1
BswSendSignalGroupQueueLength	1..1

Parameter Name	BswOsApplicationRef	
Label	BSW Os Application	
Description	A reference to an OsApplication to which the BSW belongs.	
Multiplicity	0..1	
Type	REFERENCE	
Configuration class	PreCompile:	VariantPostBuild
	PreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	BswOsTaskPeriod	
Label	Period of Bsw Os Task	
Description	The period of the BSW task in seconds.	
Multiplicity	0..1	
Type	FLOAT	
Default value	0.1	
Range	>0	
Configuration class	PreCompile:	VariantPostBuild

	PreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	**BswOsTaskRef**	
Label	BSW Os Task	
Description	A reference to an OsTask which is used by the Rte to make calls to the basic software (e.g. Com) for inter-core communication. The task must belong to the BSW Os application.	
Multiplicity	0..1	
Type	REFERENCE	
Configuration class	**PreCompile:**	VariantPostBuild
	PreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	**BswSendSignalQueueLength**	
Label	BSW Send Signal Queue Length	
Description	The length of the send signal request queue of the BSW task.	
Multiplicity	1..1	
Type	INTEGER	
Default value	1	
Range	<=65535	
	>=1	
Configuration class	**PreCompile:**	VariantPostBuild
	PreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	**BswSendSignalGroupQueueLength**	
Label	BSW Send Signal Group Queue Length	
Description	The length of the send signal group request queue of the BSW task.	
Multiplicity	1..1	
Type	INTEGER	
Default value	1	
Range	<=65535	
	>=1	

Configuration class	PreCompile:	VariantPostBuild
	PreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

5.2.1.17. OsCounterAssignments

Containers included		
Container name	**Multiplicity**	**Description**
OsCounterAssignment	0..n	**Label:** Os Counter Assignment Contains a counter configuration for each partition. For each partition, a separate Os counter configuration can be specified here. If no special counter is configured for a partition, the default counter configuration is taken. The default counter configuration is located on the General tab.

5.2.1.18. OsCounterAssignment

Parameters included	
Parameter name	**Multiplicity**
OsApplicationRef	1..1
OsCounterRef	1..1

Parameter Name	OsApplicationRef	
Label	Os Application	
Description	A reference to an OsApplication for this counter configuration.	
Multiplicity	1..1	
Type	REFERENCE	
Configuration class	VariantPostBuild:	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	OsCounterRef

Label	Os Counter
Description	A reference to an OsCounter (HW or SW) used by the Rte for this partition. The Rte generator requires the counter to setup alarms and schedule tables.
Multiplicity	1..1
Type	REFERENCE
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH

5.2.1.19. CooperativeTasks

Containers included		
Container name	**Multiplicity**	**Description**
CooperativeTask	0..n	**Label:** Task in Group A list of OsTasks that cannot interrupt each other regardless of their configured priorities, schedule, application assignment, and core assignment. This list will be taken into account by the Rte generator to optimize locks and to reduce the number of implicit buffers.

5.2.1.20. CooperativeTask

Parameters included	
Parameter name	**Multiplicity**
TaskRef	0..1

Parameter Name	TaskRef
Label	Task in Group
Description	Reference to an OsTask that is part of a cooperative task group. One OsTask can be contained in multiple cooperative task groups.
Multiplicity	0..1
Type	REFERENCE
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile

Origin	Elektrobit Automotive GmbH

5.2.1.21. TaskChain

Containers included		
Container name	**Multiplicity**	**Description**
ChainedTask	0..n	**Label:** Task Chain Specifies a subsequent OsTask for a given OsTask. The subsequent OsTask will be activated via ChainTask() if the given OsTask has finished. This is only possible for tasks that are basic tasks and where only timing events are mapped to. The Rte will not configure any expiry points in the schedule table if the subsequent OsTask is chained. The first task in the chain must have a cycle times that is the greatest common divisor of all task cycle times within the chain. The cycle time is calculated by the timing events that are mapped to these tasks.

5.2.1.22. ChainedTask

Parameters included	
Parameter name	**Multiplicity**
PredecessorTaskRef	0..1
SuccessorTaskRef	0..1

Parameter Name	PredecessorTaskRef	
Label	Predecessor Task	
Description	Reference to the OsTask that shall run before the Successor OsTask.	
Multiplicity	0..1	
Type	REFERENCE	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	SuccessorTaskRef

Label	Successor Task
Description	Reference to the OsTask that shall run after the Predecessor OsTask.
Multiplicity	0..1
Type	REFERENCE
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH

5.2.1.23. RteImplicitCommunication

Containers included		
Container name	**Multiplicity**	**Description**
RteSoftwareComponentInstanceRef	1..n	**Label:** Software Component Instance Reference to a SwComponentPrototype. This denotes the instances of the VariableAccess belonging to the RteImplicitCommunication.

Parameters included	
Parameter name	**Multiplicity**
RteCoherentAccess	1..1
RteImmediateBufferUpdate	1..1
RteVariableReadAccessRef	0..n
RteVariableWriteAccessRef	0..n

Parameter Name	RteCoherentAccess
Label	Coherent Access
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..1	
Type	BOOLEAN	
Default value	false	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	**RteImmediateBufferUpdate**	
Label	Immediate Buffer Update	
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..1	
Type	BOOLEAN	
Default value	false	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	**RteVariableReadAccessRef**	
Label	Variable Read Access	
Description	Reference to the VariableAccess in the dataReadAccess role.	
Multiplicity	0..n	
Type	FOREIGN-REFERENCE	
Configuration class	**PreCompile:**	VariantPostBuild
	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	**RteVariableWriteAccessRef**	
Label	Variable Write Access	
Description	Reference to the VariableAccess in the dataWriteAccess role.	
Multiplicity	0..n	
Type	FOREIGN-REFERENCE	
Configuration class	**PreCompile:**	VariantPostBuild

	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

5.2.1.24. RteSoftwareComponentInstanceRef

Parameters included	
Parameter name	**Multiplicity**
TARGET	1..1
CONTEXT	0..n

Parameter Name	TARGET
Label	Target
Multiplicity	1..1
Type	REFERENCE
Origin	AUTOSAR_ECUC

Parameter Name	CONTEXT
Label	Context
Multiplicity	0..n
Type	REFERENCE
Range	ROOT-SW-COMPOSITION-PROTOTYPE
Origin	AUTOSAR_ECUC

5.2.1.25. RteInitializationBehavior

Parameters included	
Parameter name	**Multiplicity**
RteInitializationStrategy	1..1
RteSectionInitializationPolicy	1..n

Parameter Name	RteInitializationStrategy
Description	Definition of the initialization strategy applicable for the SectionInitializationPolicys selected by RteSectionInitializationPolicy.
Multiplicity	1..1

Type	ENUMERATION
Default value	RTE_INITIALIZATION_STRATEGY_AT_RTE_START_AND_PARTITION_RES-TART
Range	RTE_INITIALIZATION_STRATEGY_AT_DATA_DECLARATION
	RTE_INITIALIZATION_STRATEGY_AT_DATA_DECLARATION_AND_-PARTITION_RESTART
	RTE_INITIALIZATION_STRATEGY_AT_RTE_START_AND_PARTITION_RES-TART
	RTE_INITIALIZATION_STRATEGY_NONE

Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	**RteSectionInitializationPolicy**	
Label	Section Initialization Policy	
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..n	
Type	STRING	
Configuration class	**PreCompile:**	VariantPostBuild
	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

5.2.1.26. RteInitializationRunnableBatch

Parameters included	
Parameter name	**Multiplicity**
RteMappedToTaskRef	0..1

Parameter Name	RteMappedToTaskRef	
Label	Mapped to Task	
Description	Reference to the OsTask the RteInitializationBehavior will be called in its context. For the mode switch events activated on-entry and referencing the initial mode it is mandatory, for the init event is optional (as it can be defined in the EventToTaskMapping)	
Multiplicity	0..1	
Type	REFERENCE	
Configuration class	**PreCompile:**	VariantPostBuild
	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

5.2.1.27. RteOsInteraction

Containers included		
Container name	**Multiplicity**	**Description**
RteModeToSched-uleTableMapping	0..n	**Label:** Mode to Schedule Table Mapping *The functionality related to this parameter is not supported by the current implementation.* Provides configuration input in which modes of a ModeDe-clarionGroupPrototype of a mode manager a OsSched-uleTable shall be active. The mode manager is either specified as a SwComponentPrototype (RteModeSchtblMapSwc) or as a BSW module (RteModeSchtblMapBsw).
RteUsedOsActivation	0..n	**Label:** Used Os Activation *The functionality related to this parameter is not supported by the current implementation.* Attributes used in the activation of OsTasks and Runnable Entities.

5.2.1.28. RteModeToScheduleTableMapping

Containers included		
Container name	**Multiplicity**	**Description**
RteModeSchtblMapBsw	0..1	**Label:** Mode Schedule Table Mapping BSW *The functionality related to this parameter is not supported by the current implementation.* Specifies an instance of a ModeDeclarationGroupPrototype of a BSW module.
RteModeSchtblMapSwc	0..1	**Label:** Mode Schedule Table Mapping SWC *The functionality related to this parameter is not supported by the current implementation.* Specifies an instance of a ModeDeclarationGroupPrototype of a SwComponentPrototype.

Parameters included	
Parameter name	**Multiplicity**
RteModeSchtblMapModeDeclarationRef	1..n
RteModeScheduleTableRef	1..1

Parameter Name	RteModeSchtblMapModeDeclarationRef	
Label	Mode Declaration	
Description	*The functionality related to this parameter is not supported by the current implementation.* Reference to the ModeDeclarations.	
Multiplicity	1..n	
Type	FOREIGN-REFERENCE	
Configuration class	**PreCompile:**	VariantPostBuild
	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	RteModeScheduleTableRef
Label	Schedule Table
Description	*The functionality related to this parameter is not supported by the current implementation.*

	Reference to the OsScheduleTable which shall be active in the specified Rte-ModeSchblMapModeDeclarationRefs.
Multiplicity	1..1
Type	REFERENCE
Configuration class	**VariantPostBuild:** VariantPostBuild
	VariantPreCompile: VariantPreCompile
Origin	AUTOSAR_ECUC

5.2.1.29. RteModeSchtblMapBsw

Parameters included	
Parameter name	**Multiplicity**
RteModeSchtblMapBswProvidedModeGroupRef	1..1
RteModeSchtblMapBswInstanceRef	1..1

Parameter Name	**RteModeSchtblMapBswProvidedModeGroupRef**
Label	Provided Mode Group
Description	*The functionality related to this parameter is not supported by the current implementation.* Reference to an instance of a ModeDeclarationGroupPrototype of a BSW module.
Multiplicity	1..1
Type	FOREIGN-REFERENCE
Configuration class	**VariantPostBuild:** VariantPostBuild
	VariantPreCompile: VariantPreCompile
Origin	AUTOSAR_ECUC

Parameter Name	**RteModeSchtblMapBswInstanceRef**
Label	BSW Module
Description	*The functionality related to this parameter is not supported by the current implementation.* Reference to an instance specification of a BSW module.
Multiplicity	1..1

Type	REFERENCE	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

5.2.1.30. RteModeSchtblMapSwc

Parameters included	
Parameter name	**Multiplicity**
RteModeSchtblMapSwcPortRef	1..1
RteModeSchtblMapSwcInstanceRef	1..1

Parameter Name	**RteModeSchtblMapSwcPortRef**	
Label	SWC Port	
Description	*The functionality related to this parameter is not supported by the current implementation.* Reference to the PPortPrototype of a SwComponentPrototype.	
Multiplicity	1..1	
Type	FOREIGN-REFERENCE	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	**RteModeSchtblMapSwcInstanceRef**	
Label	SWC Instance	
Description	*The functionality related to this parameter is not supported by the current implementation.* Reference to an instance specification of a SwComponentPrototype.	
Multiplicity	1..1	
Type	REFERENCE	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile

Origin	AUTOSAR_ECUC

5.2.1.31. RteUsedOsActivation

Parameters included	
Parameter name	**Multiplicity**
RteExpectedActivationOffset	1..1
RteExpectedTickDuration	1..1
RteActivationOsAlarmRef	0..1
RteActivationOsSchTblRef	0..1
RteActivationOsTaskRef	0..1

Parameter Name	RteExpectedActivationOffset	
Label	Expected Activation Offset (s)	
Description	*The functionality related to this parameter is not supported by the current implementation.* 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..1	
Type	FLOAT	
Default value	0.0	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	RteExpectedTickDuration
Label	Expected Tick Duration (s)
Description	*The functionality related to this parameter is not supported by the current implementation.* The expected tick duration in seconds which shall be configured to drive the Os-ScheduleTables 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..1	
Type	FLOAT	
Default value	0.0	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	RteActivationOsAlarmRef	
Label	Os Alarm	
Description	*The functionality related to this parameter is not supported by the current implementation.* Reference to an OsAlarm.	
Multiplicity	0..1	
Type	REFERENCE	
Configuration class	**PreCompile:**	VariantPostBuild
	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	RteActivationOsSchTblRef	
Label	Os Schedule Table	
Description	*The functionality related to this parameter is not supported by the current implementation.* Reference to an OsScheduleTable.	
Multiplicity	0..1	
Type	REFERENCE	
Configuration class	**PreCompile:**	VariantPostBuild
	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	RteActivationOsTaskRef	
Label	Os Task	
Description	*The functionality related to this parameter is not supported by the current implementation.*	

	Reference to an OsTask.	
Multiplicity	0..1	
Type	REFERENCE	
Configuration class	**PreCompile:**	VariantPostBuild
	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

5.2.1.32. RtePostBuildVariantConfiguration

Parameters included	
Parameter name	**Multiplicity**
[RtePostBuildUsedPredefinedVariant](#)	1..n

Parameter Name	**RtePostBuildUsedPredefinedVariant**	
Label	Used Predefined Variant	
Description	*The functionality related to this parameter is not supported by the current implementation.* Reference to the PredefinedVariant element which defines the values for PostBuildVariationCriterion elements.	
Multiplicity	1..n	
Type	FOREIGN-REFERENCE	
Configuration class	**PreCompile:**	VariantPostBuild
	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

5.2.1.33. RteRips

Parameters included	
Parameter name	**Multiplicity**
[RteRipsSupport](#)	1..1
[RteRipsPluginConfigurationRef](#)	0..n

Parameter Name	**RteRipsSupport**

Label	Rips Support
Description	Globally enables or disables the support for Rte Implementation Plug-Ins (RIPS).
Multiplicity	1..1
Type	ENUMERATION
Default value	RTE_RIPS_OFF
Range	RTE_RIPS_OFF
	RTE_RIPS_ON
Configuration class	**VariantPostBuild:** VariantPostBuild
	VariantPreCompile: VariantPreCompile
Origin	AUTOSAR_ECUC

Parameter Name	**RteRipsPluginConfigurationRef**
Label	Reference To Rips Plugin
Description	Reference to the configuration container of the RTE Implementation Plug-in holding the RIPS relevant settings.
Multiplicity	0..n
Type	URI-REFERENCE
Configuration class	**VariantPostBuild:** VariantPostBuild
	VariantPreCompile: VariantPreCompile
	VariantPostBuild: VariantPostBuild
	VariantPreCompile: VariantPreCompile
Origin	AUTOSAR_ECUC

5.2.1.34. RteSwComponentInstance

Containers included		
Container name	**Multiplicity**	**Description**
DataMappings	0..1	**Label:** Mapping of Data Elements Configuration of communication mapping
RteEventToTaskMapping	0..n	**Label:** Event to Task Mapping Maps a RunnableEntity onto one OsTask based on the activating RTEEvent. Even if a RunnableEntity shall be executed via a direct function call this RteEventToTaskMapping shall

Containers included

		be specified, but no RteMappedToTask and RtePositionIn-Task elements given.
RteEventToIsrMapping	0..n	**Label:** Event to Task Mapping Maps a RunnableEntity onto one ISR based on the activating ExternalTriggerOccurredEvent or TimingEvent.
RteExclusiveAreaImplementation	0..n	**Label:** Exclusive Area Implementation Specifies the implementation to be used for the data consistency of this ExclusiveArea.
RteExternalTriggerConfig	0..n	Defines the configuration of External Trigger Event Communication for Software Components.
RteInternalTriggerConfig	0..n	Defines the configuration of Inter Runnable Triggering for Software Components.
RteNvRamAllocation	0..n	**Label:** NVRAM Allocation Specifies the relationship between the AtomicSwComponentType's NVRAMMapping / NVRAM needs and the NvM module configuration.

Parameters included

Parameter name	Multiplicity
MappedToOsApplicationRef	0..1
RteSoftwareComponentInstanceRef	0..1

Parameter Name	MappedToOsApplicationRef	
Label	Mapped to Os Application	
Description	A reference to an Os Application to which the software component instance shall be mapped. This parameter must be configured if partioning is used.	
Multiplicity	0..1	
Type	REFERENCE	
Configuration class	**PreCompile:**	VariantPostBuild
	PreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	RteSoftwareComponentInstanceRef
Label	Software Component Instance
Description	Reference to a SwComponentPrototype.
Multiplicity	0..1

Type	FOREIGN-REFERENCE	
Configuration class	PreCompile:	VariantPostBuild
	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

5.2.1.35. DataMappings

Containers included		
Container name	**Multiplicity**	**Description**
DataSRMapping	0..n	**Label:** Data Mappings for Sender/Receiver Communication In addition to configure the mapping of variable data prototypes to system signals and signal groups for sender/receiver communication in the system description, this vendor-specific parameters also support to specify the data mapping in the ECU configuration.

5.2.1.36. DataSRMapping

Containers included		
Container name	**Multiplicity**	**Description**
VariableDataPrototypeInstanceRef	1..1	**Label:** Variable Data Prototype Reference to an instance of a VariableDataPrototype
DataSRKindMapping	1..1	**Label:** Mapping to Com Signals/Signal Groups A primitive type and an array of bytes must be mapped to a Com signal. A complex type must be mapped to a Com signal group, each element of the complex type must be mapped to a signal of the signal group.

5.2.1.37. VariableDataPrototypeInstanceRef

Parameters included	
Parameter name	**Multiplicity**
TARGET	1..1
CONTEXT	0..n

Parameter Name	TARGET	
Label	Variable Data Prototype	
Description	Reference to the VariableDataPrototype which shall be mapped.	
Multiplicity	1..1	
Type	REFERENCE	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	CONTEXT	
Label	Reference to the associated Port	
Description	Reference to the PortPrototype to which the VariableDataPrototype belongs.	
Multiplicity	0..n	
Type	REFERENCE	
Range	PortPrototype	
Configuration class	**PreCompile:**	VariantPostBuild
	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

5.2.1.38. DataSRKindMapping

Containers included		
Container name	**Multiplicity**	**Description**
DataSRPrimitive	1..1	**Label:** Primitive Type Mapping Mapping for primitive types.
DataSRComplex	1..1	**Label:** Complex Type Mapping Mapping for complex types.

5.2.1.39. DataSRPrimitive

Parameters included	
Parameter name	**Multiplicity**

Parameters included	
SignalRef	1..1

Parameter Name	SignalRef	
Label	Com Signal	
Description	A reference to a Com signal for this primitive type.	
Multiplicity	1..1	
Type	SYMBOLIC-NAME-REFERENCE	
Configuration class	VariantPostBuild:	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

5.2.1.40. DataSRComplex

Parameters included	
Parameter name	Multiplicity
SignalGroupRef	1..1
SignalRef	0..n

Parameter Name	SignalGroupRef	
Label	Signal Group	
Description	A reference to a Com signal group for the complex type.	
Multiplicity	1..1	
Type	SYMBOLIC-NAME-REFERENCE	
Configuration class	VariantPostBuild:	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	SignalRef
Label	Com Group Signals
Description	SignalRef is a list of all signals of this signal group. Each element of a complex type must be mapped to an own signal. The linking between signals and elements of the complex types is specified by the order of the signals defined in this container according to the following rule: The Rte Generator shall match signals

	in the input with elements in the complex type by performing a depth first traversal of the data structure and using the signals in the order that they are given in the input.	
Multiplicity	0..n	
Type	SYMBOLIC-NAME-REFERENCE	
Configuration class	**PreCompile:**	VariantPostBuild
	PreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

5.2.1.41. RteEventToTaskMapping

Parameters included	
Parameter name	**Multiplicity**
RteActivationOffset	0..1
RtePeriod	0..1
RteImmediateRestart	1..1
RteOsSchedulePoint	0..1
RtePositionInTask	0..1
RteServerQueueLength	0..1
RteEventRef	1..1
RteMappedToTaskRef	0..1
RteAllowInterPartitionDirectCall	1..1
RteRipsFillRoutineRef	0..1
RteRipsFlushRoutineRef	0..1
RteRipsInvocationHandlerRef	0..1
RteUsedInitFnc	0..1
RteUsedOsAlarmRef	0..1
RteUsedOsEventRef	0..1
RteUsedOsSchTblExpiryPointRef	0..1
RteVirtuallyMappedToTaskRef	0..1

Parameter Name	**RteActivationOffset**
Label	Activation Offset (s)

Description	Activation offset in seconds.	
Multiplicity	0..1	
Type	FLOAT	
Default value	0.0	
Configuration class	**PreCompile:**	VariantPostBuild
	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	**RtePeriod**	
Label	Period (s) for OperationInvokedEvents	
Description	Period in seconds for OperationInvokedEvents. Enabling this parameter disables the event based triggering of any runnable assigned to this event.	
Multiplicity	0..1	
Type	FLOAT	
Default value	0.1	
Range	>0	
Configuration class	**PreCompile:**	VariantPostBuild
	PreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	**RteImmediateRestart**	
Label	Immediate Restart	
Description	*The functionality related to this parameter is not supported by the current implementation.* When RteImmediateRestart is set to true the RunnableEntity 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..1	
Type	BOOLEAN	
Default value	false	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile

Origin	AUTOSAR_ECUC

Parameter Name	RteOsSchedulePoint	
Label	Schedule Point	
Description	*The functionality related to this parameter is not supported by the current implementation.* 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	ENUMERATION	
Range	CONDITIONAL	
	NONE	
	UNCONDITIONAL	
Configuration class	**PreCompile:**	VariantPostBuild
	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	RtePositionInTask	
Label	Position in Task	
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.	
Multiplicity	0..1	
Type	INTEGER	
Default value	0	
Range	<=65535	
	>=0	
Configuration class	**PreCompile:**	VariantPostBuild
	PreCompile:	VariantPreCompile

Origin	AUTOSAR_ECUC

Parameter Name	RteServerQueueLength	
Label	Server Queue Length	
Description	Specifies the length of the queue for the server call serialization. This value overwrites the queueLength specified at the ServerComSpec.	
Multiplicity	0..1	
Type	INTEGER	
Default value	1	
Range	<=65535	
	>=0	
Configuration class	**PreCompile:**	VariantPostBuild
	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	RteEventRef	
Label	RTE Event	
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..1	
Type	FOREIGN-REFERENCE	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	RteMappedToTaskRef	
Label	Mapped to Task	
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.	
Multiplicity	0..1	
Type	REFERENCE	
Configuration class	**PreCompile:**	VariantPostBuild

PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC

Parameter Name	RteAllowInterPartitionDirectCall	
Label	Allow interpartition direct call	
Description	When RteMappedToTaskRef is empty and RteBswAllowInterPartitionDirectCall is set to true the inter-partition direct-call is allowed.	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	false	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	RteRipsFillRoutineRef	
Label	Rips Fill Routine	
Description	Reference to a Buffer-Fill Routine implemented by an RTE Implementation Plug-In.	
Multiplicity	0..1	
Type	URI-REFERENCE	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
	VariantPostBuild:	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	RteRipsFlushRoutineRef	
Label	Rips Flush Routine	
Description	Reference to a Buffer-Flush Routine implemented by an RTE Implementation Plug-In.	
Multiplicity	0..1	
Type	URI-REFERENCE	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile

	VariantPostBuild:	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	RteRipsInvocationHandlerRef	
Description	Reference to a Buffer-Fill Routine implemented by an RTE Implementation Plug-In.	
Multiplicity	0..1	
Type	CHOICE-REFERENCE	
Configuration class	VariantPostBuild:	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
	VariantPostBuild:	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	RteUsedInitFnc	
Description	The RunnableEntity is executed during initialization in the context of the Rte_-Init_<InitContainer> function.	
Multiplicity	0..1	
Type	REFERENCE	
Configuration class	PreCompile:	VariantPostBuild
	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	RteUsedOsAlarmRef	
Label	Used Os Alarm	
Description	*The functionality related to this parameter is not supported by the current implementation.* If an OsAlarm is used to activate the OsTask this RteEvent is mapped to it shall be referenced here.	
Multiplicity	0..1	
Type	REFERENCE	
Configuration class	PreCompile:	VariantPostBuild
	PreCompile:	VariantPreCompile

Origin	AUTOSAR_ECUC

Parameter Name	RteUsedOsEventRef	
Label	Used Os Event	
Description	*The functionality related to this parameter is not supported by the current implementation.* If an OsEvent is used to activate the OsTask this RteEvent is mapped to it shall be referenced here.	
Multiplicity	0..1	
Type	REFERENCE	
Configuration class	**PreCompile:**	VariantPostBuild
	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	RteUsedOsSchTblExpiryPointRef	
Label	Used Schedule Table Expiry Point	
Description	*The functionality related to this parameter is not supported by the current implementation.* If an OsScheduleTableExpiryPoint is used to activate the OsTask this RteEvent is mapped to it shall be referenced here.	
Multiplicity	0..1	
Type	REFERENCE	
Configuration class	**PreCompile:**	VariantPostBuild
	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	RteVirtuallyMappedToTaskRef
Label	Virtually Mapped to Task
Description	*The functionality related to this parameter is not supported by the current implementation.* 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	
Configuration class	**PreCompile:**	VariantPostBuild
	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

5.2.1.42. RteEventToIsrMapping

Parameters included	
Parameter name	**Multiplicity**
[RtePositionInIsr](#)	0..1
[RteEventRef](#)	1..1
[RteMappedToIsrRef](#)	0..1

Parameter Name	**RtePositionInIsr**	
Label	Position in ISR	
Description	Each RunnableEntity activation mapped to an ISR has a specific position within the ISR execution. For event driver activation this is the order of evaluation which actual RunnableEntity has to be executed.	
Multiplicity	0..1	
Type	INTEGER	
Default value	0	
Range	<=65535	
	>=0	
Configuration class	**PreCompile:**	VariantPostBuild
	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	**RteEventRef**
Label	RTE Event
Description	Reference to the description of the ExternalTriggerOccurredEvent or TimingEvent which is pointing to the RunnableEntity being mapped. This allows a fine grained mapping of RunnableEntites based on the activating RTEEvent.
Multiplicity	1..1
Type	FOREIGN-REFERENCE

Configuration class	VariantPostBuild:	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	RteMappedToIsrRef	
Label	Mapped to ISR	
Description	Reference to the ISR the RunnableEntity activated by the RteEventRef is mapped to.	
Multiplicity	0..1	
Type	REFERENCE	
Configuration class	PreCompile:	VariantPostBuild
	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

5.2.1.43. RteExclusiveAreaImplementation

Parameters included	
Parameter name	Multiplicity
RteExclusiveAreaImplMechanism	1..1
RteExclusiveAreaRef	1..1
ExclusiveAreaUserCalloutEnter	0..1
ExclusiveAreaUserCalloutExit	0..1
RteExclusiveAreaOsResourceRef	0..1

Parameter Name	RteExclusiveAreaImplMechanism
Label	Exclusive Area Implementation
Description	To be used implementation mechanism for the specified ExclusiveArea.
Multiplicity	1..1
Type	ENUMERATION
Range	ALL_INTERRUPT_BLOCKING
	COOPERATIVE_RUNNABLE_PLACEMENT
	OS_INTERRUPT_BLOCKING
	OS_RESOURCE

	EB_FAST_LOCK	
	NO_LOCK	
	USER_CALLOUT	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	**RteExclusiveAreaRef**	
Label	Exclusive Area	
Description	Reference to the ExclusiveArea.	
Multiplicity	1..1	
Type	FOREIGN-REFERENCE	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	**ExclusiveAreaUserCalloutEnter**	
Label	Exclusive Area UserCalloutEnter	
Description	The name of the function that is called by the Rte to enter the exclusive area if RteExclusiveAreaImplMechanism is set to USER_CALLOUT.	
Multiplicity	0..1	
Type	STRING	
Configuration class	**PreCompile:**	VariantPostBuild
	PreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	**ExclusiveAreaUserCalloutExit**	
Label	Exclusive Area UserCalloutExit	
Description	The name of the function that is called by the Rte to leave the exclusive area if RteExclusiveAreaImplMechanism is set to USER_CALLOUT.	
Multiplicity	0..1	
Type	STRING	
Configuration class	**PreCompile:**	VariantPostBuild
	PreCompile:	VariantPreCompile

Origin	Elektrobit Automotive GmbH

Parameter Name	RteExclusiveAreaOsResourceRef	
Label		
Description	*The functionality related to this parameter is not supported by the current implementation.* Optional reference to an OsResource in case RteExclusiveAreaImplMechanism is configured to OS_RESOURCE for this ExclusiveArea.	
Multiplicity	0..1	
Type	REFERENCE	
Configuration class	**PreCompile:**	VariantPostBuild
	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

5.2.1.44. RteExternalTriggerConfig

Containers included		
Container name	**Multiplicity**	**Description**
RteSwcTriggerSourceRef	1..1	Reference to a Trigger instance in the pPortPrototype of the related component instance.

Parameters included	
Parameter name	**Multiplicity**
RteTriggerSourceQueueLength	1..1

Parameter Name	RteTriggerSourceQueueLength	
Description	Length of trigger queue on the trigger source side.	
Multiplicity	1..1	
Type	INTEGER	
Default value	0	
Range	<=4294967295	
	>=0	
Configuration class	**VariantPostBuild:**	VariantPostBuild

	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

5.2.1.45. RteSwcTriggerSourceRef

Parameters included	
Parameter name	Multiplicity
TARGET	1..1
CONTEXT	0..n

Parameter Name	TARGET
Multiplicity	1..1
Type	REFERENCE
Origin	AUTOSAR_ECUC

Parameter Name	CONTEXT
Multiplicity	0..n
Type	REFERENCE
Range	P-PORT-PROTOTYPE
Origin	AUTOSAR_ECUC

5.2.1.46. RteInternalTriggerConfig

Parameters included	
Parameter name	Multiplicity
RteTriggerSourceQueueLength	1..1
RteSwcTriggerSourceRef	1..1

Parameter Name	RteTriggerSourceQueueLength
Description	Length of trigger queue on the trigger source side.
Multiplicity	1..1
Type	INTEGER
Default value	0

Range	<=4294967295	
	>=0	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	**RteSwcTriggerSourceRef**	
Description	Reference to an InternalTriggeringPoint of the related component instance.	
Multiplicity	1..1	
Type	FOREIGN-REFERENCE	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

5.2.1.47. RteNvRamAllocation

Parameters included	
Parameter name	**Multiplicity**
RteNvmRamBlockLocationSymbol	0..1
RteNvmRomBlockLocationSymbol	0..1
RteSwNvRamMappingRef	0..1
RteNvmBlockRef	1..1
RteSwNvBlockDescriptorRef	1..1

Parameter Name	**RteNvmRamBlockLocationSymbol**	
Label	RAM Block Location Symbol	
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	LINKER-SYMBOL	
Configuration class	**PreCompile:**	VariantPostBuild
	PreCompile:	VariantPreCompile

Origin	AUTOSAR_ECUC

Parameter Name	RteNvmRomBlockLocationSymbol	
Label	ROM Block Location Symbol	
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	LINKER-SYMBOL	
Configuration class	PreCompile:	VariantPostBuild
	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	RteSwNvRamMappingRef	
Label	SW NVRAM Mapping	
Description	Reference to the SwServiceDependency which is used to specify the NvBlockNeeds.	
Multiplicity	0..1	
Type	FOREIGN-REFERENCE	
Configuration class	VariantPostBuild:	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	RteNvmBlockRef	
Label	NvM Block	
Description	Reference to the used NvM block for storage of the NVRAMMapping information.	
Multiplicity	1..1	
Type	SYMBOLIC-NAME-REFERENCE	
Configuration class	VariantPostBuild:	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	RteSwNvBlockDescriptorRef
Label	NvBlockDescriptor

Description	Reference to the NvBlockDescriptor in case the RTE needs to call the NvM directly	
Multiplicity	1..1	
Type	FOREIGN-REFERENCE	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

5.2.1.48. RteSwComponentType

Containers included		
Container name	**Multiplicity**	**Description**
[RteComponentTypeCalibration](#)	0..1	**Label:** Component Type Calibration 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.

Parameters included	
Parameter name	**Multiplicity**
[RteComponentTypeRef](#)	1..1
[RteImplementationRef](#)	0..1

Parameter Name	**RteComponentTypeRef**	
Label	Component Type	
Description	Reference to either AtomicSwComponentType or ParameterSwComponentType.	
Multiplicity	1..1	
Type	FOREIGN-REFERENCE	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	**RteImplementationRef**

Label	Implementation
Description	The Implementation which shall be assigned to the SwComponentType.
Multiplicity	0..1
Type	FOREIGN-REFERENCE
Configuration class	**PreCompile:**	VariantPostBuild
	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC

5.2.1.49. RteComponentTypeCalibration

Parameters included	
Parameter name	**Multiplicity**
RteCalibrationSupportEnabled	1..1
RteCalibrationSwAddrMethodRef	0..n

Parameter Name	RteCalibrationSupportEnabled	
Label	Calibration Support	
Description	Enables calibration support for the specified ParameterSwComponentType or AtomicSwComponentType.	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	true	
Configuration class	**VariantPostBuild:**	VariantPostBuild
	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	RteCalibrationSwAddrMethodRef	
Label	Calibration Sw Addr Method	
Description	Reference to the SwAddrMethod for which software calibration support shall be enabled.	
Multiplicity	0..n	
Type	FOREIGN-REFERENCE	
Configuration class	**PreCompile:**	VariantPostBuild

	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

5.2.2. Application programming interface (API)

5.2.2.1. Type definitions

5.2.2.1.1. Rte_Instance

Purpose	Run-Time Environment Instance.
Type	`const Rte_CDS_APPL *const`
Description	For each Atomic Software Component Type 'APPL' the Rte_Instance type is defined in the Software Component's Application Header File as a pointer to the Software Component's Component Data Structure Rte_CDS_APPL. Rte_Instance defines the data type of the Software Component's instance handles that are used to distinguish instance specific information.

5.2.2.1.2. Rte_ModeType_M

Purpose	Run-Time Environment Mode Type.
Type	`uint8_OR_uint16`
Description	The Rte_ModeType_M is defined for each Mode Declaration Group 'M'. Depending on the number of modes in the Mode Declaration Group, Rte_ModeType_M is defined as uint8 (if 'M' contains 256 or less modes) or as uint16 (if 'M' contains more than 256 modes).

5.2.2.1.3. Rte_PortHandle_I_RP

Purpose	Run-Time Environment Port Handle.
Type	`const Rte_PDS_C_I_RP *`

Description	The Rte_PortHandle_I_RP is defined in a Software Component's Application Header File for each interface 'I' and each usage (R: Interface is required, P: Interface is provided) as a pointer to the Port Data Structure Rte_PDS_C_I_RP, where 'C' is the Software Component Type Name. Rte_PortHandle_I_RP defines the data type necessary for accessing arrays of port data structures when using the indirect API.

5.2.2.1.4. Std_ReturnType

Purpose	Standard Return Type.
Type	`uint8`
Description	Defines the status and error values returned by API functions.

5.2.2.2. Macro constants

5.2.2.2.1. ApplicationError

Purpose	Error Code.
Value	1,...,63
Description	Generic application error. Application errors can be returned in client/server operation if an application error but no other error occurred. The symbolic names of the application errors and their error codes are defined as part of the client/server interface.

5.2.2.2.2. RTE_E_COMMS_ERROR

Purpose	Error Code.
Value	128
Description	An communication error has occured. Defined for backward compatibility with AUTOSAR 2.1, replaced by RTE_E_COM_STOPPED in AUTOSAR 3.0

5.2.2.2.3. RTE_E_COM_STOPPED

Purpose	Error Code.
Value	128

Description	An IPDU group was disabled while the application was waiting for the transmission acknowledgement. No value is available. This is not considered a fault, since the IPDU group is switched off on purpose.

5.2.2.2.4. RTE_E_HARD_TRANSFORMER_ERROR

Purpose	Error Code.
Value	138
Description	An error during transformation occurred.

5.2.2.2.5. RTE_E_INVALID

Purpose	Error Code.
Value	1
Description	Standard application error which indicates that an invalidated signal has been received in sender/receiver communication.

5.2.2.2.6. RTE_E_LIMIT

Purpose	Error Code.
Value	130
Description	An internal limit has been exceeded.

5.2.2.2.7. RTE_E_LOST_DATA

Purpose	Error Code.
Value	64
Description	Incoming data with event semantics (isQueued = 'TRUE') in sender/receiver communication was lost due to a queue overflow or an error of the communication stack.

5.2.2.2.8. RTE_E_MAX_AGE_EXCEEDED

Purpose	Error Code.

Value	64
Description	Available data with data semantics (isQueued = 'FALSE') in sender/receiver communication has exceeded the Alive Timeout limit on the receiver side.

5.2.2.2.9. RTE_E_NEVER_RECEIVED

Purpose	Error Code.
Value	133
Description	No data received for the corresponding unqueued data element since system start or partition restart.

5.2.2.2.10. RTE_E_NO_DATA

Purpose	Error Code.
Value	131
Description	An explicit read API call returned no data or the result of a server invocation is not yet available. This is not an error.

5.2.2.2.11. RTE_E_OK

Purpose	Error Code.
Value	0
Description	No error occurred.

5.2.2.2.12. RTE_E_SHUTDOWN_NOTIFICATION

Purpose	Error Code.
Value	156
Description	A shutdown notification was received in a blocking Rte API call (non standard). No value is available.

5.2.2.2.13. RTE_E_SOFT_TRANSFORMER_ERROR

Purpose	Error Code.

Value	140
Description	An error during transformation occurred which shall be notified to the SWC but still produces valid data as output (comparable to a warning).

5.2.2.2.14. RTE_E_TIMEOUT

Purpose	Error Code.
Value	129
Description	A timeout was detected, no value is available.

5.2.2.2.15. RTE_E_TRANSFORMER_LIMIT

Purpose	Error Code.
Value	139
Description	Buffer for transformation operation could not be created.

5.2.2.2.16. RTE_E_TRANSMIT_ACK

Purpose	Error Code.
Value	132
Description	A transmission acknowledgment was received. This is not an error.

5.2.2.2.17. RTE_E_UNCONNECTED

Purpose	Error Code.
Value	134
Description	The port used for communication is not connected.

5.2.2.2.18. RTE_MODULE_ID

Purpose	Module Information.
Value	240

Description	Module ID.

5.2.2.2.19. RTE_VENDOR_ID

Purpose	Module Information.
Value	1
Description	Vendor ID.

5.2.2.2.20. SCHM_E_IN_EXCLUSIVE_AREA

Purpose	Error Code.
Value	135
Description	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 or is running in an ExclusiveArea.

5.2.2.2.21. SCHM_E_LIMIT

Purpose	Error Code.
Value	130
Description	An internal Basic Software Scheduler limit has been exceeded. The request could not be handled. OUT buffers were not modified.

5.2.2.2.22. SCHM_E_LOST_DATA

Purpose	Error Code.
Value	64
Description	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.

5.2.2.2.23. SCHM_E_NO_DATA

Purpose	Error Code.

Value	131
Description	An explicit read API call returned no data. (This is not an error.)

5.2.2.2.24. SCHM_E_OK

Purpose	Error Code.
Value	0
Description	No error occurred.

5.2.2.2.25. SCHM_E_TIMEOUT

Purpose	Error Code.
Value	129
Description	The configured timeout exceeds before the intended result was ready.

5.2.2.2.26. SCHM_E_TRANSMIT_ACK

Purpose	Error Code.
Value	132
Description	Transmission acknowledgement received.

5.2.2.3. Functions

5.2.2.3.1. Rte_CData_NAME

Purpose	Provides access to the calibration parameter which is internally defined in an atomic software component.
Synopsis	`TYPE Rte_CData_NAME ( Rte_Instance self );`
Service ID	0x1F
Sync/Async	Synchronous
Reentrancy	Reentrant

Parameters (in)	self	Instance Handle, which is omitted if the Software Component has supportsMultipleInstantiation = 'FALSE'.
Return Value	Value of the calibration parameter in case of primitive type or a pointer to the parameter for string, record and array types.	
Description	Provides access to the calibration parameter which is internally defined in an atomic software component. Name Elements: ► NAME: Name of the calibration parameter	

5.2.2.3.2. Rte_COMCbkInv_SG

Purpose	Callback to indicate a signal group invalidation.
Synopsis	void **Rte_COMCbkInv_SG** (void);
Service ID	0x98
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Description	This callback function indicates that COM has received a signal group and parsed it as "invalid". Name Elements: ► SG: Name of the signal group

5.2.2.3.3. Rte_COMCbkInv_SN

Purpose	Callback to indicate a signal invalidation.
Synopsis	void **Rte_COMCbkInv_SN** (void);
Service ID	0x92
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Description	This callback function indicates that COM has received a signal and parsed it as "invalid". Name Elements: ► SN: Name of the signal

5.2.2.3.4. Rte_COMCbkRxTOut_SG

Purpose	Callback to indicate an outdated signal group.
Synopsis	`void Rte_COMCbkRxTOut_SG ( void );`
Service ID	0x99
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Description	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). Name Elements: ► SG: Name of the signal group

5.2.2.3.5. Rte_COMCbkRxTOut_SN

Purpose	Callback to indicate an outdated signal.
Synopsis	`void Rte_COMCbkRxTOut_SN ( void );`
Service ID	0x93
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Description	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). Name Elements: ► SN: Name of the signal

5.2.2.3.6. Rte_COMCbkTAck_SG

Purpose	Callback to indicate a signal group transmission acknowledgement.
Synopsis	`void Rte_COMCbkTAck_SG ( void );`
Service ID	0x96
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Description	This callback function indicates that the signals of the composite data item/event is already handed over by COM to the PDU router. Name Elements:

	SG: Name of the signal group

5.2.2.3.7. Rte_COMCbkTAck_SN

Purpose	Callback to indicate a signal transmission acknowledgement.
Synopsis	`void Rte_COMCbkTAck_SN ( void );`
Service ID	0x90
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Description	This callback function indicates that the signal of the primitive data item/event is already handed over by COM to the PDU router. Name Elements: ► SN: Name of the signal

5.2.2.3.8. Rte_COMCbkTErr_SG

Purpose	Callback to indicate a signal group transmission error.
Synopsis	`void Rte_COMCbkTErr_SG ( void );`
Service ID	0x97
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Description	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. Name Elements: ► SG: Name of the signal group

5.2.2.3.9. Rte_COMCbkTErr_SN

Purpose	Callback to indicate a signal transmission error.
Synopsis	`void Rte_COMCbkTErr_SN ( void );`
Service ID	0x91
Sync/Async	Synchronous
Reentrancy	Non-Reentrant

Description	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. Name Elements:
	► SN: Name of the signal

5.2.2.3.10. Rte_COMCbkTxPrep_MN

Purpose	Callback to indicate transmission preparation.
Synopsis	void **Rte_COMCbkTxPrep_MN** (void);
Service ID	0xA8
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Description	This callback function indicates that the signals/signal groups to be sent via a dedicated Com_MainFunctionTx instance will now be prepared for transmission. Name Elements:
	► MN: Name of the Com_MainFunctionTx instance

5.2.2.3.11. Rte_COMCbkTxTOut_SG

Purpose	Callback to indicate a timeout for a signal group.
Synopsis	void **Rte_COMCbkTxTOut_SG** (void);
Service ID	0x9A
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Description	This callback function indicates that the timeout of TransmissionAcknowledgementRequest for sending the signal group of the composite data item/event has expired. Name Elements:
	► SG: Name of the signal group

5.2.2.3.12. Rte_COMCbkTxTOut_SN

Purpose	Callback to indicate a transmission timeout.

Synopsis	void **Rte_COMCbkTxTOut_SN** (void);
Service ID	0x94
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Description	This callback function indicates that the timeout of TransmissionAcknowledgementRequest for sending the signal of the primitive data item/event has expired. Name Elements: ► SN: Name of the signal

5.2.2.3.13. Rte_COMCbk_SG

Purpose	Callback to indicate a signal group reception.
Synopsis	void **Rte_COMCbk_SG** (void);
Service ID	0x95
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Description	This callback function indicates that the signals of the composite data item/event or the arguments of an operation are ready for reception. Name Elements: ► SG: Name of the signal group

5.2.2.3.14. Rte_COMCbk_SN

Purpose	Callback to indicate a signal reception.
Synopsis	void **Rte_COMCbk_SN** (void);
Service ID	unspecified
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Description	This callback function indicates that the signal of the primitive data item/event is ready for reception. Name Elements: ► SN: Name of the signal

5.2.2.3.15. Rte_Call_Asynchronous_P_O

Purpose	Initiates an asynchronous client/server communication.
Synopsis	`Std_ReturnType` **`Rte_Call_Asynchronous_P_O`** `( Rte_Instance self ,` `TYPE_1 ARGUMENT_1 , ... , TYPE_N ARGUMENT_N );`
Service ID	0x1C
Sync/Async	Asynchronous
Reentrancy	Reentrant

Parameters (in)	`self`	Instance Handle, which is omitted if the Software Component has supportsMultipleInstantiation = 'FALSE'.
	`ARGUMENT_1-ARGUMENT_N`	Parameters of the Operation Prototype corresponding to the Argument Prototypes of direction IN and IN/OUT. Regardless of the direction of the Argument Prototype, all parameters are in parameters. All parameters are passed by value, except parameters of complex type or String Type.
Return Value	Standard Return Code	
	`RTE_E_OK`	No error occurred during the server execution.
	`RTE_E_LIMIT`	An outstanding asynchronous client/server communication to the same server exists for this client. The server invocation is discarded.
	`RTE_E_COM_STOPPED`	A communication error occurred (inter-ECU communication only). The request has not been successfully passed to the communication service.
	`RTE_E_UNCONNECTED`	Indicates that the receiver port is not connected.
Description	Initiates an asynchronous client/server communication. Name Elements: ► P: Name of the Require Port ► O: Name of the Operation Prototype ► TYPE_1, ... , TYPE_N: Types of the Argument Prototypes of direction IN and IN/OUT	

5.2.2.3.16. Rte_Call_Synchronous_P_O

Purpose	Initiates a synchronous client/server communication.
Synopsis	`Std_ReturnType` **`Rte_Call_Synchronous_P_O`** `( Rte_Instance self , TYPE_1 ARGUMENT_1 , ... , TYPE_N ARGUMENT_N , Rte_Transformer-Error transformerError );`
Service ID	0x1C
Sync/Async	Synchronous
Reentrancy	Depends on the configuration (e.g. it is not re-entrant if the server runnable entity cannot be invoked concurrently).

Parameters (in)	`self`	Instance Handle, which is omitted if the Software Component has supportsMultipleInstantiation = 'FALSE'.
	`ARGUMENT_1-ARGUMENT_N`	Parameters of the Operation Prototype. Depending on the direction of the corresponding Argument Prototype, the parameters are either in, in/out or out parameters. Input parameters (in) of primitive type except of String Type are passed by value, all other parameters are passed by reference.

Parameters (out)	`ARGUMENT_1-ARGUMENT_N`	Parameters of the Operation Prototype. Depending on the direction of the corresponding Argument Prototype, the parameters are either in, in/out or out parameters. Input parameters (in) of primitive type except of String Type are passed by value, all other parameters are passed by reference.
	`transformerError`	The optional parameter contains the transformer error which occurred during execution of the transformer chain.

Return Value	Standard Return Code.	
	`RTE_E_OK`	No error occurred during the server execution.
	`RTE_E_COM_STOPPED`	A communication error occurred (inter-ECU communication only). The request

		has not been successfully passed on to the communication service. The buffers of the in/out and out parameters are not modified.
	RTE_E_TIMEOUT	No reply was received within the configured timeout (inter-task or inter-ECU communication only), or the server invocation queue is full, or the partition of the client is not active, or the partition of the server is not active. The buffers of the in/out and out parameters are not modified.
	RTE_E_SHUTDOWN_NOTIFICATION	A shutdown notification has been received by the Rte while waiting for the server to terminate (inter-ECU communication and intra-ECU communication with timeout monitoring only). The buffers of the in/out and out parameters are not modified.
	RTE_E_SOFT_TRANSFORMER_ERROR	An error during transformation occurred which shall be notified to the SWC but still produces valid data as output (comparable to a warning).
	RTE_E_HARD_TRANSFORMER_ERROR	An error during transformation occurred.
	RTE_E_TRANSFORMER_LIMIT	Buffer for transformation operation could not be created.
	ApplicationError	The application error from a server is returned if none of the infrastructure errors listed above (other than RTE_E_OK) have occurred.
	RTE_E_UNCONNECTED	Indicates that the receiver port is not connected.
Description	Initiates a synchronous client/server communication. Name Elements: ► P: Name of the Require Port ► O: Name of the Operation Prototype ► TYPE_1, ... , TYPE_N: Types of the Argument Prototypes	

5.2.2.3.17. Rte_Calprm_P_NAME

Purpose	Provides access to the calibration parameter which is defined by a parameter component. This API is only generated if the AUTOSAR 3.2 Rte Wrapper generation is enabled.
Synopsis	`TYPE` **`Rte_Calprm_P_NAME`** `( Rte_Instance self );`
Service ID	-
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	`self` — Instance Handle, which is omitted if the Software Component has supportsMultipleInstantiation = 'FALSE'.
Return Value	Value of the calibration parameter in case of primitive type or a pointer to the parameter for string, record and array types.
Description	Provides access to the calibration parameter which is defined by a parameter component. Name Elements: ► P: Name of the Require Port ► NAME: Name of the calibration parameter

5.2.2.3.18. Rte_DRead_P_D

Purpose	Explicitly reads a Data Element Prototype with data semantics and returns it by value.
Synopsis	`TYPE` **`Rte_DRead_P_D`** `( Rte_Instance self );`
Service ID	0x1A
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	`self` — Instance Handle, which is omitted if the Software Component has supportsMultipleInstantiation = 'FALSE'.
Return Value	The value of the read Data Element Prototype
Description	Explicitly reads a Data Element Prototype with data semantics (isQueued = 'FALSE') and returns the result by value. Name Elements:

	► P: Name of the Provide Port
	► D: Name of the Data Element Prototype
	► TYPE: Type of the Data Element Prototype

5.2.2.3.19. Rte_Enter_NAME

Purpose	Enters the Exclusive Area.
Synopsis	void **Rte_Enter_NAME** (Rte_Instance self);
Service ID	0x2A
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	self	Instance Handle, omitted if the Software Component has supportsMultipleInstantiation = FALSE.
Description	Enters the Exclusive Area. Name Elements: ► NAME: Name of the Exclusive Area

5.2.2.3.20. Rte_Exit_NAME

Purpose	Leaves the Exclusive Area.
Synopsis	void **Rte_Exit_NAME** (Rte_Instance self);
Service ID	0x2B
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	self	Instance Handle, omitted if the Software Component has supportsMultipleInstantiation = FALSE.
Description	Enters the Exclusive Area. Name Elements: ► NAME: Leaves the Exclusive Area

5.2.2.3.21. Rte_Feedback_P_D

Purpose	Provided feedback for transmissions and communications.	
Synopsis	`Std_ReturnType` **`Rte_Feedback_P_D`** `( Rte_Instance self );`	
Service ID	0x17	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	`self`	Instance Handle, which is omitted if the Software Component has supportsMultipleInstantiation = 'FALSE'.
Return Value	Standard Return Code	
	`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).
	`RTE_E_TRANSMIT_ACK`	A transmission acknowledgment has been received from the communication service.
	`RTE_E_COM_STOPPED`	No data was received within the specified timeout because the corresponding IPDU group was disabled (inter-ECU communication only).
	`RTE_E_TIMEOUT`	No data was received within the specified timeout
	`RTE_E_UNCONNECTED`	Indicates that the sender port is not connected.
Description	Provides access for a sender to transmission acknowledgment or error notifications for explicit sender/receiver communication. Name Elements: ► P: Name of the Provide Port ► D: Name of the Data Element Prototype	

5.2.2.3.22. Rte_IFeedback_P_D

Purpose	Provided feedback for transmissions and communications.
Synopsis	`Std_ReturnType` **`Rte_IFeedback_P_D`** `( Rte_Instance self );`

Service ID	0x2F	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	`self`	Instance Handle, which is omitted if the Software Component has supportsMultipleInstantiation = 'FALSE'.
Return Value	Standard Return Code	
	`RTE_E_NO_DATA`	No acknowledgments or error notifications were received from COM when the runnable entity was started.
	`RTE_E_TRANSMIT_ACK`	A transmission acknowledgment has been received from the communication service.
	`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.
	`RTE_E_TIMEOUT`	(Inter-ECU only) A timeout notification was received from COM before any error notification.
	`RTE_E_UNCONNECTED`	Indicates that the sender port is not connected.
	`RTE_E_SOFT_TRANSFORMER_ERROR`	The return value of at least one transformer in the transformer chain was a soft error and no hard error occurred in the transformer chain.
	`RTE_E_HARD_TRANSFORMER_ERROR`	The return value of one transformer in the transformer chain represented a hard transformer error.
Description	Provides access for a sender to transmission acknowledgment or error notifications for implicit sender/receiver communication. Name Elements: ► P: Name of the Provide Port ► D: Name of the Data Element Prototype	

5.2.2.3.23. Rte_IInvalidate_RE_P_D

Purpose	Invalidates the Runnable Entity's copy of the Data Element Prototype.
Synopsis	void **Rte_IInvalidate_RE_P_D** (Rte_Instance self);
Service ID	0x24
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	self	Instance Handle, which is omitted if the Software Component has supportsMultipleInstantiation = 'FALSE'.
Description	Invalidates the Runnable Entity's copy of the Data Element Prototype. The invalidation of the Runnable Entity's copy is notified by the connected receivers after the Runnable Entity terminates. Name Elements: ► RE: Name of the Runnable Entity for which the Data Read Access is specified ► P: Name of the Provide Port ► D: Name of the Data Element Prototype ► TYPE: Type of the Data Element Prototype	

5.2.2.3.24. Rte_IReadRef_RE_P_D

Purpose	Provides read access to the Runnable Entity's copy of the Data Element Prototype.
Synopsis	TYPE **Rte_IReadRef_RE_P_D** (Rte_Instance self);
Service ID	0x32
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	self	Instance Handle, which is omitted if the Software Component has supportsMultipleInstantiation = 'FALSE'.
Return Value	The reference to the data stored in the Runnable Entity's copy of the Data Element Prototype.	
Description	Provides read access to the Runnable Entity's copy of the Data Element Prototype by reference. The copy is updated before the Runnable Entity is started. Name Elements:	

	▶ RE: Name of the Runnable Entity for which the Data Read Access is specified
	▶ P: Name of the Require Port
	▶ D: Name of the Data Element Prototype
	▶ TYPE: Type of the Data Element Prototype

5.2.2.3.25. Rte_IRead_RE_P_D

Purpose	Provides read access to the Runnable Entity's copy of the Data Element Prototype.
Synopsis	TYPE **Rte_IRead_RE_P_D** (Rte_Instance self);
Service ID	0x21
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	self	Instance Handle, which is omitted if the Software Component has supportsMultipleInstantiation = 'FALSE'.
Return Value	The data stored in the Runnable Entity's copy of the Data Element Prototype.
Description	Provides read access to the Runnable Entity's copy of the Data Element Prototype. The copy is updated before the Runnable Entity is started. Name Elements: ▶ RE: Name of the Runnable Entity for which the Data Read Access is specified ▶ P: Name of the Require Port ▶ D: Name of the Data Element Prototype ▶ TYPE: Type of the Data Element Prototype

5.2.2.3.26. Rte_IStatus_RE_P_D

Purpose	Provides access to the error status of a Data Element Prototype.
Synopsis	Std_ReturnType **Rte_IStatus_RE_P_D** (Rte_Instance self , Rte_-TransformerError transformerError);
Service ID	0x25
Sync/Async	Synchronous
Reentrancy	Reentrant

Parameters (in)	self		Instance Handle, which is omitted if the Software Component has supportsMultipleInstantiation = 'FALSE'.
Parameters (out)	transformerError		The optional parameter contains the transformer error which occurred during execution of the transformer chain.
Return Value	Standard Return Code		
	RTE_E_OK		No Error occurred.
	RTE_E_INVALID		Invalidated data has been received.
	RTE_E_MAX_AGE_EXCEEDED		Outdated data has been received.
	RTE_E_NEVER_RECEIVED		no data has been received updated since partition start.
	RTE_E_UNCONNECTED		Indicates that the receiver port is not connected.
	RTE_E_SOFT_TRANSFORMER_ERROR		An error during transformation occurred which shall be notified to the SWC but still produces valid data as output (comparable to a warning).
	RTE_E_HARD_TRANSFORMER_ERROR		An error during transformation occurred.
Description	Provides access to the error status of the Data Element Prototype that is accessed via Data Read Access. Name Elements: ► RE: Name of the Runnable Entity for which the Data Read Access is specified ► P: Name of the Require Port ► D: Name of the Data Element Prototype ► TYPE: Type of the Data Element Prototype		

5.2.2.3.27. Rte_IWriteRef_RE_P_D

Purpose	Provides write access to the Runnable Entity's copy of the Data Element Prototype by reference.
Synopsis	TYPE **Rte_IWriteRef_RE_P_D** (Rte_Instance self);
Service ID	0x23
Sync/Async	Synchronous

Reentrancy	Reentrant	
Parameters (in)	`self`	Instance Handle, which is omitted if the Software Component has supportsMultipleInstantiation = 'FALSE'.
Return Value	A reference to the corresponding data element.	
Description	Provides write access to the Runnable Entity's copy of the Data Element Prototype by reference. The changes to the Runnable Entity's copy are made available to the connected receivers after the Runnable Entity terminates. Name Elements: ► RE: Name of the Runnable Entity for which the Data Read Access is specified ► P: Name of the Provide Port ► D: Name of the Data Element Prototype ► TYPE: Type of the Data Element Prototype	

5.2.2.3.28. Rte_IWrite_RE_P_D

Purpose	Provides write access to the Runnable Entity's copy of the Data Element Prototype.	
Synopsis	`void Rte_IWrite_RE_P_D ( Rte_Instance self , TYPE data );`	
Service ID	0x22	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	`self`	Instance Handle, which is omitted if the Software Component has supportsMultipleInstantiation = 'FALSE'.
	`data`	Value to be sent.
Description	Provides write access to the Runnable Entity's copy of the Data Element Prototype. The changes to the Runnable Entity's copy are made available to the connected receivers after the Runnable Entity terminates. Name Elements: ► RE: Name of the Runnable Entity for which the Data Read Access is specified ► P: Name of the Provide Port ► D: Name of the Data Element Prototype ► TYPE: Type of the Data Element Prototype	

5.2.2.3.29. Rte_Init_InitContainer

Purpose	Initializes the RTE.
Synopsis	`void `**`Rte_Init_InitContainer`**` ( void );`
Service ID	0x75
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Description	Schedules RunnableEntitys for initialization purpose which are mapped to the related RteInitializationRunnableBatch container. Name Elements: ► InitContainer: Name of the related RteInitializationRunnableBatch container Description

5.2.2.3.30. Rte_Invalidate_P_D

Purpose	Initiate explicit invalidation of a Data Element Prototype.	
Synopsis	`Std_ReturnType `**`Rte_Invalidate_P_D`**` ( Rte_Instance self );`	
Service ID	0x16	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	`self`	Instance Handle, which is omitted if the Software Component has supportsMultipleInstantiation = 'FALSE'.
Return Value	Standard Return Code	
	`RTE_E_OK`	No Error occurred.
	`RTE_E_COM_STOPPED`	A communication error was detected when passing the data to the communication service (inter-ECU communication only)
Description	Initiates explicit invalidation of a Data Element Prototype with data semantics(isQueued = 'FALSE') for sender/receiver communication. Name Elements:	

| | ► | P: Name of the Provide Port |
| | ► | D: Name of the Data Element Prototype |

5.2.2.3.31. Rte_IrTrigger_RE_O

Purpose	Activates inter-runnable triggers.	
Synopsis	Std_ReturnType **Rte_IrTrigger_RE_O** (void);	
Service ID	0x2E	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Return Value	Standard Return Code when queuing is configured, otherwise void	
	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.
Description	Raise an internal trigger to activate Runnable entities of the same software component instance. Name Elements: ► RE: The name of the runnable entity the API might be used in ► O: The name of the InternalTriggeringPoint	

5.2.2.3.32. Rte_IrvIRead_RE_NAME

Purpose	Provides read access to the Inter Runnable Variable with implicit Communication Approach.	
Synopsis	TYPE **Rte_IrvIRead_RE_NAME** (Rte_Instance self);	
Service ID	0x26	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	self	Instance Handle, which is omitted if the Software Component has supportsMultipleInstantiation = 'FALSE'.

Return Value	Value of the accessed Inter Runnable Variable.
Description	Provides read access to the Inter Runnable Variable with implicit Communication Approach. Name Elements: ► RE: Name of the Runnable Entity for which read access to the Inter Runnable Variable is specified ► NAME: Name of the Inter Runnable Variable with implicit Communication Approach

5.2.2.3.33. Rte_IrvIWriteRef_RE_NAME

Purpose	Provide a reference to the VariableDataPrototype defined with the the implicitInter-RunnableVariable role referenced by a VariableAccess in the writtenLocalVariable role.
Synopsis	void **Rte_IrvIWriteRef_RE_NAME** (Rte_Instance self);
Service ID	0x31
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	self	Instance Handle, which is omitted if the Software Component has supportsMultipleInstantiation = 'FALSE'.
Description	Provides write access to the data write buffer of the VariableDataPrototype. Name Elements: ► RE: Name of the Runnable Entity for which write access to the Inter Runnable Variable is specified ► NAME: Name of the Inter Runnable Variable with implicit Communication Approach

5.2.2.3.34. Rte_IrvIWrite_RE_NAME

Purpose	Provides write access to the Inter Runnable Variable with implicit Communication Approach.

Synopsis	void **Rte_IrvIWrite_RE_NAME** (Rte_Instance self , TYPE data);	
Service ID	0x27	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	self	Instance Handle, which is omitted if the Software Component has supportsMultipleInstantiation = 'FALSE'.
	data	Value to be Written.
Description	Provides write access to the Inter Runnable Variable with implicit Communication Approach. Name Elements: ► RE: Name of the Runnable Entity for which write access to the Inter Runnable Variable is specified ► NAME: Name of the Inter Runnable Variable with implicit Communication Approach	

5.2.2.3.35. Rte_IrvRead_RE_NAME

Purpose	Provide read access to the Inter Runnable Variables with explicit Communication Approach.	
Synopsis	TYPE **Rte_IrvRead_RE_NAME** (Rte_Instance self);	
Service ID	0x28	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	self	Instance Handle, which is omitted if the Software Component has supportsMultipleInstantiation = 'FALSE'.
Return Value		
Description	Provides read access to the Inter Runnable Variable with explicit Communication Approach. Name Elements: ► RE: Name of the Runnable Entity for which read access to the Inter Runnable Variable is specified	

	► NAME: Name of the Inter Runnable Variable with implicit Communication Approach

5.2.2.3.36. Rte_IrvWrite_RE_NAME

Purpose	Provides write access to the Inter Runnable Variable with explicit Communication Approach.	
Synopsis	void **Rte_IrvWrite_RE_NAME** (Rte_Instance self , TYPE data);	
Service ID	0x29	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	self	Instance Handle, which is omitted if the Software Component has supportsMultipleInstantiation = 'FALSE'.
	data	Value to be Written.
Description	Provides write access to the Inter Runnable Variable with explicit Communication Approach. Name Elements: ► RE: Name of the Runnable Entity for which write access to the Inter Runnable Variable is specified ► NAME: Name of the Inter Runnable Variable with implicit Communication Approach	

5.2.2.3.37. Rte_IsAvailable_S_E

Purpose	Provides access to the availability status of an optional element of a structure.	
Synopsis	boolean **Rte_IsAvailable_S_E** (TYPE data);	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	data	The Structure data in which the availability status of an element needs to be queried.
Return Value	Availability status of the element in structure	

	TRUE	If the optional element is available in the structure.
	FALSE	If the optional element is not available in the structure.
Description	The RTE provides access to the availability status of an optional element of a structure. Name Elements:	
	► S: Name of the Structure Implementation Data Type	
	► E: Name of the optional element	
	► TYPE: The Structure Implementation Data Type	

5.2.2.3.38. Rte_IsUpdated_P_D

Purpose	Provides access to the update flag for an explicit reciever.	
Synopsis	`boolean` **`Rte_IsUpdated_P_D`** `( Rte_Instance self );`	
Service ID	0x30	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	`self`	Instance Handle, which is omitted if the Software Component has supportsMultipleInstantiation = 'FALSE'.
Return Value	TRUE DataElement updated since last read FALSE DataElement not updated since last read.	
Description	Provides access to the update flag for an explicit reciever, if the enableUpdate attribute is enabled for the associated VariableDataPrototype. Name Elements: ► P: Name of the Require Port ► D: Name of the Data Element Prototype	

5.2.2.3.39. Rte_LdComCbkCopyRxData_SN

Purpose	Callback to provide the received data of an I-PDU segment (N-PDU) to the upper layer.

Synopsis	BufReq_ReturnType **Rte_LdComCbkCopyRxData_SN** (const PduInfoType * SduInfoPtr , PduLengthType * RxBufferSizePtr);	
Parameters (in)	SduInfoPtr	Provides the source buffer (SduDataPtr) and the number of bytes to be copied (SduLength). An SduLength of 0 can be used to query the current amount of available buffer in the upper layer module In this case, the SduDataPtr may be a NULL_PTR.
Parameters (out)	RxBufferSizePtr	Available receive buffer after data has been copied. {Non Reentrant for same sn, otherwise Reentrant}
Return Value		
Description	Within this API, the passed signal data (or the indicated portion) is copied to the previously locked reception buffer. Name Elements: ► SN: Name of the LdCom signal	

5.2.2.3.40. Rte_LdComCbkCopyTxData_SN

Purpose	Callback to copy the indicated number of bytes to the provided destination.	
Synopsis	BufReq_ReturnType **Rte_LdComCbkCopyTxData_SN** (const PduInfoType * SduInfoPtr , RetryInfoType * RetryInfoPtr , PduLengthType * TxDataCntPtr);	
Parameters (in)	SduInfoPtr	Provides the destination buffer (SduDataPtr) and the number of bytes to be copied (SduLength).
	RetryInfoPtr	Will not be handled by LdCom and its upper layer.
Parameters (out)	TxDataCntPtr	Indicates the remaining number of bytes that are available in the upper layer module's Tx buffer. {Non Reentrant for same SN, otherwise Reentrant}
Return Value		
Description	Name Elements: ► SN: Name of the LdCom signal	

5.2.2.3.41. Rte_LdComCbkRxIndication_SN

Purpose	Callback to indicate the reception of a received LdCom signal.
Synopsis	void **Rte_LdComCbkRxIndication_SN** (const PduInfoType * info);
Parameters (in)	info	Length and pointer to the buffer of the data {Non Reentrant for same sn, otherwise Reentrant}
Description	This callback function indicates that the signal of the primitive data item/event is ready for reception. Name Elements: ► SN: Name of the LdCom signal

5.2.2.3.42. Rte_LdComCbkStartOfReception_SN

Purpose	Callback to indicate whether the connection is accepted or not.
Synopsis	BufReq_ReturnType **Rte_LdComCbkStartOfReception_SN** (PduLengthType SduLength , PduLengthType * RxBufferSizePtr);
Parameters (in)	SduLength	Pointer to a PduInfoType structure containing the payload data (without protocol information) and payload length of the first frame or single frame of a transport protocol I-PDU reception, and the Meta- Data related to this PDU. If neither first/single frame data nor MetaData are available, this parameter set to NULL_PTR.
Parameters (out)	RxBufferSizePtr	Available receive buffer in the receiving module. This parameter will be used to compute the Block Size (BS) in the transport protocol module. {Non Reentrant for same sn, otherwise Reentrant}
Return Value		
Description	Within this API, the corresponding reception buffer is locked. Name Elements:

	►	SN: Name of the LdCom signal

5.2.2.3.43. Rte_LdComCbkTpRxIndication_SN

Purpose	Callback to indicate whether the transmission was successful or not.	
Synopsis	void **Rte_LdComCbkTpRxIndication_SN** (NotifResultType result);	
Service ID	0xA3	
Sync/Async	Synchronous	
Parameters (in)	`result`	Result of the reception.
Description	Within this API, the RTE unlock the previously locked reception buffer. Name Elements: ► SN: Name of the LdCom signal {Non Reentrant for same SN, otherwise Reentrant}	

5.2.2.3.44. Rte_LdComCbkTpTxConfirmation_SN

Purpose	Callback to indicate whether the transmission was successful or not.	
Synopsis	void **Rte_LdComCbkTpTxConfirmation_SN** (NotifResultType result);	
Service ID	0xA5	
Sync/Async	Synchronous	
Parameters (in)	`result`	Result of the reception.
Description	The RTE unlock the signal buffer after the invocation of this callback. Name Elements: ► SN: Name of the LdCom signal {Non Reentrant for same SN, otherwise Reentrant}	

5.2.2.3.45. Rte_LdComCbkTriggerTransmit_SN

Purpose	Callback to indicate the request for transmission of an LdCom signal.
Synopsis	Std_ReturnType **Rte_LdComCbkTriggerTransmit_SN** (const PduInfoType * PduInfoPtr);

Parameters (in,out)	PduInfoPtr	Length and pointer to the buffer of the I-PDU. On return, the service will indicate the length of the copied SDU data in Sdu-Length. {Non Reentrant for same sn, otherwise Reentrant}
Return Value		
Description	Within this API, the lower layer module (calling module) requests the data to be copied into the buffer provided by PduInfoPtr->SduDataPtr and update the length of the actual copied data in PduInfoPtr->SduLength. Name Elements: ► SN: Name of the LdCom signal	

5.2.2.3.46. Rte_Mode_P_MDP

Purpose	Provides the currently active, the previous and the next Rte Mode.	
Synopsis	`Rte_ModeType_M` **`Rte_Mode_P_MDP`** `( Rte_Instance self , Rte_Mode-Type_M * previousMode , Rte_ModeType_M * nextMode );`	
Service ID	0x2C	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	self	Instance Handle, omitted if the Software Component has supportsMultipleInstantiation = FALSE.
Parameters (out)	previousMode	Pointer to the memory location in which the previous mode will be stored if a transition occurs. Otherwise the currently active mode will be provided.
	nextMode	Pointer to the memory location in which the next mode will be stored if a transition occurs. Otherwise the currently active mode will be provided.
Return Value	Currently active mode of the Mode Declaration Group Prototype.	
Description	Provides the currently active mode in a mode port. If the mode machine instance is in transition additionally the values of the previous and the next mode are provided. Name Elements:	

	►	P: Name of the Provide Port
	►	MDP: Name of the Mode Declaration Group Prototype
	►	M: Name of the Mode Declaration Group

5.2.2.3.47. Rte_Mode_P_m

Purpose	Provides the currently active Rte mode.
Synopsis	`Rte_ModeType_M` **`Rte_Mode_P_m`** `( Rte_Instance self );`
Service ID	0x2C
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	`self`	Instance Handle, omitted if the Software Component has supportsMultipleInstantiation = FALSE.
Return Value	Currently active mode of the Mode Declaration Group Prototype.	
Description	Provides the currently active mode in a mode port. Name Elements: ► P: Name of the Provide Port ► m: Name of the Mode Declaration Group Prototype ► M: Name of the Mode Declaration Group	

5.2.2.3.48. Rte_NPorts_I_RP

Purpose	Provides the number of ports.
Synopsis	`uint8` **`Rte_NPorts_I_RP`** `( Rte_Instance self );`
Service ID	0x11
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	`self`	Instance Handle, omitted if the Software Component has supportsMultipleInstantiation = FALSE.

Return Value	Number of port data structures of the corresponding interface type and usage.
Description	Provides the number of ports of a given interface type and a given provide/require usage.
	Name Elements:
	► I: Name of the Interface
	► RP: Usage of the Interface (R: Interface is required; P: Interface is provided)

5.2.2.3.49. Rte_PartitionRestarting_PID

Purpose	Indicates restart of a partition.
Synopsis	void **Rte_PartitionRestarting_PID** (void);
Service ID	0x73
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Description	Indicate to the RTE that a Partition is going to be restarted and that the communication with the Partition shall be ignored.
	Name Elements:
	► PID: Name of the EcucPartition according to the ECU Configuration Description

5.2.2.3.50. Rte_PartitionTerminated_PID

Purpose	Indicates termination of a partition.
Synopsis	void **Rte_PartitionTerminated_PID** (void);
Service ID	0x72
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Description	Indicate to the RTE that a partition is going to be terminated, and the communication with the Partition shall be ignored.
	Name Elements:
	► PID: Name of the EcucPartition according to the ECU Configuration Description

5.2.2.3.51. Rte_Pim_NAME

Purpose	Provides access to a per-instance memory section.
Synopsis	`TYPE Rte_Pim_NAME ( Rte_Instance self );`
Service ID	0x1E
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	`self`	Instance Handle, omitted if the Software Component has supportsMultipleInstantiation = FALSE.
Return Value	A typed pointer to the per-instance memory.	
Description	Provides access to a per-instance memory section. Name Elements: ► NAME Name of the Per-Instance Memory ► TYPE Type of the Per-Instance Memory	

5.2.2.3.52. Rte_Port_R

Purpose	Provides access to a single port.
Synopsis	`Rte_PortHandle_I_RP Rte_Port_R ( Rte_Instance self );`
Service ID	0x12
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	`self`	Instance Handle, omitted if the Software Component has supportsMultipleInstantiation = FALSE.
Return Value	Pointer to the port data structure of the specified port.	
Description	Provides access to the port data structure of a single port of a software component instance. Name Elements:	

> ► I: Name of the Interface
>
> ► P: Name of the port (Provide Port or Require Port)
>
> ► RP: Usage of the Interface (R: Interface is required; P: Interface is provided)

5.2.2.3.53. Rte_Ports_I_RP

Purpose	Initiates a synchronous client/server communication.
Synopsis	`Rte_PortHandle_I_RP` **`Rte_Ports_I_RP`** `( Rte_Instance self );`
Service ID	0x10
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	`self`	Instance Handle, omitted if the Software Component has supportsMultipleInstantiation = FALSE.
Return Value	Array of port data structures of the corresponding interface type and usage.
Description	Provides an array of the ports of a given interface type and a given provide/require usage. Name Elements: ► I: Name of the Interface ► RP: Usage of the Interface (R: Interface is required; P: Interface is provided)

5.2.2.3.54. Rte_Prm_P_NAME

Purpose	Provides access to the calibration parameter which is defined by a parameter component.
Synopsis	`TYPE` **`Rte_Prm_P_NAME`** `( Rte_Instance self );`
Service ID	0x20
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	`self`	Instance Handle, which is omitted if the Software Component has supportsMultipleInstantiation = 'FALSE'.

Return Value	Value of the calibration parameter in case of primitive type or a pointer to the parameter for string, record and array types.
Description	Provides access to the calibration parameter which is defined by a parameter component. Name Elements: ► P: Name of the Require Port ► NAME: Name of the calibration parameter

5.2.2.3.55. Rte_Read_P_D

Purpose	Explicitly reads a Data Element Prototype with data semantics and returns it by argument.	
Synopsis	Std_ReturnType **Rte_Read_P_D** (Rte_Instance self , TYPE * data , Rte_TransformerError transformerError);	
Service ID	0x19	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	self	Instance Handle, which is omitted if the Software Component has supportsMultipleInstantiation = 'FALSE'.
Parameters (out)	data	Pointer to the memory location where the received data shall be stored.
	transformerError	The optional parameter contains the transformer error which occurred during execution of the transformer chain.
Return Value	Standard Return Code	
	RTE_E_OK	No error occurred.
	RTE_E_INVALID	Invalidated data has been received.
	RTE_E_MAX_AGE_EXCEEDED	Outdated data has been received.
	RTE_E_NEVER_RECEIVED	no data has been received updated since partition start.
	RTE_E_UNCONNECTED	Indicates that the receiver port is not connected.
	RTE_E_SOFT_TRANSFORMER_ERROR	An error during transformation occurred which shall be notified to the SWC but still

		produces valid data as output (comparable to a warning).
	RTE_E_HARD_TRANSFORMER_ERROR	An error during transformation occurred.
	RTE_E_TRANSFORMER_LIMIT	Buffer for transformation operation could not be created.
Description	Explicitly reads a Data Element Prototype with data semantics (isQueued = 'FALSE') and returns the result by argument. Name Elements: ► P: Name of the Provide Port ► D: Name of the Data Element Prototype ► TYPE: Type of the Data Element Prototype	

5.2.2.3.56. Rte_Receive_P_D

Purpose	Explicitly reads a Data Element Prototype with event semantics.	
Synopsis	Std_ReturnType **Rte_Receive_P_D** (Rte_Instance self , TYPE * data , Rte_TransformerError transformerError);	
Service ID	0x1B	
Sync/Async	Synchronous	
Reentrancy	Non-Reentrant	
Parameters (in)	self	Instance Handle, which is omitted if the Software Component has supportsMultipleInstantiation = 'FALSE'.
Parameters (out)	data	Pointer to the memory location where the received data shall be stored.
	transformerError	The optional parameter contains the transformer error which occurred during execution of the transformer chain.
Return Value	Standard Return Code	
	RTE_E_OK	No error occurred.
	RTE_E_NO_DATA	No data was received and no other error occurred when reception of the data was attempted (non-blocking reception only).
	RTE_E_SOFT_TRANSFORMER_ERROR	An error during transformation occurred which shall be notified to the SWC but still

		produces valid data as output (comparable to a warning).
	RTE_E_HARD_TRANSFORMER_ERROR	An error during transformation occurred.
	RTE_E_TIMEOUT	No data was received within the specified timeout and no other error occurred when reception of the data was attempted (intra-ECU communication only).
	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 layer. This error does not affect the data returned in the OUT parameter.
	RTE_E_UNCONNECTED	Indicates that the receiver port is not connected.
	RTE_E_SHUTDOWN_NOTIFICATION	A shutdown notification has been received by the Rte while waiting for new data (blocking reception only). No data is available.
Description	Explicitly reads a Data Element Prototype with event semantics (isQueued = 'TRUE'). Name Elements: ► P: Name of the Require Port ► D: Name of the Data Element Prototype ► TYPE: Type of the Data Element Prototype	

5.2.2.3.57. Rte_RestartPartition_PID

Purpose	Restarts a partition.
Synopsis	void **Rte_RestartPartition_PID** (void);
Service ID	0x74
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Description	Initialize the RTE resources allocated for a partition. Name Elements:

	▶	PID: Name of the EcucPartition according to the ECU Configuration Description

5.2.2.3.58. Rte_Result_P_O

Purpose	Retrieves the result of an asynchronous client/server communication.	
Synopsis	`Std_ReturnType` **`Rte_Result_P_O`** `( Rte_Instance self , TYPE_1 AR-` `GUMENT_1 , ...  , TYPE_N ARGUMENT_N );`	
Service ID	0x1D	
Sync/Async	Synchronous	
Reentrancy	Non-Reentrant	
Parameters (in)	`self`	Instance Handle, which is omitted if the Software Component has supportsMultipleInstantiation = 'FALSE'.
Parameters (out)	`ARGUMENT_1-ARGUMENT_N`	Parameters of the Operation Prototype corresponding to the Argument Prototypes of direction IN/OUT and OUT. Depending on the direction of the Argument Prototype, the parameters are either in/out or out parameters. All parameters are passed by reference.
Return Value	Standard Return Code	
	`RTE_E_OK`	No error occurred during the server execution.
	`RTE_E_NO_DATA`	The server's result is not available but no other error occurred within the API call (non-blocking reception only). The buffers of the in/out and out parameters are not modified.
	`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 are not modified.
	`RTE_E_COM_STOPPED`	A communications error occurred (inter-ECU communication only). The request has not been successfully passed to the communication service. The buffers

		for the in/out and out parameters are not modified.
	`RTE_E_SHUTDOWN_NOTIFICATION`	A shutdown notification has been received by the Rte while waiting for the server to terminate (inter-ECU communication and intra-ECU communication with timeout monitoring only). The buffers of the in/out and out parameters are not modified
	`ApplicationError`	The application error from a server is returned if none of the infrastructure errors listed above (other than RTE_E_OK) have occurred.
	`RTE_E_UNCONNECTED`	Indicates that the receiver port is not connected.
Description	Retrieves the result of an asynchronous client/server communication. Name Elements: ► P: Name of the Require Port ► O: Name of the Operation Prototype ► TYPE_1, ... , TYPE_N: Types of the Argument Prototypes of direction IN/OUT and OUT	

5.2.2.3.59. Rte_Send_P_D

Purpose	Initiate an explicit sender-receiver transmission of data elements.	
Synopsis	`Std_ReturnType` **`Rte_Send_P_D`** `( Rte_Instance self , TYPE data ,` `Rte_TransformerError transformerError );`	
Service ID	0x13	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	`self`	Instance Handle, which is omitted if the Software Component has supportsMultipleInstantiation = 'FALSE'.
	`data`	Value to be sent.
Parameters (out)	`transformerError`	The optional parameter contains the transformer error which occurred during execution of the transformer chain.

Return Value	Standard Return Code	
	RTE_E_OK	No Error occurred.
	RTE_E_COM_STOPPED	A communication error was detected when passing the data to the communication service (inter-ECU communication only)
	RTE_E_SOFT_TRANSFORMER_ERROR	An error during transformation occurred which shall be notified to the SWC but still produces valid data as output (comparable to a warning).
	RTE_E_HARD_TRANSFORMER_ERROR	An error during transformation occurred.
	RTE_E_LIMIT	The event has been discarded because the receiver's receive queue was full (intra-ECU communication only)
Description	Initiates explicit sender/receiver transmission for Data Element Prototypes with event semantics (isQueued = 'TRUE'). Name Elements: ► P: Name of the Provide Port ► D: Name of the Data Element Prototype ► TYPE: Type of the Data Element Prototype	

5.2.2.3.60. Rte_SetAvailable_S_E

Purpose	Provides access to set the availability status of an optional element of a structure.	
Synopsis	void **Rte_SetAvailable_S_E** (TYPE * data , boolean available);	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	available	Availability status of the optional element.
Parameters (in,out)	data	Pointer to the memory location where the structure data is stored.
Description	The RTE provides access to the set the availability status of an optional element of a structure. Name Elements: ► S: Name of the Structure Implementation Data Type	

	►	E: Name of the optional element
	►	TYPE: The Structure Implementation Data Type

5.2.2.3.61. Rte_Start

Purpose	Initializes the Rte.	
Synopsis	`Std_ReturnType` **`Rte_Start`** `( void );`	
Service ID	0x70	
Sync/Async	Synchronous	
Reentrancy	Non-Reentrant	
Return Value	Standard Return Code	
	`RTE_E_OK`	No error occurred.
	`RTE_E_LIMIT`	An internal limit was exceeded because the allocation of a required resource had failed.
Description	Initializes the Rte.	

5.2.2.3.62. Rte_StartTiming

Purpose	Starts the triggering of recurrent events.
Synopsis	`void` **`Rte_StartTiming`** `( void );`
Service ID	0x76
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Description	Releases the activation of RunnableEntitys triggered by TimingEvents and BackgroundEvents after the last call of a Rte_Init_InitContainer function.

5.2.2.3.63. Rte_Stop

Purpose	Terminates the Rte.
Synopsis	`Std_ReturnType` **`Rte_Stop`** `( void );`

Service ID	0x71	
Sync/Async	Synchronous	
Reentrancy	Non-Reentrant	
Return Value	Standard Return Code	
	`RTE_E_OK`	No error occurred.
	`RTE_E_LIMIT`	A resource could not be released.
Description	Terminates the Rte.	

5.2.2.3.64. Rte_SwitchAck_P_m

Purpose	Provides access to mode switch acknowledgement.	
Synopsis	`Std_ReturnType` **`Rte_SwitchAck_P_m`** `( Rte_Instance self );`	
Service ID	0x18	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	`self`	Instance Handle, omitted if the Software Component has supportsMultipleInstantiation = FALSE.
Return Value	Standard Return Code	
	`RTE_E_NO_DATA`	(non-blocking read) The mode switch is still in progress.
	`RTE_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.
	`RTE_E_TRANSMIT_ACK`	The mode switch has been completed.
	`RTE_E_UNCONNECTED`	Indicates that the mode provider port is not connected.
Description	Provide access to mode switch completed acknowledgements and error notifications to mode managers. Name Elements: ▸ P: Name of the Provide Port ▸ m: Name of the Mode Declaration Group Prototype	

5.2.2.3.65. Rte_Switch_P_m

Purpose	Initiates a Rte mode switch.	
Synopsis	Std_ReturnType **Rte_Switch_P_m** (Rte_Instance self , Rte_Mode-Type_M MODE);	
Service ID	0x15	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	self	Instance Handle, omitted if the Software Component has supportsMultipleInstantiation = FALSE.
	MODE	Next mode.
Return Value	Standard Return Code	
	RTE_E_OK	No error occurred.
	RTE_E_LIMIT	The mode switch has been discarded because the queue for mode switches was full.
Description	Initiates a Rte mode switch.	
	Name Elements:	
	► P: Name of the Provide Port	
	► m: Name of the Mode Declaration Group Prototype	
	► M: Name of the Mode Declaration Group	

5.2.2.3.66. Rte_Trigger_P_O

Purpose	Activates external triggers.
Synopsis	Std_ReturnType **Rte_Trigger_P_O** (void);
Service ID	0x2D
Sync/Async	Synchronous
Reentrancy	Reentrant
Return Value	Standard Return Code when queuing is configured, otherwise void

	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.
Description	Raise a external trigger of a trigger port. Name Elements: ► P: The name of the port ► O: The name of the trigger within the trigger interface	

5.2.2.3.67. Rte_Write_P_D

Purpose	Initiate an explicit sender-receiver transmission of data elements.	
Synopsis	Std_ReturnType **Rte_Write_P_D** (Rte_Instance self , TYPE data , Rte_TransformerError transformerError);	
Service ID	0x14	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	self	Instance Handle, which is omitted if the Software Component has supportsMultipleInstantiation = 'FALSE'.
	data	Value to be sent.
Parameters (out)	transformerError	The optional parameter contains the transformer error which occurred during execution of the transformer chain.
Return Value	Standard Return Code	
	RTE_E_OK	No Error occurred.
	RTE_E_COM_STOPPED	A communications error was detected when passing the data to the communication service (inter-ECU communication only)
	RTE_E_SOFT_TRANSFORMER_ERROR	An error during transformation occurred which shall be notified to the SWC but still produces valid data as output (comparable to a warning).

	`RTE_E_HARD_TRANSFORMER_ERROR`	An error during transformation occurred.
	`RTE_E_TRANSFORMER_LIMIT`	Buffer for transformation operation could not be created.
Description	Initiates explicit sender/receiver transmission for Data Element Prototypes with data semantics (isQueued = 'FALSE'). Name Elements: ► P: Name of the Provide Port ► D: Name of the Data Element Prototype ► TYPE: Type of the Data Element Prototype	

5.2.2.3.68. SchM_ActMainFunction

Purpose	Triggers the activation of internal triggers.
Synopsis	`void ` **`SchM_ActMainFunction`** ` ( void );`
Service ID	0x05
Sync/Async	Synchronous
Reentrancy	Depends on the configuration (e.g. it is not re-entrant if the bsw schedulable entity cannot be invoked concurrently).
Description	Triggers the activation of the BswSchedulableEntity which is associated with an activationPoint of the same or Basic Software Module.

5.2.2.3.69. SchM_Call

Purpose	Invokes a Client-Server operation between BSW modules, possibly crossing partition boundaries.	
Synopsis	`Std_ReturnType ` **`SchM_Call`** ` ( TYPE returnValue , TYPE_1 ARGUMENT_1 , ... , TYPE_N ARGUMENT_N );`	
Service ID	0x0C	
Sync/Async	Synchronous (BswSynchronousServerCallPoint) / Asynchronous (BswAsynchronousServerCallPoint)	
Reentrancy	Depends on the configuration (e.g. it is not re-entrant if the server called entity has isReentrant of the corresponding module entry set to false	
Return Value		

Description	Invokes a Client-Server operation between BSW modules, possibly crossing partition boundaries, synchronously or asynchronously.

5.2.2.3.70. SchM_Deinit

Purpose	Finalizes the Basic Software Scheduler.
Synopsis	`void SchM_Deinit ( void );`
Service ID	0x01
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Description	Finalizes the Basic Software Scheduler part of the RTE.

5.2.2.3.71. SchM_Enter_MP_VI_AI_NAME

Purpose	Enters the Exclusive Area.
Synopsis	`void SchM_Enter_MP_VI_AI_NAME ( void );`
Service ID	0x03
Sync/Async	Synchronous
Reentrancy	Reentrant
Description	Enters the Exclusive Area. Name Elements: ► MP: The module prefix of the calling BSW module ► VI: Optional, the vendor ID ► AI: Optional, the vendor API infix ► NAME: Name of the Exclusive Area

5.2.2.3.72. SchM_Exit_MP_VI_AI_NAME

Purpose	Leaves the Exclusive Area.
Synopsis	`void SchM_Exit_MP_VI_AI_NAME ( void );`

Service ID	0x04
Sync/Async	Synchronous
Reentrancy	Reentrant
Description	Enters the Exclusive Area. Name Elements: ► MP: The module prefix of the calling BSW module ► VI: Optional, the vendor ID ► AI: Optional, the vendor API infix ► NAME: Leaves the Exclusive Area

5.2.2.3.73. SchM_GetVersionInfo

Purpose	Returns the version information of the Basic Software Scheduler.	
Synopsis	void **SchM_GetVersionInfo** (Std_VersionInfoType * versioninfo);	
Service ID	0x02	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (out)	versioninfo	The parameter versioninfo points to the memory location that holds the version information of the Basic Software Scheduler.
Description	SchM_GetVersionInfo returns the version information of the RTE module which includes the Basic Software Scheduler.	

5.2.2.3.74. SchM_Init

Purpose	Initializes the Basic Software Scheduler.
Synopsis	void **SchM_Init** (void);
Service ID	0x00
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Description	Initializes the Basic Software Scheduler part of the RTE.

5.2.2.3.75. SchM_Mode_MP_VI_AI_m

Purpose	Provides the currently active SchM mode.
Synopsis	`SchM_ModeType_M` **`SchM_Mode_MP_VI_AI_m`** `( void );`
Service ID	0x07
Sync/Async	Synchronous
Reentrancy	Reentrant
Return Value	Currently active mode of the Mode Declaration Group Prototype.
Description	Provides the currently active mode of a required Mode Declaration Group Prototype. Name Elements: ► MP: The module prefix of the calling BSW module ► VI: Optional, the vendor ID ► AI: Optional, the vendor API infix ► m: Name of the Mode Declaration Group Prototype ► M: Name of the Mode Declaration Group

5.2.2.3.76. SchM_Receive

Purpose	Reads a Data Element Prototype with event semantics.	
Synopsis	`Std_ReturnType` **`SchM_Receive`** `( TYPE * data );`	
Service ID	0x0B	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Return Value	Standard Return Code	
	`SCHM_E_OK`	No error occurred.
	`SCHM_E_NO_DATA`	No "events" (means queued data) were received and no other error occurred when the read was attempted.
	`SCHM_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 other error.
Description	Perfoms an "explicit" sender-receiver reception of data elements with "event" semantic (queued) between BSW modules.	

5.2.2.3.77. SchM_Result

Purpose	Get the result of an asynchronous call of a BswModuleEntry.
Synopsis	`Std_ReturnType `**`SchM_Result`**` ( TYPE returnValue , TYPE_1 ARGUMENT_1 , ... , TYPE_N ARGUMENT_N );`
Service ID	0x0D
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Return Value	
Description	Get the result of an asynchronous call of a BswModuleEntry.

5.2.2.3.78. SchM_Send

Purpose	Initiates an explicit sender-receiver transmission of data elements.	
Synopsis	`Std_ReturnType `**`SchM_Send`**` ( TYPE data );`	
Service ID	0x0A	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Return Value	Standard Return Code	
	`SCHM_E_OK`	No Error occurred.
	`SCHM_E_LIMIT`	An 'event' has been discarded due to a full queue by one of the partition local receivers.
Description	Initiate an "explicit" sender-receiver transmission of data elements with "event" semantic (queued) between BSW modules.	

5.2.2.3.79. SchM_SwitchAck_MP_VI_AI_m

Purpose	Provides access to SchM mode switch acknowledgement.

Synopsis	`Std_ReturnType` **`SchM_SwitchAck_MP_VI_AI_m`** `( SchM_ModeType_M MODE );`	
Service ID	0x08	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	`MODE`	Next mode.
Return Value	Standard Return Code	
	`SCHM_E_NO_DATA`	(non-blocking read) no error is occurred when the SchM_SwitchAck read was attempted.
	`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
	`SCHM_E_TIMEOUT`	The partition of the mode users is stopped or restarting or has been restarted while the mode switch was requested.
Description	Provides access to SchM mode switch completed acknowledgements and error notifications to mode managers. Name Elements: ▸ MP: The module prefix of the calling BSW module ▸ VI: Optional, the vendor ID ▸ AI: Optional, the vendor API infix ▸ m: Name of the Mode Declaration Group Prototype ▸ M: Name of the Mode Declaration Group	

5.2.2.3.80. SchM_Switch_MP_VI_AI_m

Purpose	Initiates a SchM mode switch.
Synopsis	`Std_ReturnType` **`SchM_Switch_MP_VI_AI_m`** `( SchM_ModeType_M MODE );`
Service ID	0x06
Sync/Async	Synchronous
Reentrancy	Reentrant

Parameters (in)	MODE	Next mode.
Return Value	Standard Return Code	
	SCHME_OK	No error occurred.
	SCHM_E_LIMIT	The mode switch has been discarded because the queue for mode switches was full.
Description	Initiates a SchM mode switch. Name Elements: ► MP: The module prefix of the calling BSW module ► VI: Optional, the vendor ID ► AI: Optional, the vendor API infix ► m: Name of the Mode Declaration Group Prototype ► M: Name of the Mode Declaration Group	

5.2.2.3.81. SchM_Trigger

Purpose	Triggers the activation of external triggers.
Synopsis	`void SchM_Trigger ( void );`
Service ID	0x09
Sync/Async	Synchronous
Reentrancy	Depends on the configuration (e.g. it is not re-entrant if the bsw schedulable entity cannot be invoked concurrently).
Description	Triggers the activation of connected BswSchedulableEntitys of the same or other Basic Software Modules.

5.2.3. Integration notes

5.2.3.1. Exclusive areas

The Rte does not use exclusive areas. Instead, the Rte protects internal data using a configurable data consistency mechanism if atomic or cooperative access to the data cannot be guaranteed. Refer to the section

called "Configuring the default data consistency mechanism" in the chapter "Run-Time Environment" in the User's Guide that is contained in the EB tresos AutoCore Generic documentation.

5.2.3.2. Production errors

Production errors are not reported by the `Rte` module.

5.2.3.3. Memory mapping

General information about memory mapping is provided in the EB tresos AutoCore Generic documentation. Refer to the section `Memory mapping and compiler abstraction` in the `Integration notes` section for details.

The `Rte` defines memory sections in the Basic Software Module Description that can be used by the generated `Rte`.

5.2.3.4. Integration requirements

WARNING	Integration requirements list is not exhaustive
⚠	The following list of integration requirements helps you to integrate your product. However, this list is not exhaustive. You also require information from the user's guide, release notes, and EB tresos AutoCore known issues to successfully integrate your product.

5.2.3.4.1. lim.Rte.EB_INTREQ_Rte_0002

Description	Timeout monitoring for intra-ECU transmission acknowledgment not supported for non-blocking calls. Timeout monitoring for intra-ECU transmission acknowledgment is only supported in connection with a wait point and not with non-blocking Rte_Feedback and Rte_IFeedback calls.
Rationale	Without using a wait point, the implementation would have to use alarm callbacks to realize the timeout detection. Alarm callbacks shall only be used in scalability classes 1 (OS242).

5.2.3.4.2. lim.Rte.EB_INTREQ_Rte_0006

Description	The port interface mapping is not applied for inter-ECU communication if data elements are directly mapped to Com signals. A port interface mapping is specified for a

	connector or for a chain of connectors from a provide port to a require port where both ports are located on different ECUs. In this case the port interface mapping cannot be applied if the data elements are directly mapped to COM signals. The port interface mapping is only applied if the data elements are mapped to system signals in the system description.
Rationale	If data elements are directly mapped to COM signals, the information about the port interface mapping gets lost. In this case, the integrator must map the signals by hand according to the specified port interface mapping. Only if the system signal mapping is used, the Rte can calculate which system signal corresponds to which data element on the basis of the port interface mapping.

5.2.3.4.3. lim.Rte.EB_INTREQ_Rte_0007

Description	Rte_IsUpdated might return true even if no data has been updated since last read. If the runnable that calls Rte_Write()/Rte_IWrite can interrupt the runnable that calls Rte_Read(), it might happen that the update flag is set, although the receiver has already read the updated data.
Rationale	Due to performance reasons, no data consistency mechanisms are applied for the sequence of writing data and setting the update flag.

5.2.3.4.4. lim.Rte.EB_INTREQ_Rte_0009

Description	Mode disabling dependencies for queued ExternalTriggerOccurredEvents are not supported. If a RteTriggerSourceQueueLength greater than 1 is configured for a Trigger port, the Rte requires that no mode disabling dependencies are configured for the corresponding ExternalTriggerOccurredEvents.
Rationale	The queueing of trigger events requires that all runnable entities be executed.

5.2.3.4.5. lim.Rte.EB_INTREQ_Rte_0011

Description	No data consistency for the buffer initialization during Rte_Start/Rte_Restart is guaranteed. The user must guarantee that each partition is started/restarted from a high priority task so that no data corruption can occour during the startup/restart phase.
Rationale	The task context of Rte_Start/Rte_Restart is not known by the Rte. Using a data consistency mechanism here would also require a lock in all other APIs which are accessing that buffer. This would have major drawbacks for the overall runtime performance of the system.

5.2.3.4.6. lim.Rte.EB_INTREQ_Rte_0012

Description	Communication timeout monitoring is not supported for Basic software mode switch acknowledgement request. The communication timeout for Basic software mode switch acknowledgement request is not supported.
Rationale	

5.2.3.4.7. lim.Rte.EB_INTREQ_Rte_0013

Description	Rte_LdComCbkTriggerTransmit callbacks must be scheduled in a way that they do not interrupt data send preparations (e.g. transformer execution or buffer copy operations) for the corresponding LdCom_Transmit. Otherwise Rte_LdComCbkTriggerTransmit will return E_NOT_OK.
Rationale	

5.2.3.4.8. lim.Rte.EB_INTREQ_Rte_0014

Description	If inter-ECU C/S communication is used with meta data and the LdCom IF API, direct data provision must be guaranteed for the response signal of the server. Thus, the lower communication interface must directly copy the data when LdCom_Transmit() is called.
Rationale	If triggered data provision is used, the Rte would need to wait after each server response until the signal is completly transmitted. Since no reliable mechanism is currently available to detect the end or an error of a signal transmission, only direct data provision is supported.

5.2.3.4.9. lim.Rte.EB_INTREQ_Rte_0015

Description	If inter-ECU C/S communication is used with meta data and the LdCom TP API, it must be ensured that the server only produces a result after the transmission of the last response signal has finished.
Rationale	Similar to lim.Rte.EB_INTREQ_Rte_0014, a response signal might get lost if the transmission is ongoing and the Tx buffer is still locked.

6. Bibliography

Bibliography

[1] *AUTOSAR Specification of RTE Software*, Issue Version 3.2.0, Release 4.0.3, Revision 0003, Publisher: AUTOSAR

[2] *AUTOSAR Specification of Communication*, Issue Version 4.2.0, Release 4.0.3, Revision 0003, Publisher: AUTOSAR

[3] *EB tresos AutoCore OS documentation*, Publisher: Elektrobit Automotive GmbH

[4] *AUTOSAR Software Component Template*, Issue Version 3.1.0, Release 3.1, Revision 0001, Publisher: AUTOSAR

[5] *AUTOSAR General Specification of Transformers*, Issue Release 4.2.1, Revision 0001, Publisher: AUTOSAR

[6] *AUTOSAR Specification of RTE Software*, Issue AUTOSAR CP Release 4.3.1, Publisher: AUTOSAR