Document Title	Requirements on Runtime Environment
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	83

Document Status	published
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	R24-11

Document Change History

Date	Release	Changed by	Description
2024-11-27	R24-11	AUTOSAR Release Management	• Editorial changes
2023-11-23	R23-11	AUTOSAR Release Management	• Removed PartitionRestart • Set Requirements related to RTE_Implementation_Plug-ins and ClassicPlatformFlexibility to valid • Added chapter "Change history of AUTOSAR traceable items"
2022-11-24	R22-11	AUTOSAR Release Management	• Removed Support for Compiler Abstraction
2021-11-25	R21-11	AUTOSAR Release Management	• No content changes
2020-11-30	R20-11	AUTOSAR Release Management	• Added support for ClassicPlatfrom Flexibility [SRS_Rte_00318] - [SRS_Rte_00321] • Added missing requirement for MetaData support [SRS_Rte_00322] • Data filtering on sender side

△

2019-11-28	R19-11	AUTOSAR Release Management	• RTE Implementation Plug-ins: Set [SRS_Rte_00300] - [SRS_Rte_00317] to type valid • Extended Serialization for Data Structures in SOME/IP with tag/length/value encoding (TLV): Set [SRS_Rte_00261] to valid • Category 2 interrupts and RunnableEntities • Changed Document Status from Final to published
2018-10-31	4.4.0	AUTOSAR Release Management	• Added support for RTE Implementation Plug-ins: [SRS_Rte_00300] - [SRS_Rte_00317] • Added support for Extended Serialization for Data Structures in SOME/IP with tag/length/value encoding (TLV): [SRS_Rte_00261]
2017-12-08	4.3.1	AUTOSAR Release Management	• Editorial changes
2016-11-30	4.3.0	AUTOSAR Release Management	• Added support for ExtendedBufferAccess: [SRS_Rte_00254], [SRS_Rte_00255], [SRS_Rte_00256], [SRS_Rte_00257], [SRS_Rte_00258], [SRS_Rte_00259], [SRS_Rte_00260]
2015-07-31	4.2.2	AUTOSAR Release Management	• Added requirement: [SRS_Rte_00253]
2014-10-31	4.2.1	AUTOSAR Release Management	• Added support for concepts: – NVDataHandlingRTE: [SRS_Rte_00245] – EfficientCOMforLargeData: [SRS_Rte_00246] – SenderReceiverSerialization: [SRS_Rte_00247], [SRS_Rte_00248], [SRS_Rte_00249], [SRS_Rte_00250], [SRS_Rte_00251] • Added requirement: [SRS_Rte_00252]

▽

△

2013-10-31	4.1.2	AUTOSAR Release Management	• Removed requirement: [SRS_Rte_00125]
2013-03-15	4.1.1	AUTOSAR Release Management	• Added support for concepts: – Enhance Port Compatibility: [SRS_Rte_00236] – Refined Scheduling of Runnables: [SRS_Rte_00237] – Provide Activating RTE Event: [SRS_Rte_00238] – Enhanced BSW Allocation: [SRS_Rte_00241], [SRS_Rte_00242], [SRS_Rte_00243] – Rapid Prototyping Support for AUTOSAR ECUs: [SRS_Rte_00244] • Added requirements: [SRS_Rte_00239], [SRS_Rte_00240] • Changed requirements: [SRS_Rte_00084]
2011-12-22	4.0.3	AUTOSAR Release Management	• Changed requirements: [SRS_Rte_00155], [SRS_Rte_00154] • Added requirements: [SRS_Rte_00234], [SRS_Rte_00235]
2011-04-15	4.0.2	AUTOSAR Release Management	• Changed requirements: [SRS_Rte_00210], [SRS_Rte_00020]
2009-12-18	4.0.1	AUTOSAR Release Management	• Added support for concepts: – AUTOSAR Scheduler harmonization – RTE API enhancement – Triggered Event – Enhance Measurement and Calibration – Avoidance of duplicated Type Definitions – Integrity and Scaling at ports – Implicit Communication Enhancement – A2L Generation Support – Support of large data types

▽

▽

			– Fixed Data Exchange – Variant Handling – Time Determinism – DLT Concept – Memory related Concepts – Build System Enhancement – Multi Core Architectures – Memory Partitioning – Error Handling – VMM AMM Concept • Legal disclaimer revised
2009-02-04	3.1.2	AUTOSAR Release Management	• Changed requirement: [SRS_Rte_00005] • Removed requirement: [SRS_Rte_00044]
2008-08-13	3.1.1	AUTOSAR Release Management	• Legal disclaimer revised
2007-12-21	3.0.1	AUTOSAR Release Management	• Document meta information extended • Small layout adaptations made
	2.1.1	AUTOSAR Release Management	• "Advice for users" revised • "Revision Information" added
2006-11-28	2.1	AUTOSAR Release Management	• Added requirements: [SRS_Rte_00153], [SRS_Rte_00154], [SRS_Rte_00155], [SRS_Rte_00156], [SRS_Rte_00157], [SRS_Rte_00158], [SRS_Rte_00159], [SRS_Rte_00160], [SRS_Rte_00161] • Changed requirement: [SRS_Rte_00151] • Legal disclaimer revised

△

2006-05-16	2.0	AUTOSAR Release Management	• Added requirement: [SRS_Rte_00152] • Changed requirements: [SRS_Rte_00133], [SRS_Rte_00013], [SRS_Rte_00077], [SRS_Rte_00075] • Removed requirement: [SRS_Rte_00136] • Date format changed to dd-mm-yyyy
2005-05-31	1.0	AUTOSAR Release Management	• Initial release

Disclaimer

Contents

1 Scope of this document 9
1.1 Document Conventions . . . 9
2 Functional Overview 10
3 Requirements Tracing 11
4 Requirements on RTE 13
4.1 Functional Requirements . . . 13
4.1.1 Interaction with AUTOSAR OS . . . 13
4.1.2 Interaction with AUTOSAR COM . . . 16
4.1.3 Interaction with Application Software Components . . . 20
4.1.4 Interaction with Basic Software Components . . . 31
4.1.5 Generation of the BSW Scheduler . . . 35
4.1.6 Support for Measurement and Calibration . . . 42
4.1.7 General Requirements . . . 45
4.1.8 VFB Tracing . . . 73
4.1.9 Application Software Component Initialization and Finalization 76
4.1.10 API . . . 78
4.1.11 C/C++ API . . . 90
4.1.12 Initialization and Finalization Operation . . . 90
4.1.13 Partition Restarting and Termination . . . 91
4.1.14 Fault Operation . . . 91
4.1.15 RTE Implementation Plug-Ins . . . 92
4.2 Non-Functional Requirements . . . 102
4.2.1 General Requirements . . . 102
5 Change history of AUTOSAR traceable items 103
5.1 Traceable item history of this document according to AUTOSAR Release R24-11 . . . 103
5.1.1 Added Requirements in R24-11 . . . 103
5.1.2 Changed Requirements in R24-11 . . . 103
5.1.3 Deleted Requirements in R24-11 . . . 103
5.2 Traceable item history of this document according to AUTOSAR Release R23-11 . . . 103
5.2.1 Added Requirements in R23-11 . . . 103
5.2.2 Changed Requirements in R23-11 . . . 103
5.2.3 Deleted Requirements in R23-11 . . . 104
5.3 Traceable item history of this document according to AUTOSAR Release R22-11 . . . 104
5.3.1 Added Requirements in R22-11 . . . 104
5.3.2 Changed Requirements in R22-11 . . . 104
5.3.3 Deleted Requirements in R22-11 . . . 104

References

[1] Standardization Template
AUTOSAR_FO_TPS_StandardizationTemplate

[2] Requirements on Standardization Template
AUTOSAR_FO_RS_StandardizationTemplate

[3] Virtual Functional Bus
AUTOSAR_CP_EXP_VFB

[4] Software Component Template
AUTOSAR_CP_TPS_SoftwareComponentTemplate

[5] Basic Software Module Description Template
AUTOSAR_CP_TPS_BSWModuleDescriptionTemplate

[6] Requirements on Mode Management
AUTOSAR_CP_RS_ModeManagement

[7] Specification of Memory Mapping
AUTOSAR_CP_SWS_MemoryMapping

[8] Specification of Platform Types for Classic Platform
AUTOSAR_CP_SWS_PlatformTypes

[9] General Requirements on Basic Software Modules
AUTOSAR_CP_RS_BSWGeneral

[10] Specification of Diagnostic Log and Trace
AUTOSAR_CP_SWS_DiagnosticLogAndTrace

1 Scope of this document

The goal of AUTOSAR and of this document is to define the requirements and behavior of the AUTOSAR Run-time environment.

It is not within the remit of AUTOSAR to consider how the RTE is implemented but however all requirements and behavioral specifications are reviewed internally to ensure that at least one feasible implementations is possible.

1.1 Document Conventions

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

The verbal forms for the expression of obligation specified in [TPS_STDT_00053] shall be used to indicate requirements, see Standardization Template, chapter Support for Traceability ([1]).

2 Functional Overview

The Run-Time Environment (RTE) is at the heart of the AUTOSAR ECU architecture. The RTE is the realization (for a particular ECU) of the interfaces of the AUTOSAR Virtual Function Bus (VFB) and thus provides the infrastructure services for communication between Application Software Components as well as facilitating access to basic software components including the OS.

Application Software Components contain system software that is CPU and location independent. This means that, subject to constraints imposed by the system designer, an Application Software Component can be mapped to any available ECU during system configuration. The RTE is responsible for ensuring that components can communicate and that the system continues to function as expected wherever the components are mapped.

The RTE encompasses both the *variable elements* of the system infrastructure that arise from the different mappings of components to ECUs as well as *standardized* RTE services. The RTE is generated and/or configured for each ECU to ensure that the RTE is optimal for the ECU.

3 Requirements Tracing

The following table references the requirements specified in [2] and links to the fulfillments of these.

Requirement	Description	Satisfied by
[RS_BRF_00057]	AUTOSAR shall define a memory mapping mechanism	[SRS_Rte_00169] [SRS_Rte_00170]
[RS_BRF_01024]	AUTOSAR shall provide naming rules for public symbols	[SRS_Rte_00164] [SRS_Rte_00165] [SRS_Rte_00166] [SRS_Rte_00167] [SRS_Rte_00168] [SRS_Rte_00252]
[RS_BRF_01136]	AUTOSAR shall support variants of configured BSW data resolved after system start-up	[SRS_Rte_00191] [SRS_Rte_00201] [SRS_Rte_00202] [SRS_Rte_00203] [SRS_Rte_00204] [SRS_Rte_00206] [SRS_Rte_00207] [SRS_Rte_00229]
[RS_BRF_01160]	AUTOSAR shall support BSW distribution on multi-core MCUs	[SRS_Rte_00241] [SRS_Rte_00242] [SRS_Rte_00243]
[RS_BRF_01216]	AUTOSAR OS shall support to synchronize ScheduleTables to an outside time source	[SRS_Rte_00232]
[RS_BRF_01240]	AUTOSAR OS shall support communication between OSApplications	[SRS_Rte_00210]
[RS_BRF_01248]	AUTOSAR OS shall support to terminate and restart OSApplications	[SRS_Rte_00196]
[RS_BRF_01304]	AUTOSAR RTE shall support broadcast communication	[SRS_Rte_00179] [SRS_Rte_00183]
[RS_BRF_01316]	AUTOSAR RTE shall support data transformation transparent to the Software Components	[SRS_Rte_00247] [SRS_Rte_00248] [SRS_Rte_00249] [SRS_Rte_00250] [SRS_Rte_00251] [SRS_Rte_00253]
[RS_BRF_01320]	AUTOSAR RTE shall schedule SWC and BSW modules	[SRS_Rte_00049] [SRS_Rte_00116] [SRS_Rte_00211] [SRS_Rte_00212] [SRS_Rte_00213] [SRS_Rte_00214] [SRS_Rte_00215] [SRS_Rte_00216] [SRS_Rte_00217] [SRS_Rte_00218] [SRS_Rte_00219] [SRS_Rte_00220] [SRS_Rte_00221] [SRS_Rte_00222] [SRS_Rte_00229] [SRS_Rte_00230]
[RS_BRF_01328]	AUTOSAR RTE shall support scheduling of executable entities on defined events	[SRS_Rte_00162] [SRS_Rte_00163] [SRS_Rte_00216] [SRS_Rte_00230] [SRS_Rte_00235]
[RS_BRF_01376]	AUTOSAR RTE shall support automatic re-scaling and conversion of port data elements	[SRS_Rte_00181] [SRS_Rte_00182]
[RS_BRF_01384]	AUTOSAR RTE shall support automatic range checks of data	[SRS_Rte_00180]
[RS_BRF_01392]	AUTOSAR RTE shall support a bypass implementation	[SRS_Rte_00244]
[RS_BRF_01393]	AUTOSAR RTE shall support a bypass selectable after generation of an ECU image	[SRS_Rte_00254] [SRS_Rte_00255] [SRS_Rte_00256] [SRS_Rte_00257] [SRS_Rte_00258] [SRS_Rte_00259] [SRS_Rte_00260]
[RS_BRF_01394]	AUTOSAR shall support a memory interface for RTE-managed buffer access	[SRS_Rte_00255] [SRS_Rte_00256] [SRS_Rte_00257] [SRS_Rte_00258] [SRS_Rte_00259] [SRS_Rte_00260]

△

Requirement	Description	Satisfied by
[RS_BRF_01560]	AUTOSAR communication shall support mapping of signals into transferrable protocol data units	[SRS_Rte_00251]
[RS_BRF_01568]	AUTOSAR communication stack shall support fixed size and dynamic size signals	[SRS_Rte_00190]
[RS_BRF_01616]	AUTOSAR communication shall support initial values for signals	[SRS_Rte_00184]
[RS_BRF_01649]	AUTOSAR communication shall support communication of large and dynamic data in a dedicated optimized module	[SRS_Rte_00246]
[RS_BRF_01816]	AUTOSAR non-volatile memory functionality shall organize persistent data based on logical memory blocks	[SRS_Rte_00176] [SRS_Rte_00177] [SRS_Rte_00178] [SRS_Rte_00228] [SRS_Rte_00245]
[RS_BRF_02056]	AUTOSAR OS shall support timing protection	[SRS_Rte_00193]
[RS_BRF_02272]	AUTOSAR shall offer tracing of application software behavior	[SRS_Rte_00003] [SRS_Rte_00004] [SRS_Rte_00005] [SRS_Rte_00008] [SRS_Rte_00045] [SRS_Rte_00192]
[RS_Main_00150]	AUTOSAR shall support the deployment and reallocation of AUTOSAR Application Software	[SRS_Rte_00318] [SRS_Rte_00319] [SRS_Rte_00321]
[RS_Main_00200]	Resource Efficiency	[SRS_Rte_00300] [SRS_Rte_00301] [SRS_Rte_00302] [SRS_Rte_00303] [SRS_Rte_00304] [SRS_Rte_00305] [SRS_Rte_00306] [SRS_Rte_00307] [SRS_Rte_00309] [SRS_Rte_00310] [SRS_Rte_00311] [SRS_Rte_00312] [SRS_Rte_00313] [SRS_Rte_00314] [SRS_Rte_00315] [SRS_Rte_00316] [SRS_Rte_00317] [SRS_Rte_00318] [SRS_Rte_00319] [SRS_Rte_00320] [SRS_Rte_00321]
[RS_Main_00280]	Standardized Automotive Communication Protocols	[SRS_Rte_00261] [SRS_Rte_00322]

Table 3.1: Requirements Tracing

4 Requirements on RTE

4.1 Functional Requirements

4.1.1 Interaction with AUTOSAR OS

The requirements in this section all concern how the RTE interacts with the AUTOSAR OS. The AUTOSAR ECU architecture defines all interactions to occur over a *standardized interface*.

[SRS_Rte_00020] Access to OS ⌈

Description:	The RTE shall abstract the features of Os from AUTOSAR Software Components. Only the access to the service interface of the RTE shall be available for AUTOSAR Software Components directly.
Rationale:	The Application Software Components are intended to be Os independent and therefore should not access any particular Os function directly, except for the service interface. For example, the RTE uses task-based functionality (tasks, resources, events, .) to provide Runnable Entity functionality to the application. The existence of OS tasks is not made visible to the application.
Dependencies:	[SRS_Rte_00025]
Use Case:	The Os offers a standardized interface. This interface is accessed by Software Components only via the RTE API and hence access is controlled by the RTE. The Os offers a service interface. This is directly accessible by the Software Components.
Supporting Material:	Specification of the Virtual Functional Bus [3] The AUTOSAR ECU architecture defines a standardized interface for the OS and an AUTOSAR interface for Application Software Components and therefore there can be no direct interaction.

⌋

[SRS_Rte_00099] Decoupling of interrupts ⌈

Description:	The RTE shall not permit category 1 interrupt context to be propagated to Application Software Components. To ensure low latency times and determinism, the interrupt context may have to be propagated to the RTE.
Rationale:	If Application Software Components were able to execute within an category 1 interrupt context they would be able to block the system schedule for unacceptably long periods of time.
Dependencies:	–
Use Case:	The RTE 'intercepts' interrupts and enables a Runnable Entity to handle the notification. The Runnable Entity executes in the context of a task.

▽

△

Supporting Material:	Specification of the Virtual Functional Bus [3] In this requirement, blocking meant to indicate that the RTE shall not suspend (Running->Waiting) the thread of control executing the callback. It is not meant to indicate that the thread cannot be pre-empted. i.e. blocking is "suspended" but not "pre-empted"

⌋

[SRS_Rte_00036] Assignment to OS Applications ⌈

Description:	When partitioning is in use, the RTE generator shall reject configurations where the Runnable Entities of one Software Component instance are not assigned to tasks within the same OS-Application.
Rationale:	All objects (e.g. resources, alarms) which belong to one OS-Application have access to each other - the OS will kill tasks that attempt direct access without being mapped to the same OS application.
Dependencies:	[SRS_Rte_00018]
Use Case:	Efficient access is required - if mapped to different OS applications then the RTE would be required to implement the protection mode switches which would have a significant impact on efficiency.
Supporting Material:	Where memory protection is used the tasks mapped for a component instance form a single OS Application - this permits intra-component interactions to occur with minimum overhead.

⌋

[SRS_Rte_00049] Construction of task bodies

Upstream requirements: RS_BRF_01320

⌈

Description:	The RTE generator shall construct task/ISR2 bodies to execute Runnable Entities and Basic Software Schedulable Entities in a form suitable for the AUTOSAR OS - this will typically be as a function exported with C linkage. The SW-Component description declares the Runnable Entities present in a component. The Basic Software Module description declares the Basic Software Schedulable Entities present in a BSW Module.
Rationale:	The mapping of Runnable Entities and Basic Software Schedulable Entities to tasks/ISR2s forms part of the input to the generator. Automatic mapping is too complex a task (and insufficient data is present in the input) to be considered as part of AUTOSAR at this stage.
Dependencies:	[SRS_Rte_00219]

▽

△

Use Case:	Runnable Entities and Basic Software Schedulable Entities in a sequence mapped to the same task. It is possible to provide • tasks/ISR2s with only Runnable Entities • tasks/ISR2s with support for interlaced execution of Runnable Entites and Basic Software Schedulable Entities • tasks/ISR2s with only Basic Software Schedulable Entities.
Supporting Material:	–

⌋

[SRS_Rte_00193] Support for Runnable Entity execution chaining

Upstream requirements: RS_BRF_02056

⌈

Description:	The RTE Generator shall allow Runnable Entity monitoring by mapping of the monitored Runnable Entities to one or several OS Tasks and - in case of the usage of several OS Tasks - chain the execution of these OS Tasks.
Rationale:	In order to monitor the execution time of Runnable Entities per individual (or several) Runnable Entity, the mechanism of task chaining shall be used so the monitoring can be implemented using OS Task monitoring. The RTE shall be able to activate (chain) the execution of the chained tasks in order to get the desired Runnable Entity execution order.
Dependencies:	–
Use Case:	Runnable Entities which are configured to be executed sequentially in one OS Task shall be able to be spilt to several OS Tasks in order to apply OS execution time monitoring on a fine grained level (individual Runnable Entity or several Runnable Entities). To get the same behavior the chain task mechanism shall be applied.
Supporting Material:	–

⌋

[SRS_Rte_00210] Support for inter OS application communication

Upstream requirements: RS_BRF_01240

⌈

Description:	RTE shall support the communication between Software Component instance that are allocated to different OS applications, where different OS applications may be located on different memory partitions and/or cores.

▽

△

Rationale:	As Software Component instances from different OS applications may be located on different cores and different memory partitions, the communication between them may require the use of dedicated communication and signaling methods like the use of a Cross OS Application Communication Module.
Dependencies:	[SRS_Rte_00011]
Use Case:	Multi core support, memory partitioning support
Supporting Material:	–

⌋

4.1.2 Interaction with AUTOSAR COM

The requirements in this section all concern how the RTE interacts with the AUTOSAR COM. The AUTOSAR ECU architecture defines all interaction to occur over a *standardized interface*.

[SRS_Rte_00068] Signal initial values ⌈

Description:	The RTE generator shall ensure that signals for which an INIT_VALUE is specified are initialized.
Rationale:	Data can be read before COM or Efficient COM for large data has provided a first value and applications should be prevented from reading uninitialized data.
Dependencies:	[SRS_Rte_00108] The INVALIDATE attribute can be used in conjunction with an INIT_VALUE to indicate to an Application Software Component that no data has been received since COM or Efficient COM for large data or the RTE started. The INVALIDATE attribute shall be initialized too.
Use Case:	–
Supporting Material:	Specification of the Virtual Functional Bus [3]

⌋

[SRS_Rte_00069] Communication timeouts ⌈

Description:	The RTE generator shall include run-time checks for monitoring timeouts specified in the ECU Configuration for blocking communication. When synchronous intra-task client server communication is optimized to a direct function call, no timeout can occur though clients can still be written to expect a timeout were the configuration to change. Therefore this requirement does not apply when the synchronous client server call is optimized to a direct function call.

▽

△

Rationale:	Prevent infinite blocking of receivers.
Dependencies:	[SRS_Rte_00147]
Use Case:	A Runnable Entity performs a blocking "read" of a data item. A blocking read will wait forever if no data arrives unless a timeout is applied.
Supporting Material:	Specification of the Virtual Functional Bus [3] - timeouts are required within components to prevent infinite blocking and thus apply both to inter-ECU communication (that uses COM or Efficient COM for large data) and intra-ECU communication (that may or may not use COM or Efficient COM for large data).

⌋

[SRS_Rte_00073] Atomic transport of Data Elements ⌈

Description:	The RTE shall ensure that the transmission and reception of data elements (regardless whether they are simple or composite), and all arguments of a single RTE operation are treated as atomic units. Where a parameter is passed by reference rather than by value the RTE is forced to rely on the component not modifying the target of the reference while the parameter is in use by the RTE.
Rationale:	–
Dependencies:	[SRS_Rte_00032] - data consistency. This should not be read as requiring COM to treat the transmission as atomic - it (or a lower layer in the COM stack) shall remain free to split across multiple (network) frames as long as the split is not visible to the RTE.
Use Case:	Elements of a record cannot be handled separately by the Application Software Component but, instead, the whole record should be treated as a single atomic unit.
Supporting Material:	Specification of the Virtual Functional Bus [3] Software Component Template [4]

⌋

[SRS_Rte_00082] Standardized communication protocol ⌈

Description:	The RTE shall define and implement the protocol (e.g. message sequences) for inter-ECU client-server communication. For communication mechanisms that are not directly provided by the AUTOSAR COM layer (e.g. client-server communication) a standardized protocol that is implemented by every AUTOSAR RTE has to be defined.
Rationale:	This ensures that RTEs of different vendors are interoperable.
Dependencies:	[SRS_Rte_00062]
Use Case:	If C/S is mapped to paired COM or Efficient COM for large data message channels then both RTEs shall implement the same mapping to be compatible.
Supporting Material:	[SRS_Rte_00091] - common protocol for COM or Efficient COM for large data transmissions.

⌋

[SRS_Rte_00091] Inter-ECU Marshalling ⌈

Description:	The RTE shall use a common format for transmitting/receiving data elements or parameters of operations between ECUs. On transmission the (target specific) signal data shall be converted to the common format and the reverse operation performed on reception. It shall be possible to exchange record types between components written in different programming languages.
Rationale:	The RTE is responsible for ensuring that data elements or parameters of operations (e.g. records, parameter lists, .) can be sent between ECUs. Since each ECU may define signals differently in memory a straight transmission cannot be performed and, instead, the sender shall convert the data elements or parameters to a common format before transmission and the reverse transformation shall be performed by the receiving RTE. A common communication protocol enables RTEs from different vendors to interoperate.
Dependencies:	[SRS_Rte_00082] - defines common message sequence for client-server.
Use Case:	–
Supporting Material:	Software Component Template [4] The term "Marshalling" is used synonymously to "Serialization".

⌋

[SRS_Rte_00181] Conversion between internal and network data types

Upstream requirements: RS_BRF_01376

⌈

Description:	The RTE shall generate conversion routines - when configured - for different data types for data elements used in sender-receiver communication within one ECU (high resolution) and data elements used between ECUs (low resolution) to save serial data link bandwidth and non-productive development work.
Rationale:	Save bandwidth in inter-ECU communication.
Dependencies:	–
Use Case:	Within one ECU the SWCs want to use data types with high resolutions in computations to get a small epsilon (computational deviation). When the same data is transported to another ECU it is converted into the network representation on the sender side (with loss of precision). On the receiver side the data is converted back to the original type.
Supporting Material:	–

⌋

[SRS_Rte_00246] Support of Efficient COM for large data

Upstream requirements: RS_BRF_01649

⌈

Description:	RTE shall support more than one Interaction layer module. Currently COM and Efficient COM for large data are supported.
Rationale:	Besides the "traditional" COM a lean module shall be supported giving the option to omit either of them in case they are not needed.
Dependencies:	–
Use Case:	An ECU has many Signals of large and dynamic size. The usage of Efficient COM for large data will save resources in the communication stack (e.g. buffers). Another ECU has only large and dynamic signals being transmitted on request. This ECU may omit COM if no use case out of COM is needed.
Supporting Material:	–

⌋

[SRS_Rte_00251] Array based signal group handling with Com

Upstream requirements: RS_BRF_01316, RS_BRF_01560

⌈

Description:	The Rte shall use the uint8-array API to pass the serialized representation of a composite data to COM when this configured.
Rationale:	The AUTOSAR transformer chain provides means to serialize composite data into a uint8-array representation. This serialized uint8-array shall be passed as one entity to COM.
Dependencies:	–
Use Case:	Usage of transformer with Com-based serialization and Com Interaction to enable the communication with a fixed communication matrix.
Supporting Material:	–

⌋

[SRS_Rte_00322] Support of Metadata

Upstream requirements: RS_Main_00280

⌈

Description:	The Rte shall support (transparently) forwarding meta data from/to (Ld-)COM and SWCs if configured.
Rationale:	MetaData is used for the interaction with COM or LdCom to provide additional information about the actual payload of a PDU instance. This can be e.g. addresses of client ECUs forwarded transparently, allowing to keep RTE bus agnostic.
Dependencies:	–
Use Case:	Distinguishing C/S requests/responses originating from different client(ECUs).
Supporting Material:	–

⌋

4.1.3 Interaction with Application Software Components

Includes Application Software Components, sensor components and actuator components.

[SRS_Rte_00011] Support for multiple Application Software Component instances. ⌈

Description:	The RTE shall support multiple instances of the same Application/Sensor/Actuator Software Component type mapped to the same ECU.
Rationale:	Repetition of the same Application Software Component type on an ECU to promote component reuse.
Dependencies:	Name space - rules for "instantiating" an application / sensor / actuator have to be defined. [SRS_Rte_00210]
Use Case:	• Having one Application Software Component type for the window lifter which is instantiated four times in a vehicle. • Homogenous SW redundancy is a basic/simple solution for increasing fault tolerance.
Supporting Material:	Software Component Template [4]

⌋

[SRS_Rte_00012] Multiple instantiated AUTOSAR software components delivered as binary code shall share code ⌈

Description:	The RTE generator shall implement multiple instantiations (delivered as binary code) of the AUTOSAR software component type (on the same ECU) through sharing the same Application Software Component code for all instances. Source code deliverables can be instantiated either by code sharing or code duplication. This depends on the implementation of the RTE.
Rationale:	1) VFB metamodel. 2) Requirement to minimise code space. 3) Cannot modify binary-code software components and therefore all instances must use the same object-code.
Dependencies:	[SRS_Rte_00133]
Use Case:	–
Supporting Material:	Software Component Template [4] Code that is to be shared between instances should be re-entrant because of the pre-emptive environment it which it will run. Re-entrant code is indicated by the "supportsMultipleInstantiation" flag in the software component description.

⌋

[SRS_Rte_00013] Per-instance memory ⌈

Description:	The RTE shall provide per-instance memory to Application Software Components, where each Application Software Component instance has its own copy of memory, not shared with other instances of the same Application Software Component type.
Rationale:	Require variables with lifetime greater than that of a Runnable Entity but remain protected from different instances of the same Software Component type. RTE shall not provide memory shared between instances of an Application Software Component type. Also required for multiple instance support.
Dependencies:	[SRS_Rte_00075] - API for accessing per-instance memory
Use Case:	–
Supporting Material:	Software Component Template [4]

⌋

[SRS_Rte_00077] Instantiation of per-instance memory ⌈

Description:	The RTE generator shall instantiate each per-instance memory section of a software component according to the attributes given in its software component description. The instantiation of per-instance memory shall be either derived from input information or instantiated automatically by the RTE generator (where such information is not available). In the latter case the address of the per-instance memory is assigned by the RTE generator and therefore is not subject to control by the system integrator.
Rationale:	Required by the Software Component Template
Dependencies:	–
Use Case:	RAM mirror from NVRAMManager.
Supporting Material:	The per-instance memory presented as input to the RTE generator shall be uniquely identifiable. Software Component Template [4]

⌋

[SRS_Rte_00017] Rejection of inconsistent component implementations ⌈

Description:	The RTE generator shall ensure that the compiler can detect (and reject) access to undefined RTE API calls. The RTE generator is required to reject "invalid" configurations, part of this is rejecting invalid APIs (i.e. calls to an unknown port at compile time).
Rationale:	–
Dependencies:	–
Use Case:	The component description defines the names of the ports that a component "requires" and "provides" and the associated interfaces. The RTE generator can then define only the valid API calls. For example, consider a component that has a Port ‘p1’ with a data items ‘a’ and ‘b’. In this case, the RTE generator will only create an API for the send of data items ‘a’ and ‘b’. Thus an attempt by the component to send any invalid data item, e.g. ‘c’ on the interface shall be detected by the compiler and a warning issued.
Supporting Material:	Software Component Template [4]

⌋

[SRS_Rte_00134] Runnable Entity categories supported by the RTE ⌈

Description:	The RTE shall support the Runnable Entity categories 1a, 1b and 2 Runnable Entity category: • 1a) The Runnable Entity is only allowed to use implicit reading (DataReadAccess) and writing (DataWriteAcess). A category 1a Runnable Entity cannot block and cannot use explicit read/write. • 1b) The Runnable Entity can use explicit reading and writing (DataReadAccess). A category 1b Runnable Entity cannot block. Implicit read/write is also allowed. • 2) The Runnable Entity may use explicit reading/writing including blocking behavior.
Rationale:	Support several kinds of runnable entity implementation kinds.
Dependencies:	[SRS_Rte_00128], [SRS_Rte_00129] - Implicit reception and transmission. [SRS_Rte_00098] - Explicit transmission.
Use Case:	It is easier to reason about time behavior for category 1a Runnable Entities that do not invoke RTE API calls that (may) not execute in constant time.
Supporting Material:	Software Component Template [4]

⌋

[SRS_Rte_00072] Activation of Runnable Entities ⌈

Description:	The RTE shall start/resume a Runnable Entity according to the RTEEvents to which it is linked.
Rationale:	Activations of Runnable Entities due to arrival of data from other components, invocation of operations of one port or time based execution of Runnable Entities is based on the RTEEvent model [4].
Dependencies:	[SRS_Rte_00160], [SRS_Rte_00161]
Use Case:	Cyclic, time based activation of Runnable Entities; activation of a Runnable Entity due to the arrival of data using the sender-receiver communication pattern.
Supporting Material:	Software Component Template [4]

⌋

[SRS_Rte_00160] Debounced start of Runnable Entities ⌈

Description:	The RTE shall allow the configuration of a debounce start time of Runnable Entities to avoid the same Runnable Entity being executed shortly after each other.

▽

△

Rationale:	In case several RTE Events occur within a short time interval there shall only be a limited amount of executions of the Runnable Entity. It shall be possible to define a minimum time which in which all activations are noticed, but the Runnable Entity will start only after that period has passed.
Dependencies:	[SRS_Rte_00072]
Use Case:	Runnable Entities being activated with along timing period and additionally activated on several DataReceivedEvents. If the Runnable Entity has just been executed the RTE shall wait for the defined period until the Runnable Entity is executed again.
Supporting Material:	–

⌋

[SRS_Rte_00161] Activation offset of Runnable Entities ⌈

Description:	The RTE shall allow the definition of an activation offset of Runnable Entites.
Rationale:	In order to allows optimizations in the scheduling (smooth cpu load, mapping of Runnable Entities with different periods in the same task to avoid data sharing, etc.), the RTE has to handle the activation offset information from a task shared reference point for time trigger Runnable Entites.
Dependencies:	[SRS_Rte_00072]
Use Case:	–
Supporting Material:	–

⌋

[SRS_Rte_00031] Multiple Runnable Entities ⌈

Description:	The RTE shall support multiple Runnable Entities in one Software Component type.
Rationale:	Runnable Entities are used for servers, receivers, feedback, . etc and therefore each component can have many Runnable Entities.
Dependencies:	–
Use Case:	–
Supporting Material:	VFB Metamodel

⌋

[SRS_Rte_00032] Data consistency mechanisms ⌈

Description:	The RTE shall support one or more mechanism for ensuring data consistency within an Application Software Component instance. No direct access to data 'outside' the component instance is possible within AUTOSAR. The scope of the mechanism (e.g. exclusive area) shall be all Runnable Entities (that statically specify the same exclusive area - RTE_IN004) in the software component instance. If consistency between component instances is required then an additional software component can be created to provide appropriate access semantics to the encapsulated data. A side effect of a data consistency mechanism may be to prevent other Runnable Entities in different component instances from executing, for example, the RTE may lock out all interrupts for a short period of time. However this is not deemed to be an illegal (non-AUTOSAR) communication channel since the set of affected Runnable Entities is not defined and therefore cannot be relied upon by a component author.
Rationale:	Multiple Runnable Entities can be active within an Application Software Component and therefore a mechanism shall exist to prevent concurrency conflicts. An Application Software Component cannot access the OS directly and therefore the RTE shall provide the mechanism.
Dependencies:	–
Use Case:	Exclusive areas are an example of a mechanism suitable, e.g.: void RTERunnable_a(RTEInstance self) } . RTEEnter_<region>(self); /* read-modify-write data */ RTEExit_<region>(self); An Application Software component, which server function is concurrently called by several clients: in this case multiple executions of runnable entities (associated to each call, supporting "canBeInvokedConcurrently") usually need to access shared data.
Supporting Material:	VFB Requirements (VFB_C60, Sched70) To permit an exclusive area to affect all instances of a software component type would be incorrect since component instances are independent and would also open a non-AUTOSAR communication channel between the components. Note: The APIs using in this requirement that are mentioned are only examples of how the APIs may look like - the API presented in the SWS is subject to change.

⌋

[SRS_Rte_00046] Support for "Executable Entity runs inside" Exclusive Areas ⌈

Description:	The RTE shall support exclusive areas where a Runnable Entity or a Basic Software Schedulable Entity is declared as "running inside" the Exclusive Area. All Runnable Entities in a SW-Component or Basic Software Schedulable Entities that specify the same "runs inside" Exclusive Area shall be scheduled non preemptively with respect to other Runnable Entities respectively Basic Software Schedulable Entities in the set.
Rationale:	"Runs inside" Exclusive Areas satisfies the requirement from the SW-Component template and Basic Software Module Description template that certain Exclusive Areas can be defined that are automatically entered whenever a Executable Entity is invoked by the RTE / Basic Software Scheduler.
Dependencies:	[SRS_Rte_00032]
Use Case:	–
Supporting Material:	SW-Component Template [4] BSWMD Template [5]

⌋

[SRS_Rte_00142] Support for InterRunnableVariables ⌈

Description:	The RTE shall support InterRunnableVariables. A Software Component shall be able to declare one or more InterRunnableVariables used for data consistency purposes. An InterRunnableVariable is use when several Runnable Entities of the same Software Component instance access the same data item. InterRunnableVariables are used to store data item copies to avoid concurrent Runnable Entity accesses to the one original data item.
Rationale:	InterRunnableVariables satisfy the requirement from the software component template that certain InterRunnableVariables can be defined that can be accessed by Runnable Entities of same software component instance to implement the "variable copies" strategy.
Dependencies:	[SRS_Rte_00032]
Use Case:	Data item write access by Runnable Entity in 10ms task. Several data item read accesses by Runnable Entity in 50ms task. 50ms task can be preempted by 10ms task. Data inconsistency can be prevented if Runnable Entity in 50ms task gets a copy of the original data in an InterRunnableVariable to work on during activation.
Supporting Material:	SW-Component Template [4]

⌋

[SRS_Rte_00033] Serialized execution of Server Runnable Entities ⌈

Description:	The RTE shall support serialized and non-serialized execution of Server Runnable Entities. The RTE shall complete the processing of one serialized service request before it accepts and dispatches the next request for that server. The serialization is applied on the service level, so one server can handle multiple service calls concurrently (this implies that the service's Runnable Entities are mapped to different tasks and there is no shared data between them). If serialization is supported by a server and how big the actual queue is shall be configurable.
Rationale:	A serialized server only accepts and processes requests atomically and thus avoids potential conflicting concurrent access. Invocation of the server's Runnable Entity shall be encapsulated within an exclusive area when simultaneous intra-ECU and inter-ECU execution is possible.
Dependencies:	[SRS_Rte_00032] - Per-instance scope of exclusive areas. [SRS_Rte_00110] - Support for buffering.
Use Case:	The NVRAM Manager is capable of handling multiple requests. If for each stored data item there is a separate port generated during configuration the generic serialization mechanism works fine.
Supporting Material:	The execution of a server is independent of how it is invoked, in particular, whether the call is synchronous or asynchronous is a property of the client and not the server. This requirement explicitly enforces strict serialization of Server Runnable Entities. An RTE generator can optimize a client/server call to a direct function call only if serialization is maintained. This can be done through the insertion of resource locks or other mechanisms.

⌋

[SRS_Rte_00133] Concurrent invocation of Runnable Entities ⌈

Description:	RTE has to allow and support the concurrent invocation of a Runnable Entity (means several activations of same Runnable Entity at same time) for those Runnable Entities whose attribute "canBeInvokedConcurrently" is set to TRUE. The RTE generator shall reject input configurations requiring several concurrent activations of a Runnable Entity when the attribute "canBeInvokedConcurrently" of the Runnable Entity is set to FALSE. Note that this is independent of the Runnable Entities ability to be multiple instantiated or not.
Rationale:	Requirement from SwCT Runnable Entities description
Dependencies:	[SRS_Rte_00012]
Use Case:	Direct client-server calls implementation with Runnable Entity implemented as a server. E.g. needed for Basic-SW services.
Supporting Material:	Software Component Template [4]

⌋

[SRS_Rte_00143] Mode Switches ⌈

Description:	The RTE shall implement the functionality of ModeSwitchEvent and ModeDisablingDependency.
Rationale:	ModeDisablingDependency is the only mean by which AUTOSAR allows to define sets of Runnable Entities that run only in certain modes. ModeSwitchEvent allows to trigger Runnable Entities on the transitions between modes.
Dependencies:	[SRS_Rte_00144]
Use Case:	Use cases are, e.g.: Initialization and finalization phases, different communication modes (telling, whether the SW-C can expect the communication partners of the ports to be available).
Supporting Material:	Software Component Template [4] Requirements on Mode Management [6]

⌋

[SRS_Rte_00176] Sharing of NVRAM data

Upstream requirements: RS_BRF_01816

⌈

Description:	Several Application Software Components shall be able to access the same data - through ports - defined in NvBlockComponentType.
Rationale:	This permits to lower the amount of NVRAM needed on an ECU and to avoid duplication of data and the maintenance of this duplicated data in the engineering processes.
Dependencies:	–
Use Case:	Reuse of common parameters. Reduce NVRAM usage.
Supporting Material:	–

⌋

[SRS_Rte_00180] DataSemantics range check during runtime

Upstream requirements: RS_BRF_01384

⌈

Description:	Provide functionality in the RTE to enable checking the ranges of data element values for both S/R and C/S communication. If specified at SW-Component level a violation shall be reported to the SW-Component and a correction strategy shall be implemented. If configured during ECU Configuration a violation shall be reported to the DET.

▽

△

Rationale:	For integration of SW-Components from different providers the check of the actual contract is needed in order to detect violations. For debugging it is useful to check whether SWCs sending data do provide the data within the specified ranges. In case there is a violation a Development Error may be raised.
Dependencies:	–
Use Case:	Even when a SW-Component specifies in the PortInterface that a certain DataElementPrototype value shall be within 0-100 it is technically possible to send a value of 110. The RTE shall be able to check for such range violations. Range check on sender/client side: No propagation of illegal values through the RTE (local and remote). Reaction in case of error: "Out Of Bounds" error code in the status value. Range check on receiver/server side: Protection against reception of out of bounds values on the receiver side (assuming the sender has not already checked). Reaction in case of error: "Out Of Bounds" error code in the status value.
Supporting Material:	–

⌋

[SRS_Rte_00182] Self Scaling Signals at Port Interfaces

Upstream requirements: RS_BRF_01376

⌈

Description:	When explicitly specified, the RTE shall support the connection of ports whose interfaces have incompatible data types or incompatible data semantics according to the compatibility rules of the SWC-T. The RTE shall allow specifying the scaling of signals for ports on the sender/server side and the receiver/client side to allow automatic re-scaling in the RTE. A deterministic generation of the re-scaling shall be supported by the RTE generator independent of its implementation. Dedicated connector needs to be modeled on VFB level, and based on that the RTE generator has to create the adapter. The RTE generator is not allowed to do it completely on its own.
Rationale:	Avoid writing SWC glue code to interface two different SWCs conversion code provided by the integrator / sub-system designer (e.g. using a COMPU-METHOD). Hence, no recalculation of signal resolution in the affected SWCs is necessary. In order to allow deterministic generation of the re-scaling code additional input information might be need to specify the details of the desired re-scaling.
Dependencies:	–

▽

△

Use Case:	In diagnostics the resolution of signals has to be provided as specified in the ISO document for OBDII (e.g. engine speed). If the RTE could provide these signals in the correct resolution the SWC is not required to do this re-calculation. For the re-scaling of LINEAR to LINEAR data an optional/alternative call-out function from the RTE might be utilized in order to support deterministic conversion formula specification.
Supporting Material:	–

⌋

[SRS_Rte_00236] Support for ModeInterfaceMapping ⌈

Description:	The RTE shall support that ports are connected which are typed by different ModeSwitchInterfaces and where the ModeDeclarationGroupPrototype of the provide port are typed by ModeDeclarationGroup which ModeDeclarations are mapped to ModeDeclarations of the require port. Hereby the number of ModeDeclarations of the require port might be different than the number of ModeDeclarations of the provide port.
Rationale:	Mode Manager and mode User are designed independently from each other.
Dependencies:	–
Use Case:	Receiving Software Component has to be connected to a Mode Manager providing a ModeDeclarationGroup with different ModeDeclaration than the user. In the case that Software Component is reused scenarios happens that the ModeDeclarationGroup used by the ModeManager is incompatible to the ModeDeclarationGroup used by the ModeUser. For instance A) the ModeManager is basically able to differentiate sub-states more fin grained as it is required by the generic ModeUser. But due to the definition of the mode communication it is not possible to use two Mode Switch Ports at the Mode Manager because this would lead to two independent and unsynchronized ModeMachineInstances in the RTE. B) the generic Mode User is basically able to support additionally modes which are not used by all Mode Manager.
Supporting Material:	–

⌋

[SRS_Rte_00237] Time recurrent activation of Runnable Entities ⌈

Description:	The RTE shall support the time recurrent activation of Runnable Entities. The applicable time period shall be definable by the software component type and be overwriteable per component instance.

▽

△

Rationale:	Support closed loop controllers with different time base
Dependencies:	–
Use Case:	Reuse of components in different rasters
Supporting Material:	–

⌋

4.1.4 Interaction with Basic Software Components

[SRS_Rte_00152] Support for port-defined argument values ⌈

Description:	The mechanism of "port-defined argument values", as defined in the AUTOSAR Software Component Template [4], has to be supported.
Rationale:	To allow the interaction of Application Software Components with the infrastructural basic software.
Dependencies:	–
Use Case:	Access of a SW-C to the NVRAM Manager.
Supporting Material:	AUTOSAR Software Component Template [4]

⌋

[SRS_Rte_00022] Interaction with call-backs ⌈

Description:	The RTE shall not suspend execution while executing a call-back.
Rationale:	Blocking COM (e.g. in a call-back) could prevent reception of data and therefore lead to data loss.
Dependencies:	[SRS_Rte_00099] - decoupling of interrupts.
Use Case:	–
Supporting Material:	If the RTE cannot process (e.g. pass the information to a Runnable Entity) the call-back immediately then the information must be queued and processed at a later point. A call-back is not the same as activation of a Runnable Entity.

⌋

[SRS_Rte_00062] Local access to basic software components ⌈

Description:	The RTE shall permit application and basic software components to directly (via RTE) access the AUTOSAR interfaces of basic software components located on the same ECU. The RTE generator shall prevent direct access to the AUTOSAR interfaces of remote basic software components. One exception is the BswM module which is allowed to be accessed on other ECUs as well. Indirect access to the AUTOSAR interfaces of basic software components located on a remote ECU shall be possible via the inclusion of an Application Software Component on the remote ECU to 'export' an appropriate AUTOSAR interface to the basic software component.
Rationale:	This requirement is imposed for two reasons: • Efficiency - remote access to a basic software component permits only the lowest level of optimization. For example, the RTE generated would be unable to take advantage of intra-task access to optimize communication to either a direct function call (client-server) or queue write (sender-receiver). • Control - an ECU integrator can know, a priori, that scheduling will not be affected by components on remote ECUs accessing the basic software and blocking access by local components.
Dependencies:	[SRS_Rte_00018] - rejection invalid configurations.
Use Case:	• On a given ECU, sensor/actuators components are not allowed to communicate with remote ECU abstraction. This means that sensor/actuator SWCs SHALL be mapped to the same ECU to which the sensor/actuator devices are mapped. • Exception: A mode request to the local BswM is also distributed by the RTE to other ECUs which are configured to need the mode request as well.
Supporting Material:	See VFB chapter 4.4.2.2. The location of a service can be implemented by a proxy implemented as an AUTOSAR software component.

⌋

[SRS_Rte_00169] Map code and memory allocated by the RTE to memory sections

Upstream requirements: RS_BRF_00057

⌈

Description:	The RTE shall map its generated code and allocated memory to RTE memory sections.
Rationale:	Enable memory mapping control of the whole ECU, not only BSWM and SWC.
Dependencies:	–
Use Case:	Efficiency of the interactions between SWC Runnable Entities and generated code. Memory Mapping of calibration parameters. Memory partitioning and memory mapping of PIM and other variables in the same partition as the users.
Supporting Material:	–

⌋

[SRS_Rte_00170] Provide used memory sections description

Upstream requirements: RS_BRF_00057

⌈

Description:	The RTE generator shall produce an XML file with a description of memory sections used by the means of the BSW Module template.
Rationale:	Enable memory mapping control of the whole ECU, not only BSWM and SWC.
Dependencies:	[SRS_Rte_00169] - Map code and memory allocated by the RTE to memory sections.
Use Case:	Efficiency of the interactions between SWC Runnable Entities and generated code. Memory Mapping of calibration parameters. Memory partitioning and memory mapping of PIM and other variables in the same partition as the users.
Supporting Material:	Basic Software Module Description template [5]

⌋

[SRS_Rte_00177] Support of NvBlockComponentType

Upstream requirements: RS_BRF_01816

⌈

Description:	The RTE shall support NvBlockComponentType by providing a NvRAM and NvROM block to the NVRAM Manager and access to the data stored in the block.
Rationale:	Allow data to be grouped together in the same NVRAM block to lower the amount of needed NVRAM blocks.
Dependencies:	–
Use Case:	Storage of several small flags in a large NvRAM block.
Supporting Material:	–

⌋

[SRS_Rte_00228] Fan-out NvBlock callback function

Upstream requirements: RS_BRF_01816

⌈

Description:	The RTE shall support the fan-out (take one incoming callback function and distribute it into several callback function calls) of the NvBlock callback function from the NVRAM Manager to multiple Software Component instances which use the corresponding NvBlock.
Rationale:	It is possible to define several users for one NvBlock,but the NVRAM Manager is not able to handle several callback functions. Therefore the callback function has to be fan-out by the RTE.

▽

△

Dependencies:	[SRS_Rte_00177]
Use Case:	Two Software Component Instances accessing the same NvBlock.
Supporting Material:	–

⌋

[SRS_Rte_00233] Generation of the Basic Software Module Description ⌈

Description:	The RTE Generator shall provide a Basic Software Module Description of the actual generated RTE.
Rationale:	The Basic Software Module Description provides information how the described module interacts with other modules and needs to be integrated.
Dependencies:	[SRS_Rte_00169], [SRS_Rte_00170]
Use Case:	Support the integration of AUTOSAR software.
Supporting Material:	Specification of Basic Software Module Description [5]

⌋

[SRS_Rte_00241] Support for Local or Remote Handling of BSW Service Calls on Partitioned Systems

Upstream requirements: RS_BRF_01160

⌈

Description:	For systems where the BSW modules can be executed in multiple partitions, the RTE generator shall redirect the BSW service call from a SWC either to the local or to a remote partition based on the partition mapping(s) assigned to the BSW Module.
Rationale:	If a module is available on a local partition, execution within that partition is preferable for performance reasons. If it is not available, the RTE is responsible for routing service calls to the partition as configured.
Dependencies:	–
Use Case:	BSW running on multi core systems.
Supporting Material:	–

⌋

[SRS_Rte_00245] Support of Writing Strategies for NV data

Upstream requirements: RS_BRF_01816

⌈

Description:	The RTE shall provide a mechanism in order to write updated NV data of RAM Blocks to NV memory with a certain timing schema (writing strategy). The availability of this mechanism shall be configurable.
Rationale:	Support different write strategies for NV data.
Dependencies:	–
Use Case:	SW-Cs that have to fulfill different functional requirements on NV data handling.
Supporting Material:	–

⌋

4.1.5 Generation of the BSW Scheduler

[SRS_Rte_00211] Cyclic time based scheduling of BSW Schedulable Entities

Upstream requirements: RS_BRF_01320

⌈

Description:	The RTE shall support to the cyclic time based scheduling of BSW Schedulable Entities.
Rationale:	Many BSW Modules rely on the cyclic time based call of their Schedulable Entities in order to fulfill their functionality.
Dependencies:	[SRS_Rte_00072]
Use Case:	Call of the function "Com_MainFunctionTx" to achieve periodic sending of IPdus.
Supporting Material:	–

⌋

[SRS_Rte_00212] Activation Offset of BSW Schedulable Entities

Upstream requirements: RS_BRF_01320

⌈

Description:	The RTE shall allow the definition of an Activation Offset of BSW Schedulable Entities.
Rationale:	In order to allow optimizations in the scheduling the RTE has to handle the activation offset information from a task shared reference point for time trigger BSW Schedulable Entities.
Dependencies:	[SRS_Rte_00211], [SRS_Rte_00161]
Use Case:	Mapping of BSW Schedulable Entities with different periods in the same task
Supporting Material:	–

⌋

[SRS_Rte_00213] Mode Switches for BSW Modules

Upstream requirements: RS_BRF_01320

⌈

Description:	The RTE shall support Mode Switches for BSW modules. BSW Schedulable Entities are scheduled dependent on modes or are activated by entering and exiting a mode.
Rationale:	Conditional scheduling of BSW Schedulable Entities dependent on different operating modes of the ECU.
Dependencies:	[SRS_Rte_00144]
Use Case:	• Initialization and finalization phases • Different communication modes
Supporting Material:	–

⌋

[SRS_Rte_00214] Common Mode handling for Basic SW and Application SW

Upstream requirements: RS_BRF_01320

⌈

Description:	The RTE shall support the coordinated switching of a Mode affecting BSW Modules and Application Software Components.
Rationale:	Synchronized behavior during a mode transition controlling AUTOSAR BSW Modules and Application Software Components.
Dependencies:	[SRS_Rte_00144], [SRS_Rte_00143], [SRS_Rte_00213]
Use Case:	ECU initialization and finalization phase.

▽

△

Supporting Material:	–

⌋

[SRS_Rte_00215] API for Mode switch notification to the SchM

Upstream requirements: RS_BRF_01320

⌈

Description:	The SchM shall provide APIs for communication of active modes from the BSW mode manager to the SchM.
Rationale:	The SchM needs to disable the execution of certain BSW Schedulable Entities depending on modes, therefore it needs to know the current modes.
Dependencies:	[SRS_Rte_00213], [SRS_Rte_00214] This might be implemented via a Port-Based API or a direct C-API.
Use Case:	See [SRS_Rte_00213]
Supporting Material:	–

⌋

[SRS_Rte_00216] Triggering of BSW Schedulable Entities by occurrence of External Trigger

Upstream requirements: RS_BRF_01328, RS_BRF_01320

⌈

Description:	The RTE shall support the triggering of BSW Schedulable Entities by the occurrence of External Triggers. Particular BSW Schedulable Entities in BSW dependent from External Triggers shall be executed after occurrence of the event in a defined and deterministic order specified by the integrator. The occurrence of the External Trigger is either reported via API to the RTE or by means of the OS (e.g. expiration of an OS Alarm). Restriction: This is only applicable for intra-ECU usage.
Rationale:	Sporadic and non timing based periodic activation of BSW Schedulable Entities in different BSW Modules.
Dependencies:	[SRS_Rte_00218]
Use Case:	Angle periodic triggering of the ignition for a combustion engine.
Supporting Material:	–

⌋

[SRS_Rte_00230] Triggering of BSW Schedulable Entities by occurrence of Internal Trigger

Upstream requirements: RS_BRF_01328, RS_BRF_01320

⌈

Description:	The Basic Software Scheduler shall support the triggering of BSW Schedulable Entities by the occurrence of Internal Trigger. Particular BSW Schedulable Entities in BSW dependent from Internal Events shall be executed after occurrence of the event in a defined and deterministic order specified by the integrator. The occurrence of the Internal Trigger is either reported via API to the RTE or by means of the OS (e.g. expiration of an OS Alarm). Restriction: This is only applicable for intra-ECU usage.
Rationale:	Decoupling from Interrupt Context inside a Basic Software Module.
Dependencies:	–
Use Case:	An interrupt which shall not exceed a certain WCET activates a BSW Schedulable Entity to process more time consuming algorithms.
Supporting Material:	–

⌋

[SRS_Rte_00217] Synchronized activation of Runnable Entities and BSW Schedulable Entities

Upstream requirements: RS_BRF_01320

⌈

Description:	The RTE shall support the triggering of both, Runnable Entities and BSW Schedulable Entities by the same Triggered Events.
Rationale:	Synchronous activation of routines in AUTOSAR BSW modules and Application Software Components.
Dependencies:	–
Use Case:	Angle periodic triggering of the routines in Application Software Components and Complex Drivers for a combustion engine.
Supporting Material:	–

⌋

[SRS_Rte_00218] API for Triggering BSW modules by Triggered Events

Upstream requirements: RS_BRF_01320

⌈

Description:	The RTE shall provide an API usable for BSW modules to notify the RTE about the occurrence of Triggered Events.
Rationale:	The sources for Triggered Events may be captured in the BSW and need to be forwarded to the application SWCs.
Dependencies:	[SRS_Rte_00216], [SRS_Rte_00217]
Use Case:	See [SRS_Rte_00216]
Supporting Material:	–

⌋

[SRS_Rte_00219] Support for interlaced execution sequences of Runnable Entities and BSW Schedulable Entities

Upstream requirements: RS_BRF_01320

⌈

Description:	The RTE shall support the interlaced execution sequences of Runnable Entities and BSW Schedulable Entities within the same Os Task. The whole execution sequence for all Runnable Entities and BSW Schedulable Entities which are mapped to the same OS Task can be arbitrarily defined.
Rationale:	Usage of OS Tasks for scheduling of Application Software Components and BSW Modules.
Dependencies:	–
Use Case:	Reduce response time of a closed loop control by configuration of a signal flow orientated calculation sequence (plant determination, controller, actuator). Reduce number of tasks.
Supporting Material:	–

⌋

[SRS_Rte_00220] ECU life cycle dependent scheduling

Upstream requirements: RS_BRF_01320

⌈

Description:	The RTE shall support the exclusive Scheduling of BSW modules dependent from the ECU life cycle. Before the RTE is fully initialized (call of Rte_Start) or after the RTE is finalized (call of Rte_Stop) only BSW Schedulable Entities shall be scheduled.
Rationale:	Support different life-cycles of BSW Modules and Application Software Components.

▽

△

Dependencies:	–
Use Case:	Exclusive Scheduling of BSW Modules during start-up and shut-down phase of the ECU.
Supporting Material:	–

⌋

[SRS_Rte_00221] Support for "BSW integration" builds

Upstream requirements: RS_BRF_01320

⌈

Description:	The RTE generator shall provide means to generate the API for only for BSW Modules (with the related BSW Scheduling code), excluding the API for Application Software Components. When the input information contains Application Software Component parts these shall be ignored in this mode. The complete input information must be valid, including the ignored parts.
Rationale:	Support integration of BSW Modules without Application Software Components.
Dependencies:	–
Use Case:	Pre-integration and test of BSW Module packages.
Supporting Material:	–

⌋

[SRS_Rte_00222] Support shared exclusive areas in BSW Service Modules and the corresponding Service Component

Upstream requirements: RS_BRF_01320

⌈

Description:	The RTE shall provide APIs to enter or exit exclusive areas for both, BSW Service Modules and the corresponding Service Component.
Rationale:	Coordinated access to shared memory between BSW Schedulable Entities and BSW Runnable Entities of the SAME module (i.e. the AUTOSAR Service which has both, a BSW aspect and a SW-C aspect).
Dependencies:	[SRS_Rte_00046]
Use Case:	Coordinate the access to the NvM job buffer from the NvM module Schedulable Entities and the NvM server Runnable Entities called by the Application SW-Components.
Supporting Material:	–

⌋

[SRS_Rte_00229] Support for Variant Handling of BSW Modules

Upstream requirements: RS_BRF_01320, RS_BRF_01136

⌈

Description:	The RTE Generator shall support the generation of the BSW scheduling where "PreCompileTime" and "PostBuild" variability is left open and shall be resolved after the generation.
Rationale:	Some variability may stay in the input information after the RTE Generation phase.
Dependencies:	[SRS_Rte_00201]
Use Case:	A simple NvM only needs one Schedulable Entity while the full-featured NvM requires several Schedulable Entities. Both variants can be described in one input information and the generated RTE shall contain means to switch between both variants after the generation.
Supporting Material:	–

⌋

[SRS_Rte_00243] Support for inter-partition communication of BSW modules

Upstream requirements: RS_BRF_01160

⌈

Description:	On multi-core systems that use the SchM for parallel execution of BSW modules, the SchM shall provide APIs for the inter-partition communication of modules, including APIs for the initialization of satellites and functions for service invocation on a specific partition.
Rationale:	The SchM needs to provide mechanisms for inter-partition communication, because invocations of a module function are not assigned to specific partitions.
Dependencies:	–
Use Case:	BSW running on multi core systems.
Supporting Material:	–

⌋

4.1.6 Support for Measurement and Calibration

[SRS_Rte_00153] Support for Measurement ⌈

Description:	The RTE generator shall create code allowing read out of ECU internal communication data and variable contents. Responsibility of RTE is to supply RAM locations where the measurement data can be read by other SW (e.g. Basic SW, external measurement tools). This read out might be asynchronous to all RTE actions. The RTE is not responsible to deliver the measurement values to ECU external instances.
Rationale:	Measurement is needed to get knowledge about ECU internal behavior when ECU is running.
Dependencies:	–
Use Case:	Monitor SWC internal signals (e.g.InterRunnableVariables), VFB communication or mode states.
Supporting Material:	Software Component Template [4] chapter "Measurement & Calibration"

⌋

[SRS_Rte_00154] Support for Calibration ⌈

Description:	RTE and BSW Scheduler shall support calibration process of ECUs. RTE and BSW Scheduler are not responsible to make parameter or interpolation curve/map modifications by itself but must support calibration data emulation. RTE and BSW Scheduler are neither responsible to handle exchange of calibration parameter values nor for communication with ECU external instances.
Rationale:	Calibration is the process of adjusting an ECU SW to fulfill its tasks to control physical processes resp. to fit to special project needs or environments.
Dependencies:	[SRS_Rte_00153] - Support of Measurement: calibration needs means of measurement to work properly.
Use Case:	Adapt ECU SW to motor specific properties. Environment specific adaptation of ECU SW.
Supporting Material:	Software Component Template [4] chapter "Measurement & Calibration" ASAP Standard (www.asam.net)

⌋

[SRS_Rte_00156] Support for different calibration data emulation methods ⌈

Description:	The RTE generator shall support these data emulation methods for calibration purposes: directAccess Calibration data is stored in ROM and accessed directly. This method can be used with appropriate calibration hardware. Single pointered method Calibration data accesses are done via one indirection over a pointer table in RAM Double pointered method Calibration data accesses are done via a base pointer in RAM and over a pointer table in ROM/FLASH InitRAM parameter method RTE accesses calibration parameters located in RAM directly (without any indirection) and copies the values from ROM/FLASH during startup Methods 2-4 need SW support from RTE.
Rationale:	Projects in different domains have different requirements and different RAM availabilities.
Dependencies:	[SRS_Rte_00154] - Support of Calibration.
Use Case:	DirectAccess method: No overhead. Appropriate HW support present or after rebuild for production Single pointered method: more available RAM present than with InitRAM method, only 1 indirection, no time for initial copy Double pointered method: less RAM needs than single pointered method when calibration is off, activate several modified parameters simultaneously InitRAM parameter method: Only few parameters to calibrate
Supporting Material:	Software Component Template [4] chapter "Measurement & Calibration"

⌋

[SRS_Rte_00157] Support for calibration parameters in NVRAM ⌈

Description:	RTE shall support allocation of calibration parameters in NVRAM
Rationale:	Allocation in NVRAM allows independent parameter manipulation by other instances without re-flashing the ECU
Dependencies:	[SRS_Rte_00154] - Support of Calibration.
Use Case:	Modify a NVRAM calibration parameter via a diagnostic service, e.g. modify window lifter speed or enable a SW option
Supporting Material:	Software Component Template [4] chapter "Measurement & Calibration"

⌋

[SRS_Rte_00158] Support separation of calibration parameters ⌈

Description:	RTE shall support separation of calibration parameters
Rationale:	Separation required e.g. due to security reasons
Dependencies:	[SRS_Rte_00154] - Support of Calibration.
Use Case:	Separate calibration parameters for monitoring purposes from the other calibration parameters to get independency from parameters for normal functional operation in case of partly corrupted memory.
Supporting Material:	Software Component Template [4] chapter "Measurement & Calibration"

⌋

[SRS_Rte_00159] Sharing of calibration parameters ⌈

Description:	Several software components (and also several instances of software components) shall be able to share same calibration parameters defined in CalprmComponentTypes.
Rationale:	Avoids potential inconsistencies between several calibration parameters for same item on 1 ECU, reduces ECU resource consumption
Dependencies:	[SRS_Rte_00154] - Support of Calibration.
Use Case:	Common use of calibration parameters like maximum vehicle speed, left/right steering wheel, temperature sensor interpolation curve, ..
Supporting Material:	Software Component Template [4] chapter "Measurement & Calibration"

⌋

[SRS_Rte_00189] A2L Generation Support ⌈

Description:	The RTE generator shall support the generation of output information in order to support the later generation of a complete A2L file.
Rationale:	The RTE generator is allocating the variables which shall be measurable. Therefore the information about the allocated variables shall be exported for further usage by subsequent tools.
Dependencies:	–
Use Case:	In order to measure some RTE allocated variables the symbols used in the allocation have to be available for the measurement tools.
Supporting Material:	–

⌋

4.1.7 General Requirements

[SRS_Rte_00021] Per-ECU RTE customization ⌈

Description:	The RTE shall be customizable (generated and/or configured) for each ECU. The RTE generator should avoid, where possible, the use of generic functions and should, instead, favor functions that are configured/generated to specifically implement the required communication patterns.
Rationale:	Generic functions are considered to be too computationally expensive since the function needs to dynamically determine what actions to perform (e.g. switch on parameters). In contrast, statically configured/generated functions know implicitly what needs to be done and therefore avoid these costs and are therefore considered necessary for the production of optimal systems.
Dependencies:	–
Use Case:	–
Supporting Material:	An ECU with two or more micro-controllers can be configured using either shared memory (and hence a single OS, single basic software set, etc) or with separate memory (multiple OSs, multiple basic software sets, etc.). In the first case there is only a single ECU according to the AUTOSAR ECU architecture and therefore only one RTE. In the second case there are multiple, independent, ECUs and therefore multiple RTEs.

⌋

[SRS_Rte_00065] Deterministic generation ⌈

Description:	A given version of a RTE generator from a vendor - for an identical set of input files - shall reproduce, every time it is invoked, the same RTE code with the exception of time-related information in code comments, like ‘generated at..’, which may differ.
Rationale:	The generated RTE code shall be equal in case the same generator (version) and input information is used.
Dependencies:	–
Use Case:	There shall be no difference (other than information in comments) between RTEs generated by the same generator for the same input files.
Supporting Material:	–

⌋

[SRS_Rte_00028] "1:n" Sender-receiver communication ⌈

Description:	The RTE shall support "1:n" sender-receiver communication. Sender-receiver communication is message passing and the RTE shall support scenarios with a single-sender-multiple-receivers ("1:n").

▽

△

Rationale:	VFB Specification requires support for single-sender-multiple-receiver ("1:n")
Dependencies:	[SRS_Rte_00131] - "n:1" communication.
Use Case:	–
Supporting Material:	VFB Specification

⌋

[SRS_Rte_00131] "n:1" Sender-receiver communication ⌈

Description:	The RTE shall support "n:1" sender-receiver communication. Sender-receiver communication is message passing and the RTE shall support scenarios with multiple-senders-single-receiver ("n:1").
Rationale:	VFB Specification requires support for multiple-senders-one-receiver ("n:1")
Dependencies:	[SRS_Rte_00028] - "1:n" communication.
Use Case:	–
Supporting Material:	VFB Specification

⌋

[SRS_Rte_00029] "n:1" Client-server communication ⌈

Description:	The RTE shall support multiple-client-single-server ("n:1") client-server (function invocation) communication. Individual clients are independent - there is no coordination of requests between clients. Single-client-multiple-server ("1:n") communication is not required. Such communication raises issues about buffering and selection of results that are application dependent and therefore not considered to be the domain of the RTE.
Rationale:	–
Dependencies:	VFB requires support multiple-clients-one-server ("n:1") but explicitly does not require to support single-client-multiple-server ("1:n") communication
Use Case:	The fog light shall serve as backup for the brake light this can be implemented by one "fog light" - server switching fog light on/off and two clients, the "fog light"-client and the "brake light"-client both switching fog lights on and off for different purposes.
Supporting Material:	VFB Specification

⌋

[SRS_Rte_00079] Single asynchronous client-server interaction ⌈

Description:	The RTE shall support at most one asynchronous call at a time from a single operation in a required port categorized by a client-server interface (i.e. there can only be one outstanding request per "AsynchronousServerCallPoint"). Note that a single client can simultaneously have multiple outstanding requests provided each is to different server operations. When a SW-component instance restarts it may receive a stale reply - replies to a request made before the component was restarted. The RTE shall forward stale replies and it is the job of the SW-component instance to detect and reject the reply, for example, through sequence numbers.
Rationale:	Requirement from VFB spec (4.1.4.2 Client-Server Communication). There is no queuing (of parameters and return locations) on the client-side and therefore only one single outstanding request can be supported.
Dependencies:	–
Use Case:	–
Supporting Material:	Software Component Template [4] VFB Specification [3]

⌋

[SRS_Rte_00080] Multiple requests of servers ⌈

Description:	The RTE shall support the queuing of concurrent calls to a server (by different clients). A server specified using the "BUFFERING queue(n)" attribute may have queued requests from multiple clients. Requests shall be read from the server's queue using first-in-first-out semantics. Depending on the RTE implementation the queue may be present in the either in the RTE or in COM.
Rationale:	Requirement from VFB spec (4.1.4.2 Client-Server Communication)
Dependencies:	[SRS_Rte_00033]
Use Case:	–
Supporting Material:	Queues are applied at the operation level, i.e. each operation in a client-server interface has a dedicated queue.

⌋

[SRS_Rte_00162] "1:n" External Trigger communication

Upstream requirements: RS_BRF_01328

⌈

Description:	The RTE shall support the communication of External Trigger events from one trigger source to multiple trigger sinks ("1:n"). The Runnable Entity(s) in the trigger sink(s) linked to the event shall be executed after occurrence of the event in a defined and deterministic order defined by the integrator. Restriction: This is only applicable for intra-ECU usage.

▽

△

Rationale:	Sporadic and non timing based periodic activation of Runnable Entities in different Software Components.
Dependencies:	–
Use Case:	Angle periodic triggering of the Mass Air Flow calculation for a combustion engine.
Supporting Material:	–

⌋

[SRS_Rte_00163] Support for InterRunnableTriggering

Upstream requirements: RS_BRF_01328

⌈

Description:	The RTE shall support InterRunnableTriggering. A software component instance shall be able to explicitly trigger the execution of Runnable Entities of the same component instance.
Rationale:	Decoupling of calculation and processing sequences inside a Software Component instance.
Dependencies:	–
Use Case:	A time base triggered Runnable Entity which shall not exceed a certain WCET activates a second Runnable Entity of the same SW-C instance in case of error to process more time consuming exception handling.
Supporting Material:	–

⌋

[SRS_Rte_00235] Support queued triggers

Upstream requirements: RS_BRF_01328

⌈

Description:	External and internal trigger event communication shall support queuing the number of triggers issued by a trigger source. When the trigger source is informed of the end of execution of all triggered executable entities, the RTE shall (if any trigger is in the queue) dequeue a trigger by activating again the triggered executable entities.
Rationale:	–
Dependencies:	–
Use Case:	There are use cases existing where it is important to queue the number of activations done by a trigger source when triggers are faster issued in the trigger source as the runnables in the trigger target can be activated.
Supporting Material:	–

⌋

[SRS_Rte_00025] Static communication ⌈

Description:	The RTE shall support only those communication connections known when the RTE is generated - the source(s) and destination(s) of all communication shall be known statically. Static communication is considered to include Application Software Component access to the publisher-subscriber service - components are statistically configured to access the service and then subscribers are dynamically chosen from a statically configured set.
Rationale:	Dynamic communication is deemed too expensive (both at run-time and in code overhead) and would therefore limit the range of devices for which the RTE is suitable.
Dependencies:	–
Use Case:	–
Supporting Material:	VFB Specification In AUTOSAR (and in COM) only static communication connections are permitted. If dynamic communication will be allowed in future, all specifications have to be reworked.

⌋

[SRS_Rte_00144] RTE shall support the notification of mode switches via AUTOSAR interfaces ⌈

Description:	RTE shall use the well defined mechanisms of AUTOSAR interfaces for the communication of active modes from the mode manager to the mode dependent software component.
Rationale:	Use the flexibility and configuration mechanisms defined for AUTOSAR interfaces.
Dependencies:	[SRS_Rte_00143]
Use Case:	See [SRS_Rte_00143]
Supporting Material:	AUTOSAR. Software Component Template. Version 1.04 - Final, 04 2005 (p. 61, L 7-8)

⌋

[SRS_Rte_00018] Rejection of invalid configurations ⌈

Description:	The RTE generator shall detect, and reject where appropriate, the invalid deployment and communication configuration of application and basic software components.
Rationale:	The RTE is required to reject "invalid" configurations, e.g. wait point in category 1a or 1b Runnable Entity, interface incompatibility, .
Dependencies:	[SRS_Rte_00062] - local access to basic software.

▽

△

Use Case:	Multiple instantiation of a component where the "supportsMultipleInstantiation" flag is not set. The RTE generator shall reject the mapping of event-triggered and "communication triggered" Runnable Entities to the same basic task. (An implementation is possible, if inefficient, for extended tasks).
Supporting Material:	A valid RTE cannot be generated for an invalid configuration. For example, AUTOSAR is required to be interoperable with "legacy ECUs" (Requirement RS_Main_00190, AUTOSAR_MainRequirements_v2.2_r.doc, p. 32). The capabilities of such ECUs may not be precisely compatible with AUTOSAR and therefore some configurations, e.g. client-server communication with the legacy ECU, should be rejected - that's an invalid configuration.

⌋

[SRS_Rte_00055] RTE use of global namespace ⌈

Description:	The RTE specification shall define standard naming conventions for all the symbols created by the RTE generator that are visible within the global namespace. Creating symbol definitions within the global namespace using this naming convention is the exclusive right of the RTE generator. Application and/or basic software components shall not create symbols defined by this naming convention within the global namespace.
Rationale:	Prevents conflicts with symbols created by application and/or basic software components.
Dependencies:	–
Use Case:	–
Supporting Material:	All symbols use the prefix "RTE".

⌋

[SRS_Rte_00164] Ensure a unique naming of generated types visible in the global namespace

Upstream requirements: RS_BRF_01024

⌈

Description:	The RTE shall define standard naming conventions for all the type symbols created by the RTE generator that are visible within the global namespace. The naming convention shall be defined in a way, that each type requiring an own implementation within the global namespace is named uniquely.
Rationale:	Prevent type conflicts within the global namespace caused by integration of SWC provided from different suppliers.
Dependencies:	[SRS_Rte_00055]
Use Case:	–

▽

△

Supporting Material:	–

⌋

[SRS_Rte_00165] Suppress identical "C" type re-definitions

Upstream requirements: RS_BRF_01024

⌈

Description:	The RTE generator shall suppress the re-definition of compatible type declarations resulting in identical "C" type definition.
Rationale:	ECU Extract can contain (compatible) type definitions which are resulting in the identical type definitions.
Dependencies:	–
Use Case:	Improvement of code quality by avoidance of type re-definitions. Enabling stricter type checking and suppress compiler errors/warnings.
Supporting Material:	–

⌋

[SRS_Rte_00166] Use the AUTOSAR Standard Types in the global namespace if the AUTOSAR data type is mapped to an AUTOSAR Standard Type

Upstream requirements: RS_BRF_01024

⌈

Description:	The RTE shall suppress the re-declaration of AUTOSAR data types mapped to the AUTOSAR Standard Types and shall use directly the AUTOSAR Standard Types instead.
Rationale:	Improving code quality by avoidance of type casts in case of communication with Basic Software and in case of library calls.
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋

[SRS_Rte_00167] Encapsulate a Software Component local name space

Upstream requirements: RS_BRF_01024

⌈

Description:	The RTE generator shall encapsulate a Software Component local name space for declarations and definitions for APIs and types related to one Software Component.
Rationale:	From a SW-C providers point of view this is the only name space which can be ensured by the SW-C provider. Therefore the RTE Generator has to provide the mapping from the SW-C locally used names within the application header file to the names used within the global namespace.
Dependencies:	[SRS_Rte_00055], [SRS_Rte_00087]
Use Case:	Support embedding of SW-Components in different ECUs by avoiding type conflicts.
Supporting Material:	–

⌋

[SRS_Rte_00252] Encapsulate a BSW Module local name space

Upstream requirements: RS_BRF_01024

⌈

Description:	The RTE generator shall encapsulate a BSW Module local name space for declarations and definitions for APIs and types related to one BSW Module.
Rationale:	From a BSW Module providers point of view this is the only name space which can be ensured by the BSW Module provider.
Dependencies:	[SRS_Rte_00055], [SRS_Rte_00087]
Use Case:	Support embedding of BSW Modules in different ECUs by avoiding type conflicts.
Supporting Material:	–

⌋

[SRS_Rte_00126] C language support ⌈

Description:	The RTE generator shall support SW-Components and BSW Modules created using ‘ANSI C’.
Rationale:	Support common used programming language.
Dependencies:	–
Use Case:	–
Supporting Material:	Specification of the Virtual Functional Bus [3] ANSI/ISO 9899-1989, "Programming Languages - C"

⌋

[SRS_Rte_00138] C++ language support ⌈

Description:	The RTE generator shall support SW-Components and BSW Modules created using ISO C++.
Rationale:	Support common used programming language.
Dependencies:	–
Use Case:	–
Supporting Material:	Specification of the Virtual Functional Bus [3] ISO/IEC 14882-1998, "Programming Languages - C++"

⌋

[SRS_Rte_00051] RTE API mapping ⌈

Description:	The RTE specification shall define a standard naming convention for all RTE API artifacts visible by a component author that are created by the RTE generator. The names of RTE API artifacts shall not include the component instance names.
Rationale:	The requirement for an API mapping enables the signature of generated RTE functions to be hidden from users and permits targeted optimization depending on configuration. The hiding of signatures is desirable for two reasons: The names of generated RTE functions may be long (to ensure name uniqueness) and therefore unwieldy for users to reference directly. The generated function name may include information not known when the component is compiled, such as the instance name.
Dependencies:	–
Use Case:	–
Supporting Material:	At the point the component is written the component instance name is not defined (deployment has not be performed) and therefore the component instance name cannot be included in the API. However, the instance name is required by the RTE generator when actually generating the RTE to ensure name uniqueness and therefore the RTE generator shall implement a well-defined API mapping from the RTE API to the generated RTE API functions.

⌋

[SRS_Rte_00048] RTE Generator input ⌈

Description:	The RTE generator shall accept input consisting of zero or more databases/files.
Rationale:	It is not reasonable to expect input as a single file. The RTE generator shall collect information from multiple input files and check their consistency.
Dependencies:	–

▽

△

Use Case:	• Provide each Software Component type description in a separate input file. • Provide the System description in a separate input file. • Be prepared to not find any input file and provide proper error reporting.
Supporting Material:	AUTOSAR design flow does not restrict input to one source and therefore RTE generator must be flexible.

⌋

[SRS_Rte_00023] RTE Overheads ⌈

Description:	The RTE generator shall provide a configuration option specifying the overall design goal of the generated RTE - minimizing memory and/or run-time overhead.
Rationale:	Unfortunately, the different feasible directions of optimizations can contradict each other and only a trade-off close to the overall design goal can be found. But this may sufficient in order to meet the constraints of the ECU or the mapping of that special functionality to the ECU is not possible, anyway. Thus, this requirement requests that the RTE generator should be able to generate RTEs that fit on a wide a range of devices as possible (obviously depending on configuration and component deployment).
Dependencies:	–
Use Case:	The RTE generator shall generate RTEs for the ECUs needs with respect to the given resources (processor speed, memory, etc.).
Supporting Material:	VFB_C20 - Rephrased since requirement is inherently untestable - one can never know when requirements have been minimized (e.g. local minima). However, rejection does not absolve an implementation from a general requirement to be "efficient" with ECU resources. The test is - does the generated RTE fit for the ECUs resources, but not more!

⌋

[SRS_Rte_00024] Source-code AUTOSAR software components ⌈

Description:	The RTE shall support AUTOSAR software components where the source is available ("source-code software components").
Rationale:	AUTOSAR software components as source-code increase the optimization potential for the generated RTE.
Dependencies:	–
Use Case:	–
Supporting Material:	Software Component Template [4]

⌋

[SRS_Rte_00140] Binary-code AUTOSAR software components ⌈

Description:	The RTE shall support AUTOSAR software components where only the object code ("binary-code software components") is available.
Rationale:	Binary-code AUTOSAR software components are required for IP hiding.
Dependencies:	–
Use Case:	–
Supporting Material:	Software Component Template [4] Support for binary-code AUTOSAR software components requires the same compiler type and compiler version.

⌋

[SRS_Rte_00083] Optimization for source-code components ⌈

Description:	The RTE generator should provide optimized communication when the source-code of an Application Software Component is available. Optimizations envisaged include elimination of the RTE for that communication channel.
Rationale:	VFB_C20
Dependencies:	[SRS_Rte_00023] - minimize overheads, has been rephrased from the specification of VFB_C20 to be testable. This requirement is considered testable provided the "contemporary" solution is suitable for comparison.
Use Case:	Conversion of intra-task S-R communication to direct variable write. This can only be performed for source-code components since the deployment is not known when a binary-code component is compiled.
Supporting Material:	VFB_C20 requires that optimizations should enable the RTE to impose zero overhead when compared with "contemporary" implementations. When comparing solutions, one should make sure that the AUTOSAR system has the same features as the "contemporary solution", for example, by using the "SupportsMultipleInstantiation" attribute of the "Implementation" class.

⌋

[SRS_Rte_00027] VFB to RTE mapping shall be semantic preserving ⌈

Description:	The RTE generator shall configure the RTE to implement the specified communication paths while retaining their semantics.
Rationale:	The mapping from VFB model expressed in the XML input to generated RTE is required to be semantic preserving. This requirement applies regardless of whether communication is done by COM, by the RTE directly or if the RTE generator optimizes the generated RTE to bypasses the RTE completely for certain communication paths.
Dependencies:	–
Use Case:	The RTE generator is not permitted to modify the semantics of communication, for example, converting synchronous to asynchronous.

▽

△

Supporting Material:	VFB_C10

⌋

[SRS_Rte_00190] Support for variable-length Data Types

Upstream requirements: RS_BRF_01568

⌈

Description:	The RTE shall support the transfer of data with variable size (could be bigger than 8 bytes but has a fixed maximum size). The variable length support shall be applicable to • strings • primitive byte arrays
Rationale:	The support of variable length data allows a more efficient utilization of resources in the ECU and on the communication busses. It also supports the implementation of dynamic communication protocols (like SAE J1939). Handling of strings with always a fixed length is not preferable, since a lot of ECU resources will be allocated by this approach. The usage of an always fixed size would result in wasting runtime and bandwidth in case of not used but reserved space. The alternative usage of a couple of interfaces and signals to serve different sizes will complicate the APIs and waste configuration freedom.
Dependencies:	–
Use Case:	• Transferring message and information strings of variable length between ECU's, e.g. Central ECU and Instrument Cluster. • Transferring the content of a received SMS to display it. • Transferring (variable) strings to a display. • Primitive byte arrays which are similar to strings but are not zero terminated.
Supporting Material:	–

⌋

[SRS_Rte_00234] Support for Record Type sub-setting ⌈

Description:	The RTE shall support that ports are connected which are typed by different interfaces and where the elements of the provide port are typed by composite data types which composite elements are mapped to elements of the require port. Hereby the require port might contain only a sub set of the elements contained in the provide port.

▽

△

Rationale:	Since the handling of data in a consistent manner requires using a record type, it shall be allowed at the receiver of a RecordType to only receive a sub-set of the sent record data elements. Since different receivers do require a different sub-set of the provided data.
Dependencies:	–
Use Case:	4 wheel speed signals and the movement direction signal are provided in one record. If a receiver is only interested in the movement direction information all of the other information from this record does not have to be considered at this specific receiver.
Supporting Material:	–

⌋

[SRS_Rte_00098] Explicit Sending ⌈

Description:	The RTE shall provide a mechanism for making requests for explicit sending of AUTOSAR signals (i.e. an implementation of DataSendPoint). The DataSendPoint of a Runnable Entity references an instance of a data-element in a provided port. Using the DataSendPoint, a Runnable Entity can use an explicit RTE API call to write new values of the specified data-element (which may cause an immediate send depending on component and communication configuration).
Rationale:	Implementation of internal component model from VFB Specification.
Dependencies:	[SRS_Rte_00134], [SRS_Rte_00128] and [SRS_Rte_00129] - The current SwCT and VFB specifications require that the Runnable Entity is of cat 2 for explicit sending. This situation is being revised so that cat 1b and 2 will be able to access DataSendPoints (e.g. extended to cat 1b).
Use Case:	–
Supporting Material:	VFB Specification [3] Software Component Template [4]

⌋

[SRS_Rte_00129] Implicit Sending ⌈

Description:	The RTE shall provide a mechanism for the implicit sending of data elements. The mechanism shall grant write-access to a data element of a provided port that may be freely changed until the Runnable Entity returns. The presence of DataWriteAccess means that the Runnable Entity will potentially modify the DataElement in the pPort. The Runnable Entity has free access to the data-element while it is running but the Runnable Entity should ensure that the data-element is in a consistent state when it returns. When using DataWriteAccess the new values of the data-element are made available, by the RTE, when the Runnable Entity returns. Depending on the configuration the RTE may either have nothing to do or it may need to actually initiate sending of the data element.

▽

△

Rationale:	Implementation of internal component model from VFB Specification.
Dependencies:	[SRS_Rte_00134] and [SRS_Rte_00128] - Previous SwCT and VFB specifications required that the Runnable Entity is (at most?) of cat 1b. This situation is being revised so that cat 1a are allowed to access DataReadAccess and DataWriteAccess.
Use Case:	–
Supporting Material:	VFB Specification [3] Software Component Template [4]

⌋

[SRS_Rte_00128] Implicit Reception ⌈

Description:	The RTE shall provide a mechanism for the implicit reception of data elements. The mechanism shall grant read-access to a data element of a required port that will not be modified by the RTE and may be freely read until the Runnable Entity returns. The presence of DataReadAccess means that the Runnable Entity will require access to the DataElement in the rPort. The Runnable Entity expects that the contents of this data does not change during execution of the Runnable Entity.
Rationale:	–
Dependencies:	[SRS_Rte_00134] and [SRS_Rte_00129] - Previous SwCT and VFB specifications required that the Runnable Entity is (at most?) of cat 1b. This situation is being revised so that cat 1a are allowed to access DataReadAccess and DataWriteAccess.
Use Case:	–
Supporting Material:	VFB Specification [3] Software Component Template [4]

⌋

[SRS_Rte_00141] Explicit Reception ⌈

Description:	The RTE shall provide a mechanism for making requests for explicit reception of AUTOSAR signals (i.e. an implementation of DataReceivePoint). The DataReceivePoint of a Runnable Entity references an instance of a data-element in a required port. Using the DataReceivePoint, a Runnable Entity can use an explicit RTE API call to receive new values of the specified data-element (e.g. the 'next' value is read out of the local queue).
Rationale:	Implementation of internal component model from VFB Specification.
Dependencies:	[SRS_Rte_00134], [SRS_Rte_00128], [SRS_Rte_00098], and [SRS_Rte_00129]
Use Case:	–
Supporting Material:	VFB Specification [3]

⌋

[SRS_Rte_00092] Implementation of VFB model "waitpoints" ⌈

Description:	The RTE API shall support wait points at which Runnable Entities will block until an "RTEEvent" occurs. A category 2 Runnable Entity shall, through the RTE API, be able to suspend its execution (i.e. block) until a well-defined event occurs. This requirement does not mean that "wait points" shall be explicitly specified in the API and could be satisfied by blocking calls that suspend the caller until an event (defined in the VFB meta-model) occurs.
Rationale:	Runnable Entities need to be able to suspend execution (block) until a defined event occurs.
Dependencies:	[SRS_Rte_00027]
Use Case:	• Waiting for the arrival of data • Waiting for the return of a call
Supporting Material:	This requirement is a special case of [SRS_Rte_00027]. Software Component Template [4]

⌋

[SRS_Rte_00145] Compatibility mode ⌈

Description:	The RTE Generator shall provide a compatibility operating mode that guarantees compatibility between different RTE implementations both for source code and object code components.
Rationale:	For IP hiding purposes a component may be delivered as object code only. Then it has to be precompiled against a header file created by an RTE implementation that may not be the RTE implementation that is used in the integration environment.
Dependencies:	[SRS_Rte_00146]
Use Case:	–
Supporting Material:	–

⌋

[SRS_Rte_00146] Vendor mode ⌈

Description:	The RTE Generator may provide a vendor operating mode allowing vendor-specific optimizations.
Rationale:	The vendor mode does not need to rely on predefined data structures and gives the individual RTE implementations the freedom for further optimizations for an additional reduction of the RTE overhead.
Dependencies:	[SRS_Rte_00145]
Use Case:	–
Supporting Material:	–

⌋

[SRS_Rte_00148] Support "Specification of Memory Mapping" ⌈

Description:	The document "Specification of Memory Mapping" shall be supported by RTE implementations.
Rationale:	To allow the integration of several software modules in one ECU.
Dependencies:	–
Use Case:	–
Supporting Material:	Specification of Memory Mapping [7]

⌋

[SRS_Rte_00150] Support "Specification of Platform Types" ⌈

Description:	The document "Specification of Platform Types" shall be supported by RTE implementations.
Rationale:	–
Dependencies:	–
Use Case:	–
Supporting Material:	Specification of Platform Types [8]

⌋

[SRS_Rte_00151] Support RTE relevant requirements of the "General Requirements on Basic Software Modules" ⌈

Description:	The following requirements of the "General Requirements on Basic Software Modules" shall be supported by RTE implementations: [SRS_BSW_00007] [SRS_BSW_00101] [SRS_BSW_00161] [SRS_BSW_00300] [SRS_BSW_00305] [SRS_BSW_00307] [SRS_BSW_00308] [SRS_BSW_00310] [SRS_BSW_00312] [SRS_BSW_00326] [SRS_BSW_00327] [SRS_BSW_00330] [SRS_BSW_00336] [SRS_BSW_00337] [SRS_BSW_00338] [SRS_BSW_00342] [SRS_BSW_00345] [SRS_BSW_00346] [SRS_BSW_00347] [SRS_BSW_00353] [SRS_BSW_00397] [SRS_BSW_00399] [SRS_BSW_00400] [SRS_BSW_00405] [SRS_BSW_00407] [SRS_BSW_00415] [SRS_BSW_00447]
Rationale:	–
Dependencies:	–
Use Case:	–
Supporting Material:	General Requirements on Basic Software Modules [9], only a subset of the requirements needs to be taken into account for the RTE.

⌋

[SRS_Rte_00171] Support for fixed and constant data ⌈

Description:	The RTE shall support the access to fixed and constant data shareable between SWCs.
Rationale:	SWCs shall be able to access fixed data which are commonly defined for several SWCs.
Dependencies:	–
Use Case:	Definition of general constant values to be used by many SWCs (like pi, avogadro).
Supporting Material:	–

⌋

[SRS_Rte_00178] Data consistency of NvBlockComponentType

Upstream requirements: RS_BRF_01816

⌈

Description:	The RTE shall protect the data defined in NvBlockComponentType against concurrent write and read access (by SWCs or NVRAM Manager).
Rationale:	The data of a NvBlockComponentType is shared amongst SWCs (and also with the NVM). The data needs to be protected against concurrent write and read access.
Dependencies:	[SRS_Rte_00176]
Use Case:	–
Supporting Material:	Software Component Template [4]

⌋

[SRS_Rte_00179] Support of Update Flag for Data Reception

Upstream requirements: RS_BRF_01304

⌈

Description:	In case of Sender Receiver communication with last is best semantics, if the configuration requires, RTE shall support an update flag that indicates whether there has been an update of the data since the last read operation from the Software Component to the data element. The update flag shall be set during reception of the data by the RTE and reset during the read operation from the software component.
Rationale:	Allows polling for updates.
Dependencies:	[SRS_Rte_00110]

▽

△

Use Case:	This allows a Runnable Entity - that is, e.g., triggered by the FlexRay cycle - to take action depending on the availability of new data. It shall be possible to refrain from re-reading the data element, if the data is not updated.
Supporting Material:	–

⌋

[SRS_Rte_00184] RTE Status "Never Received"

Upstream requirements: RS_BRF_01616

⌈

Description:	The RTE shall support the RTE-Status "never received". This is the new initial status of each data element for which it is configured. This initial status will be cleared when the first reception occurs.
Rationale:	This additional status establishes the possibility to check, whether a data element has been changed since system start or partition restart.
Dependencies:	–
Use Case:	Get the information whether involved data have been received at any time since system start or partition restart.
Supporting Material:	–

⌋

[SRS_Rte_00191] Support for Variant Handling

Upstream requirements: RS_BRF_01136

⌈

Description:	The RTE shall support the resolution and implementation of the AUTOSAR Variant Handling mechanism.
Rationale:	With the support of variant handling it is also possible to leave some variation until the RTE generation. Then the RTE Generator needs to support the resolution of these variation points.
Dependencies:	–
Use Case:	• Using the same ECU (hardware and software) for several vehicle lines. This leads to different communication matrices, which need to be used depending in which vehicle line the ECU is build in. Also the functionality may be slightly different in each vehicle line. • A project can choose out of different existing implementations of AUTOSAR SWCs respectively SW-compositions with same or compatible interfaces to implement the sum of the system functionality. This addresses pre-built variant handling.

▽

△

Supporting Material:	–

⌋

[SRS_Rte_00201] Contract Phase with Variant Handling support

Upstream requirements: RS_BRF_01136

⌈

Description:	The RTE shall support the generation of the Application Software Component header file where "PreCompileTime" or "PostBuild" variability is left open and shall be resolved later.
Rationale:	Some variability may be defined as having the binding time "PreCompileTime" or "PostBuild". This variability has to be represented in the generated application software component header file.
Dependencies:	–
Use Case:	The existence of a Port is specified to be variable with a binding time "PreCompile". Then in the application software component header file the corresponding APIs could be wrapped in #IFDEFs so the compiler can actually benefit from the binding during compiling. When the binding time is "PostBuild" the application software component header file shall consider the superset of all possible variants.
Supporting Material:	–

⌋

[SRS_Rte_00202] Support for array size variants

Upstream requirements: RS_BRF_01136

⌈

Description:	The RTE generator shall be able generate arrays whose size is depending on a model attribute.
Rationale:	Software supporting a variable - but fixed during runtime - number of entities.
Dependencies:	This shall be resolved until "PreCompile" time.
Use Case:	"Length of Arrays, size of Calibration Axis": depending on the system, the number of cylinders varies, values required per cylinder are combined in arrays to implement scalable algorithms.
Supporting Material:	–

⌋

[SRS_Rte_00204] Support the selection / de-selection of SWC prototypes

Upstream requirements: RS_BRF_01136

⌈

Description:	The RTE shall support the configuration - post-build time - a software component to run or not. When the SWC does not run, the RTE needs to behave as if the software component does not exist, therefore all RTE events for that SWC's Runnable Entities shall be suppressed.
Rationale:	Handling of optional functionality.
Dependencies:	The behavior of a "deselected" SW-Component prototype shall be defined.
Use Case:	Disabling trailer functionality if not used.
Supporting Material:	–

⌋

[SRS_Rte_00206] Support the selection of a signal provider

Upstream requirements: RS_BRF_01136

⌈

Description:	RTE shall support selection of a signal provider at "PostBuild" time.
Rationale:	The provider of a signal could come from one/several internal software component, or via a network signal. The RTE shall support the selection of the source post-build loadable. Support variants where the source of a signal can be either internal or external to the ECU.
Dependencies:	–
Use Case:	A system where in one variant the temperature sensor is connected to the local ECU and is available locally, another variant the temperature value is provided via the network.
Supporting Material:	–

⌋

[SRS_Rte_00207] Support N to M communication patterns while unresolved variations are affecting these communications

Upstream requirements: RS_BRF_01136

⌈

Description:	In a system N:M communication shall be allowed when this communication is subject to a variation point. After all affecting variants have been resolved only 1:N or N:1 communication shall be possible. This shall be available for Postbuild variants.

▽

△

Rationale:	Some variants will have multiple components publish the same data. The receivers require the data from any publisher. But after variant resolving only one publisher is actually sending the data.
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋

[SRS_Rte_00231] Support native interface between Rte and Com for Strings and uint8 arrays ⌈

Description:	Com does support the sending / receiving of an array of uint8 natively. The Rte shall use this native interface if the communicated data type is supported by Com.
Rationale:	Allow an easy access to data natively supported by Com.
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋

[SRS_Rte_00232] Synchronization of runnable entities

Upstream requirements: RS_BRF_01216

⌈

Description:	The RTE shall support the synchronization of runnable entities. Synchronization means that the mechanisms that implement the runnable entity activation (OsAlarm, OsTask, OsEvent, etc...) are triggered (expired, activated, set, etc...) at the same point of time. Synchronization shall be possible even if runnable entity are mapped to different OsTasks, different OsPartitions, different cores or even different ECUs.
Rationale:	In some cases, multiple sensors or actuators have to be acquired or driven by different runnable entities at the same point of time.
Dependencies:	Global time service for synchronization over ECU is required.
Use Case:	On a Flexray cluster, some processus are time triggered and shall be synchronized between ECUs.
Supporting Material:	–

⌋

[SRS_Rte_00238] Allow enabling of RTE-Feature to get the activating Event of Executable Entity ⌈

Description:	It is possible to request the activation of one Executable Entity by several Events. It shall be possible to identify the activating Event during the execution of the activated Executable Entity.
Rationale:	The Executable Entity's code shall be able to identify the actually activating Event which caused the current execution of this Executable Entity.
Dependencies:	–
Use Case:	A Executable Entity is defined to be activated by a "TimingEvent" as well as by a "DataReceivedEvent". During the execution of the Executable Entity the code needs to distinguish which activation source actually triggered the execution.
Supporting Material:	–

⌋

[SRS_Rte_00244] Support for bypass

Upstream requirements: RS_BRF_01392

⌈

Description:	The RTE shall provide support for implementation of software component bypass. A bypass consists in reading/modifying/writing a data for testing or rapid prototyping purpose.
Rationale:	To support the integration on a standard RTE of different implementations of bypass tools and software.
Dependencies:	–
Use Case:	A Rapid Prototyping tool/software vendor provides an implementation that can be integrated on a standard RTE. A Tier 1 supplier integrates the Rapid Prototyping software in an AUTOSAR ECU. The Rapid Prototyping tool is used by an OEM on the ECU provided by the Tier 1 supplier to evaluate/test new control algorithms.
Supporting Material:	–

⌋

[SRS_Rte_00254] Selectable RP Preparation

Upstream requirements: RS_BRF_01393

⌈

Description:	The RTE shall support selectable data preparation for RP.
Rationale:	A "hookable" flag attached to data permits the RTE generator to enable RP preparation for the data element and thus ensure that resources are only allocated where they are required.
Dependencies:	–
Use Case:	"Interesting" signals generated/consumed by SWCs are prepared for RP and subsequently made available for presentation to RP tool users by the RTE generator.
Supporting Material:	–

⌋

[SRS_Rte_00255] RP Memory Interface

Upstream requirements: RS_BRF_01393, RS_BRF_01394

⌈

Description:	The RTE shall support a write-read access pattern for RP prepared data.
Rationale:	The write-read pattern results in the RTE generator writing each associated RP prepared data element to a RP buffer and subsequently reading from the RP buffer rather than the data element. These modifications ensure that if an RP tool patches the write to the RP buffer then the value that is written by the RP tool to the RP buffer will be used by subsequent RTE generated code instead of the actual API parameter.
Dependencies:	–
Use Case:	An RP tool intercepts writes to the RP global buffer and uses bypass functionality to change the written value. Since the RTE no longer uses the original but performs subsequent reads from the RP global buffer it will use the value as modified by the RP tool.
Supporting Material:	–

⌋

[SRS_Rte_00256] Conditional Bypass

Upstream requirements: RS_BRF_01393, RS_BRF_01394

⌈

Description:	The RP memory interface provided by the RTE shall support the enabling or disabling of bypass functionality.
Rationale:	RP enables the bypass of existing functionality however the selection is dynamic and therefore it may be necessary to re-enable the original runnable entity and thus disable the bypass functionality.
Dependencies:	–
Use Case:	Algorithm under test, i.e. the bypass functionality, is erroneous and the original algorithm must be re-enabled.
Supporting Material:	–

⌋

[SRS_Rte_00257] RunnableEntity Bypass

Upstream requirements: RS_BRF_01393, RS_BRF_01394

⌈

Description:	The RTE generator shall support disabling the execution of runnable entities.
Rationale:	RP enables the bypass of existing functionality and therefore it may be necessary to disable the original runnable entity to prevent unwanted side-effects. The enable/disable decision is dynamic and may be selected by, for example, calibration tools, either prior to system start or change during run-time. The conditional execution of the original RunnableEntity is unrelated to the normal conditionality of the invocation, e.g. due to the presence of pre-scalers created by the RTE generator when multiple RteEvents are mapped to the task.
Dependencies:	–
Use Case:	The introduction of bypass functionality replaces an existing algorithm and to reduce system load the original algorithm is not executed. The switch between the two algorithms occurs during run-time so that the behavior of the two can be compared.
Supporting Material:	–

⌋

[SRS_Rte_00258] RTE Generated Service Points

Upstream requirements: RS_BRF_01393, RS_BRF_01394

⌈

Description:	The RTE shall support the generation of service points.
Rationale:	A service point is a call of a service function provided by the service component (typically a BSW module, not a SwServiceComponentType). The service function is responsible for sampling (reading) and stimulating (writing) the bypass data. The action of sampling may then trigger the RP system to perform the bypass (this may involve the communication of the sampled data to an external system for computation) ready for reading when the stimulation occurs. Data is sampled and/or stimulated at service points. During either sampling or stimulation the data is read and/or written from the memory associated with the data to/from a local buffer during the execution of the service point and hence transferred to/from the RP tool.
Dependencies:	–
Use Case:	A service point placed before a runnable entity can be used to trigger the RP system. A service point placed after a runnable entity can be used to sample bypass values.
Supporting Material:	–

⌋

[SRS_Rte_00259] Manually Inserted Service Points

Upstream requirements: RS_BRF_01393, RS_BRF_01394

⌈

Description:	The RTE generator shall support manually inserted service points within SWCs.
Rationale:	In this scenario the service function signature of the BSW that provides the service is known by the SWC developer and manually inserted into the SWC's code when it is developed. The description of these service points is both an input to, and output from, the RTE generator however the generator's role is restricted to validating the manual service points do not conflict with the generated service points. Note that there is no requirement for the RTE generator to insert calls within generated code for manually inserted service points. However the RTE generator must ensure that the description of the SWC's service hooks is exported for subsequent tools.
Dependencies:	–
Use Case:	A SWC developer implements the service function calls at the required positions within the RunnableEntity's code, typically one right before and a second one right after every area to be prepared for bypassing. This mechanism might be used in migration scenarios where a RunnableEntity contains multiple functionality.

▽

△

Supporting Material:	–

⌋

[SRS_Rte_00260] RP Interface Documentation

Upstream requirements: RS_BRF_01393, RS_BRF_01394

⌈

Description:	The RP interface documentation shall describe the elements of the RP memory interface provided by the RTE and their relationship to the bypassed SWCs and service points.
Rationale:	The RP interface documentation provides the standardized mechanism for an RP tool to determine the use of memory, e.g. buffers and flags, of the RP prepared RTE. The same documentation also describes the generated service points (including their relationship to SWCs) and their usage (e.g. the use as pre- and post-hooks).
Dependencies:	–
Use Case:	The RP interface documentation describes the symbol of the buffer used by the RP memory interface to implement the write-read cycle. Using this information, along with address information from the linker output, the RP tool can provide direct access to the buffer.
Supporting Material:	–

⌋

[SRS_Rte_00247] The Rte shall execute transformer chains for SWC communication

Upstream requirements: RS_BRF_01316

⌈

Description:	The Rte shall execute transformer chains for SWC communication for Senders, Receivers, Servers and Clients of Sender/Receiver and Client/Server inter-ECU communication.
Rationale:	Modification or extension of data is often necessary within the communication between Software Components in inter-ECU communication in a way which is transparent for the Software Components.
Dependencies:	–
Use Case:	Serialize complex data which are sent over a communication bus and de-serialized them at the receiver's RTE. Extend the data with checksums which are created at the sender's and checked at the receiver's side RTE.

▽

△

Supporting Material:	–

⌋

[SRS_Rte_00248] The Rte shall provide the buffer for the data transformation

Upstream requirements: RS_BRF_01316

⌈

Description:	The Rte shall provide the buffer for the data transformation.
Rationale:	Data Transformation requires space to be executed on. As the Rte coordinates the execution of transformers, also the Rte has to provide the buffer the transformers work on.
Dependencies:	SRS_Rte_00247
Use Case:	–
Supporting Material:	–

⌋

[SRS_Rte_00249] The Rte shall provide transformation errors to the SWCs

Upstream requirements: RS_BRF_01316

⌈

Description:	The Rte shall provide transformation errors to the SWCs to enable them to react accordingly to errors.
Rationale:	SWCs are not only interested in the fact that communication failed but they also might need more detailed information which transformer failed with which error.
Dependencies:	SRS_Rte_00247
Use Case:	Safety related SWCs want to distinguish between deserialization error and failed checksum checking.
Supporting Material:	–

⌋

[SRS_Rte_00253] The RTE shall execute data transformation for SWC/BSW communication within one ECU

Upstream requirements: RS_BRF_01316

⌈

Description:	The Rte shall execute data transformation for SWC/BSW communication for Senders, Receivers of Sender/Receiver intra-ECU communication.
Rationale:	Transformation of data is often necessary within the communication between Software Components or Basic Software Modules in intra-ECU communication in a transparent way.
Dependencies:	–
Use Case:	Transform different representations of data structures between Software Components or Basic Software Modules within one ECU - e.g. between NvBlockSwComponentType and DCM.
Supporting Material:	–

⌋

[SRS_Rte_00250] The Rte shall provide size indications of variable size arrays to SWCs

Upstream requirements: RS_BRF_01316

⌈

Description:	The Rte shall provide size indications of variable size arrays to SWCs.
Rationale:	SWCs which send variable size arrays to other SWCs are the only instances which know how many elements of the variable size array are filled with valid data. These SWC can indicate the number of valid elements to the Rte. Then, the Rte can consider that information and transmit only the valid elements to the receiver.
Dependencies:	SRS_Rte_00202, SRS_Rte_00247
Use Case:	Bus load efficient transmission of arrays with a variable size.
Supporting Material:	–

⌋

[SRS_Rte_00261] The RTE shall support optional struct members.

Upstream requirements: RS_Main_00280

⌈

Description:	The RTE shall support optional struct members.
Rationale:	A support for optional members in the RTE allows that the SOME/IP Transformer can use this information to skip members which are not available during serialization.
Dependencies:	SRS_Xfrm_00106
Use Case:	Optional members are helpful in large data structures where not all members exist at the same time. The provider only has to provide members which are currently available. Only this data needs to be transferred to a consumer.
Supporting Material:	–

⌋

4.1.8 VFB Tracing

[SRS_Rte_00005] The RTE generator shall support "trace" builds

Upstream requirements: RS_BRF_02272

⌈

Description:	If the RTE provides means for tracing which cost additional RAM and/or ROM and/or RUNTIME in the ECU. It shall be possible to switch these features off statically during RTE generation.
Rationale:	Allow monitoring of VFB communication and runtime behavior.
Dependencies:	[SRS_Rte_00045], [SRS_Rte_00008]
Use Case:	DLT, Debugging
Supporting Material:	VFB90, VFB_C10

⌋

[SRS_Rte_00045] Standardized VFB tracing interface

Upstream requirements: RS_BRF_02272

⌈

Description:	In 'trace' build the RTE implementation shall provide a standardized interface to make data values and events available to VFB tracing tools. When in 'trace' build the RTE generator inserts hook calls at interesting points (e.g. API invocation, interactions with COM, task start, Runnable Entity start, etc).
Rationale:	By defining a standardized VFB tracing interface tool vendors can adapt existing trace tools quickly - this will promote the adoption of AUTOSAR ECUs.
Dependencies:	[SRS_Rte_00005]
Use Case:	–
Supporting Material:	VFB Specification (VFB90); VFB Specification V1.03, Sect. 4.4.3, p. 94 An RTE implementation can define 'null' implementations of the hooks - to enable an RTE to build - but then permit them to be re-implemented by tool vendors to target vendor specific interfaces to standard tracing tools. The VFB tracing interface shall be configurable to interface with the AUTOSAR Diagnostic Log and Trace [10] functionality.

⌋

[SRS_Rte_00008] VFB tracing configuration

Upstream requirements: RS_BRF_02272

⌈

Description:	If the RTE is configured for tracing communication across the VFB, it shall be possible to detail the configuration of the RTE for what has to be logged and traced.
Rationale:	Tracing only of interesting signals/activations/system states, to reduce overhead in RAM+RUNTIME.
Dependencies:	[SRS_Rte_00005]
Use Case:	–
Supporting Material:	VFB Specification (VFB90)

⌋

[SRS_Rte_00192] Support multiple trace clients

Upstream requirements: RS_BRF_02272

⌈

Description:	The RTE Generator shall be able to support multiple trace clients for the same trace event.
Rationale:	It shall be possible to configure several trace functions on the same trace event. The individual trace functions shall not need to know about each other.
Dependencies:	[SRS_Rte_00008]
Use Case:	The Debugger and the Diagnostic Log and Trace (DLT) are interested in the same trace event.
Supporting Material:	–

⌋

[SRS_Rte_00003] Tracing of sender-receiver communication

Upstream requirements: RS_BRF_02272

⌈

Description:	The 'trace' builds of the RTE generator shall support tracing of sender-receiver signals on the VFB. The RTE should provide means for the tracing of transported signals of sender-receiver communication. It should be possible to trace both intra-ECU and inter-ECU communication.
Rationale:	Log data and supply it for debugging purposes.
Dependencies:	[SRS_Rte_00005]
Use Case:	–
Supporting Material:	VFB Specification (VFB90)

⌋

[SRS_Rte_00004] Tracing of client-server communication

Upstream requirements: RS_BRF_02272

⌈

Description:	The 'trace' builds of the RTE generator shall support the tracing of client-server communication. The RTE should provide means for the tracing of transported signals of client-server communication. It should be possible to trace both intra-ECU and inter-ECU communication.
Rationale:	Log data and supply it for debugging purposes.
Dependencies:	[SRS_Rte_00005]
Use Case:	–

▽

△

Supporting Material:	VFB Specification (VFB90)

⌋

4.1.9 Application Software Component Initialization and Finalization

[SRS_Rte_00052] Initialization and finalization of components ⌈

Description:	The RTE shall support initialization and finalization of Application Software Components. The term "initialization of a component" refers to the phase of a software components life cycle which will be executed before entering the normal operational mode, normally in order to set up an appropriate environment for executing the application. The term "finalization of a component" refers to the phase of a software components life cycle which will be executed after the normal operational mode, normally in order to reset the operational environment to a determined state.
Rationale:	The general ECU life-cycle is characterized by transitions from inoperational states to run states and back to the inoperational states of the ECUStateManager. When in run states the OS scheduler is responsible for the ECUs schedule and Runnable Entities will be executed with respect to the OS scheduler. In all other states the ECUStateManager is responsible for schedule of the ECU, but Runnable Entities can only be executed in synchronous and sequential way. Since the initialization and finalization phase of components refer to the transition from inoperational state to run states or vice versa, the Runnable Entities given for initialization and finalization can be executed synchronously as well as asynchronously depending on the intention of the system designer. That means the RTE shall provide means to invoke those Runnable Entities in the one or the other way and shall ensure that Runnable Entities of Application Software Components always communicate at least conceptually with modules of the basic software via the RTE.
Dependencies:	SWS ECUStateManager
Use Case:	Reading application specific parameters from non-volatile RAM while initialization application specific software component, writing application specific parameters to the non-volatile RAM while finalization of the Application Software Component.
Supporting Material:	VFB Specification (Sched42)

⌋

[SRS_Rte_00070] Invocation order of Runnable Entities ⌈

Description:	The RTE generator shall respect the invocation order of Runnable Entities as specified in the input information.
Rationale:	The invocation order of Runnable Entities may be of importance to the functionality of the algorithm or its timing. The invocation order may be provided with the description of the SW-Components or in the configuration of the RTE.
Dependencies:	–
Use Case:	A control loop implemented with several SW-Components. The execution of the individual Runnable Entities of this control loop shall be respected by the generated RTE code.
Supporting Material:	Software Component Template [4]

⌋

[SRS_Rte_00239] Support rule-based initialization of composite DataPrototypes and compound primitive DataPrototypes ⌈

Description:	The RTE shall support the rule-based initialization of composite DataPrototypes and compound primitive DataPrototypes. The rule shall support the following cases: • same value for all elements • explicit list of values, rest shall be initialized with a given value
Rationale:	Init values are required at sender and receiver side if the UNCONNECTED R-Port feature of RTE shall be used which ends up in a plenty of ValueSpecifications.
Dependencies:	–
Use Case:	SWCs with many interfaces
Supporting Material:	–

⌋

[SRS_Rte_00240] Support of init runnables for initialization purposes ⌈

Description:	The RTE shall support the execution of initialization runnable entities as a part of startup sequence controlled by the Bsw Manager and delay the start of timing events until the end of the startup sequence.
Rationale:	Simpler approach to initialize data inside a software-component other than to use mode-management.
Dependencies:	–
Use Case:	Define initialization runnables without being aware of integration details like ECU modes.

▽

△

Supporting Material:	Software Component Template [4]

⌋

4.1.10 API

[SRS_Rte_00100] Compiler independent API ⌈

Description:	The RTE API (for a particular programming language) shall be compiler and platform independent.
Rationale:	There shall be no need to change an Application Software Component source-code when the component is moved between ECUs and/or the compiler is changed.
Dependencies:	–
Use Case:	–
Supporting Material:	In addition the RTE should, ideally, also be retargetable (i.e. portable) with minimum effort. This is explicitly not stated as an RTE requirement since it is a statement about RTE implementation and not RTE behavior.

⌋

[SRS_Rte_00168] Typing of RTE API.

Upstream requirements: RS_BRF_01024

⌈

Description:	For parameters which are typed by an AUTOSAR data type the RTE API shall either use data types related to the whole AUTOSAR data type or data types related to the used Base Type. The kind of API is defined in the SWC-Description and is part of the contract phase input.
Rationale:	Dependent from the SWCs implementation either usage of Base Type related types (comprising only the "C" implementation aspect) or full AUTOSAR data type related types (comprising the semantic of the data type as well) is more advantageous. This depends how many library functions are used in the SW-C's implementation. In general the RTE shall support both typing to avoid unnecessary type casts as far as possible and to support strong type checking.
Dependencies:	–
Use Case:	Strong type checking with a static code analyzer.
Supporting Material:	–

⌋

[SRS_Rte_00059] RTE API shall pass "in" primitive data types by value ⌈

Description:	An API input parameter that is a primitive data type (with the exception of a string) shall be passed by value.
Rationale:	Pass by value is efficient for small data types.
Dependencies:	–
Use Case:	–
Supporting Material:	In the context of this requirement, primitive data types include integers (both signed and unsigned), floating point and opaque types.

⌋

[SRS_Rte_00060] RTE API shall pass "in" composite data types by reference ⌈

Description:	The RTE API shall pass all input parameters that are composite data types (i.e. a record or an array) or strings by reference.
Rationale:	Pass by reference is efficient for large data types.
Dependencies:	–
Use Case:	–
Supporting Material:	[SRS_Rte_00036] - Assignment to OS Applications.

⌋

[SRS_Rte_00061] "in/out" and "out" parameters ⌈

Description:	The RTE API shall pass ‘in/out’ and "out" formal parameters by reference.
Rationale:	Required so that modifications to the actual parameters made by the called function are visible to the caller.
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋

[SRS_Rte_00115] API for data consistency mechanism ⌈

Description:	The RTE shall provide an API to access the data consistency mechanism(s).
Rationale:	The data mechanism may rely on AUTOSAR OS mechanisms (e.g. resources) which cannot be accessed directly by Application Software Components.
Dependencies:	[SRS_Rte_00032] - data consistency mechanism.
Use Case:	–
Supporting Material:	VFB Requirements (VFB_C60, Sched70)

⌋

[SRS_Rte_00075] API for accessing per-instance memory ⌈

Description:	The RTE generator shall generate an API in the application header file through which the Runnable Entities of a component instance can access their per-instance memory for reading and writing.
Rationale:	Required by the software component template
Dependencies:	[SRS_Rte_00013] - per-instance memory.
Use Case:	–
Supporting Material:	"per-instance memory" is synonymous to "Individual data of a component instance" described in Software Component Template [4] The RTE does not impose a data consistency mechanism on access to per-instance memory. If a component requires consistency then the RTEEnter and RTEExit API calls should be used.

⌋

[SRS_Rte_00107] Support for INFORMATION_TYPE attribute ⌈

Description:	The RTE generator shall support the INFORMATION_TYPE attribute with values "data" and "event". The RTE generator shall support different information types for each data item in an AUTOSAR interface. The RTE generator shall raise a configuration-time error if the specification of INFORMATION_TYPE is inconsistent for sender and receiver.
Rationale:	Required by VFB Specification.
Dependencies:	–
Use Case:	–
Supporting Material:	VFB Specification v1.03, p. 40 VFB Specification v1.03, Section 4.1.7.4 VFB Specification v1.03, Table 4-15 VFB Specification v1.04, p. 61 line 14 When "data" is specified, the RTE shall presume the following; Receive mode shall be (explicit read) "data_read_access". Buffering shall be "last_is_best". Specification of an initial value is required. Specification of "TIME_FOR_RESYNC" is required. Specification of "LIVELIHOOD" is required. When "event" is specified, the RTE shall presume the following: The "TIME_FOR_RESYNC" is not specified. The "LIVELIHOOD" is not specified. An attempt to redefine the presumptions shall cause a configuration time error. Failure to fulfill the presumptions shall cause a configuration time error.

⌋

[SRS_Rte_00108] Support for INIT_VALUE attribute ⌈

Description:	The RTE generator shall support the INIT_VALUE attribute for both intra-ECU and inter-ECU communication (though the latter is expected to requires no direct support if AUTOSAR COM is used). If an initial value is specified for a receiver and not a sender (or vice versa) the RTE generator shall apply the same initial value to both sender and receiver. If an initial value is specified for both sender and receiver the RTE generator shall use the specifications for the receiver.
Rationale:	The input information can contain conflicting values for the INIT_VALUE attribute (sender and receiver).
Dependencies:	[SRS_Rte_00068] - Signal initial values.
Use Case:	The INIT_VALUE is different in the sender- and receiver com spec.
Supporting Material:	VFB Specification v1.03, p. 42

⌋

[SRS_Rte_00109] Support for RECEIVE_MODE attribute ⌈

Description:	The RTE generator shall support the RECEIVE_MODE attribute with the values "data_read_access", "wake_up_of_wait_point" and "activation_of_runnable_entity". The RTE generator shall support different receive modes for each data item in an AUTOSAR interface.
Rationale:	Derived from the VFB Specification.
Dependencies:	–
Use Case:	–
Supporting Material:	VFB Specification v1.03, p. 43 When "data_read_access" is specified the RTE generator shall create a non-blocking read API for the data item. The name of the API could include the port name and data item name. When "wake_up_of_wait_point" is specified the RTE generator shall create a blocking read API for the data item. The name of the API shall include the port name and data item name. The API could support a timeout specified at configuration time. When "activation_of_runnable_entity" is specified the RTE generator shall invoke a Runnable Entity when data is received passing the received data as parameters to the Runnable Entity. The name of the Runnable Entity could include the port name and data item name.

⌋

[SRS_Rte_00110] Support for BUFFERING attribute ⌈

Description:	The RTE generator shall support the BUFFERING attribute with the values "last_is_best" (sender/receiver only), "queue" and "no" (client/server only). The RTE generator shall support different buffering specifications for each data item in an AUTOSAR interface. Note the queues may be implemented by either the RTE or by COM.
Rationale:	–
Dependencies:	[SRS_Rte_00033] - Serialization of Server Runnable Entities.
Use Case:	–
Supporting Material:	VFB Specification v1.03, p. 43 When "last_is_best" is specified the RTE generator Shall create a non-consuming read API for the data item. The name of the API shall include the port name and data item name. Shall store received data shall be stored in a single-element queue and new data shall overwrite existing data. When "queue" is specified for sender/receiver the RTE generator Shall create a consuming read API for the data item. The name of the API shall include the port name and data item name. Shall store received data in a queue (the length of which is specified by the "queue" attribute value) accessed on a first-in-first-out basis. Shall discard new data if the queue is full. When "no" or "queue" is specified for client/server see [SRS_Rte_00033].

⌋

[SRS_Rte_00111] Support for CLIENT_MODE attribute ⌈

Description:	The RTE generator shall support the CLIENT_MODE attribute with the values "synchronous" and "asynchronous". The RTE generator shall support different client mode specifications for each operation in an AUTOSAR interface.
Rationale:	–
Dependencies:	[SRS_Rte_00049] - construction of task bodies.
Use Case:	–
Supporting Material:	VFB Specification v1.03, p. 51 When "synchronous" is specified the RTE generator Shall create an API that invokes the operation synchronously. The name of the API could include the port name and operation name. Shall support a timeout specified at the configuration time. The RTE generator should ignore any timeout specified for intra-task communication. When "asynchronous" is specified the RTE generator Shall create an API that invokes the operation asynchronously. The name of the API could include the port name and operation name. Reject configurations that specify asynchronous invocation of server where both Runnable Entities are mapped to the same task.

⌋

[SRS_Rte_00121] Support for FILTER attribute ⌈

Description:	The RTE generator shall support the FILTER attribute. If specified, the attribute value shall specify the filter type used. The RTE generator shall support different filter specifications for each data item in an AUTOSAR interface. The RTE generator shall apply value-based filtering regardless whether communication occurs via COM or is handled by the RTE.
Rationale:	Same behavior independent of RTE implementation and component deployment.
Dependencies:	–
Use Case:	–
Supporting Material:	VFB Specification v1.03, p. 44

⌋

[SRS_Rte_00147] Support for communication infrastructure time-out notification ⌈

Description:	The RTE shall support the notification of time-outs on cyclically received signals/signal-groups via COM. The deadline monitoring has to be enabled for these signals and the callback has to be configured in COM. This is only applicable for sender-receiver communication with info-type "data".
Rationale:	Indicate the missing update of signals received via COM.
Dependencies:	[SRS_Rte_00069]
Use Case:	When the "vehicle speed" signals is not updated because of communication infrastructure errors it needs to be indicated to the SW-Components interested.
Supporting Material:	nonBSW Feature list (v0.40) F049 The value is specified as "aliveTimeout" in the Software Component Template [4] (Required ComSpec). The value is specified as "timeout" in combination with "needsOutdatedIndication" in the System Template (SystemSignal). COM already performs the "deadline monitoring" and notifies the RTE.

⌋

[SRS_Rte_00078] Support for Data Element Invalidation ⌈

Description:	The RTE shall support the invalidation of Data Elements. The RTE shall provide an API to invalidate a Data Element and to query if a Data Element has been invalidated. Also a notification on the reception of an invalid Data Element shall be supported.
Rationale:	The "canInvalidate" communication attribute shall be visible to the software components.
Dependencies:	–

▽

△

Use Case:	Data invalid
Supporting Material:	–

⌋

[SRS_Rte_00122] Support for Transmission Acknowledgement ⌈

Description:	The RTE generator shall support Transmission Acknowledgement for outgoing communication.
Rationale:	Allow the transmitting Application Software Component to wait for the acknowledgement and handle the successful / failed transmission.
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋

[SRS_Rte_00094] Communication and Resource Errors ⌈

Description:	The RTE shall handle errors related to communication or resources. The RTE is required to handle communication errors (e.g. message transmission failed) and resource errors (e.g. network not available) and to notify the relevant software component through the RTE API.
Rationale:	–
Dependencies:	[SRS_Rte_00084] - Support infrastructural errors.
Use Case:	–
Supporting Material:	Software Component Template [4] VFB v1.03 explicitly delegates the definition of the error handling mechanism to be defined by WP RTE.

⌋

[SRS_Rte_00084] Support infrastructural errors ⌈

Description:	The RTE API shall support the forwarding of infrastructural errors (see [3]) to components. This can occur synchronously with API calls (e.g. read, send) or asynchronously (i.e. activation of Runnable Entity). Infrastructural errors include communication and resource errors and are split into two groups: Immediate Infrastructure Error: If the RTE detects an error which is specific to the current processed data. The remaining bit0..bit5 may describe the specific error. Overlayed Error: If the RTE detects an error which is not specific to the current processed data. Overlayed Errors are set using bit6.

▽

△

Rationale:	VFB Specification [3]
Dependencies:	[SRS_Rte_00094] - Communication and Resource Errors.
Use Case:	–
Supporting Material:	VFB Specification [3] Software Component Template [4]

⌋

[SRS_Rte_00123] The RTE shall forward application level errors from server to client ⌈

Description:	The RTE shall pass the application error ID together with the communication reply from the server to the client. The RTE shall only pass the application error, if no structural error is present.
Rationale:	For client-server communication, the application SW components require a method to transfer application specific errors under the condition that there are no structural errors on the communication path.
Dependencies:	[SRS_Rte_00124] - API for application level server errors.
Use Case:	A crypto library provides a server. An application error should be returned if the arguments of the call are miss-configured.
Supporting Material:	–

⌋

[SRS_Rte_00124] API for application level errors during Client Server communication ⌈

Description:	The RTE shall communicate application level errors on the same path as structural errors of the communication stack. The RTE shall receive error information from the server operation's return value.
Rationale:	This requirement enables the efficient use of return values to pass error IDs. By a common use of the return value for structural and application errors, the application only has to check once for "OK".
Dependencies:	[SRS_Rte_00123] - Forwarding of application level server errors.

▽

△

Use Case:	Rte_StatusType
	sqrt(Rte_Instance self,
	Double p, Double *result)
	}
	if (p < (Double)0.0) }
	/* Set application error
	* (API to be defined) */
	return ERROR_IMAGINARY_NUMBER;
	{
	*result = (Double)sqrt(p);
	return RTE_E_OK;
	{

Supporting Material:	–

⌋

[SRS_Rte_00089] Independent access to interface elements ⌈

Description:	The RTE API shall support independent access to data items (sender-receiver interface) or operations (client-server interface).
Rationale:	Required by the VFB Specification
Dependencies:	–
Use Case:	–
Supporting Material:	VFB Specification Data items (or operations) in an interface form multiple logical channels between the same end-points (ports). Each logical channel is handled independently - data can be sent and received or operations invoked without reference to other logical channels. Since logical channels are independent there is not guarantee of consistency between sends over different channels.

⌋

[SRS_Rte_00137] API for mismatched ports ⌈

Description:	The RTE generator shall provide null API calls for data elements or operations for ports where more elements/operations are provided than required. The API for an unconnected provided data element or operation shall discard the input parameters and return "no error".
Rationale:	–
Dependencies:	–
Use Case:	A provided sender-receiver port defines two data elements ‘a’ and ‘b’ yet is required by a port with only element ‘a’. The API call for ‘b’ shall be generated but shall have no effect.
Supporting Material:	Software Component Template [4]

⌋

[SRS_Rte_00139] Support for unconnected ports ⌈

Description:	The RTE shall handle ports, whether required or provided, that are not connected. The APIs for an unconnected required sender/receiver port shall return a dedicated status code that the sender is not connected. The result value shall be the init value. The API to call a client-server port for an unconnected required port shall return a dedicated status code that the server is not connected. The API to collect the result from an asynchronous client-server port for an unconnected required port shall return a dedicated status code that the server is not connected. The API for an unconnected provided sender/receiver port shall discard the input parameters and return "no error".
Rationale:	In a component based system design there is a high chance to end up with unconnected ports for software components. In a variant rich system design unconnected ports can occur.
Dependencies:	[SRS_Rte_00200]
Use Case:	A not connected port of a software component.
Supporting Material:	Software Component Template [4]

⌋

[SRS_Rte_00200] Support of unconnected R-Ports ⌈

Description:	The RTE Generator shall support generating an RTE with unconnected R-Ports. The strict checking of unconnected R-Ports shall be supported via configuration as well.
Rationale:	During the development of an ECU there are intermediate building states when not all needed communication partners are available yet, however it shall be possible to generate an RTE anyways.
Dependencies:	[SRS_Rte_00139]
Use Case:	The RTE Generator issues a warning for each unconnected R-Port during the generation if strict checking is enabled.
Supporting Material:	–

⌋

[SRS_Rte_00155] API to access calibration parameters ⌈

Description:	The SW-C and basic software module source code shall be independent from the actual calibration method (data emulation with SW or HW support) chosen for the needed calibration parameters. To abstract from the different access methods to calibration parameters the RTE and SchM shall provide an API.

▽

△

Rationale:	The SW-C source code shall use a dedicated API to access the calibration parameters.
Dependencies:	[SRS_Rte_00154] - Support of Calibration.
Use Case:	The SW-C code uses the same API call regardless whether the calibration parameter is stored directly in ROM or is stored in a structure to support data emulation with SW support.
Supporting Material:	Software Component Template [4] chapter "Measurement & Calibration"

⌋

[SRS_Rte_00183] RTE Read API returning the dataElement value

Upstream requirements: RS_BRF_01304

⌈

Description:	For explicit sender-receiver communication, when it is configured, the RTE generator shall provide an API returning the value of a primitive DataElement directly in the C-language return statement of the API.
Rationale:	When the SWC implementation does not need the Std_ReturnType value, this more efficient API returns the result as a return value.
Dependencies:	–
Use Case:	Many calls to RTE_Read() are expected for Application SWCs. Allow the RTE to generate efficient code which does not need special code optimization compilers to benefit from.
Supporting Material:	–

⌋

[SRS_Rte_00185] RTE API with Rte_IFeedback ⌈

Description:	The RTE shall support the generation of an API to retrieve the transmission acknowledgement in an implicit communication.
Rationale:	Allow to query the transmission state also for implicit communication.
Dependencies:	–
Use Case:	Enable a Runnable Entity to check whether the information provided from the last execution (in an implicit API) has actually been transmitted.
Supporting Material:	–

⌋

[SRS_Rte_00203] API to read system constant

Upstream requirements: RS_BRF_01136

⌈

Description:	The RTE shall provide an API to read a system constant value. This API shall be usable for the C-preprocessor code or the C-compiler code. Support for the following kinds if information shall be generated: • Read the actual value of the SystemConstantDef • Read the setting of an attribute (e.g. array size) • Check the existence of a variable element
Rationale:	The software module implementation can use the value of the system constant in its execution code. The software module implementation can query the existence of a variable element. This is applicable for Basic Software as well as for application software components.
Dependencies:	This requirement is dedicated to variant handing, a more generic requirement is [SRS_Rte_00171] which might lead to the same implementation.
Use Case:	The existence of a Port is specified to be variable with a binding time "PreCompile". The implementation of the SWC can query the value of the system constant used to enable/disable the port, in order to change its behavior.
Supporting Material:	–

⌋

[SRS_Rte_00242] Support for Cross-Core Exclusive Areas

Upstream requirements: RS_BRF_01160

⌈

Description:	On multi-core systems, the APIs to enter and exit exclusive areas shall ensure that no two entities may run inside the same exclusive area at any one time, neither by preemption nor by parallel execution of multiple entities on different cores.
Rationale:	For systems where the BSW modules can safely be executed in multiple partitions, possibly running on multiple cores, access to shared data shall be protected across partition and core boundaries.
Dependencies:	[SRS_Rte_00046]
Use Case:	BSW running on multi core systems.
Supporting Material:	–

⌋

4.1.11 C/C++ API

[SRS_Rte_00087] Software Module Header File generation ⌈

Description:	The RTE Generator shall create exactly one software module header file to be explicitly included in each C/C++ application or basic software component type that defines that component's RTE API. There may be a hierarchy of include files implicitly included.
Rationale:	Required to define API mapping and to perform optimizations and monitoring targeted for specific components. The software component header file is generated and can therefore include component specific information, including task header files for zero-overhead access to OS facilities.
Dependencies:	–
Use Case:	–
Supporting Material:	AUTOSAR design flow. This requirement does not preclude a component including its own header files.

⌋

4.1.12 Initialization and Finalization Operation

Requirements for component finalization are considered above [SRS_Rte_00052].

[SRS_Rte_00116] RTE Initialization and finalization

Upstream requirements: RS_BRF_01320

⌈

Description:	The RTE generator shall provide mechanisms to initialize and finalize the RTE in two steps: Step 1: set all initial values of the communication and mode machine instances start the schedule of BSW modules treat the communication with application SW-Cs like unconnected remote communication Step2: start the schedule and communication of all application SW-Cs The RTE startup shall support the startup of SW-C and BSW modules mapped to different OS applications, specifically different cores and memory partitions.
Rationale:	Support the initialization of the BSW Scheduler and the RTE.
Dependencies:	–
Use Case:	Initialize the mode management and the scheduling of BSW before initializing and executing the Application SW-Components.
Supporting Material:	–

⌋

4.1.13 Partition Restarting and Termination

[SRS_Rte_00196] Inter-partition communication consistency

Upstream requirements: RS_BRF_01248

⌈

Description:	When two SWCs communicate, if one SWC is on a terminated partition (resp. a partition being restarted), it should behave for the initiating SWC as if it was mapped on a shutdown remote ECU (resp. an ECU being restarted). The RTE may provide the feedback with a timeout error immediately.
Rationale:	Servers on terminated partitions will not answer and server calls should result in timeouts for the clients. Servers' feedback should be dropped if the client is terminated or has been restarted since it sent the request. Received data during a restart should be discarded and init values should be proposed to the SWCs.
Dependencies:	–
Use Case:	Avoid transmission of potentially inconsistent data. Avoid inconsistent feedback to clients (sequence number needed).
Supporting Material:	–

⌋

4.1.14 Fault Operation

Errors are directly reported to invoking Software Component (see [SRS_Rte_00094]).

4.1.15 RTE Implementation Plug-Ins

[SRS_Rte_00318] Modular Runtime Environment

Upstream requirements: RS_Main_00200, RS_Main_00150

⌈

Description:	The RTE shall support to split the implementation of the VFB functionality between RTE Generator and so called Rte Implementation Plug-Ins. Thereby two different specialization of Rte Implementation Plug-Ins shall be supported: • Local Software Cluster Communication Plug-Ins generally take care about the implementation of preemption and concurrency locks, protection of data accesses, and implicit communication buffering inside a `Software Cluster`. • Cross Software Cluster Communication Plug-Ins generally handling the communication towards a non-software-cluster-local communication partner.
Rationale:	Runtime and memory optimization of semaphore mechanisms in complex scheduling scenarios. Non AUTOSAR standardized buffering schema, e.g. LET Cross Software Cluster Communication
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋

[SRS_Rte_00300] RTE Implementation Plug-Ins for explicit communication

Upstream requirements: RS_Main_00200

⌈

Description:	The RTE shall support to implement for explicit communication following sub functionality • getting a semaphore • releasing a semaphore • implementation of the semaphore • accessing the global copy of the data • invocation of transformers • APIs to enable the communication across different ASIL levels • APIs to enable the communication between Software Clusters via RTE Implementation Plug-Ins outside the RTE Generator. This includes port based communication as well as Inter Runnable Variables.

▽

△

Rationale:	Runtime and memory optimization of semaphore mechanisms in complex scheduling scenarios.
Dependencies:	–
Use Case:	Activation of RunnableEntities accessing the same data. The access is mutually exclusive due to functional conditions.
Supporting Material:	–

⌋

[SRS_Rte_00301] RTE Implementation Plug-Ins for implicit communication

Upstream requirements: RS_Main_00200

⌈

Description:	The RTE shall support an implicit communication implementation with the following sub-functionalities: • access buffer or global copy by a RunnableEntity • decision about buffering • implementation of the buffering • buffer fill and buffer flush routines via RTE Implementation Plug-Ins outside the RTE Generator. This includes port based communication as well as Inter Runnable Variables.
Rationale:	Runtime and memory optimization of implicit communication buffering. Customized buffering strategies differing from the RTE standardized ones.
Dependencies:	–
Use Case:	Time triggered buffering supporting the Logical Execution Time paradigms.
Supporting Material:	–

⌋

[SRS_Rte_00320] RTE Implementation Plug-Ins for implicit communication II

Upstream requirements: RS_Main_00200

⌈

Description:	The RTE shall support for an implicit communication implementation that the `Local Software Cluster Communication Plug-In` can control point in time when the `Cross Software Cluster Communication Plug-In` reads and writes data from / to other Software Clusters
Rationale:	–
Dependencies:	–

▽

△

Use Case:	Time triggered buffering supporting the Logical Execution Time (LET) paradigms.
Supporting Material:	–

⌋

[SRS_Rte_00302] RTE Implementation Plug-Ins for exclusive areas

Upstream requirements: RS_Main_00200

⌈

Description:	The RTE shall support an exclusive area implementation with the following sub-functionalities: • getting a semaphore • releasing a semaphore • implementation of the semaphore via RTE Implementation Plug-Ins outside the RTE Generator.
Rationale:	Runtime and memory optimization of semaphore mechanisms in complex scheduling scenarios.
Dependencies:	–
Use Case:	Activation of RunnableEntities accessing the same data. The access is mutually exclusive due to functional conditions.
Supporting Material:	–

⌋

[SRS_Rte_00303] RTE Implementation Plug-Ins for global copy instantiation

Upstream requirements: RS_Main_00200

⌈

Description:	The RTE shall support to implement the instantiation of the global copy via RTE Implementation Plug-Ins outside the RTE Generator.
Rationale:	Data consistency mechanisms with various global copies.
Dependencies:	RTE measurement support with symbols for measurement buffers.
Use Case:	Data consistency by ring-buffer principle.
Supporting Material:	–

⌋

[SRS_Rte_00304] Multiple RTE Plug-Ins

Upstream requirements: RS_Main_00200

⌈

Description:	The RTE shall support the implementation of specific sub-functionalities in distinct communication graphs. It shall be realized either by dedicated RTE Implementation Plug-Ins or by the RTE Generator. Thereby all accesses to one communication graph are handled either • by RTE only OR • by RTE + exactly one `Local Software Cluster Communication Plug-In` OR • by RTE + exactly one `Cross Software Cluster Communication Plug-In` OR • by RTE + exactly one `Local Software Cluster Communication Plug-In` + exactly one `Cross Software Cluster Communication Plug-In` Different communication graphs might have a different configuration w.r.t. to the mentioned scenarios above.
Rationale:	Multiple RTE Implementation Plug-In instances in one environment.
Dependencies:	[SRS_Rte_00300]; [SRS_Rte_00301]; [SRS_Rte_00302]; [SRS_Rte_00303] ; [SRS_Rte_00311]; [SRS_Rte_00312]; [SRS_Rte_00315]
Use Case:	In a large set of software components only few communication graphs are utilizing RTE features which are not supported by the RTE Plug-In provider. Such communication graphs are fully implemented by RTE Generator whereas the majority of communication can still be optimized. In a large ECU different sets of software components applying different scheduling principles. Different scheduling principles require different RTE Implementation Plug-Ins in one environment. The cross cluster communication is implemented by another vendors solution as the internal communication.
Supporting Material:	–

⌋

[SRS_Rte_00305] Graduated validation strategy

Upstream requirements: RS_Main_00200

⌈

Description:	The RTE shall support a graduated validation strategy to detect and to reject invalid configurations. In case a communication graph is assigned to an RTE Implementation Plug-In the RTE Generator shall only apply validation checks safeguarding the sub-functionality in the RTE implementation. In addition the RTE Implementation Plug-In provider shall apply checks safeguarding the sub-functionality in the RTE Implementation Plug-In.
Rationale:	An RTE Implementation Plug-In may handle configurations which are not supported by a standard RTE Generator.
Dependencies:	[SRS_Rte_00018]
Use Case:	The standard implicit buffering in RTE is not capable to handle pre-emptiv scheduled Runnable since this would require multiple buffers for one data access. In case the RTE Implementation Plug-In offers functionality to handle multiple buffers for one data access of a Runnable it can accept this configuration.
Supporting Material:	–

⌋

[SRS_Rte_00306] Standardized interfaces for RTE Implementation Plug-Ins

Upstream requirements: RS_Main_00200

⌈

Description:	In case a communication graph is assigned to one or more RTE Implementation Plug-In(s) the RTE Generator shall provide a standardized interface to implement the selected sub-functionality by the assigned RTE Implementation Plug-In(s).
Rationale:	RTE Implementation Plug-In and RTE Generator are provided by different vendors.
Dependencies:	[SRS_Rte_00300]; [SRS_Rte_00301]; [SRS_Rte_00302]; [SRS_Rte_00303] ; [SRS_Rte_00311]; [SRS_Rte_00312]; [SRS_Rte_00315]
Use Case:	–
Supporting Material:	–

⌋

[SRS_Rte_00307] RTE Implementation Plug-Ins for cross core communication

Upstream requirements: RS_Main_00200

⌈

Description:	The RTE shall support the implementation of cross core communication between software components in an RTE Implementation Plug-In.
Rationale:	Distribute whole software components to different cores.
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋

[SRS_Rte_00309] RTE Implementation Plug-Ins for cross safety partition communication

Upstream requirements: RS_Main_00200

⌈

Description:	The RTE shall support the implementation of cross safety partition communication between software components with a different ASIL level in an RTE Implementation Plug-In.
Rationale:	ISO26262 "Freedom from interference" for safety related functionalities
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋

[SRS_Rte_00310] Shared mode queue

Upstream requirements: RS_Main_00200

⌈

Description:	The RTE shall queue mode switch requests of configurable sets of mode machine instances in common queues. Thereby the mode machine instances may be assigned to different partitions.
Rationale:	Preserve global execution order of mode switch requests.
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋

[SRS_Rte_00311] Core synchronous transitions for mode switches

Upstream requirements: RS_Main_00200

⌈

Description:	The RTE shall provide interfaces for the graduated ramp-down and ramp-up of the task schedule during a mode switch. Thereby the interfaces shall support the notification of a task coordinator about:: • the reception of the mode switch notification • the execution of on-entry ExecutableEntitys, on-transition ExecutableEntitys, on-exit ExecutableEntitys • the dequeuing of the mode switch notification Additionally the interface shall support: • delaying all pending and newly activated tasks until the transition is over • waiting until currently executed tasks have been finished. • resuming the task system on all cores/partitions in the group
Rationale:	Deterministic task execution during mode switches.
Dependencies:	–
Use Case:	The mode transition in a multi core configuration with many OS task shall be performed in a managed way.
Supporting Material:	–

⌋

[SRS_Rte_00312] RTE Implementation Plug-Ins for transformers in client server communication

Upstream requirements: RS_Main_00200

⌈

Description:	The RTE shall support to implement the invocation of transformers for client server communication in an RTE Implementation Plug-In.
Rationale:	Adapt the transformer buffer allocation strategy (static, stack based) to the ECU SW architecture needs.
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋

[SRS_Rte_00317] RTE Implementation Plug-Ins for transformers in trigger communication

Upstream requirements: RS_Main_00200

⌈

Description:	The RTE shall support to implement the invocation of transformers for trigger communication in an RTE Implementation Plug-In.
Rationale:	Adapt the transformer buffer allocation strategy (static, stack based) to the ECU SW architecture needs.
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋

[SRS_Rte_00319] RTE Implementation Plug-Ins for parameter communication

Upstream requirements: RS_Main_00200, RS_Main_00150

⌈

Description:	The RTE shall support to implement for parameter communication following sub functionality: • accessing the global copy of the parameter • APIs to enable the reading from other Software Clusters.
Rationale:	Cross Software Cluster Communication with the restriction that parameters are read-only
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋

[SRS_Rte_00313] Description of RTE Implementation Plug-in properties

Upstream requirements: RS_Main_00200

⌈

Description:	The RTE Implementation Plug-In shall describe its implementation properties relevant for the RTE Generator.
Rationale:	The to be generated RTE code can depend on RTE Implementation Plug-In properties.
Dependencies:	–
Use Case:	In case of a buffering strategy without global copy the RTE Generator shall not provide a global copy for the related communication graph.

▽

△

Supporting Material:	–

⌋

[SRS_Rte_00314] Avoid nesting of critical sections

Upstream requirements: RS_Main_00200

⌈

Description:	The RTE code shall prevent from nesting of critical sections caused from the interleaved usage of macro based RTE API.
Rationale:	In case the RTE API gets implemented as macro and the SWC uses one RTE API as argument for another RTE API it may occur that the critical sections (and according protection calls) might be interleaved with the critical sections of the other macros. Such a code can cause dead locks.
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋

[SRS_Rte_00315] Protection of mode machine instance access

Upstream requirements: RS_Main_00200

⌈

Description:	The RTE shall support to protect the access to mode machine instances and shared mode queues related data via the RTE Implementation Plug-in.
Rationale:	Runnables accessing the same mode machine instance from different cores. Runnables accessing mode machines instances handled in a common shared mode queue from different cores.
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋

[SRS_Rte_00321] RTE Implementation Plug-Ins for mode communication

Upstream requirements: RS_Main_00200, RS_Main_00150

⌈

Description:	The RTE shall support to implement for mode communication following sub functionality: • inform the `Cross Software Cluster Communication Plug-In` about start of a mode switch • inform the `Cross Software Cluster Communication Plug-In` about end of a mode switch • dequeuing of the mode switch notification by RTE after the `Cross Software Cluster Communication Plug-In` has synchronized the mode switches of the receiving `Software Cluster`
Rationale:	Cross Software Cluster Communication of mode switches
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋

[SRS_Rte_00316] RTE Implementation Plug-Ins for compatibility mode

Upstream requirements: RS_Main_00200

⌈

Description:	The RTE and RTE Implementation Plug-ins shall support to apply RTE Implementation Plug-ins for Software Components using the compatibility mode.
Rationale:	–
Dependencies:	[SRS_Rte_00145]
Use Case:	A Software Component is integrated as object code.
Supporting Material:	–

⌋

4.2 Non-Functional Requirements

4.2.1 General Requirements

[SRS_Rte_00064] AUTOSAR methodology ⌈

Description:	The RTE generator shall operate according to the AUTOSAR methodology.
Rationale:	–
Dependencies:	–
Use Case:	–
Supporting Material:	AUTOSAR Methodology

⌋

[SRS_Rte_00019] RTE is the communication infrastructure ⌈

Description:	All communication between Application Software Components and between Application Software Components and basic software components shall occur, at least conceptually, via the RTE. Note that communication between modules within the basic software and to shared libraries does NOT occur through the RTE. This is a basic requirement and ensures that the RTE controls all communication involving Application Software Components. This requirement is not intended to prevent the RTE generator providing optimizations that bypass the RTE such as optimizing client-server to direct function call.
Rationale:	AUTOSAR ECU architecture
Dependencies:	–
Use Case:	–
Supporting Material:	VFB Specification This requirement applies regardless of whether communication is done by COM, by the RTE directly or if the RTE generator optimizes the generated RTE to bypasses the RTE completely for certain communication paths. The phrase "at least conceptually" is used to indicate that on the conceptual (model M2/M1 levels) all communication occurs via the virtual function bus and, since the RTE is the realization of the VFB for an ECU, via the RTE. However this is only conceptual since the actual implementation (model M0 level) may not use the RTE for communication. For example, client-server communication conceptually occurs via the RTE but may be implemented as a direct function call (and hence bypass the RTE).

⌋

5 Change history of AUTOSAR traceable items

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

5.1 Traceable item history of this document according to AUTOSAR Release R24-11

5.1.1 Added Requirements in R24-11

none

5.1.2 Changed Requirements in R24-11

none

5.1.3 Deleted Requirements in R24-11

none

5.2 Traceable item history of this document according to AUTOSAR Release R23-11

5.2.1 Added Requirements in R23-11

none

5.2.2 Changed Requirements in R23-11

Number	Heading
[SRS_Rte_00318]	Modular Runtime Environment
[SRS_Rte_00319]	RTE Implementation Plug-Ins for parameter communication
[SRS_Rte_00321]	RTE Implementation Plug-Ins for mode communication

Table 5.1: Changed Requirements in R23-11

5.2.3 Deleted Requirements in R23-11

Number	Heading
[SRS_Rte_00195]	No activation of Runnable Entities in terminated or restarting partitions
[SRS_Rte_00223]	Callout for partition termination notification
[SRS_Rte_00224]	Callout for partition restart request

Table 5.2: Deleted Requirements in R23-11

5.3 Traceable item history of this document according to AUTOSAR Release R22-11

5.3.1 Added Requirements in R22-11

none

5.3.2 Changed Requirements in R22-11

Number	Heading
[SRS_Rte_00201]	Contract Phase with Variant Handling support
[SRS_Rte_00206]	Support the selection of a signal provider
[SRS_Rte_00207]	Support N to M communication patterns while unresolved variations are affecting these communications

Table 5.3: Changed Requirements in R22-11

5.3.3 Deleted Requirements in R22-11

Number	Heading
[SRS_Rte_00149]	Support "Specification of Compiler Abstraction"

Table 5.4: Deleted Requirements in R22-11