Elektrobit

EB tresos® Safety RTE safety manual

EB tresos Safety RTE

Date: 2022-11-14, Document version 3.5, Status: RELEASED

Technical support

https://www.elektrobit.com/support

Legal disclaimer

Confidential information.

Table of Contents

1. Document history

Version	Date	Author	State	Description
0.1	2013-06-13	Elektrobit Automotive GmbH	DRAFT	Initial version
0.2	2013-09-12	Elektrobit Automotive GmbH	DRAFT	Reworked after EB tresos Safety RTE feature discussion.
0.2.1	2015-03-02	Elektrobit Automotive GmbH	DRAFT	Added safety requirements
0.2.2	2015-03-09	Elektrobit Automotive GmbH	DRAFT	► Adapted document structure ► Added document information
0.2.3	2015-03-18	Elektrobit Automotive GmbH	DRAFT	► Adapted document structure ► Added appendix defining the Safety Element out of Context
0.2.4	2015-03-26	Elektrobit Automotive GmbH	DRAFT	Reworked formatting of specification objects.
0.2.5	2015-05-29	Elektrobit Automotive GmbH	DRAFT	Updated assumptions, limitations and limitation checks to the AUTOSAR 4.0.3 HAZOPs for sender/receiver communication.
0.2.6	2015-08-13	Elektrobit Automotive GmbH	DRAFT	Updated review instructions for intra-ECU sender/receiver communication.
0.2.7	2015-09-03	Elektrobit Automotive GmbH	DRAFT	► Added review instructions for lifecycle APIs and tasks. ► Added review instructions for API data types (error and status codes).
0.2.8	2015-09-08	Elektrobit Automotive GmbH	DRAFT	Added limitations, limitation checks, and review instructions for: ► Implementation data types ► Partitioning ► Inter-ECU sender/receiver communication

Version	Date	Author	State	Description
0.2.9	2015-09-18	Elektrobit Automotive GmbH	PROPOSED	▶ Updated and corrected limitations, limitation checks, and review instructions. ▶ Added safe use chapter and restructured some existing chapters in that process.
0.3.0	2015-12-04	Elektrobit Automotive GmbH	PROPOSED	▶ Moved almost complete chapter *Using EB tresos Safety RTE* from the *EB tresos Safety RTE user's guide* except from chapter *Generating the RTE test cases*. ▶ Updated all links in moved chapters. ▶ Corrected exclusive area reviews. ▶ Added definition of process words and terms for the assumed safety requirements. ▶ Added description of the EB tresos Safety RTE workflow.
0.3.1	2015-12-07	Elektrobit Automotive GmbH	PROPOSED	Fixed inspection findings in the safety requirements
0.3.2	2015-12-10	Elektrobit Automotive GmbH	PROPOSED	▶ Added reviews for explicit exclusive areas. ▶ Updated limitations and assumptions.
0.4.0	2016-01-27	Elektrobit Automotive GmbH	PROPOSED	Added review instructions for inter-ECU sender/receiver communication
0.4.1	2016-02-22	Elektrobit Automotive GmbH	PROPOSED	Added additional review instructions for implementation data types.
0.5.0	2016-02-23	Elektrobit Automotive GmbH	PROPOSED	Added review instructions for inter-runnable variables.
0.5.1	2016-02-25	Elektrobit Automotive GmbH	PROPOSED	Updated macro for calling OS implementation for `WaitGetClearEvent` in detailed review instructions.
0.6.0	2016-02-26	Elektrobit Automotive GmbH	PROPOSED	▶ Updated supported EB tresos AutoCore to version EB tresos AutoCore Generic 6.4.7. ▶ Updated supported EB tresos Studio to version 14.4.1.

Version	Date	Author	State	Description
0.7.0	2016-05-25	Elektrobit Automotive GmbH	PROPOSED	▶ Added additional review instructions for the partitioning mechanism of the RTE. ▶ Added review instructions for synchronous client/server communication. ▶ Added instruction how to verify the integrity of EB tresos Safety RTE. ▶ Updated workflow description. ▶ Incorporated review findings from TE.
0.7.1	2016-07-18	Elektrobit Automotive GmbH	PROPOSED	▶ Updated assumptions, limitations, limitation checks and review instructions according to inspection findings. ▶ Incorporated review findings from TE.
0.7.2	2016-07-19	Elektrobit Automotive GmbH	PROPOSED	Added reference to the *EB tresos Safety RTE release notes*.
1.0.0	2016-07-20	Elektrobit Automotive GmbH	RELEASED	Inspection passed without further findings.
2.0.0	2016-12-09	Elektrobit Automotive GmbH	PROPOSED	Updated reviewed instructions for the synchronous client/server API.
2.1.0	2016-12-14	Elektrobit Automotive GmbH	PROPOSED	Updated review instructions for the inter-runnable variables.
2.1.1	2017-03-23	Elektrobit Automotive GmbH	PROPOSED	▶ Updated review instructions for inter-ECU and intra-ECU sender/receiver API. ▶ TE review of all chapters which are related to inter-runnable variables. ▶ TE review of all chapters which are related to client/server API. ▶ Updated assumption for a validated OS
2.2.0	2017-03-24	Elektrobit Automotive GmbH	PROPOSED	Created release for EB tresos Safety RTE 3.0.2.

Version	Date	Author	State	Description
2.2.1	2017-05-03	Elektrobit Automotive GmbH	PROPOSED	▶ Updated review instructions for the lifecycle and task APIs. ▶ Reintroduced manual review instructions for inter-runnable variables. ▶ Added additional conditions for review instruction `SafetyRTE.Review.SenderReceiver.IntraECU.CriticalSectionProtection`. ▶ Updated review instructions for implementation data types. ▶ Updated review instructions for explicit exclusive area API. ▶ Updated versions to ACG7 ▶ Added reference to open issue list ▶ Reintroduced manual review instructions for inter-ECU and intra-ECU sender/receiver communication. ▶ Added chapter "Executing static code checks"
2.2.2	2017-05-12	Elektrobit Automotive GmbH	PROPOSED	Fixed inspection findings.
2.2.3	2017-05-17	Elektrobit Automotive GmbH	PROPOSED	Updated review instruction regarding signal queue implementation.
2.2.4	2017-06-12	Elektrobit Automotive GmbH	PROPOSED	▶ Removed review instructions regarding implementation data types after implementing code analysis. ▶ Removed review instructions regarding the SchM Lifecycle API and the generation of OsTasks. These are now verified with automated code checks.
2.2.5	2017-06-13	Elektrobit Automotive GmbH	PROPOSED	Fixed inspection findings.

Version	Date	Author	State	Description
2.3.0	2017-06-14	Elektrobit Automotive GmbH	RELEASED	Inspection passed without further findings.
2.3.1	2017-11-09	Elektrobit Automotive GmbH	RELEASED	▶ Updated review instructions for exclusive areas. ▶ Updated limitation regarding supported implementation datatypes. ▶ Removed review instructions regarding the Rte Lifecycle API. These are now verified with automated code checks. ▶ Added assumptions for the compiler usage. ▶ Removed review instructions regarding the explicit sender receiver (Rte_Write and Rte_Read) API. These are now verified with automated code checks. ▶ Fixed sentence structures of lists according to editorial guidelines. ▶ Removed limitation regarding not supporting references to SwVariableRefProxy. Added limitation about not supporting ForeignModelReference. ▶ Updated wording regarding Test Generator, to reflect that the use of the Test Generators is optional. ▶ Added new limitation for cooperative tasks feature. ▶ Removed review instructions regarding the sender receiver Smc buffer usage. These are now verified with automated code checks.
2.3.2	2018-03-21	Elektrobit Automotive GmbH	RELEASED	▶ Removed the description of static code checker. The contents are moved to user guide. ▶ Added safety requirements for data transformers. ▶ Added review for the initialization of the execution counter. ▶ Added limitation for /Rte/RteGeneration/ComCbkNotInterruptable configuration parameter.

Version	Date	Author	State	Description
				▶ Added review for com signal group length.
2.3.3	2018-07-06	Elektrobit Automotive GmbH	RELEASED	▶ Partially removed review instructions regarding the Inter-runnable variables. These are now verified with automated code checks. ▶ Moved the instructions for evaluating the results of the test application to user guide.
2.3.4	2018-08-29	Elektrobit Automotive GmbH	PROPOSED	▶ Added new limitations due to new features in the RTE. ▶ Added new limitation reviews due to new features in the RTE.
2.3.5	2018-09-05	Elektrobit Automotive GmbH	RELEASED	Inspection passed without further findings.
2.3.6	2018-11-12	Elektrobit Automotive GmbH	RELEASED	▶ Added limitation for /Rte/RteGeneration/InterPartitionCommunication configuration parameter. ▶ Added limitation for single partition.
2.3.7	2018-12-13	Elektrobit Automotive GmbH	RELEASED	▶ Updated exclusive area APIs table. ▶ Added limitations for RIPS feature. ▶ Added limitations for init events and initialization of runnable batches feature.
2.3.8	2019-01-15	Elektrobit Automotive GmbH	RELEASED	▶ Added missing NoVfbTracing limitations. ▶ Rephrased text of SingleSenderReceiver Limitation Review. ▶ Rephrased a sender/receiver limitation review.
2.3.9	2019-01-22	Elektrobit Automotive GmbH	RELEASED	Fixed inspection findings.
2.3.10	2019-01-24	Elektrobit Automotive GmbH	PROPOSED	Added review instructions for the function AtomicThreadFence.
2.3.11	2019-01-30	Elektrobit Automotive GmbH	RELEASED	Fixed inspection findings.

Version	Date	Author	State	Description
2.3.12	2019-03-07	Elektrobit Automotive GmbH	PROPOSED	▶ Added limitation for OneSendSignalQueuePerCore for multicore ECUs. ▶ Removed limitation and limitation review for transformer buffer length type. ▶ Updated limitation review for NoDataFilters for sender receiver.
3.0.0	2019-04-05	Elektrobit Automotive GmbH	PROPOSED	▶ Added description of validation activities for the Rte data model. ▶ Removed RteCV limitations chapter. ▶ Added limitation and review for the supported implementation data type categories.
3.0.1	2019-04-23	Elektrobit Automotive GmbH	PROPOSED	Added limitation and review for the base type category of primitive implementation data types.
3.0.2	2019-08-08	Elektrobit Automotive GmbH	PROPOSED	▶ Removed OsResource from SafetyRTE.RS.BswExclusiveAreas requirement. ▶ Updated HAZOP destination versions after Smc changes. ▶ Updated reviews after Smc changes. ▶ Updated the detailed partitioning review instructions. ▶ Removed obsolete OneSendSignalQueuePerCore limitation and added limitation for SendSignalQueueStrategy. ▶ Added new limitation for InterEcu E2ETransformerConfiguration.
3.0.3	2019-08-30	Elektrobit Automotive GmbH	PROPOSED	▶ Added requirement for multi-core support. ▶ Updated sender/receiver requirements for multi-core support. ▶ Updated the Bsw exclusive area requirement mechanisms for multi-core support. ▶ Added and updated term definitions for multi-core support.

Version	Date	Author	State	Description
3.0.4	2019-09-12	Elektrobit Automotive GmbH	PROPOSED	▶ Removed SafetyRTE.Review.SenderReceiver.Smc.ReceiveBuffer after implementing code check. ▶ Added spinlock data consistency mechanism.
3.0.5	2019-09-20	Elektrobit Automotive GmbH	PROPOSED	▶ Fixed inspection findings. ▶ Fixed requirements tracing.
3.0.6	2019-10-11	Elektrobit Automotive GmbH	PROPOSED	▶ Adapted for multi-core feature. ▶ Fixed requirements tracing for SafetyTransformer HAZOP.
3.0.7	2019-10-21	Elektrobit Automotive GmbH	PROPOSED	Fixed inspection findings.
3.0.8	2019-11-21	Elektrobit Automotive GmbH	PROPOSED	▶ Added limitation for internal OS resources. ▶ Adapted chapter for validating the RTE data model report.
3.0.9	2019-12-02	Elektrobit Automotive GmbH	PROPOSED	▶ Added limitation for inter-ECU sender/receiver connections via compatible ports. ▶ Added limitation review for unconnected ports. ▶ Added manual review instructions for the RTE data model report. ▶ Fixed inspection findings.
3.0.10	2019-12-04	Elektrobit Automotive GmbH	RELEASED	Set to released: ASCRTECV-1761
3.0.11	2019-12-19	Elektrobit Automotive GmbH	PROPOSED	Removed limitation and limitation review for inter-ECU sender/receiver connections with compatible port interfaces.
3.0.12	2020-03-30	Elektrobit Automotive GmbH	PROPOSED	▶ Updated reviews SafetyRTE.Review.SenderReceiver.Buffer.InitialValue and SafetyRTE.Review.InterRunnableVariables.SharedBuffer.InitialValue. ▶ Added review SafetyRTE.Review.Lifecycle.Rte_Common_Init.Implementation. ▶ Updated SafetyRTE.Assumption.CoverageAnalysis.

Version	Date	Author	State	Description
				▶ Combined and rephrased reviews for inter and intra ECU sender reciever 1-to-1 Communication. ▶ Added note in SafetyRTE.Assumption.RteLifecycleUsage.
3.0.13	2020-04-09	Elektrobit Automotive GmbH	RELEASED	▶ Fixed inspection findings. ▶ Set to released: ASCRTECV-1825
3.0.14	2020-06-08	Elektrobit Automotive GmbH	PROPOSED	Replaced RTE data model by serialized RTE intermediate model.
3.0.15	2020-08-03	Elektrobit Automotive GmbH	PROPOSED	▶ Added new assumption to reject all test results for unsupported `RTE` features. ▶ Added new limitation and limitation review for unsupported transformer buffer length computation. ▶ Updated limitation review for enumeration constants.
3.0.16	2020-08-24	Elektrobit Automotive GmbH	PROPOSED	▶ Fixed inspection findings. ▶ Removed obsolete SafetyRTE.Limitation.SenderReceiver.NoInternalBufferBitFieldOptimization. ▶ Added SafetyRTE.Limitation.SenderReceiver.DynamicLengthComSignal.
3.0.17	2020-08-27	Elektrobit Automotive GmbH	RELEASED	▶ Fixed inspection findings. ▶ Set to released: ASCRTECV-1885
3.0.18	2020-12-16	Elektrobit Automotive GmbH	PROPOSED	▶ Updated the trace hooks requirement. ▶ Added AUTOSAR compatibility statement. ▶ Removed obsolete limitation SafetyRTE.Limitation.ClientServer.ServerArgumentImplPolicy and the according review. ▶ Harmonize the usage of the term application software in the entire document.

Version	Date	Author	State	Description
3.0.19	2020-12-17	Elektrobit Automotive GmbH	RELEASED	Set to released: ASCRTECV-2063
3.0.20	2021-03-15	Elektrobit Automotive GmbH	PROPOSED	▶ Extended the exclusive area appendix. ▶ Added new definition for term serialization. ▶ Removed unnecessary coverages.
3.0.21	2021-03-16	Elektrobit Automotive GmbH	PROPOSED	▶ Added limitation SafetyRTE.Limitation.ClientServer.NoInterEcuCommunication. ▶ Added limitation SafetyRTE.Limitation.SenderReceiver.NoUnmappedRunnables.
3.0.22	2021-03-29	Elektrobit Automotive GmbH	RELEASED	Fixed inspection findings.
3.0.23	2021-05-17	Elektrobit Automotive GmbH	PROPOSED	▶ Added limitation SafetyRTE.Limitation.Lifecycle.NoSingleScheduleTable. ▶ Added limitation and review for unconnected client/server ports. ▶ Changes for introspection checker. ▶ Added limitation and review for RTE ASIL-B tooling review comments. ▶ Added assumption for freedom from interference qualification for inter-partition communication.
3.0.24	2021-05-18	Elektrobit Automotive GmbH	RELEASED	▶ Fixed inspection findings. ▶ Set to released: ASCRTECV-2191
3.0.25	2021-07-12	Elektrobit Automotive GmbH	PROPOSED	Added limitations for ExecuteDespiteDataUnavailability and ExecuteTransformersDespiteLocalDataUnqueued.
3.0.26	2021-07-23	Elektrobit Automotive GmbH	RELEASED	Set to released: ASCRTECV-2057.
3.0.27	2021-08-09	Elektrobit Automotive GmbH	PROPOSED	Updated safety requirement for BSW scheduler lifecycle APIs `SchM_Init_<partition>()` and `SchM_Deinit_<partition>()`.

Version	Date	Author	State	Description
3.0.28	2021-08-25	Elektrobit Automotive GmbH	RELEASED	▶ Fixed inspection findings. ▶ Set to released: ASCRTECV-2096.
3.1.0	2021-11-03	Elektrobit Automotive GmbH	PROPOSED	Added support for runsInsideExclusiveAreas to client-server connections
3.1.1	2021-12-06	Elektrobit Automotive GmbH	RELEASED	▶ Fixed inspection findings. ▶ Set to released: ASCRTECV-2335.
3.1.2	2022-02-02	Elektrobit Automotive GmbH	PROPOSED	Added glossary terms and linkage to the glossay terms where necessary.
3.1.3	2022-02-03	Elektrobit Automotive GmbH	PROPOSED	Added assumption for the parameter Rte/RteGeneration/BswConfiguration/BswOsApplicationRef.
3.2.0	2022-04-20	Elektrobit Automotive GmbH	PROPOSED	Added requirement for implicit intra-ECU sender/receiver communication.
3.2.1	2022-04-30	Elektrobit Automotive GmbH	RELEASED	▶ Fixed inspection findings. ▶ Set to released: ASCRTECV-699.
3.2.2	2022-08-19	Elektrobit Automotive GmbH	PROPOSED	▶ Replace the detailed review instruction for `SafetyRTE.Review.Tasks.EventDeclaration` ▶ Added limitation NoInitializeImplicitWriteBufferBeforeRunnableExecutes and updated maintenance changes for ACG-8.8.6 ▶ Updated `SafetyRTE.RS.ImplDataTypes` for optional structure elements. ▶ Removed `SafetyRTE.Limitation.General.NoApplicationDataTypes` limitation. ▶ Added `SafetyRTE.Limitation.ApplicationDataTypes.DataTypeMapping` limitation. ▶ Updated review instructions for `Rte_Components_Shared.h`

Version	Date	Author	State	Description
3.2.3	2022-08-22	Elektrobit Automotive GmbH	RELEASED	Set to released: ASCRTECV-2443.
3.2.4	2022-10-28	Elektrobit Automotive GmbH	PROPOSED	Improve phrasing for buffers definitions in documents.
3.3.0	2022-11-07	Elektrobit Automotive GmbH	PROPOSED	Removed references to Test Generator.
3.4	2022-11-11	Elektrobit Automotive GmbH	PROPOSED	▶ Added new limitations and updated requirement ImplDataTypes. ▶ Added `SafetyRTE.Assumption.ValidatedDataModel`. ▶ Added enabling RTE data model validation. ▶ The document version scheme was changed.
3.5	2022-11-14	Elektrobit Automotive GmbH	RELEASED	▶ Fixed inspection findings. ▶ Set to released: ASCRTECV-2509.

Table 1.1. Document history

2. Document information

2.1. Objective

The objective of this document is to provide you with all the information necessary to ensure that *EB tresos Safety RTE* is used in a safe way in your project.

2.2. Scope and audience

This manual describes the usage of EB tresos Safety RTE in *system applications* which have safety allocations up to ASIL-D. It is valid for all projects and organizations which use EB tresos Safety RTE in a safety-related environment.

The intended audience of this document is:

- Application developers
- Software *integrators*

These persons with the given roles are responsible for performing the verification methods defined in this manual.

They should at least fulfill the following requirements:

- They should have C-programming skills.
- They should be experienced in programming AUTOSAR-compliant ECUs.

Furthermore, advanced knowledge in the following topics is recommended:

- AUTOSAR *RTE*
- AUTOSAR meta model
- Experience with safety standards
- Experience with software development in safety related environments

2.3. Motivation

This manual provides the information on how to correctly use EB tresos Safety RTE in projects for safety-related environments. If EB tresos Safety RTE is used differently, the generated `RTE` code might not comply with

the assumed safety requirements of EB tresos Safety RTE, which are defined in Section 4.2.2, “Assumed safety requirements of EB tresos Safety RTE”. You must take appropriate actions to ensure that your safety requirements are fulfilled.

2.4. Structure

This document contains the following chapters:

Chapter 2, “Document information“
: Provides a brief introduction into this document and its structure.

Chapter 4, “About EB tresos Safety RTE“
: Introduces EB tresos Safety RTE and the environment in which it can be used.

Chapter 5, “Using EB tresos Safety RTE safely“
: Describes how to correctly use EB tresos Safety RTE.

Chapter 6, “Application software constraints and requirements“
: Describes the following characteristics of EB tresos Safety RTE:

- The Safety Element out of Context (SEooC) approach
- Safety requirements
- Assumptions
- Limitations

Chapter 7, “Verification criteria“
: Describes the verification criteria.

3. EB tresos product line support

https://www.elektrobit.com/support

4. About EB tresos Safety RTE

EB tresos Safety RTE is an AUTOSAR RTE implementation which supports the usage of the generated code in ASIL-D environments for a specific subset of features. This chapter introduces EB tresos Safety RTE and describes the environment in which it is used.

4.1. Architecture of the surrounding system

The EB tresos Safety RTE is compatible to AUTOSAR 4.0.3. with selected features of later versions. For more information on AUTOSAR, see [ASRWEB].

Additional assumptions on the software running on the same ECU like EB tresos Safety RTE are defined in Section 6.4.2, “Assumptions”.

4.2. Description of EB tresos Safety RTE

EB tresos Safety RTE consists of three components:

- *EB tresos AutoCore Generic 8 RTE*
- *Rte Code Verifier (RteCV)*
- User documentation including the EB tresos Safety RTE user's guide [USERSGUIDE] and this Safety Manual.

EB tresos AutoCore Generic 8 RTE is the implementation of an AUTOSAR RTE module. The RTE is a code generator used to generate the implementation of an ECU-specific RTE based on the provided configuration and system description.

The RteCV is a tool to verify the RTE code which is generated by EB tresos AutoCore Generic 8 RTE. The RteCV performs static code checks on the generated RTE code. The EB tresos Safety RTE user's guide [USERSGUIDE] describes the usage of the tool.

The user documentation contains review instructions for the generated RTE code. These review instructions cover parts that are not verified by static code checks.

You must perform these reviews for the generated RTE code. You can find the review instructions in Chapter 7, “Verification criteria“.

NOTE **Unsupported features**

EB tresos AutoCore Generic 8 RTE supports many features which are not supported by the RteCV and are also not covered by the review instructions in this document. The safety requirements in Section 4.2.2, “Assumed safety requirements of EB tresos Safety RTE” specify which features are supported for use in safety-related environments. If you use any of these features in the generated `RTE` code, you must verify that the features are correctly implemented and that these do not interfere each other.

The following list shows the basic activities required to safeguard the generated `RTE`:

1. Verify the input of the EB tresos AutoCore Generic 8 RTE generator.
2. Use the RteCV to perform the static code checks on the generated `RTE` code.
3. Perform the reviews defined in Chapter 7, “Verification criteria“.

Step two, running the static code checks for your `RTE`, is described in [USERSGUIDE] in more detail. Additionally. in Section 5.5.3, “Checking the generated RTE code” it is described how to execute the static code checks.

4.2.1. Functional requirements of EB tresos Safety RTE

For more information on the functional requirements of EB tresos Safety RTE, see Chapter 6, “Application software constraints and requirements“.

4.2.2. Assumed safety requirements of EB tresos Safety RTE

For more information on the assumed safety requirements of EB tresos Safety RTE, see Chapter 6, “Application software constraints and requirements“.

4.2.3. Assumptions of EB tresos Safety RTE

For more information on the assumptions on the environment of EB tresos Safety RTE, see Chapter 6, “Application software constraints and requirements“.

4.2.4. Limitations of EB tresos Safety RTE

For more information on the limitations of the feature set of EB tresos Safety RTE, see Chapter 6, “Application software constraints and requirements“.

4.2.5. Safety mechanism used by EB tresos Safety RTE

EB tresos Safety RTE does not use any safety mechanism within the generated code. Instead, the generated EB tresos AutoCore Generic 8 RTE is verified independently to ensure that it behaves as expected.

4.2.6. Robustness of EB tresos Safety RTE

4.2.6.1. Robustness against hardware faults

EB tresos Safety RTE is not robust against hardware faults. It is assumed that the hardware uses fault detection mechanisms such as dual redundant processing, e.g. lock-step mode, error detection, and correction codes (ECC) etc.

4.2.6.2. Robustness against systematic software errors

The generated code of EB tresos Safety RTE is verified in accordance to [ISO26262_1ST] to detect systematic software errors in the generated `RTE` code.

There are no means implemented in EB tresos Safety RTE which make it robust against systematic errors in the calling software.

4.2.6.3. Robustness against configuration errors

EB tresos Safety RTE is sensitive to configuration errors. It is based on the assumption that the provided configuration and system model is correct and therefore only performs basic configuration checks. The integrator is responsible for the correctness of the configuration and the system model.

4.2.6.4. Robustness against resource conflicts

EB tresos Safety RTE uses AUTOSAR OS services and tasks to implement its functionality and to protect critical sections. It relies on a correct OS configuration and the reliable implementation of the corresponding services as well as task handling by the OS.

EB tresos Safety RTE uses the AUTOSAR Com stack to perform *inter-ECU communication*. Therefore you must make sure to use end-to-end protection for safety-relevant inter-ECU communication.

EB tresos Safety RTE does not use other software or hardware resources beyond those mentioned before in the features which are supported for safety related environments. For all other features, the integrator must handle potential resource conflicts.

4.2.6.5. Robustness against interrupt overload

EB tresos Safety RTE does not handle task scheduling and interrupts and is therefore not robust against interrupt overload. The integrator must make sure to configure and build the system so that interrupt overload cannot happen or is handled without jeopardizing the safety relevant parts.

4.2.6.6. Robustness against input errors

EB tresos Safety RTE must be used in accordance to the AUTOSAR specification and the EB tresos Safety RTE user's guide. Input errors caused by incorrect usage of `RTE` API functions must be avoided by verifying the *application software* according to its safety integrity level.

4.2.6.7. Robustness against data starvation

EB tresos Safety RTE does not expect any data input from external sources, so it cannot be starved of data.

4.2.7. What EB tresos Safety RTE does not do

EB tresos Safety RTE does not validate the provided AUTOSAR model. It assumes that the provided model is correct. It is the responsibility of the integrator to ensure this behavior.

EB tresos AutoCore Generic 8 RTE provides functionality that is not verified for the usage in ASIL-D environment. If you want to use any feature that is not supported by EB tresos Safety RTE you must verify it. This includes a verification that the feature is correctly implemented and that this feature does not interfere any other feature.

EB tresos Safety RTE does not verify the linker file of the system application. Instead, the tests are executed with their own linker file. The integrator must ensure that this memory mapping is similar to the mapping in the system application. Otherwise the tests do not cover partitioning related memory mapping issues.

4.2.8. Backward compatibility

EB tresos Safety RTE references a specific EB tresos AutoCore Generic 8 RTE. Compatibility to other EB tresos AutoCore Generic 8 RTE releases is not guaranteed.

5. Using EB tresos Safety RTE safely

EB tresos Safety RTE is developed as a safety element out of context (SEooC). Therefore, Elektrobit Automotive GmbH assumes that the environment meets particular requirements so that the generated `RTE` code behaves appropriately and safely. In addition to that, the feature set which is supported by EB tresos Safety RTE for use in ASIL-D environments is limited to a subset of the AUTOSAR `RTE` features. For more information on these assumptions and limitations, see Chapter 6, “Application software constraints and requirements“.

Each release of EB tresos Safety RTE is developed with and tested against a particular EB tresos AutoCore Generic 8 RTE version, which is specified within the EB tresos Safety RTE release notes [RELNOTES]. This EB tresos AutoCore Generic 8 RTE version is delivered as part of the EB tresos Safety RTE delivery. Do not use EB tresos Safety RTE with a different EB tresos AutoCore Generic 8 RTE version.

For more information on intended usage and possible misuse of EB tresos Safety RTE, see the EB tresos Safety RTE user's guide [USERSGUIDE] chapter *Safe and correct use*.

5.1. Getting started

This section describes the system environment required by EB tresos Safety RTE and how to install EB tresos Safety RTE.

5.1.1. Prerequisites

The RteCV requires the following tools and modules to assure that the validation of the generated `RTE` code can be performed:

1. EB tresos Studio
2. EB tresos® AutoCore Generic. It is recommended to update to the EB tresos® AutoCore Generic version of the ACG8 RTE when you use EB tresos Safety RTE.

The product versions of EB tresos Studio and ACG8 RTE which are supported by this release of EB tresos Safety RTE are documented in the EB tresos Safety RTE release notes [RELNOTES].

NOTE **EB tresos AutoCore Generic 8 RTE**

EB tresos Safety RTE contains a version of EB tresos AutoCore Generic 8 RTE. It is mandatory that you use that version of EB tresos AutoCore Generic 8 RTE. Do not use any other version of the `RTE` with EB tresos Safety RTE.

5.1.2. Installing EB tresos Safety RTE

Before you install EB tresos Safety RTE:

1. Make sure that the correct EB tresos Studio version is already installed.
2. Remove EB tresos AutoCore Generic 8 RTE from the EB tresos Studio installation.

To install EB tresos Safety RTE, follow the steps described in chapter *Installing additional products to EB tresos Studio* in the *EB tresos installation guide* which is part of the file which you have downloaded from EB Command.

NOTE **Install all packages**

You must install all packages from the file which you have downloaded, including EB tresos AutoCore Generic 8 RTE which is part of that delivery.

5.1.3. Verifying the integrity of EB tresos Safety RTE

The EB tresos Safety RTE delivery contains the checksums with SHA-1 hashes of all files that are part of the shipped package. You can find these files in the `$TRESOS_BASE/checksums` folder. This folder contains the following checksum files:

- `RteCV_TS_<version>.sha1`

  This file contains the SHA-1 hashes of all RteCV files.

- `RteCV_TS_<version>.sha1.sha1`

  This file contains the SHA-1 hash of the file `RteCV_TS_<version>.sha1`.

- `Rte_TS_<version>.sha1`

  This file contains the SHA-1 hashes of all RTE files.

- `Rte_TS_<version>.sha1.sha1`

  This file contains the SHA-1 hash of the file `Rte_TS_<version>.sha1`.

The checksums allow you to verify the integrity of the delivered sources and thus to detect if any file became corrupted, e.g. by a failing hard drive or an accidental change.

You can process the SHA-1 checksums automatically with a **validateChecksums.bat** script. To perform the check, follow these steps:

1. Open the `$TRESOS_BASE/checksums` folder.
2. Run the **validateChecksums.bat** script to verify all files with a SHA-1 hash.

Verify the output of the tool and check whether the following entries exist:

- `* checking RteCV_TS_<version>`
- `* checking RteCV_TS_<version>.sha1`
- `* checking Rte_TS_<version>`
- `* checking Rte_TS_<version>.sha1`

If all files are intact, the message `All checksums are valid` is printed.

If any file is corrupt, the message `Checksum validation errors` is printed with a list of affected files.

WARNING **Use EB tresos Safety RTE only with correct files**

Use EB tresos Safety RTE only if all RteCV and RTE files are verified and intact.

In case of corrupted files, reinstall the package that contains EB tresos Safety RTE, and watch out for any warnings that appear during this process. Contact Elektrobit Automotive GmbH support if verification still fails.

5.2. Field Monitoring

Periodically, the EB tresos AutoCore known issues document is automatically created for each delivery under maintenance and is provided to the customer via the EB Command platform. The document provides a list of known problems with the following information:

- Related release version
- List of published bugs with the following information:
  - Unique ID of bug
  - Affected module and version
  - Defect description
  - Workaround (if applicable)

You need credentials for EB Command to access the EB tresos AutoCore known issues.

5.3. EB tresos Safety RTE workflow

To use EB tresos Safety RTE, you have to perform the following general steps:

1. The generation of the RTE for your system application.
2. The validation of the generated RTE.

You can find a detailed overview of the EB tresos Safety RTE workflow in Appendix A, “EB tresos Safety RTE workflow“.

Generation of the RTE

The first step in using EB tresos Safety RTE is the configuration and generation of the RTE for your system application. This step takes place during the system application development and is typically done several times, until the AUTOSAR model and with it the generated RTE for the application software is no longer changing.

For more information on how to configure and generate your RTE, see Section 5.4, “Generating your RTE”.

NOTE

Limited feature set

EB tresos Safety RTE only supports a subset of the features of EB tresos AutoCore Generic 8 RTE for their usage in system applications with a safety allocation. If you use unsupported features, you must define and perform your own validation means for using the generated RTE code in your system application. Therefore, it is recommended that you restrict the used features to features that are supported by EB tresos Safety RTE in order to reduce or prevent additional validation overhead.

- Chapter 6, “Application software constraints and requirements“

  Describes the supported feature set including the limitations imposed by EB tresos Safety RTE.

- Section 7.1, “Limitation checks”

  Describes checks to verify if the generated RTE code contains unsupported features.

Validation of the generated RTE

After the final RTE for your system application was generated, that RTE must be validated. Therefore, this step takes place later in the development.

For more information on how to validate your generated RTE, see Section 5.5, “Validating your generated RTE”.

NOTE

Repeated validation during the development

While the validation of early variants of the generated RTE helps to find potential problems more early in the development, this is expensive and not necessary for each iteration. Keep in mind that the final generated RTE must be validated.

It is strongly recommended to check early if the features used in your generated RTE are supported by EB tresos Safety RTE. Otherwise, additional validation beyond the means provided by EB tresos Safety RTE are probably necessary.

It is also recommended to perform static code checks using the RteCV early to see if your AUTOSAR model uses elements that are not supported or that are not allowed.

5.4. Generating your RTE

For information on how to configure and generate the RTE, see the [AUTOCOREUSRDOC], chapter *User's guide → Run-Time Environment → Configuring and generating the RTE*.

5.5. Validating your generated RTE

After the RTE is generated for your system application it is necessary to validate the generated RTE code. In order to do that, you must perform the following steps:

1. Check that the validation of the *RTE data model* is enabled. For more information, see Section 5.5.1, “Enabling the validation of the RTE data model”.
2. Validate the generated *RTE data model report*. For more information, see Section 5.5.2, “Validating the RTE data model report”.
3. Check your generated RTE code with the static code checker. For more information, see Section 5.5.3, “Checking the generated RTE code”.
4. Review your generated RTE. For more information, see Section 5.5.4, “Reviewing the generated RTE code”.
5. Validate unsupported features of your generated RTE. For more information, see Section 5.5.5, “Validating unsupported features of the RTE”.

5.5.1. Enabling the validation of the RTE data model

The validation of the RTE data model is enabled by default. But the validation can be disabled via a command line parameter or via an option in the option file. The command line parameter has the higher priority. For more information see [USERSGUIDE].

Therefore, you have to make sure that the validation of the RTE data model is enabled.

WARNING **RTE data model validation**

The validation of the RTE data model must be enabled for the final qualification.

You must validate that the RTE data model validation is not disabled.

Validating that the RTE data model validation is not disabled

Step 1
Validate that the option `RteCV.Option.DataModelDisableValidation` is not set in the option file.

Step 2
Validate that the command line parameter `disable-data-model-validation` is not used.

5.5.2. Validating the RTE data model report

The RTE data model report is a representation of the RTE data model. Each time the RteCV is executed it imports the *serialized RTE intermediate model*, transforms it into the RTE data model and generates the RTE data model report out of it. The RTE data model is used by the RteCV to verify the generated RTE code. Hence the RTE data model contains all relevant data.

The serialized RTE intermediate model contains all required information from the configuration and system description of your system application. During the serialization this information is stored in a file. For more information about how to enable the export of the serialized RTE intermediate model, see [USERSGUIDE], chapter *Using the RteCV → Introduction to RteCV → Exporting the serialized RTE intermediate model*.

Since the data of the RTE data model is provided from the RTE you have to make sure that nothing went wrong during the import of the configuration and the system description into EB tresos Studio and the export of the data. This means you have to ensure that e.g. the mapping of software components to partitions is correct or all artifacts have the correct values.

WARNING **Invalid validation results**

Each deviation between the RTE data model report and your configuration and system description of your system application will lead to invalid validation results. Consequently it is not safe to use your system application in an ASIL-D environment.

You must validate the RTE data model report before you continue with the validation of your generated RTE.

Validation of the `ImplementationDataType` artifacts in the RTE data model report:

Step 1
Validate that the `AtomicAccess` attribute is correct according to the platform specification.

Step 2
Validate that the `RteEmitter` and `TypeReference` attributes correspond to what was originally defined in the AUTOSAR data model that was imported into EB tresos Studio.

Step 3
Validate for all `ImplementationDataType` artifacts where the attributes `RteEmitter` and `TypeReference` are set to `true` that all other attributes correspond to what was originally defined in the AUTOSAR data model that was imported into EB tresos Studio.

Validation of the RTE data model report:

Step 1
Validate that the RTE data model report contains all expected artifacts. For example, all partitions, software components, ports, etc. are found within the RTE data model report.

Step 2
Validate that the artifacts contain all expected artifacts. For example, the partitions contain the correct software components, the partitions contain the correct tasks, etc.

Step 3
Validate that the attributes of these artifacts corresponds to what was originally defined in the AUTOSAR data model that was imported into EB tresos Studio.

Step 4
Validate that when changes are made to the EB tresos Studio data model, the relevant changes appear in the RTE data model report.

Step 5
Validate that the RTE data model report contents are consistent with each other.

5.5.3. Checking the generated RTE code

To validate your RTE with the static code checks, you must perform the following steps which are described in the next sections:

1. Run the pattern based code checks on the generated RTE code. For more information, see Section 5.5.3.1.1, “Executing pattern based code checks”.
2. Run the abstract syntax tree based code checks on the generated RTE code. For more information, see Section 5.5.3.1.2, “Executing abstract syntax tree based code checks”.
3. Evaluate the test results. For more information, see Section 5.5.3.2, “Evaluating the results of the static code checks”.

NOTE

Use the generated RTE code from your system application

Keep in mind that the static code checker shall verify the RTE code which is later used in your system application. Thus do not change anything in the RTE configuration and do not generate a new RTE for the static code checks. Just reuse the generated RTE code from your system application.

5.5.3.1. Executing static code checks

5.5.3.1.1. Executing pattern based code checks

After the generation of the RTE, the RteCV shall perform pattern based static code checks on the generated RTE code. This is done by executing the RteCVExt.CodeChecker_TS_TxDxM5I5R0 plugin. To perform the pattern based code analysis on your generated RTE code, follow the instructions in the [USERSGUIDE], chapter *Using the RteCV → Performing RTE static code checks with the RteCV*.

5.5.3.1.2. Executing abstract syntax tree based code checks

After the generation of the RTE, the RteCV shall perform abstract syntax tree based static code checks on the generated RTE code. This is done by executing the RteCVExt.Introspection_TS_TxDxM5I5R0 plugin. To perform the abstract syntax tree based code analysis on your generated RTE code, follow the instructions in the [USERSGUIDE], chapter *Using the RteCV → Performing RTE static code checks with the RteCV*.

5.5.3.2. Evaluating the results of the static code checks

You can evaluate the generated RteCV verification reports after all static code checks are executed.

NOTE

To evaluate the reports, follow the instructions in the [USERSGUIDE], chapter *Using the RteCV → Performing `RTE` code checks with the RteCV →* Evaluating the results of the static code analysis

5.5.4. Reviewing the generated RTE code

In addition to the generated tests from the [USERSGUIDE], chapter *Using the RteCV → Generating the RTE tests with the RteCV*, EB tresos Safety RTE provides review specifications to validate the generated `RTE` code. This section provides the information on how to perform reviews to validate the generated `RTE` code.

Input

The input for the code reviews consists of the following parts:

- The AUTOSAR model that describes the ECU.
- The generated `RTE` code
- The review instructions

The AUTOSAR model used as input for the reviews, must be the same one, which was used to generate the `RTE`.

The generated `RTE` code must be the `RTE` code, which runs on the ECU.

The review instructions are defined in the Section 7.2, “Reviews”.

Prerequisites

Read the review instructions carefully. It is of paramount importance that you understand what must be done for a review and when the reviewed code passes or fails that review. Otherwise, it is likely that you do not perform the reviews correctly, which might lead to erroneous code in your system application that causes problems.

Performing the reviews

In order to validate the generated `RTE` code for usage in a system application with a safety allocation, you must perform *all* specified reviews as they go hand in hand with the generated tests. Document the review results in a format that is appropriate for your project.

TIP

Document which code was reviewed

In addition to the review results, it is recommended that you document which code part was reviewed as well. This helps in identifying unsupported features which are necessary to identify further validation efforts.

For detailed instructions on what to review, see Section 7.2, “Reviews”.

5.5.5. Validating unsupported features of the RTE

EB tresos Safety RTE does not support all features of EB tresos AutoCore Generic 8 RTE for their usage in system applications with a safety allocation. For more information, see Section 4.2, “Description of EB tresos Safety RTE”. This means, that based on your AUTOSAR model EB tresos AutoCore Generic 8 RTE can generate code that is not validated by EB tresos Safety RTE. In this case you must validate such generated `RTE` code to prevent the violation of any safety requirements you have.

Identification of unsupported features

There are two ways to identify unsupported features:

1. Limitation checks, see Section 7.1, “Limitation checks”
2. Test coverage measurement and review coverage analysis

The *limitation checks* are the first means to identify unsupported features. Due to them being reviews, they can be used early during the development when the `RTE` as well as the system application is not yet final. In this way the usage of unsupported features can be reduced and for those unsupported features, which are necessary, validation methods can be defined and planned early.

Test coverage measurement and review coverage analysis must be performed in later stages, when the generated `RTE` tests are executed (test coverage measurement) and the generated `RTE` code is reviewed (compare Section 5.5.4, “Reviewing the generated RTE code”). You must validate each function and macro which is not covered by either tests or reviews.

Some thoughts on test coverage measurement and review coverage analysis:

- Test coverage measurement is not a 100% secure way to identify unsupported features. For example if you use the unsupported `Rte_IsUpdated()` API for sender/receiver communication, test coverage tells that handling the flag in the `Rte_Read_<p>_<o>()/Rte_Write_<p>_<o>()` API is already tested. That is however not true, as `Rte_IsUpdated()` was never called to check for the correct value of the update flag after such an API call. Therefore you also must validate the corresponding functionality in the `Rte_Read_<p>_<o>()/Rte_Write_<p>_<o>()` APIs.

- The test coverage measurement and review coverage analysis do not only deliver unsupported features. They also identify gaps in the generated tests and reviews. You must validate such gaps in the same way as you must do for any unsupported features.

TIP

RteCV log

The RteCV log which is generated during the test generation potentially contains warnings about unsupported and ignored features. This log is therefore also a good place to check when you look for what must be validated beyond what is covered by EB tresos Safety RTE. It might especially give hints on why some functionality was not tested.

Validation criteria

You must ensure to check the following validation criteria:

- Freedom from interference
- Correctness

Freedom from interference means that you must validate, that the validated functionality does not interfere with:

1. Other `RTE` functions or macros that do not belong to that feature.
2. Other `RTE` functions or macros that belong to that feature in an unexpected or detrimental way.
3. Other functionality from the system application in a detrimental or unexpected way.

Examples might be that the validated functionality may not overwrite or change variables used by other functionality or that it may not interfere with scheduling (in each case depending on the unsupported functionality).

In addition to that you must validate such functionality for *correctness* if there are safety requirements on that functionality.

Validation means

There are no specific restrictions on the means to be used for validating unsupported features other than they must be appropriate to validate the functionality according to the safety level allocated to your system application.

NOTE **Validation mechanism selection**

It is your responsibility to choose and apply appropriate validation mechanisms.

5.5.6. Interpretation of the validation results

If all static code checks and manual reviews were successful and also no issues were found regarding the validation of the unsupported features and the RTE data model report, the generated and validated `RTE` code may be used in an system application with a safety allocation.

The following section provides information on how to interpret the validation results if the validation failed. In this case, the exact interpretation depends on which validation mechanism was used.

Static code checks

Static code checks can fail due to several reasons:

- The generated `RTE` code is erroneous.
- The generated `RTE` code is correct, but an API function or a variant of the API function is not supported by the EB tresos Safety RTE.

WARNING **Do not use an RTE API which failed a code check**

If there are any code checks which failed for the generated `RTE` code, that code may not be used in any system application with a safety allocation.

The only exception to this is if you analyze the reason for the failed code check and make sure through appropriate means that the affected functionality cannot lead to the violation of any of your safety requirements. The pattern generated by a code checker for the RTE API can be compared to the generated RTE code to determine if missing or additional functionality is present within the generated RTE code.

Limitation checks

While the limitation checks are not a validation mechanism, it is important to understand what a failing check means and how you must react to it.

If a limitation check fails, this means that you are using a feature which is not supported by EB tresos Safety RTE for its usage in system applications with a safety allocation. In such a case, there are no mechanisms in EB

tresos Safety RTE to validate that this feature works as expected and does not interfere with other functionality in an unintended way. Therefore, it is your responsibility to validate unsupported features.

For more information on how to validate unsupported features, see Section 5.5.5, “Validating unsupported features of the RTE” .

Reviews

Reviews can fail due to several reasons. Therefore there are several possible outcomes during their evaluation:

- There are reviews which fail if your generated `RTE` code uses features, which are not supported by EB tresos Safety RTE. In this case you must take appropriate measures to validate the unsupported feature as well as the review object, for which the review failed. For more information, see Section 5.5.5, “Validating unsupported features of the RTE”.
- Reviews can succeed or fail because the review description is misleading or erroneous or was not correctly understood independent on if the generated `RTE` code is correct or not. If you are in doubt about a review or its result, assume that the review failed and handle as below.
- Reviews fail if the generated `RTE` code is erroneous.

WARNING **Do not use an RTE which failed a review**

If there are any reviews which failed for the generated `RTE` code, that code may not be used in any system application with a safety allocation.

The only exception to this is if you analyze the reason for the failed review and make sure through appropriate means that the affected functionality cannot lead to the violation of any of your safety requirements.

Validation of unsupported features

WARNING **Do not use an RTE with not validated unsupported features**

If there are any unsupported features which could not be validated according to Section 5.5.5, “Validating unsupported features of the RTE”, the generated `RTE` code may not be used in any system application with a safety allocation.

6. Application software constraints and requirements

This chapter describes the following characteristics of the EB tresos Safety RTE:

- The definition of EB tresos Safety RTE as Safety Element out of Context (SEooC)
- The definition of the assumed safety requirements
- The assumptions made by EB tresos Safety RTE on the environment
- The limitations of the feature set

6.1. Safety element out of context (SEooC) definition

EB tresos Safety RTE is developed as a Safety Element out of Context (SEooC) according to [ISO26262-10_1ST].

6.1.1. Functional scope of the SEooC

6.1.1.1. Provided functionality

The functional scope of the SEooC is defined by the requirements as specified in Section 6.3, “Safety requirements”.

WARNING	**Assumed safety requirements**
	The requirements listed in Section 6.3, “Safety requirements” are not sufficient to fulfill all safety requirements imposed on the software architecture of the entire system. Depending on the system application, additional safety mechanisms may be necessary to guarantee system safety.

6.1.1.2. Assumed employment

EB tresos Safety RTE assumes that you employ all of its parts according to the specification given in the safety manual.

6.1.1.3. Assumed external functionality

EB tresos Safety RTE assumes that the following safety mechanisms are used on the ECU to ensure the integrity of safety related functionality:

- End-to-end protection for inter-ECU communication, e.g. via EB tresos Safety E2E Protection
- Memory protection for spatial freedom from interference, e.g. provided by EB tresos Safety OS
- Control flow monitoring, e.g. provided by EB tresos Safety TimE Protection

Additionally EB tresos Safety RTE requires trusted critical section protection APIs, e.g. provided by EB tresos Safety OS

6.1.2. Implementation scope of the SEooC

The safety-related part of EB tresos Safety RTE is the implementation of the supported features in the generated `RTE` code. The supported features are defined by the safety requirements.

The generated `RTE` code which is not directly related to those features is not part of the safety-related part of EB tresos Safety RTE.

NOTE

Mixed safety-related and non-safety-related code

Each file in the generated `RTE` code can contain safety-related as well as non-safety-related code. There is no obvious separation between these two.

6.2. Definitions

This section defines the processes and terms used in Section 6.3, “Safety requirements”.

6.2.1. Process words

This section defines all *processes* according to the scheme for defining process words specified in the [RMP], in chapter *Specification of requirements*.

Process	Definition
to create	In *EB tresos Safety RTE* *to create* is defined as implementing something in the EB tresos Safety RTE source code.
to provide	In EB tresos Safety RTE *to provide* is defined as making something available.

to support	In EB tresos Safety RTE *to support* is defined as allowing the use or operation of something.

Table 6.1. Definition of processes

6.2.2. Term definitions

This section defines all *terms* according to the scheme for defining terms specified in the [RMP], in chapter *Specification of requirements*.

Term	Definition
1:1 communication	In EB tresos Safety RTE a *1:1 communication* is defined as the limitation of sender/receiver communication to one sending and one receiving entity.
API data type	In EB tresos Safety RTE an *API data type* is defined as a standardized type or preprocessor macro specified by AUTOSAR for the *RTE* API.
application error	In EB tresos Safety RTE an *application error* is defined as a value, which represents an error code within an specified range.
basic software module	In EB tresos Safety RTE a *basic software module* is defined as a collection of software files (code and description) specified by AUTOSAR that define a certain basic software functionality present on an *ECU*.
basic software scheduler	In EB tresos Safety RTE a *basic software scheduler* is defined as a component which is part of the *RTE*. It is responsible for the management of basic software modules and the triggering of *BSW events*.
call	In EB tresos Safety RTE a *call* is defined as the invocation of a function.
client	In EB tresos Safety RTE a *client* is defined as a software entity which uses services of a server for client-server communication. Note: This definition is derived from [ASRGLOSSARY]
client-server communication	In EB tresos Safety RTE a *client-server communication* is defined as a specific form of communication, in a possibly distributed system. In this system, software entities act as clients, servers or both. It is possible to connect 1...n clients to a server. The clients request services via a specific protocol from one server. Note: This definition is derived from [ASRGLOSSARY]
COM-callback	In EB tresos Safety RTE a *COM-callback* is defined as an *RTE* function, which is called by the COM module to receive data for inter-ECU sender/receiver communication.
compatible interface	In EB tresos Safety RTE *compatible interfaces* are defined as interfaces which are compatible according to the AUTOSAR standard without using port interface mappings.

complex data type	In EB tresos Safety RTE a *complex data type* is defined as a type implemented by one of the following types: ▶ `array` ▶ `struct` ▶ `union`
core	In EB tresos Safety RTE a *core* is defined as a single processing unit of a microcontroller, which contains registers, arithmetic/logic unit and other processing hardware.
data consistency	In EB tresos Safety RTE the *data consistency* is defined as the atomic access to a variable.
direct API	In EB tresos Safety RTE a *direct API* is defined as the characteristic of an AUTOSAR *RTE* API that is supposed to be used via a direct function call without any indirection, i.e. no usage of function pointers.
direct function call	In EB tresos Safety RTE a *direct function call* is defined as the invocation of the server via a direct function call, i.e. the `OperationInvokedEvent`, which triggers the server which is not mapped to a task.
exclusive area	In EB tresos Safety RTE an *exclusive area* is defined as the concept for blocking concurrent accesses to ensure data consistency.
executable entity	In EB tresos Safety RTE an *executable entity* is defined as an abstraction for executable code of e.g. an *runnable entity*. The code is executed in the context of a task.
explicit behavior	In EB tresos Safety RTE an *explicit behavior* is defined as data reception and transmission that is triggered immediately from an API during the execution of a runnable. Explicit sender/receiver communication uses a receive buffer to store data for a sender/receiver connection.
implementation data type	In EB tresos Safety RTE an *implementation data type* (primitive data type or complex data type) is defined as the definition of a C-type for an *AUTOSAR model object* `ImplementationDataType`.
implicit behavior	In EB tresos Safety RTE an *implicit behavior* is defined as a data reception or data transmission that is not immediately triggered from an API. Here the tasks perform the operation at the beginning (reception) or the end (transmission) of its execution. Implicit sender/receiver communication uses a receive buffer and runnable specific data handle buffers to store data for a sender/receiver connection.

initial value	In EB tresos Safety RTE an *initial value* is defined as the value which is provided to the receiver of a sender/receiver communication, before data was sent yet by the sender.
inter-core communication	In EB tresos Safety RTE *inter-core communication* is defined as the communication between software components that are contained in different partitions which are mapped on different cores by using mechanisms provided by the *RTE*.
inter-ECU communication	In EB tresos Safety RTE *inter-ECU communication* is defined as the communication between *ECUs* using mechanisms provided by the *RTE* and the Com.
inter-partition communication	In EB tresos Safety RTE *inter-partition communication* is defined as the communication between software components that are contained in different partitions on the same *ECU* by using mechanisms provided by the *RTE*.
inter-runnable variable	In EB tresos Safety RTE an *inter-runnable variable* is defined as a variable which is used for exchanging data between *runnable entities* which are contained in the same software component.
interrupt blocking	In EB tresos Safety RTE *interrupt blocking* is defined as an implementation of a data consistency mechanism by disabling/suspending interrupts.
intra-partition communication	In EB tresos Safety RTE *intra-partition communication* is defined as the communication between software components that are contained in the same partition by using mechanisms provided by the *RTE*.
lifecycle	In EB tresos Safety RTE, a *lifecycle* is defined as the series of phases that EB tresos Safety RTE passes from startup until shutdown.
multi-core	In EB tresos Safety RTE a *multi-core* system is defined as a system which has two or more cores that can execute multiple instructions at the same time.
non-concurrent execution	In EB tresos Safety RTE a *non-concurrent execution* is defined as the serialized execution of an executable entity.
operation-invoked event	In EB tresos Safety RTE an *operation-invoked event* is defined as an *AUTOSAR model object* used to activate the server in a client-server communication.
optional structure elements	In EB tresos Safety RTE an *optional structure element* is defined as an optional member of a structure implementation data type. The availability of the optional elements can be set and queried at run-time using macros generated by the RTE.
partition	In EB tresos Safety RTE a *partition* is defined as a term to represent an *AUTOSAR OS application*.
partitioning	In EB tresos Safety RTE the *partitioning* is defined as a mechanism to separate the software of an *ECU* to different *partitions*. It is used to protect the software in different *partitions* from each other via memory protection.
port	In EB tresos Safety RTE a *port* is defined as an interaction point between software components.

port-defined argument	In EB tresos Safety RTE a *port-defined argument* is defined as an argument included in the signature of a server call.
primitive data type	In EB tresos Safety RTE a *primitive data type* is defined as a type implemented by one of the following types: ▶ `boolean` ▶ `integer` ▶ `floating point` ▶ `opaque type` Note: This definition is derived from [ASRRTE422]
read by argument	In EB tresos Safety RTE *read by argument* is defined as sender/receiver communication where the current value is written to the function argument.
runs inside exclusive area	In EB tresos Safety RTE a *runs inside exclusive area* is defined as the execution of an executable entity inside an exclusive area.
sender/receiver communication	In EB tresos Safety RTE a *sender/receiver communication* is defined as a communication mechanism which offers asynchronous distribution of variable data where a sender transmits data to one or more receivers, or a receiver receives data from one or more senders. Note: This definition is derived from [ASRGLOSSARY]
sequential scheduling	In EB tresos Safety RTE a *sequential scheduling* is defined as the sequentially execution of *runnable entities* in a task.
serialization	In EB tresos Safety RTE a *serialization* is defined as the process to translate data into a series of bits with a specified format. This format can be used to translate the series of bits into data that is semantically identical to the original data. In AUTOSAR the *transformers* serialize data for the RTE.
server	In EB tresos Safety RTE a *server* is defined as a software entity which provides services to clients for client-server communication. Note: This definition is derived from [ASRGLOSSARY]
shared memory communicator	In EB tresos Safety RTE a *shared memory communicator* is defined as a communication mechanism used by the *RTE* to perform inter-partition communication via the use of partition-specific and shared memory sections.
single-core	In EB tresos Safety RTE a *single-core* system is defined as a system which has a single core that can execute one instruction at a time.
single instantiation	In EB tresos Safety RTE a *single instantiation* is defined as an AUTOSAR feature for instantiating the same software component at one time on the same *ECU*.

software component	In EB tresos Safety RTE a *software component* is defined as an AUTOSAR software component, which is an architectural element, that provides and/or requires interfaces and is connected to other *software components* to fulfill architectural responsibilities. Note: This definition is derived from [ASRGLOSSARY]
spinlock	In EB tresos Safety RTE a *spinlock* is defined as an implementation of a data consistency mechanism by entering a loop until a shared OS resource can be acquired. The spinlock mechanism is only used in multi-core systems.
synchronous communication	In EB tresos Safety RTE a *synchronous communication* is defined as communication where the calling software entity is blocked until the called operation is evaluated. The calling software entity continues its operation once its result is available. Note: This definition is derived from [ASRGLOSSARY]
task blocking	In EB tresos Safety RTE *task blocking* is defined as an implementation of a data consistency mechanism by prohibiting the mutual task preemption.
timing event	In EB tresos Safety RTE a *timing event* is defined as an *AUTOSAR model object* used for periodic activation of *executable entities*.
trace hook function	In EB tresos Safety RTE a *trace hook function* is defined as a function which is used by the *RTE* to trace e.g. the activation of *runnable entities*.
unqueued	In EB tresos Safety RTE the word *unqueued* is defined as the characteristic of sender/receiver communication with last-is-best semantics.

Table 6.2. Definition of terms

6.3. Safety requirements

This section defines the assumed safety requirements of EB tresos Safety RTE.

Id:	SafetyRTE.RS.SingleCore
Doctype:	`requirement`
Status:	APPROVED
Source:	Elektrobit Automotive GmbH
Version:	2
Description:	EB tresos Safety RTE shall support the execution on a single-core system.
Needs coverage of:	`requirement`
Safety class:	ASIL-D

Id:	SafetyRTE.RS.MultiCore
Doctype:	`requirement`
Status:	APPROVED
Source:	Elektrobit Automotive GmbH
Version:	1
Description:	EB tresos Safety RTE shall support the execution on a multi-core system.
Needs coverage of:	`requirement`, `rte_assumption`
Safety class:	ASIL-D

Id:	SafetyRTE.RS.Partitioning
Doctype:	`requirement`
Status:	APPROVED
Source:	Elektrobit Automotive GmbH
Version:	2
Description:	EB tresos Safety RTE shall support partitioning.
Comment:	This means that it is supported to map software components to different partitions.
Needs coverage of:	`rte_assumption`
Safety class:	ASIL-D

Id:	SafetyRTE.RS.SMC
Doctype:	`requirement`
Status:	APPROVED
Source:	Elektrobit Automotive GmbH
Version:	2
Description:	EB tresos Safety RTE shall provide inter-partition and inter-core communication by using the shared memory communicator (SMC).
Needs coverage of:	`rte_assumption`, `hazop`
Provides coverage to:	SafetyRTE.RS.SingleCore, Version: 2 SafetyRTE.RS.MultiCore, Version: 1
Safety class:	ASIL-D

Id:	SafetyRTE.RS.ImplDataTypes
Doctype:	`requirement`
Status:	APPROVED

Source:	Elektrobit Automotive GmbH
Version:	4
Description:	EB tresos Safety RTE shall support the following implementation data types, that have the parameter `TYPE-EMITTER` set to `RTE`: ► Primitive implementation data type ► Structure implementation data type ► with and without optional structure elements ► Array implementation data type ► Redefinition implementation data type ► Pointer implementation data type
Comment:	The implementation data types are specified in [ASRRTE422], chapter *5.3.4.2*.
Needs coverage of:	`rte_assumption`, `hazop`
Safety class:	ASIL-D

Id:	SafetyRTE.RS.ApiDataTypes
Doctype:	`requirement`
Status:	APPROVED
Source:	Elektrobit Automotive GmbH
Version:	2
Description:	EB tresos Safety RTE shall support the API data types which are specified by the *RTE*.
Comment:	The API data types are specified in [ASRRTE422], chapters *5.5* and *6.4*.
Needs coverage of:	`rte_assumption`
Safety class:	ASIL-D

Id:	SafetyRTE.RS.DirectAPIs
Doctype:	`requirement`
Status:	APPROVED
Source:	Elektrobit Automotive GmbH
Version:	2
Description:	EB tresos Safety RTE shall support the direct API.
Needs coverage of:	`rte_assumption`, `hazop`
Safety class:	ASIL-D

Id:	SafetyRTE.RS.SwcInstantiation
Doctype:	`requirement`
Status:	APPROVED
Source:	Elektrobit Automotive GmbH
Version:	2
Description:	EB tresos Safety RTE shall support single instantiation of software components.
Needs coverage of:	`rte_assumption`, `hazop`
Provides coverage to:	SafetyRTE.RS.SingleCore, Version: 2 SafetyRTE.RS.MultiCore, Version: 1
Safety class:	ASIL-D

Id:	SafetyRTE.RS.CompatiblePortInterfaces
Doctype:	`requirement`
Status:	APPROVED
Source:	Elektrobit Automotive GmbH
Version:	1
Description:	EB tresos Safety RTE shall support connections of ports typed by compatible interfaces.
Needs coverage of:	`rte_assumption`
Safety class:	ASIL-D

Id:	SafetyRTE.RS.DataConsistency
Doctype:	`requirement`
Status:	APPROVED
Source:	Elektrobit Automotive GmbH
Version:	2
Description:	EB tresos Safety RTE shall provide the following data consistency mechanisms: ► Sequential scheduling ► Interrupt blocking ► Task blocking ► Spinlock
Comment:	The mechanisms to guarantee data consistency are described in [ASRRTE422], chapter *4.2.5.4* and spinlocks are described in [ASROS403], chapter *7.9.29*.
Needs coverage of:	`rte_assumption`, `hazop`
Provides coverage to:	SafetyRTE.RS.SingleCore, Version: 2 SafetyRTE.RS.MultiCore, Version: 1
Safety class:	ASIL-D

Id:	SafetyRTE.RS.InterruptBlocking
Doctype:	`requirement`
Status:	APPROVED
Source:	Elektrobit Automotive GmbH
Version:	1
Description:	EB tresos Safety RTE shall provide the following interrupt blocking APIs: ► `{Suspend|Resume}AllInterrupts()` ► `Rte_UserDefinedInt{Lock|Unlock}()`
Needs coverage of:	`rte_assumption`, `hazop`
Provides coverage to:	SafetyRTE.RS.SingleCore, Version: 2 SafetyRTE.RS.MultiCore, Version: 1
Safety class:	ASIL-D

Id:	SafetyRTE.RS.SpinLock
Doctype:	`requirement`
Status:	APPROVED
Source:	Elektrobit Automotive GmbH
Version:	1
Description:	EB tresos Safety RTE shall provide the following spinlock APIs: ▶ `{Get|Release}Spinlock()`
Needs coverage of:	`rte_assumption`, `hazop`
Provides coverage to:	SafetyRTE.RS.MultiCore, Version: 1
Safety class:	ASIL-D

Id:	SafetyRTE.RS.TraceHooks
Doctype:	`requirement`
Status:	APPROVED
Source:	Elektrobit Automotive GmbH
Version:	4
Description:	EB tresos Safety RTE shall provide the following trace hook functions: ▶ `Rte_Task_Dispatch()` ▶ `Rte_Task_EndHook()` ▶ `Rte_Task_Terminate()`
Needs coverage of:	`rte_assumption`
Safety class:	ASIL-D

Id:	SafetyRTE.RS.LifecycleAPI
Doctype:	`requirement`
Status:	APPROVED
Source:	Elektrobit Automotive GmbH
Version:	3
Description:	EB tresos Safety RTE shall provide the following lifecycle APIs for each partition: ▶ `Rte_Start_<partition>()` ▶ `Rte_Stop_<partition>()` ▶ `Rte_PartitionTerminated_<partition>()`
Comment:	`Rte_Start()` and `Rte_Stop()` exist, but must not be used. Instead there are partition-specific versions of these API functions for each partition.
Needs coverage of:	`rte_assumption`, `hazop`
Provides coverage to:	SafetyRTE.RS.SingleCore, Version: 2 SafetyRTE.RS.MultiCore, Version: 1
Safety class:	ASIL-D

Id:	SafetyRTE.RS.BswSchedulerLifeCycleAPI
Doctype:	`requirement`
Status:	APPROVED
Source:	Elektrobit Automotive GmbH
Version:	3
Description:	EB tresos Safety RTE shall provide the following basic software scheduler lifecycle APIs for each partition with at least one mapped basic software module: ▶ `SchM_Init_<partition>()` ▶ `SchM_Deinit_<partition>()`
Comment:	`SchM_Init()` and `SchM_Deinit()` exist, but shall be generated as empty stubs. Instead there are partition-specific versions of these API functions for each *BSW partition*.
Needs coverage of:	`rte_assumption`, `hazop`
Provides coverage to:	SafetyRTE.RS.SingleCore, Version: 2 SafetyRTE.RS.MultiCore, Version: 1
Safety class:	ASIL-D

Id:	SafetyRTE.RS.TimingEvents
Doctype:	`requirement`
Status:	APPROVED
Source:	Elektrobit Automotive GmbH
Version:	2
Description:	EB tresos Safety RTE shall create calls for the configured timing events in the generated OS tasks.
Comment:	This requirement means that the OS tasks are generated according to the *RTE event* to task mappings, the event offset, and the task period.
Needs coverage of:	`rte_assumption`, `hazop`
Safety class:	ASIL-D

Id:	SafetyRTE.RS.OperationInvokedEvents
Doctype:	`requirement`
Status:	APPROVED
Source:	Elektrobit Automotive GmbH
Version:	1
Description:	EB tresos Safety RTE shall create calls for the configured operation-invoked events, which are not explicitly mapped to a task for synchronous client-server communication.
Needs coverage of:	`rte_assumption`, `hazop`
Safety class:	ASIL-D

Id:	SafetyRTE.RS.SwcExclusiveAreas
Doctype:	`requirement`
Status:	APPROVED
Source:	Elektrobit Automotive GmbH
Version:	3
Description:	EB tresos Safety RTE shall provide exclusive areas for the software components, i.e. via `Rte_Enter()`/`Rte_Exit()` APIs. The supported exclusive area mechanisms are: ► Interrupt blocking (`ALL_INTERRUPT_BLOCKING`) ► OS interrupt blocking (`OS_INTERRUPT_BLOCKING`) ► OS resources (`OS_RESOURCE`) ► No lock (`NO_LOCK`), i.e. no critical section

Comment:	In this case the *RTE* guarantees the data consistency for a part of the executable code of the software component.
Needs coverage of:	`rte_assumption`, `hazop`
Provides coverage to:	SafetyRTE.RS.SingleCore, Version: 2 SafetyRTE.RS.MultiCore, Version: 1
Safety class:	ASIL-D

Id:	SafetyRTE.RS.BswExclusiveAreas
Doctype:	`requirement`
Status:	APPROVED
Source:	Elektrobit Automotive GmbH
Version:	5
Description:	EB tresos Safety RTE shall provide exclusive areas for the basic software modules, i.e. via `SchM_Enter()`/`SchM_Exit()` APIs. The supported exclusive area mechanisms are: ► Interrupt blocking (`ALL_INTERRUPT_BLOCKING`) ► OS interrupt blocking (`OS_INTERRUPT_BLOCKING`) ► OS spinlock (`OS_SPINLOCK`) ► No lock (`NO_LOCK`), i.e. no critical section
Comment:	In this case the *RTE* guarantees the data consistency for a part of the executable code of the basic software module. The exclusive area mechanism `OS_RESOURCE` is not supported because it is prone to errors and the RTE cannot guarantee correct protection as it typically does not know, to which tasks and interrupt service routines, the resource must be allocated.
Needs coverage of:	`rte_assumption`, `hazop`
Provides coverage to:	SafetyRTE.RS.SingleCore, Version: 2 SafetyRTE.RS.MultiCore, Version: 1
Safety class:	ASIL-D

Id:	SafetyRTE.RS.ImplicitExclusiveAreas
Doctype:	`requirement`
Status:	APPROVED
Source:	Elektrobit Automotive GmbH
Version:	2
Description:	EB tresos Safety RTE shall provide exclusive areas implicitly used to protect the execution of runnable entities that are triggered via: ► timing events ► operation invoked events This applies for exclusive areas in the runs inside exclusive area role.
Comment:	In this case the *RTE* guarantees the data consistency for the complete executable code of the software component or basic software module.
Needs coverage of:	`rte_assumption`, `hazop`
Provides coverage to:	SafetyRTE.RS.SingleCore, Version: 2 SafetyRTE.RS.MultiCore, Version: 1
Safety class:	ASIL-D

Id:	SafetyRTE.RS.SenderReceiverIntraEcu
Doctype:	`requirement`
Status:	APPROVED
Source:	Elektrobit Automotive GmbH
Version:	3
Description:	EB tresos Safety RTE shall provide intra-ECU sender/receiver communication with the following properties: ► Unqueued data semantics ► Explicit read and write access ► 1:1 communication ► Initial values ► Read by argument, i.e. `Rte_Read()` API ► Intra-partition communication ► Inter-partition communication ► Inter-core communication
Comment:	Note that only those features are supported for intra-ECU sender/receiver communication that are explicitly mentioned.

Needs coverage of:	`rte_assumption`, `hazop`
Provides coverage to:	SafetyRTE.RS.SingleCore, Version: 2 SafetyRTE.RS.MultiCore, Version: 1
Safety class:	ASIL-D

Id:	SafetyRTE.RS.SenderReceiverImplicitIntraEcu
Doctype:	`requirement`
Status:	APPROVED
Source:	Elektrobit Automotive GmbH
Version:	2
Description:	EB tresos Safety RTE shall provide implicit intra-ECU sender/receiver communication with the following properties: ► Unqueued data semantics ► Implicit read and write access ► 1:1 communication ► Initial values ► Intra-partition communication ► Inter-partition communication ► Inter-core communication
Comment:	Note that only those features are supported for implicit intra-ECU sender/receiver communication which are explicitly mentioned.
Needs coverage of:	`rte_assumption`, `hazop`
Provides coverage to:	SafetyRTE.RS.SingleCore, Version: 2 SafetyRTE.RS.MultiCore, Version: 1
Safety class:	ASIL-D

Id:	SafetyRTE.RS.SenderReceiverInterEcu
Doctype:	`requirement`
Status:	APPROVED
Source:	Elektrobit Automotive GmbH
Version:	3
Description:	EB tresos Safety RTE shall provide inter-ECU sender/receiver communication with the following properties: ► Unqueued data semantics ► Explicit read and write access

	▶ 1:1 communication, this means that one sender or one receiver is connected via one COM-signal or signal group
	▶ Read by argument, i.e. `Rte_Read()` API
	▶ Sender or receiver can be mapped to:
	  ▶ the same partition as the COM module (intra-partition communication)
	  ▶ a different partition on the same core as the COM module (inter-partition communication)
	  ▶ a partition on a different core as the COM module (inter-core communication)
	▶ Data reception via interruptible COM-callbacks
	▶ Non-periodic execution of the BSW OS task
	▶ Support of data transformation

Comment:	Only those features are supported for inter-ECU sender/receiver communication which are explicitly mentioned.
Needs coverage of:	`rte_assumption`, `hazop`
Provides coverage to:	SafetyRTE.RS.SingleCore, Version: 2 SafetyRTE.RS.MultiCore, Version: 1
Safety class:	ASIL-D

Id:	SafetyRTE.RS.InterRunnableVariables
Doctype:	`requirement`
Status:	APPROVED
Source:	Elektrobit Automotive GmbH
Version:	2
Description:	EB tresos Safety RTE shall provide inter-runnable variables with the following properties: ▶ Explicit read and write access ▶ sequential scheduling
Needs coverage of:	`rte_assumption`, `hazop`
Provides coverage to:	SafetyRTE.RS.SingleCore, Version: 2 SafetyRTE.RS.MultiCore, Version: 1
Safety class:	ASIL-D

Id:	SafetyRTE.RS.ClientServer
Doctype:	`requirement`

Status:	APPROVED
Source:	Elektrobit Automotive GmbH
Version:	3
Description:	EB tresos Safety RTE shall provide synchronous client-server communication with the following properties: ► Synchronous communication ► Direct function calls ► Intra-partition communication ► Port-defined arguments ► Implementation data types are compatible to the server function argument data types ► Non-concurrent execution ► Server returns application errors ► Sequential scheduling ► Server runnable entity with runs inside exclusive area
Needs coverage of:	`rte_assumption`, `hazop`
Provides coverage to:	SafetyRTE.RS.SingleCore, Version: 2 SafetyRTE.RS.MultiCore, Version: 1
Safety class:	ASIL-D

Id:	SafetyRTE.RS.DataTransformation
Doctype:	`requirement`
Status:	APPROVED
Source:	Elektrobit Automotive GmbH
Version:	2
Description:	EB tresos Safety RTE shall support data transformation with the following properties: ► Serialization with Some/IP based transformer ► Serialization with COM based transformer ► E2E safety transformer for unqueued sender/receiver communication ► In-place or out-place handling of the buffer ► Transformer error handling within `Rte_Write/Rte_Read APIs` ► Execution despite data unavailability
Needs coverage of:	`rte_assumption`

Provides coverage to:	SafetyRTE.RS.SingleCore, Version: 2 SafetyRTE.RS.MultiCore, Version: 1
Safety class:	ASIL-D

6.4. Assumptions

This section defines all assumptions on the environment which are necessary to use EB tresos Safety RTE in a safe way.

NOTE **Fulfillment of the assumptions**

The integrator must ensure that the assumptions named in this chapter apply for the project.

6.4.1. Defining assumptions

Assumptions must be defined using the following scheme:

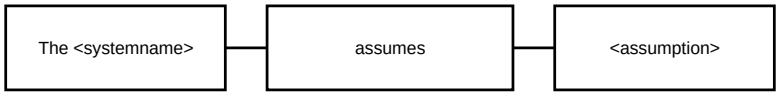

Figure 6.1. Scheme for defining assumptions

6.4.2. Assumptions

[SafetyRTE.Assumption.ReliableExecutionEnvironment]

EB tresos Safety RTE assumes that the hardware provides a *reliable execution environment*.

Note: In EB tresos Safety RTE a *reliable execution environment* is defined as a microcontroller that provides a mechanism that prevents or detects non-systematic hardware faults, e.g. data corruption or incorrect execution of instructions. Examples are lockstep mode and ECC memory.

[SafetyRTE.Assumption.ValidatedTools]

EB tresos Safety RTE assumes that the integrator takes appropriate measures to ensure the confidence in the used software tools, e.g. via tool qualifications.

Note: The used software tools include for example build environment, compiler, linker, and device flasher.

[SafetyRTE.Assumption.ReviewedToolOutput]

EB tresos Safety RTE assumes that the integrator reviews the output of each used tool and takes appropriate measures if warnings or errors exist.

Note: The used tools are for example the RteCV, EB tresos Studio, the compiler, and the linker.

[SafetyRTE.Assumption.CompilerUsage]

EB tresos Safety RTE assumes that the version and options of the used compiler are identical to the one that is specified in the [QSSAFERTE] or are approved separately by Elektrobit Automotive GmbH.
Note: If any other compiler or options are chosen, then the user has to perform additional verification measures that are adequate and sufficient to ensure that there are no faults of EB tresos Safety RTE caused by the chosen compiler or options. The used compiler or linker must at least issue a warning or error in the following cases:

- Inclusion of a non existing file
- Usage of an undeclared or undefined identifier (function, variable, macro, or type)
- Implicit declaration of a function
- Implementation of functions with different parameter types then specified in the declaration of that function
- Assignment of a void return value of a function to a variable
- Assignment of incompatible data types
- Calling function with incompatible parameter data types
- Declaration or definition of duplicated identifiers (function, variable, macro, or type)

[SafetyRTE.Assumption.CorrectAutosarModel]

EB tresos Safety RTE assumes that the AUTOSAR model used to generate the `RTE` is correct, i.e. it correctly reflects the requirements of the integrator and is valid according to AUTOSAR as well as to the restrictions of the configured modules.

[SafetyRTE.Assumption.ValidatedSoftware]

EB tresos Safety RTE assumes that any other software running on the ECU is validated according to the safety integrity level of its partition.

[SafetyRTE.Assumption.UnsupportedRteFeatureUsage]

EB tresos Safety RTE assumes that if the integrator uses `RTE` features which are not supported by EB tresos Safety RTE, the integrator validates these to comply with the item's safety requirements and to ensure freedom from interference with the other `RTE` features.

[SafetyRTE.Assumption.UnsupportedRteFeatureResults]

EB tresos Safety RTE assumes that the integrator validates the RteCV verification report and the RteCV log output and rejects the test results from all `RTE` features that are not supported by EB tresos Safety RTE.

[SafetyRTE.Assumption.RteUsage]

EB tresos Safety RTE assumes that the `RTE` is used according to [ASRRTE403]. This means among others:

- *Application software* correctly calls the `RTE` API functions necessary to perform the intended functionality.
- Return values of `RTE` APIs are checked for error conditions and if error conditions occur, these are handled appropriately.

- Any software on the ECU only calls RTE API functions, which are allowed in the corresponding context. For example a *runnable entity* only calls RTE API functions, which are configured for it.
- No software on the ECU, with exception of the RTE itself, directly accesses any RTE internals like functions or variables, even if they are visible outside of the RTE. To use an RTE feature, the official APIs and interfaces are used.
- Runnable entities are exclusively triggered by tasks defined by the RTE or RTE APIs.

Note: The startup and the shutdown of the RTE slightly differs from [ASRRTE403]. For more information, see **[SafetyRTE.Assumption.RteLifecycleUsage]**.

[SafetyRTE.Assumption.RteLifecycleUsage]

EB tresos Safety RTE assumes that the RTE is correctly started and stopped according to AUTOSAR, e.-g. the SchM is initialized before the RTE is started, with the following exceptions:

- Rte_Start() shall not be used. The Rte_Start_<partition>() functions of all configured RTE partitions shall be called instead before using the RTE functionality. Note: For example rte_sws_ext_-7577: Rte_Start_<partition>() may only be called after SchM_Init_<partition>().
- Rte_Stop() shall not be used. The Rte_Stop_<partition>() functions of all configured RTE partitions shall be called instead.
- SchM_Init() shall be generated as empty stub. The SchM_Init_<partition>() functions of all *BSW partition*s shall be called before calling Rte_Start_<partition>().
- SchM_Deinit() shall be generated as empty stub. The SchM_Deinit_<partition>() functions of all *BSW partition*s shall be used instead.
- A special task shall be configured for each partition that calls the SchM_Init_<partition>() and Rte_Start_<partition>(). These tasks have to be started by the OS after startup.

Note: <partition> is the shortName of the corresponding OsApplication.

NOTE

SchM and RTE startup

The exceptions described in **[SafetyRTE.Assumption.RteLifecycleUsage]** are necessary to prevent OS memory protection errors when calling a function on a trusted partition from a non-trusted partition.

An alternative to these exceptions is to configure memory regions and trusted functions, see [ASROS403] and the documentation of the used OS. In this case the lifecycle functions can be generated and used normally, but they have to be manually verified by the integrator.

[SafetyRTE.Assumption.Partitioning]

EB tresos Safety RTE assumes that partitioning is used to separate SWCs from each other according to their safety level.

[SafetyRTE.Assumption.MemoryProtection]

EB tresos Safety RTE assumes that memory protection is used and configured as follows:

- RTE functions have read access to the RTE internal variables and buffers of all partitions.

- `RTE` functions have write access to `RTE` internal variables and buffers of the partition, to which they belong.
- `RTE` functions which take input parameters by reference, i.e. as pointer, have read access to the location, to which the input parameter points.
- `RTE` functions with output parameters have write access to the location, to which the output parameter points.
- Functions from one partition do not have write access to variables and data of a partition with a higher safety integrity level.

[SafetyRTE.Assumption.TimingProtection]

EB tresos Safety RTE assumes that *logical program flow monitoring* and *temporal program flow monitoring* are used to secure the correct scheduling of tasks and runnable entities including the timing and sequence of the REs within a task. EB tresos Safety RTE assumes that the used safety mechanism is developed at least according to the highest safety integrity level of any SWC or software module running on the ECU.

[SafetyRTE.Assumption.SignalValidation]

EB tresos Safety RTE assumes that SWCs validate data received from SWCs with lower safety integrity level or take other appropriate measures before using the data if invalid data can compromise safety requirements.

[SafetyRTE.Assumption.FreedomFromInterferenceValidation]

EB tresos Safety RTE assumes that freedom from interference is ensured for functionality (including SMC buffers or other used buffers) located on a partition with lower safety integrity level that are used for communication with a SWC on a partition with higher safety integrity level, if invalid data can compromise safety requirements.

[SafetyRTE.Assumption.EndToEndProtection]

EB tresos Safety RTE assumes that end-to-end protection is used for inter-ECU communication with an appropriate profile to ensure the reliable transport of messages to and from other ECUs for safety-related signals. The chosen profile must support the identification of the following error conditions:

- Missing/lost messages
- Duplicate and injected (additional) messages
- Incorrect message sequence
- Invalid/corrupted data

EB tresos Safety RTE assumes that the end-to-end protection is developed at least according to the highest safety integrity level of any SWC or software module running on the ECU. In addition, it is assumed that data is read periodically in order to determine if a message has been delayed and is no longer valid.

Note: The sequence between different messages, i.e. different `ComSignal`s or `ComSignalGroup`s, is potentially not covered by standard end-to-end protection mechanisms. Make sure to either not rely on such a sequence or use an appropriate mechanism to ensure it.

[SafetyRTE.Assumption.ValidatedMemoryProtection]

EB tresos Safety RTE assumes that the memory protection is developed at least according to the highest safety integrity level of any SWC or software module running on the ECU.

[SafetyRTE.Assumption.ValidatedOs]

EB tresos Safety RTE assumes that the following AUTOSAR OS features with the corresponding API are developed at least according to the highest safety integrity level of any SWC or software module running on the ECU:

- Task execution and scheduling
- Task management:
  - `ActivateTask()`
  - `TerminateTask()`
- Interrupt handling:
  - `SuspendAllInterrupts()`
  - `ResumeAllInterrupts()`
  - `SuspendOsInterrupts()`
  - `ResumeOsInterrupts()`
- Resource management:
  - `GetResource()`
  - `ReleaseResource()`
- Spinlock management:
  - `GetSpinlock()`
  - `ReleaseSpinlock()`
- Event handling:
  - `GetEvent()`
  - `SetEvent()`
  - `ClearEvent()`
  - `WaitEvent()`
  - `OS_WaitGetClearEvent()` if available
- Error handling:
  - `ProtectionHook()`
  - `ErrorHook()`

[SafetyRTE.Assumption.CoverageAnalysis]

EB tresos Safety RTE assumes that the integrator performs a coverage analysis for the generated `RTE` code.

Note: RTE code which is not covered by code checks and also not by the review criteria must be validated by the integrator.

[SafetyRTE.Assumption.InitialValue]
EB tresos Safety RTE assumes that the integrator configures initial values for sender/receiver communication if undefined initial values violate the system's safety requirements.

[SafetyRTE.Assumption.OsInitialization]
EB tresos Safety RTE assumes that an exclusive area API is only used after the Os is started.

[SafetyRTE.Assumption.ExclusiveAreaImplementationMechanism]
EB tresos Safety RTE assumes that the configured implementation mechanism of an exclusive area provides appropriate interruption protection for all execution contexts that an *executable entity* may run inside. This means that the interruption protection is not too weak, i.e. the exclusive area can be interrupted. This also means that the interruption protection is not over-dimensioned, i.e. a weaker implementation mechanism can be used.

[SafetyRTE.Assumption.SharedExclusiveAreas]
EB tresos Safety RTE assumes that a runnable entity does not use an RTE exclusive area API for mutual exclusion protection within code shared with one or more BswModuleEntities.

[SafetyRTE.Assumption.InterEcu.BswPartitionMapping]
EB tresos Safety RTE assumes that all *BSW modules* used by the RTE for *inter-ECU communication* e.g. Com, are mapped to the partition that is referenced by parameter Rte/RteGeneration/BswConfiguration/BswOsApplicationRef.

[SafetyRTE.Assumption.ValidatedDataModel]
EB tresos Safety RTE assumes that the *RTE data model* is validated against the RteModel.xsd schema provided by the RTE. The validation is done automatically if not deactivated explicitly. To deactivate the automatic validation, an optional command line parameter can be used. See [USERSGUIDE].

6.5. Limitations

The following limitations on the usage of RTE features define the constraints, under which EB tresos Safety RTE can be used to verify that the generated RTE is correct.

The general feature set of EB tresos Safety RTE is that of EB tresos AutoCore Generic 8 RTE. For more information, see [AUTOCORERTEDOC]. For safety-related projects the features are limited to those features which are defined by the safety requirements in chapter Section 6.3, “Safety requirements” and the corresponding review and test specification.

NOTE **Handling unsupported features**

All other features are not covered by EB tresos Safety RTE and the review and test specification. Therefore the integrator is discouraged to use them in safety-related environments. If they are used in such an environment nonetheless, the integrator must ensure that the implementation of these features complies with the item's safety requirements and does not interfere with other functionality including EB tresos Safety RTE itself. For more information, see **[SafetyRTE.Assumption.UnsupportedRteFeatureUsage]** and **[SafetyRTE.Assumption.UnsupportedRteFeatureResults]**.

6.5.1. General limitations

[SafetyRTE.Limitation.General.PortInterfaceMappings]
EB tresos Safety RTE does not support port interface mappings.

[SafetyRTE.Limitation.General.NoIndirectAPI]
EB tresos Safety RTE does not support the indirect API.

[SafetyRTE.Limitation.General.NoMultipleInstantiation]
EB tresos Safety RTE does not support multiple instantiation of *software components (SWCs)*.

[SafetyRTE.Limitation.General.NoDevelopmentErrorTracing]
EB tresos Safety RTE does not support development error tracing.

[SafetyRTE.Limitation.General.NoDisabledPartitionActiveCheck]
EB tresos Safety RTE does not support disabling the partition active check.

[SafetyRTE.Limitation.General.IntraECUImplicitCommunication]
EB tresos Safety RTE supports only intra-ECU implicit sender/receiver communication.

[SafetyRTE.Limitation.General.NoOSEKCompatibilityMode]
EB tresos Safety RTE does not support the OSEK compatibility mode.

[SafetyRTE.Limitation.General.NoModeManagement]
EB tresos Safety RTE does not support mode management.

[SafetyRTE.Limitation.General.GeneratorOutput]
EB tresos Safety RTE does not support the `RTE`-only and `SchM`-only generator mode.

[SafetyRTE.Limitation.General.NoSchMVersionInfoApi]
EB tresos Safety RTE does not support the `SchM` version info API function.

[SafetyRTE.Limitation.General.NoPerInstanceMemory]
EB tresos Safety RTE does not support per-instance memory (PIM).

[SafetyRTE.Limitation.General.NoMeasurementCalibration]
EB tresos Safety RTE does not support measurement and calibration.

[SafetyRTE.Limitation.General.NoTriggerAPI]
EB tresos Safety RTE does not support the trigger API, `ExternalTriggerOccurredEvents`, and `InternalTriggerOccurredEvents`.

[SafetyRTE.Limitation.General.SchMExclusiveAreaLegacySupport]
EB tresos Safety RTE does not support the AUTOSAR 3.1 SchM API.

[SafetyRTE.Limitation.General.NoCooperativeTasksSupport]
EB tresos Safety RTE does not support cooperative task groups.

[SafetyRTE.Limitation.General.NoOneScheduleTablePerPartition]
EB tresos Safety RTE does not support the feature one schedule table per partition.

[SafetyRTE.Limitation.General.NoDisabledPartitioning]
EB tresos Safety RTE does not support configurations with partitioning disabled.

[SafetyRTE.Limitation.General.NoSinglePartition]
EB tresos Safety RTE does not support configurations with only one partition.

[SafetyRTE.Limitation.General.NoRipsAPI]
EB tresos Safety RTE does not support the RIPS API.

[SafetyRTE.Limitation.General.NoRteFreedomFromInterferenceReviewInstructions]
EB tresos Safety RTE does not support additional freedom from interference review instructions generated by the RTE.

6.5.2. Implementation data types

[SafetyRTE.Limitation.ImplementationDataTypes.NoEnumerationConstants]
EB tresos Safety RTE does not support enumeration constants for `ImplementationDataType` instances.

[SafetyRTE.Limitation.ImplementationDataTypes.SupportedCategories]
RteCV only supports `ImplementationDataType` instances with the `category` attribute set to one of the following values:

- `VALUE`
- `TYPE_REFERENCE`
- `DATA_REFERENCE`
- `STRUCTURE`
- `ARRAY`
- `UNION`

[SafetyRTE.Limitation.ImplementationDataTypes.SupportedBaseTypeCategory]
RteCV only supports `ImplementationDataType` instances with the `category` attribute set to `VALUE` that are referencing a `SwBaseType` with the `category` set to `FIXED_LENGTH`.

6.5.3. Application data types

[SafetyRTE.Limitation.ApplicationDataTypes.DataTypeMapping]
RteCV only supports application data type to implementation data type mappings that are valid according to [ASRSWCTR2011] chapter *6.2.5 "Compatibility of ApplicationDataType and ImplementationDataType"*.

6.5.4. Lifecycle

[SafetyRTE.Limitation.Lifecycle.NoStandardStartStop]
EB tresos Safety RTE does not support the API functions `Rte_Start()` and `Rte_Stop()`.

[SafetyRTE.Limitation.Lifecycle.NoStandardInitDeinit]
EB tresos Safety RTE does not support the API functions `SchM_Init()` and `SchM_Deinit()`.

[SafetyRTE.Limitation.Lifecycle.NoPartitionRestart]
EB tresos Safety RTE does not support the API functions `Rte_RestartPartition()` and `Rte_PartitionRestarting()`.

[SafetyRTE.Limitation.Lifecycle.NoTaskChains]
EB tresos Safety RTE does not support task chains.

[SafetyRTE.Limitation.Lifecycle.VfbTracing]
EB tresos Safety RTE supports only the task dispatch and terminate trace hooks. The activate task trace hook, the event trace hooks, and the executable entity trace hooks are not supported.

[SafetyRTE.Limitation.Lifecycle.NoUserScheduleTables]
EB tresos Safety RTE does not support user schedule tables.

[SafetyRTE.Limitation.Lifecycle.NoBackgroundEvents]
EB tresos Safety RTE does not support background events.

[SafetyRTE.Limitation.Lifecycle.NoInitEvents]
EB tresos Safety RTE does not support init events.

[SafetyRTE.Limitation.Lifecycle.NoRteInitializationRunnableBatch]
EB tresos Safety RTE does not support initialization of runnable batches.

[SafetyRTE.Limitation.Lifecycle.NoSingleScheduleTable]
EB tresos Safety RTE does not support a single `RTE` schedule table for all partitions.

6.5.5. Client/server communication

[SafetyRTE.Limitation.ClientServer.AsynchCommunication]
EB tresos Safety RTE does not support asynchronous client-server communication.

[SafetyRTE.Limitation.ClientServer.ClientEventMappings]
EB tresos Safety RTE does not support Rte ECU configurations where the clients are mapped to different tasks.

[SafetyRTE.Limitation.ClientServer.OperationInvokedEventMapping]
EB tresos Safety RTE does not support the implementation of client-server operations where the server's `OperationInvokedEvent` is mapped to a task. The server shall be invoked via a direct function call.

[SafetyRTE.Limitation.ClientServer.NoVfbTracing]
EB tresos Safety RTE does not support VFB tracing for client-server communication.

[SafetyRTE.Limitation.ClientServer.NoInterEcuCommunication]
EB tresos Safety RTE does not support inter-ECU client-server communication.

[SafetyRTE.Limitation.ClientServer.NoUnconnectedPorts]
EB tresos Safety RTE does not support unconnected client-server ports.

[SafetyRTE.Limitation.ClientServer.NoImplicitIRVAccess]
EB tresos Safety RTE does not support server `RunnableEntities` with access to an `implicitInterRunnableVariable`.

6.5.6. Sender/receiver communication

[SafetyRTE.Limitation.SenderReceiver.NoDataInvalidation]
EB tresos Safety RTE does not support data invalidation for sender/receiver communication.

[SafetyRTE.Limitation.SenderReceiver.NoTxAcknowledgement]
EB tresos Safety RTE does not support transmission acknowledgement for sender/receiver communication.

[SafetyRTE.Limitation.SenderReceiver.NoDataFilters]
EB tresos Safety RTE does not support data filters for sender/receiver communication.

[SafetyRTE.Limitation.SenderReceiver.NoUpdateFlag]
EB tresos Safety RTE does not support the use of the *update flag* for sender/receiver communication.

[SafetyRTE.Limitation.SenderReceiver.NoEvents]
EB tresos Safety RTE does not support the following events for the sender/receiver communication:

- `DataReceivedEvent`
- `DataReceiveErrorEvent`
- `DataSendCompletedEvent`
- `DataWriteCompletedEvent`

[SafetyRTE.Limitation.SenderReceiver.NoQueues]
EB tresos Safety RTE does not support queued sender/receiver communication.

[SafetyRTE.Limitation.SenderReceiver.NoTimeout]

EB tresos Safety RTE does not support communication timeouts and timeout notifications for sender/receiver communication.

[SafetyRTE.Limitation.SenderReceiver.1to1Communication]

EB tresos Safety RTE supports only 1:1 sender/receiver communication. It does not support other multiplicities like 1:n or n:1.

[SafetyRTE.Limitation.SenderReceiver.InterECU.NoSystemSignalFanOut]

EB tresos Safety RTE does not support inter-ECU sender/receiver communication with system signal fan-out.

[SafetyRTE.Limitation.SenderReceiver.InterECU.SendSignalQueueStrategy]

EB tresos Safety RTE supports only one send signal queue per core.

[SafetyRTE.Limitation.SenderReceiver.InterEcu.DataTransformationChain]

EB tresos Safety RTE only supports data transformation chains with an E2E transformer configured directly after the serializer.

[SafetyRTE.Limitation.SenderReceiver.InterEcu.ExecuteDespiteDataUnavailability]

EB tresos Safety RTE only supports data transformation chains with `E2E` transformer and executes the data transformation chain even if no data is available.

[SafetyRTE.Limitation.SenderReceiver.InterEcu.ExecuteTransformersDespiteLocalDataUnqueued]

EB tresos Safety RTE shall execute the data transformation chain even if no data is available for all unqueued sender/receiver connections with a data transformation.

[SafetyRTE.Limitation.SenderReceiver.NoHandleNeverReceived]

EB tresos Safety RTE does not support the *never received status* for sender/receiver communication.

[SafetyRTE.Limitation.SenderReceiver.DataConsistency]

EB tresos Safety RTE does only support the following data consistency mechanisms for sender/receiver communication:

- Interrupt locks
- Sequential scheduling

[SafetyRTE.Limitation.SenderReceiver.NoUnconnectedPorts]

EB tresos Safety RTE does not support unconnected sender/receiver ports.

[SafetyRTE.Limitation.SenderReceiver.NolocSupport]

EB tresos Safety RTE does not support the OS IOC for inter-partition sender/receiver communication.

[SafetyRTE.Limitation.SenderReceiver.NoVfbTracing]

EB tresos Safety RTE does not support VFB tracing for sender/receiver communication.

[SafetyRTE.Limitation.SenderReceiver.NoRangeChecks]

EB tresos Safety RTE does not support range checks for sender/receiver communication.

[SafetyRTE.Limitation.SenderReceiver.NoDirectReadFromCom]

EB tresos Safety RTE does not support direct read from `Com`.

[SafetyRTE.Limitation.SenderReceiver.NoReadByValue]
EB tresos Safety RTE does not support explicit message reception by value.

[SafetyRTE.Limitation.SenderReceiver.NoPeriodicBswOsTask]
EB tresos Safety RTE does not support the periodic execution of the BSW `OS` task.

[SafetyRTE.Limitation.SenderReceiver.NoMultipleBswOsTask]
EB tresos Safety RTE does not support the execution of multiple BSW `OS` tasks.

[SafetyRTE.Limitation.SenderReceiver.NoMultipleComInstances]
EB tresos Safety RTE does not support the execution of multiple `Com` instances.

[SafetyRTE.Limitation.SenderReceiver.NoComCbkNotInterruptableSupport]
EB tresos Safety RTE does not support `Rte_COMCbk` functions that are configured as not-interruptable.

[SafetyRTE.Limitation.SenderReceiver.TransformerBufferLengthComputation]
EB tresos Safety RTE does not support replacing the transformer buffer length computation by the length of the `Com` container.

[SafetyRTE.Limitation.SenderReceiver.DynamicLengthComSignal]
EB tresos Safety RTE does not support `ComSignals` with dynamic length.

[SafetyRTE.Limitation.SenderReceiver.NoUnmappedRunnables]
EB tresos Safety RTE does not support runnable entities that are not triggered by at least one `RTE` event. Note: Only runnable entities that are triggered by `RTE` events are executed in a `RTE` task context.

[SafetyRTE.Limitation.SenderReceiver.NoRteCoherentAccess]
EB tresos Safety RTE does not support the `RteCoherentAccess` feature.

[SafetyRTE.Limitation.SenderReceiver.NoRteImmediateBufferUpdate]
EB tresos Safety RTE does not support the `RteImmediateBufferUpdate` feature.

[SafetyRTE.Limitation.SenderReceiver.NoMultipleImplicitReadersOrWriters]
EB tresos Safety RTE does not support multiple implicit reading or writing runnable entities that access the same `VariableDataPrototype`.

[SafetyRTE.Limitation.SenderReceiver.NoRunnableWithReadAndWriteAccessSameVariableDataPrototype]
EB tresos Safety RTE does not support a runnable entity with implicit read access and implicit write access using the same `VariableDataPrototype` referencing a dedicated `RPortPrototype` and `PPortPrototype` (see [ASRRTE403], rte_sws_3955).

[SafetyRTE.Limitation.SenderReceiver.NoEnableTakeAddressImplicitPort]
EB tresos Safety RTE does not support the `EnableTakeAddress` feature for ports with implicit read access or implicit write access.

[SafetyRTE.Limitation.SenderReceiver.NoInitializeImplicitWriteBufferBeforeRunnableExecutes]
EB tresos Safety RTE does not support the initialization of data handle buffers at the beginning of a task with runnable entities with implicit write access.

6.5.7. Inter-runnable variables

[SafetyRTE.Limitation.InterRunnableVariables.SingleTaskMapping]

EB tresos Safety RTE supports the communication via inter-runnable variables only for runnable entities which are mapped to the same task. It does not support inter-runnable variables which are accessed from runnable entities mapped to different tasks.

[SafetyRTE.Limitation.InterRunnableVariables.NoVfbTracing]

EB tresos Safety RTE does not support VFB tracing for the communication via inter-runnable variables.

6.5.8. Exclusive areas

[SafetyRTE.Limitation.ExclusiveAreas.NoCooperativeRunnablePlacement]

EB tresos Safety RTE does not support the cooperative runnable placement mechanism for exclusive areas.

[SafetyRTE.Limitation.ExclusiveAreas.NoEbFastLock]

EB tresos Safety RTE does not support the EB fast lock mechanism for exclusive areas.

[SafetyRTE.Limitation.ExclusiveAreas.SchMOsResource]

EB tresos Safety RTE does not support the use of `OS` resources as a `SchM` exclusive area implementation mechanism.

[SafetyRTE.Limitation.ExclusiveAreas.NoVfbTracing]

EB tresos Safety RTE does not support VFB tracing for exclusive areas.

[SafetyRTE.Limitation.ExclusiveAreas.NoInternalOsResource]

EB tresos Safety RTE does not support internal `OS` resources for exclusive areas.

[SafetyRTE.Limitation.ExclusiveAreas.NoUserCallout]

EB tresos Safety RTE does not support the use of `USER_CALLOUT` as an exclusive area implementation mechanism.

6.5.9. Multi-Core

[SafetyRTE.Limitation.MultiCore.OsSupportsSpinlockLockMethod]

EB tresos Safety RTE supports only the API functions `GetSpinLock()` and `ReleaseSpinLock()` which additionally block/release all interrupts when acquiring/releasing a spinlock.

7. Verification criteria

This chapter defines the reviews to be performed.

7.1. Limitation checks

This section lists limitation checks. The integrator shall perform these checks to see if the used configuration is within the limits of EB tresos Safety RTE.

NOTE **Validation of unsupported features**

If any of the following checks fails, the corresponding element and all associated generated code must be validated by the integrator. It is not covered by EB tresos Safety RTE.

If not stated otherwise, all the checks must be performed for the relevant parts of the AUTOSAR model of the system application, for which EB tresos Safety RTE is used. Relevant parts are all elements in the AUTOSAR model, which belong to the ECU on which the system application runs.

7.1.1. General

[SafetyRTE.Review.Limitation.General.PortInterfaceMappings]
Verify that there are no `AssemblySwConnector` or `DelegationSwConnector` instances which reference a `PortInterfaceMapping`.

[SafetyRTE.Review.Limitation.General.NoIndirectAPI]
Verify that there is no `PortApiOption` in any `SwcInternalBehavior` which has the attribute `indirectAPI` set to `true` or `1`.

[SafetyRTE.Review.Limitation.General.NoMultipleInstantiation]
Verify that there is no `SwcInternalBehavior` with the attribute `supportsMultipleInstantiation` set to `true` or `1`.

[SafetyRTE.Review.Limitation.General.NoDevelopmentErrorTracing]
Verify that the `RTE` configuration parameter `/Rte/RteGeneration/RteDevErrorDetect` is disabled or set to `false` or `0`.

[SafetyRTE.Review.Limitation.General.NoDisabledPartitionActiveCheck]
Verify that the `RTE` configuration parameter `/Rte/RteGeneration/DisablePartitionActiveChecks` is set to `false` or `0`.

[SafetyRTE.Review.Limitation.General.IntraECUImplicitCommunication]

- Verify that there is no `RunnableEntity` which has any `VariableAccess` in the `dataWriteAccess` or `dataReadAccess` role, which references a `VariableDataPrototype` that is mapped to a `Signal` or `SignalGroup`.
- Verify that there is no `RunnableEntity` which has any `VariableAccess` in the `writtenLocalVariable` role or `readLocalVariable`, which references a `VariableDataPrototype` in the `implicitInterRunnableVariable` role.

[SafetyRTE.Review.General.NoOSEKCompatibilityMode]

Verify that the `RTE` configuration parameter `/Rte/RteGeneration/OSEKCompatibilityMode` is set to `false` or `0`.

[SafetyRTE.Review.Limitation.General.NoModeManagement]

Verify that there is no `PPortPrototype` or `RPortPrototype` which references a `ModeSwitchInterface`.

[SafetyRTE.Review.Limitation.General.GeneratorOutput]

Verify that the `RTE` configuration parameter `/Rte/RteGeneration/RteGeneratorOutput` is set to `FULL`.

[SafetyRTE.Review.Limitation.General.NoSchMVersionInfoApi]

Verify that the `RTE` configuration parameter `/Rte/RteBswGeneral/RteSchMVersionInfoApi` is set to `false`.

[SafetyRTE.Review.Limitation.General.NoPerInstanceMemory]

Verify that there is no software component with an `SwcInternalBehavior` which defines `perInstanceMemory` or `arTypedPerInstanceMemory`.

[SafetyRTE.Review.Limitation.General.NoMeasurementCalibration]

Verify that the configuration follows the following rules:

- The `RTE` configuration parameter `/Rte/RteGeneration/RteMeasurementSupport` is set to `false`.
- There is no software component with an `SwcInternalBehavior` which defines `perInstanceParameter`s or `sharedParameter`s.
- There is no `ImplementationDataType` or `DataPrototype` which contains `SwDataDefProps` with the attribute `swCalibrationAccess` set to something different than `notAccessible`.
- There is no `ParameterSwComponentType` defined.

[SafetyRTE.Review.Limitation.General.NoTriggerAPI]

Verify that there is no software component with an `BswInternalBehavior` with a `BswModuleEntity` which defines an `issuedTrigger`.

[SafetyRTE.Review.Limitation.General.UsedEvents]

Verify the following event usage rules:

- There is no `RTEEvent` used other than `TimingEvent` and `OperationInvokedEvent` instances.

- There is no `BswEvent` used other than `BswTimingEvent`s.

[SafetyRTE.Review.Limitation.General.SchMExclusiveAreaLegacySupport]
Verify that the configuration parameter `ASR31SchMExclusiveAreaAPISupport` is set to false.

[SafetyRTE.Review.Limitation.General.NoCooperativeTasksSupport]
Verify that the `RteGeneration` container in the Rte configuration does not have a `CooperativeTasks` entry.

[SafetyRTE.Review.Limitation.General.NoOneScheduleTablePerPartition]
Verify that the `RTE` configuration parameter `/Rte/RteGeneration/OneScheduleTablePerPartition` is disabled.

[SafetyRTE.Review.Limitation.General.NoDisabledPartitioning]
Verify that the `RTE` configuration parameter `/Rte/RteGeneration/InterPartitionCommunication` is not set to `disabled`.

[SafetyRTE.Review.Limitation.General.NoSinglePartition]
Verify that the `OS` configuration container `/Os/OsApplication` contains more than one partition.

[SafetyRTE.Review.Limitation.General.NoRipsAPI]
Verify that the `RTE` configuration container `/Rte/RteRips` is empty.

[SafetyRTE.Review.Limitation.General.NoRteFreedomFromInterferenceReviewInstructions]
Verify that the `RTE` configuration parameter `/Rte/RteGeneration/GenerateRteFreedomFromInterferenceReviewInstructions` is either disabled or set to `false`.

7.1.2. Implementation data types

[SafetyRTE.Review.Limitation.ImplementationDataTypes.NoEnumerationConstants]
Verify that there is no `ImplementationDataType` which references a `CompuMethod` of category `TEXTTABLE`, `SCALE_LINEAR_AND_TEXTTABLE`, `SCALE_RATIONAL_AND_TEXTTABLE`, or `BITFIELD_TEXTTABLE`, which contains `CompuScales` with point ranges, i.e. `CompuScales` where the lower and upper limit are equal.

[SafetyRTE.Review.Limitation.ImplementationDataTypes.SupportedCategories]
Verify that there is no `ImplementationDataType` with a category other than `VALUE`, `TYPE_REFERENCE`, `DATA_REFERENCE`, `STRUCTURE`, `ARRAY` or `UNION`.

[SafetyRTE.Review.Limitation.ImplementationDataTypes.SupportedBaseTypeCategory]
Verify that there is no `ImplementationDataType` with category `VALUE` that is referencing a `SwBaseType` with category other than `FIXED_LENGTH`.

7.1.3. Application data types

[SafetyRTE.Review.Limitation.ApplicationDataTypes.DataTypeMapping]

Verify that the following statements apply for all `DataTypeMaps`:

- `ApplicationDataTypes` of category `VALUE` are mapped to an `ImplementationDataType` of category `VALUE`.
- `ApplicationDataTypes` of category `ARRAY`, `VAL_BLK` are mapped to an `ImplementationDataType` of category `ARRAY` where:
  - The array size of both mapped data types are the same.
  - The `ArraySizeSemantics` (if it exists) of both mapped data types are the same.
  - The base types of both mapped array data types are compatible according to this review.
- `ApplicationDataTypes` of category `STRUCTURE` are mapped to an `ImplementationDataType` of category `STRUCTURE` where:
  - The corresponding pair of elements for both the mapped data types are compatible according to this review.
- `ApplicationDataTypes` of category `STRUCTURE` that represent an optional element structure are mapped to an `ImplementationDataType` of category `STRUCTURE` that represents an optional element structure where:
  - The corresponding pair of elements for both the mapped data types have the same value for the attribute `IsOptional`.
  - The corresponding pair of elements for both the mapped data types are compatible according to this review.
- `ApplicationDataTypes` of category `BOOLEAN` are mapped to an `ImplementationDataType` of category `VALUE`.
- `ApplicationDataTypes` of category
  - `COM_AXIS`, or
  - `RES_AXIS`, or
  - `CURVE`, or
  - `MAP`, or
  - `CUBOID`, or
  - `CUBE_4`, or
  - `CUBE_5`

  are mapped to an `ImplementationDataType` of category `STRUCTURE` or `ARRAY`.

- `ApplicationDataTypes` are never mapped to an `ImplementationDataType` of the following categories:
  - `UNION`
  - `DATA_REFERENCE`
  - `FUNCTION_REFERENCE`
- `ApplicationDataTypes` are never mapped to an `ImplementationDataType` that contains `SubElements` of the following categories:
  - `UNION`
  - `DATA_REFERENCE`
  - `FUNCTION_REFERENCE`

Note: Indirection created by `ImplementationDataTypes` of category `TYPE_REFERENCE` shall be resolved before applying these review instructions.

7.1.4. Lifecycle

[SafetyRTE.Review.Limitation.Lifecycle.NoStandardStartStop]

Verify the following configuration rules regarding general start and stop APIs:

- The `RTE` configuration parameter `/Rte/RteGeneration/GenerateEmptyRteStartStopStubs` is set to `true`.
- The application software does not call the general APIs `Rte_Start()` and `Rte_Stop()`.
- The application software does not call the general APIs `SchM_Init()` and `SchM_Deinit()`.

[SafetyRTE.Review.Limitation.Lifecycle.NoPartitionRestart]

Verify that the `OS` configuration parameter `/Os/OsApplication/OsRestartTask` is not enabled and set for any `OsApplication`.

[SafetyRTE.Review.Limitation.Lifecycle.NoTaskChains]

Verify that the `RTE` configuration parameter list `/Rte/RteGeneration/TaskChain` is empty.

[SafetyRTE.Review.Limitation.Lifecycle.VfbTracing]

Verify that the `RTE` configuration list `/Rte/RteGeneration/RteVfbTraceFunction` does not contain any of the following entries:

- `Rte_Task_Activate`
- `Rte_Task_SetEvent`
- `Rte_Task_WaitEvent`
- `Rte_Task_WaitEventRet`
- Executable trace hooks, i.e. no trace hook that starts with `Rte_Runnable` or `Rte_Schedulable`

[SafetyRTE.Review.Limitation.Lifecycle.NoUserScheduleTables]
Verify that the `RTE` configuration parameter `/Rte/RteGeneration/OsScheduleTableActivationMechanism` is not set to `NEXT`.

[SafetyRTE.Review.Limitation.Lifecycle.NoRteInitializationRunnableBatch]
Verify that the `RTE` configuration container list `/Rte/Rte/RteInitializationRunnableBatch` is empty.

[SafetyRTE.Review.Limitation.Lifecycle.NoSingleScheduleTable]
Verify that the `RTE` configuration parameter `/Rte/RteGeneration/SingleScheduleTablePartitionRef` is either disabled or not set.

7.1.5. Sender/receiver communication

[SafetyRTE.Review.Limitation.SenderReceiver.NoDataInvalidation]
Verify that there is no `SenderReceiverInterface` with an `InvalidationPolicy` with the attribute `handleInvalid` set to `keep` or `replace`.

[SafetyRTE.Review.Limitation.SenderReceiver.NoTxAcknowledgement]
Verify that there is no `SenderComSpec` with a `transmissionAcknowledge` container defined.

[SafetyRTE.Review.Limitation.SenderReceiver.NoDataFilters.Receiver]
Verify that each `NonqueuedReceiverComSpec` only has `DataFilters` of `DataFilterType` `ALWAYS`.

[SafetyRTE.Review.Limitation.SenderReceiver.NoDataFilters.Sender]
Verify that each `PPortPrototype` with `AdminData.SDGS` (`SpecialDataGroups`) only has `DataFilters` of `DataFilterType` `ALWAYS`.

[SafetyRTE.Review.Limitation.SenderReceiver.NoUpdateFlag]
Verify that there is no `NonqueuedReceiverComSpec` with the attribute `enableUpdate` set to `true` or `1`.

[SafetyRTE.Review.Limitation.SenderReceiver.NoQueues]
Verify that the following applies:

1. There is no `QueuedReceiverComSpec`
2. There is no `QueuedSenderComSpec`
3. There is no `ImplementationDataType` with a `swImplPolicy` set to `queued`
4. There is no `VariableDataPrototype` with a `swImplPolicy` set to `queued` that references an `ImplementationDataType` with no `swImplPolicy`.

[SafetyRTE.Review.Limitation.SenderReceiver.NoTimeout]
Verify that there is no `NonqueuedReceiverComSpec` with the attribute `aliveTimeout` different to `0`.

[SafetyRTE.Review.Limitation.SenderReceiver.1to1Communication]
Verify that each port with a sender/receiver interface is only referenced by not more than one software connector.

[SafetyRTE.Review.Limitation.SenderReceiver.InterECU.NoSystemSignalFanOut]
Verify that there is no system signal fan-out configured.

[SafetyRTE.Review.Limitation.SenderReceiver.InterECU.SendSignalQueueStrategy]
Verify that the RTE configuration parameter /Rte/RteGeneration/SendSignalQueueStrategy is set to OnePerCore.

[SafetyRTE.Review.Limitation.SenderReceiver.InterEcu.DataTransformationChain]
Verify that in each data transformation chain the E2E transformer is configured directly after the serializer.

[SafetyRTE.Review.Limitation.SenderReceiver.InterEcu.ExecuteDespiteDataUnavailability]
Verify for each data transformation chain with an E2E transformer that configuration parameter ExecuteDespiteDataUnavailability is set to true.

[SafetyRTE.Review.Limitation.SenderReceiver.InterEcu.ExecuteTransformersDespiteLocalDataUnqueued]
Verify that the RTE configuration parameter /Rte/RteGeneration/ExecuteTransformersDespiteLocalDataUnqueued is set to true.

[SafetyRTE.Review.Limitation.SenderReceiver.NoHandleNeverReceived]
Verify that there is no NonqueuedReceiverComSpec with the attribute handleNeverReceived set to true or 1.

[SafetyRTE.Review.Limitation.SenderReceiver.DataConsistency]
For the data consistency mechanism the following shall apply:

- Verify that the RTE configuration parameter /Rte/RteGeneration/DataConsistencyMechanism is set to InterruptBlocking.
- Verify that the RTE configuration parameter /Rte/RteGeneration/InterruptBlockingFunction is set to SuspendResumeAllInterrupts or Rte_UserDefinedIntLockUnlock.

[SafetyRTE.Review.Limitation.SenderReceiver.NoUnconnectedPorts]
Verify that all PPortPrototypes and RPortPrototypes that reference a SenderReceiverInterface are connected via exactly one AssemblySwConnector or DelegationSwConnector.

[SafetyRTE.Review.Limitation.SenderReceiver.NoIocSupport]
Verify that the RTE configuration parameter /Rte/RteGeneration/InterPartitionCommunication is set to SharedMemory.

[SafetyRTE.Review.Limitation.SenderReceiver.NoVfbTracing]
Verify that there is no entry in the RTE configuration list /Rte/RteGeneration/RteVfbTraceFunction which starts with Rte_WriteHook or Rte_ReadHook.

[SafetyRTE.Review.Limitation.SenderReceiver.NoRangeChecks]
Verify that the handleOutOfRange attribute is set according to the following rules:

- There is no SenderComSpec or ReceiverComSpec with a handleOutOfRange attribute set to a value other than none.

- There is no `ISignal` with an `ISignalProps` with the `handleOutOfRange` attribute set to a value other than `none`.

[SafetyRTE.Review.Limitation.SenderReceiver.NoDirectReadFromCom]
Verify that the `RTE` configuration parameter `/Rte/RteGeneration/DirectReadFromCom` is disabled or set to `false`.

[SafetyRTE.Review.Limitation.SenderReceiver.NoReadByValue]
Verify that there is no `RunnableEntity` with a `VariableAccess` in the `dataReceivePointByValue` role.

[SafetyRTE.Review.Limitation.SenderReceiver.NoPeriodicBswOsTask]
Verify that the `RTE` configuration parameter `/Rte/RteGeneration/BswOsTaskPeriod` is disabled.

[SafetyRTE.Review.Limitation.SenderReceiver.NoMultipleBswOsTask]
Verify that the `RTE` configuration container `/Rte/RteGeneration/ComTaskConfiguration` is empty.

[SafetyRTE.Review.Limitation.SenderReceiver.NoMultipleComInstances]
Verify that for all `ComConfig` containers the `Com` configuration container `/Com/ComConfig/ComTimeBase` is enabled.

[SafetyRTE.Review.Limitation.SenderReceiver.NoComCbkNotInterruptable]
Verify that the `RTE` configuration parameter `/Rte/RteGeneration/ComCbkNotInterruptable` is disabled or set to `false`.

[SafetyRTE.Review.Limitation.SenderReceiver.TransformerBufferLengthComputation]
Verify that the `RTE` configuration parameter `/Rte/RteGeneration/OverrideXfBufferComputation` is disabled or set to `false`.

[SafetyRTE.Review.Limitation.SenderReceiver.DynamicLengthComSignal]
Verify that there is no Com signal with `ComSignalType` set to `UINT8_DYN`.

[SafetyRTE.Review.Limitation.SenderReceiver.NoUnmappedRunnables]
Verify that each `RunnableEntity` is triggered by an event. Additionally, verify that for each event at least one `RTE` configuration container `/Rte/RteSwComponentInstance/RteEventToTaskMapping` referring the event via the `RTE` configuration parameter `RteEventRef` exists.

[SafetyRTE.Review.Limitation.SenderReceiver.NoRteCoherentAccess]
Verify that the configuration parameter `/Rte/RteImplicitCommunication/RteCoherentAccess` is set to `false` for all elements in the `/Rte/RteImplicitCommunication` configuration container.

[SafetyRTE.Review.Limitation.SenderReceiver.NoRteImmediateBufferUpdate]
Verify that the configuration parameter `/Rte/RteImplicitCommunication/RteImmediateBufferUpdate` is set to `false` for all elements in the `/Rte/RteImplicitCommunication` configuration container.

[SafetyRTE.Review.Limitation.SenderReceiver.NoMultipleImplicitReadersOrWriters]

- Verify that there is no `VariableDataPrototype` that is referenced by multiple runnable entities with a `VariableAccess` in the `dataReadAccess` role.

- Verify that there is no `VariableDataPrototype` that is referenced by multiple runnable entities with a `VariableAccess` in the `dataWriteAccess` role.

[SafetyRTE.Review.Limitation.SenderReceiver.NoRunnableWithReadAndWriteAccessSameVariableDataPrototype]

Verify that there is no runnable entity with more than one `VariableAccess` in the `dataReadAccess` and `dataWriteAccess` role accessing the same `VariableDataPrototype` that references a dedicated `RPortPrototype` and `PPortPrototype`.
Note: Multiple `VariableAccesses` to the same `PRPortPrototype` are supported.

[SafetyRTE.Review.Limitation.SenderReceiver.NoEnableTakeAddressImplicitPort]

Verify that the configuration parameter `EnableTakeAddress` is set to `false` for all `PortPrototype` that are referenced by a `VariableAccess` in the `dataReadAccess` or `dataWriteAccess` role.

[SafetyRTE.Review.Limitation.SenderReceiver.NoInitializeImplicitWriteBufferBeforeRunnableExecutes]

Verify that the configuration parameter `/Rte/RteGeneration/InitializeImplicitWriteBufferBeforeRunnableExecutes` is set to `false`.

7.1.6. Inter-runnable variables

[SafetyRTE.Review.Limitation.InterRunnableVariables.SingleTaskMapping]

Verify that the parameter `/Rte/RteSwComponentInstance/RteEventToTaskMapping/RteMappedToTaskRef` of all runnable entities which use the same inter-runnable variable refers to the same `OS` task.

[SafetyRTE.Review.Limitation.InterRunnableVariables.NoVfbTracing]

Verify that there is no entry in the `RTE` configuration list `/Rte/RteGeneration/RteVfbTraceFunction` which starts with `Rte_IrvWriteHook` or `Rte_IrvReadHook`.

7.1.7. Client/server communication

[SafetyRTE.Review.Limitation.ClientServer.AsynchCommunication]

Verify that there are no `AsynchronousServerCallPoint`s defined.

[SafetyRTE.Review.Limitation.ClientServer.ClientEventMappings]

Verify that all clients which are connected to the same server are mapped to the same task.

[SafetyRTE.Review.Limitation.ClientServer.OperationInvokedEventMapping]

Verify that the `OperationInvokedEvent` instances are not mapped to a task.

[SafetyRTE.Review.Limitation.ClientServer.NoVfbTracing]

Verify that there is no entry in the `RTE` configuration list `/Rte/RteGeneration/RteVfbTraceFunction` which starts with `Rte_CallHook`.

[SafetyRTE.Review.Limitation.ClientServer.NoInterEcuCommunication]
Verify that there are no `ClientServerToSignalMappings` or `ClientServerToSignalGroupMappings` defined.

[SafetyRTE.Review.Limitation.ClientServer.NoUnconnectedPorts]
Verify that all `PPortPrototypes` and `RPortPrototypes` that reference a `ClientServerInterface` are connected via at least one `AssemblySwConnector` or `DelegationSwConnector`.

[SafetyRTE.Review.Limitation.ClientServer.NoImplicitIRVAccess]
Verify that there is no `OperationInvokedEvent` that references a server `RunnableEntity` with a `VariableAccess` to a `VariableDataPrototype` in the `implicitInterRunnableVariable` role.

7.1.8. Exclusive areas

[SafetyRTE.Review.Limitation.ExclusiveAreas.NoCooperativeRunnablePlacement]
Verify that for all `RteSwComponentInstance` and `RteBswModuleInstance` containers the `RTE` configuration parameters `/Rte/RteSwComponentInstance/RteExclusiveAreaImplementation/RteExclusiveAreaImplMechanism` and `/Rte/RteBswModuleInstance/RteBswExclusiveAreaImpl/RteExclusiveAreaImplMechanism` are not set to `COOPERATIVE_RUNNABLE_PLACEMENT`.

[SafetyRTE.Review.Limitation.ExclusiveAreas.NoEbFastLock]
Verify that for all `RteSwComponentInstance` and `RteBswModuleInstance` containers the `RTE` configuration parameters `/Rte/RteSwComponentInstance/RteExclusiveAreaImplementation/RteExclusiveAreaImplMechanism` and `/Rte/RteBswModuleInstance/RteBswExclusiveAreaImpl/RteExclusiveAreaImplMechanism` are not set to `EB_FAST_LOCK`.

[SafetyRTE.Review.Limitation.ExclusiveAreas.SchMOsResource]
Verify that for all `RteBswModuleInstance` containers the `RTE` configuration parameters `/Rte/RteBswModuleInstance/RteBswExclusiveAreaImpl/RteExclusiveAreaImplMechanism` are not set to `OS_RESOURCE`.

[SafetyRTE.Review.Limitation.ExclusiveAreas.NoVfbTracing]
Verify that there is no entry in the `RTE` configuration list `/Rte/RteGeneration/RteVfbTraceFunction` which starts with `Rte_EnterHook` or `Rte_ExitHook`.

[SafetyRTE.Review.Limitation.ExclusiveAreas.NoInternalOsResource]
Verify that there is no `OS` resource with configuration parameter `/Os/OsResource/OsResourceProperty` set to `INTERNAL`.

[SafetyRTE.Review.Limitation.ExclusiveAreas.NoUserCallout]
Verify that for all `RteSwComponentInstance` and `RteBswModuleInstance` containers the `RTE` configuration parameters `/Rte/RteSwComponentInstance/RteExclusiveAreaImplementation/RteExclusiveAreaImplMechanism` and `/Rte/RteBswModuleInstance/RteBswExclusiveAreaImpl/RteExclusiveAreaImplMechanism` are not set to `USER_CALLOUT`.

7.1.9. Multi-Core

[SafetyRTE.Review.Limitation.MultiCore.OsSupportsSpinlockLockMethod]

Verify that the `RTE` configuration parameter `/Rte/RteGeneration/OsSupportsSpinlockLockMethod` is set to `true`.

[SafetyRTE.Review.Limitation.MultiCore.OsSpinlockLockMethod]

Verify that for all spinlocks the `OS` configuration parameter `/Os/OsSpinlock/OsSpinlockLockMethod` is set to `LOCK_ALL_INTERRUPTS`.

7.2. Reviews

7.2.1. Introduction

This section defines the reviews to be performed to validate the generated `RTE`.

7.2.2. General

[SafetyRTE.Review.General.ErrorCodes]

Verify that the `RTE` correctly defines necessary error and status codes.
Note: For more detailed review instructions, see Section " **[SafetyRTE.Review.General.ErrorCodes]** ".

[SafetyRTE.Review.General.ExtDepend.MemCpy.Implementation]

Verify that the `TS_MemCpy()` and the corresponding CPU-specific function for copying memory bytes is correctly implemented, i.e.:

- Verify that the C-macro `TS_MemCpy` exists and has the following definition.

```
#define TS_MemCpy (d,s,n) ([call CPU-specific function to copy memory bytes])
```

- Verify that the C-macro `TS_MemCpy` maps to the correct CPU-specific function to copy memory bytes.
- Verify that the CPU-specific function copies exactly n bytes of memory from the source data to the destination data for array sizes from 1 to n, where n is the maximum array length sent by the RTE.
- Verify that the CPU-specific function does not modify the data pointed to by the source address.
- Verify that the CPU-specific function does not modify the length value.

[SafetyRTE.Review.General.ExtDepend.AtomicThreadFence.Implementation]

Verify that the `Atomics_ThreadFence()` and the corresponding CPU-specific function for establishing a sequentially consistent memory barrier is correctly implemented, i.e.:

- Verify that the C-macro `Atomics_ThreadFence` exists and has the following definition.

```
#define Atomics_ThreadFence() ([call CPU-specific function to insert a mem-
ory barrier])
```

- Verify that the C-macro `Atomics_ThreadFence` maps to the correct CPU-specific function to establish the memory barrier.
- Verify that the CPU-specific function implements a sequentially consistent memory barrier.

[SafetyRTE.Review.General.PlatformDatatypesPropertiesFile]

Verify that the platforms data types properties file correctly defines sizes and atomicity of data types.

7.2.3. Lifecycle

[SafetyRTE.Review.Lifecycle.PartitionState.Initialization]

Verify that the `RTE` initializes each partition state correctly.

Note: For more detailed review instructions, see Section " **[SafetyRTE.Review.Lifecycle.PartitionState.Initialization]** ".

[SafetyRTE.Review.Lifecycle.Rte_Start.Implementation]

Verify that `Rte_Start_<partition>()` calls `Rte_Common_Init_<partition>()`, if there are *global buffer* assigned to the partition.

Verify that each `Rte_Start_<partition>()` API correctly updates the partition state.

[SafetyRTE.Review.Lifecycle.Rte_Stop.Implementation]

Verify that each `Rte_Stop_<partition>()` API correctly updates the partition state.

[SafetyRTE.Review.Lifecycle.UnsupportedAPIs]

Verify that the `RTE` does not implement any of the following unsupported lifecycle APIs:

- `Rte_PartitionRestarting_<partition>()`
- `Rte_RestartPartition_<partition>()`

[SafetyRTE.Review.Lifecycle.TraceHooks]

Verify that the `RTE` implements all hook functions called by `Rte_Start_<partition>()`, `Rte_Stop_<partition>()`, and `SchM_Deinit()` as `((void)(0))`.

Note: `SchM_Init()` does not call any hook function.

[SafetyRTE.Review.Lifecycle.PartitionStates]

Verify that the `SchM` and partition state constants are correctly defined.

Note: For more detailed review instructions, see Section " **[SafetyRTE.Review.Lifecycle.PartitionStates]** ".

[SafetyRTE.Review.Lifecycle.Rte_Common_Init.Implementation]

Verify that the `Rte_Common_Init_<partition>()` correctly initializes the *global buffer* which are assigned to the partition.

7.2.4. Tasks

[SafetyRTE.Review.Tasks.Existence]

Verify that the RTE implements all necessary tasks.

Note: For more detailed review instructions, see Section " **[SafetyRTE.Review.Tasks.Existence]** ".

[SafetyRTE.Review.Tasks.ExecutionCounter.Initialization]

Verify that the RTE initializes each execution counter variable correctly.

Note: For more detailed review instructions, see Section " **[SafetyRTE.Review.Tasks.ExecutionCounter.Initialization]** ".

[SafetyRTE.Review.Tasks.Implementation]

Verify that the RTE implements each task correctly.

Note: For more detailed review instructions, see Section " **[SafetyRTE.Review.Tasks.Implementation]** ".

[SafetyRTE.Review.Tasks.EventDeclaration]

Verify that the RTE correctly declares each OS event used by an extended RTE task and no further OS events beyond these.

Note: For more detailed review instructions, see Section " **[SafetyRTE.Review.Tasks.EventDeclaration]** ".

[SafetyRTE.Review.Tasks.Rte_WaitGetClearEvent.Implementation]

Verify that the RTE implements each `Rte_WaitGetClearEvent_<partition>()` function correctly.

Note: For more detailed review instructions, see Section " **[SafetyRTE.Review.Tasks.Rte_WaitGetClearEvent.Implementation]** ".

[SafetyRTE.Review.Tasks.Rte_WaitGetClearEvent.TraceHooks]

Verify that the RTE implements all hook functions called by `Rte_WaitGetClearEvent_<partition>()` as `((void)(0))`.

[SafetyRTE.Review.Tasks.TraceHooks]

Verify that the RTE implements all hook functions called by tasks as `((void)(0))`.

7.2.5. Partitioning

[SafetyRTE.Review.Partitioning.MemorySections]

If you use linking and memory protection based on AUTOSAR memory sections, verify that EB tresos Safety RTE correctly uses memory mapping and compiler abstraction. This includes among other things:

- AUTOSAR compiler abstraction is used correctly, e.g. the macros are used correctly and memory and pointer classes match the enclosing memory mapping.
- AUTOSAR memory mapping is used correctly, e.g. memory section definitions are complete and functions, variables and, constants are correctly allocated to the different memory sections.
- AUTOSAR memory allocation keywords are extended by partition-specific keywords, so that the corresponding parts, e.g. functions and variables are linked separately.

[SafetyRTE.Review.Partitioning.PointerClasses]

If your system supports different classes of pointers, e.g. near/far pointers, which can only address part of/ the whole memory, verify that EB tresos Safety RTE correctly uses compiler abstraction for all its pointers and that the used pointer class allows the access to all memory sections necessary for the corresponding pointer.

[SafetyRTE.Review.Partitioning.SeparatedPartitions]

Verify that EB tresos Safety RTE implements all partition-specific `.c` and `.h` files.

Note: For more detailed review instructions, see Section " **[SafetyRTE.Review.Partitioning.Separated-Partitions]** ".

[SafetyRTE.Review.Partitioning.SuperfluousFiles]

Verify that EB tresos Safety RTE does not implement superfluous files, i.e. files which are not necessary for the configured functionality, e.g. partition-specific files for a not existing partition.

Note: For more detailed review instructions, see Section " **[SafetyRTE.Review.Partitioning.Superfluous-Files]** ".

7.2.6. Sender/receiver communication

[SafetyRTE.Review.SenderReceiver.Buffer.DataType]

Verify that the receive buffer and data handle buffers of each sender/receiver connection have the correct data type.

Note: For more detailed review instructions, see Section " **[SafetyRTE.Review.SenderReceiver.Buffer.-DataType]** ".

[SafetyRTE.Review.SenderReceiver.Buffer.InitialValue]

Verify that the initial value of each sender/receiver connection is correct.

Note: For more detailed review instructions, see Section " **[SafetyRTE.Review.SenderReceiver.Buffer.-InitialValue]** ".

[SafetyRTE.Review.SenderReceiver.Smc.ReceiveBuffer.InitialValue]

Verify that the initial value of each Smc connection is correct.

Note: For more detailed review instructions, see Section " **[SafetyRTE.Review.SenderReceiver.Smc.-ReceiveBuffer.InitialValue]** ".

[SafetyRTE.Review.SenderReceiver.DataHandleBuffer.InitialValue]

Verify that the initial value of each DataHandle buffer is correct.

Note: For more detailed review instructions, see Section " **[SafetyRTE.Review.SenderReceiver.DataHandleBuffer.InitialValue]** ".

[SafetyRTE.Review.SenderReceiver.General.UnsupportedAPIs]

Verify that the `RTE` does not implement any of the following unsupported explicit sender/receiver APIs:

- `Rte_DRead_<p>_<o>()`
- `Rte_Feedback_<p>_<o>()`

- `Rte_Invalidate_<p>_<o>()`
- `Rte_IsUpdated_<p>_<o>()`

[SafetyRTE.Review.SenderReceiver.TraceHooks]

Verify that the `RTE` implements all hook functions called by `Rte_[I]Write_<p>_<o>()`, `Rte_[I]Read_<p>_<o>()`, and `Rte_COMCbk_<s[ng]>()` as `((void)(0))`.

[SafetyRTE.Review.SenderReceiver.SuperfluousElements]

Verify that the `RTE` does not implement any superfluous variables or functions.

Note: For more detailed review instructions, see Section " **[SafetyRTE.Review.SenderReceiver.SuperfluousElements]** ".

[SafetyRTE.Review.SenderReceiver.ComSignalGroupLength]

Verify that the length of the Com signal group is equal to or less than the length of transformer buffer.

Note: For more detailed review instructions, see Section " **[SafetyRTE.Review.SenderReceiver.ComSignalGroupLength]** ".

[SafetyRTE.Review.SenderReceiver.SpinLockResource]

Verify that the correct spinlock resource is used for an inter-core connection.

Note: For more detailed review instructions, see Section " **[SafetyRTE.Review.SenderReceiver.SpinLockResource]** ".

7.2.7. Inter-runnable variables

[SafetyRTE.Review.InterRunnableVariables.SharedBuffer]

Verify that the shared buffer of each connection via an inter-runnable variable is unique and has the correct data type.

Note: For more detailed review instructions, see Section " **[SafetyRTE.Review.InterRunnableVariables.SharedBuffer]** ".

[SafetyRTE.Review.InterRunnableVariables.SharedBuffer.InitialValue]

Verify that the initial value of each shared buffer is correct.

Note: For more detailed review instructions, see Section " **[SafetyRTE.Review.InterRunnableVariables.SharedBuffer.InitialValue]** ".

[SafetyRTE.Review.InterRunnableVariables.TraceHooks]

Verify that the `RTE` implements all hook functions called by `Rte_IrvWrite_<re>_<o>()` and `Rte_IrvRead_<re>_<o>()` as `((void) (0))`.

[SafetyRTE.Review.InterRunnableVariables.SuperfluousElements]

Verify that the `RTE` does not implement any superfluous variables or functions.

Note: For more detailed review instructions, see Section " **[SafetyRTE.Review.InterRunnableVariables.SuperfluousElements]** ".

7.2.8. Explicit exclusive areas

[SafetyRTE.Review.ExplicitExclusiveAreas.Rte_Enter.Existence]

Verify that the RTE implements all necessary Rte_Enter_<name>() APIs with the correct signature.

Note: For more detailed review instructions, see Section " **[SafetyRTE.Review.ExplicitExclusiveAreas.Rte_Enter.Existence]** ".

[SafetyRTE.Review.ExplicitExclusiveAreas.Rte_Enter.Implementation]

Verify that each Rte_Enter_<name>() API is implemented correctly and does not perform any unintended operations.

Note: For more detailed review instructions, see Section " **[SafetyRTE.Review.ExplicitExclusiveAreas.Rte_Enter.Implementation]** ".

[SafetyRTE.Review.ExplicitExclusiveAreas.Rte_Exit.Existence]

Verify that the RTE implements all necessary Rte_Exit_<name>() APIs with the correct signature.

Note: For more detailed review instructions, see Section " **[SafetyRTE.Review.ExplicitExclusiveAreas.Rte_Exit.Existence]** ".

[SafetyRTE.Review.ExplicitExclusiveAreas.Rte_Exit.Implementation]

Verify that each Rte_Exit_<name>() API is implemented correctly and does not perform any unintended operations.

Note: For more detailed review instructions, see Section " **[SafetyRTE.Review.ExplicitExclusiveAreas.Rte_Exit.Implementation]** ".

[SafetyRTE.Review.ExplicitExclusiveAreas.SchM_Enter.Existence]

Verify that the RTE implements all necessary SchM_Enter_<bsnp>[_<vi>_<ai>]_<name>() APIs with the correct signature.

Note: For more detailed review instructions, see Section " **[SafetyRTE.Review.ExplicitExclusiveAreas.SchM_Enter.Existence]** ".

[SafetyRTE.Review.ExplicitExclusiveAreas.SchM_Enter.Implementation]

Verify that each SchM_Enter_<bsnp>[_<vi>_<ai>]_<name>() API is implemented correctly and does not perform any unintended operations.

Note: For more detailed review instructions, see Section " **[SafetyRTE.Review.ExplicitExclusiveAreas.SchM_Enter.Implementation]** ".

[SafetyRTE.Review.ExplicitExclusiveAreas.SchM_Exit.Existence]

Verify that the RTE implements all necessary SchM_Exit_<bsnp>[_<vi>_<ai>]_<name>() APIs with the correct signature.

Note: For more detailed review instructions, see Section " **[SafetyRTE.Review.ExplicitExclusiveAreas.SchM_Exit.Existence]** ".

[SafetyRTE.Review.ExplicitExclusiveAreas.SchM_Exit.Implementation]

Verify that each SchM_Exit_<bsnp>[_<vi>_<ai>]_<name>() API is implemented correctly and does not perform any unintended operations.

Note: For more detailed review instructions, see Section " **[SafetyRTE.Review.ExplicitExclusiveAreas.SchM_Exit.Implementation]** ".

[SafetyRTE.Review.ExplicitExclusiveAreas.TraceHooks]

Verify that the `RTE` implements all hook functions called by `Rte_Enter_<name>()` and `Rte_Exit_<name>()` as `((void)(0))`.

7.2.9. Client/server communication

[SafetyRTE.Review.ClientServer.Rte_Call.Sync.Existence]

Verify that the `RTE` implements no further APIs for client-server communication, e.g. the `Rte_Result_<p>_<o>()` API.

[SafetyRTE.Review.ClientServer.TraceHooks]

Verify that the `RTE` implements all hook functions called by `Rte_Call_<p>_<o>()` as `((void)(0))`.

[SafetyRTE.Review.ClientServer.Rte_Call.Sync.Implementation]

Verify that each synchronous `Rte_Call_<p>_<o>()` API calls a server-runnable entity located in the same partition as the clients.

Appendix A. EB tresos Safety RTE workflow

The following diagram shows the workflow for EB tresos Safety RTE.

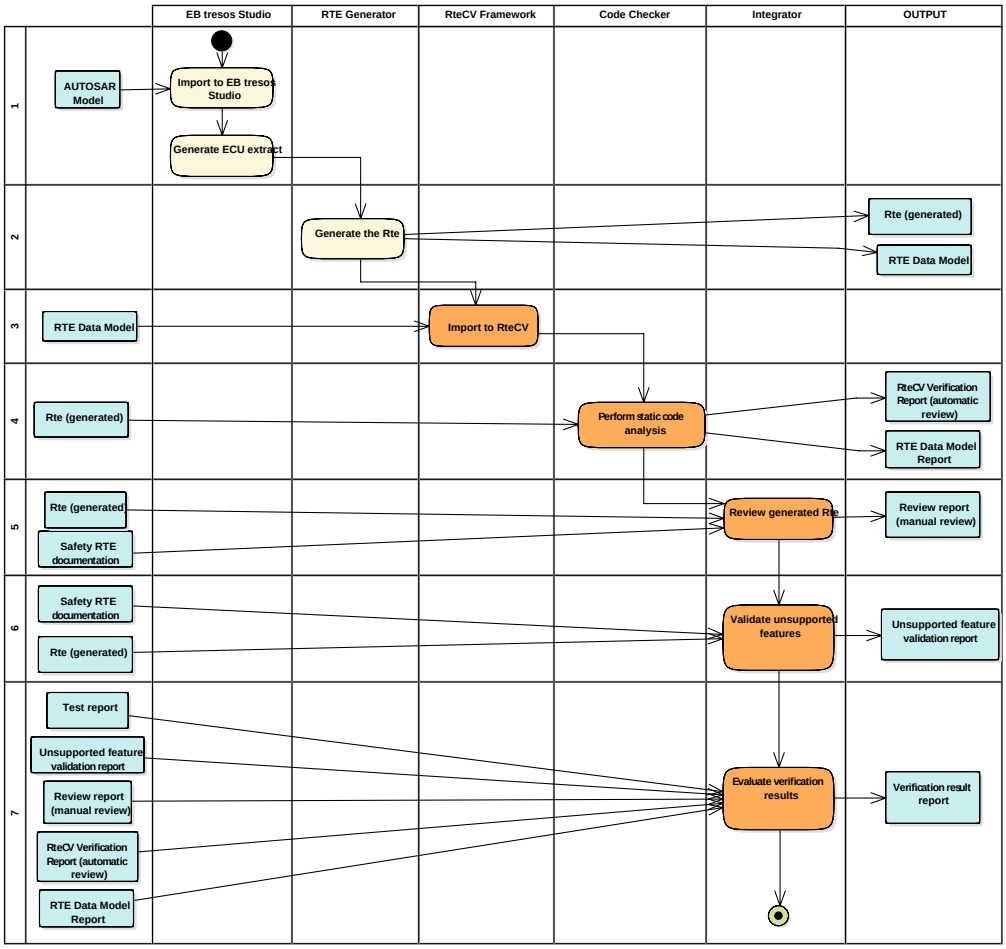

Figure A.1. Workflow of the EB tresos Safety RTE

- **Input "AUTOSAR Model"**: The AUTOSAR model is the sum of all AUTOSAR XML descriptions for a project. This includes the system description, the software component description, the basic software module description and the ECU configuration.
- **Input "Safety RTE Documentation"**: The documentation consists of this safety manual and the [USERSGUIDE].

1. **Workflow step "Import to EB tresos Studio"**: The AUTOSAR model is imported to the `EB tresos Studio` project.

- **Workflow step "Generate ECU Extract"**: The Ecu Extract Creator generates a new ECU Extract based on the selected System and ECU. The ECU Extract is stored within the `EB tresos Studio` project.

2. **Workflow step "Generate the RTE"**: The `RTE` is generated for the imported AUTOSAR model. In addtion, the serialized RTE intermediate model is generated. For more detailed information about how to configure and generate the `RTE`, see [AUTOCOREUSRDOC], chapter *User's guide → Run-Time Environment → Configuring and generating the RTE*.
3. **Workflow step "Import to RteCV"**: The serialized RTE intermediate model is imported into RTE data model of the RteCV. For more detailed information about how to import the serialized RTE intermediate model to the RteCV, see [USERSGUIDE], chapter *Using the RteCV*.
4. **Workflow step "Perform static code analysis"**: The Code Checker performs the static code checks on the generated `RTE` according to Section 5.5.3.1, “Executing static code checks”.
5. **Workflow step "Review generated RTE"**: The integrator performs the review of the generated `RTE` according to Section 5.5.4, “Reviewing the generated RTE code”.
6. **Workflow step "Validate unsupported features"**: The integrator validates unsupported features according to Section 5.5.5, “Validating unsupported features of the RTE”.
7. **Workflow step "Evaluate verification results"**: The integrator evaluates the verification results according to Section 5.5.6, “Interpretation of the validation results”.

Appendix B. Additional review instructions

B.1. Introduction

This appendix contains pseudo code listings. These are based on the C programming language and use the same conditional statements. In addition to that the conventions from the following table apply:

Convention	Item is used	Example
Square brackets, i.e. `[ ]`	For optional statements. For more information on when optional statements are available, see the detailed review instructions.	`[enter critical section]`
Angle brackets, i.e. `< >`	To indicate generic parts of identifiers	`Rte_Write_<p>_<o>()`
Curly brackets, i.e. `{ }`	To indicate alternative parts of identifiers	`Rte_{Runnable_<swc>|Schedulable}_<executable>_Start()`

Table B.1. Pseudo code conventions

B.2. General

B.2.1. Detailed review instructions

[SafetyRTE.Review.General.ErrorCodes]

Verify that the `RTE` error codes for the type `Std_ReturnType` are correctly defined according to Table B.2, "`RTE` error codes for `Std_ReturnType`".

Verify that the `SchM` error codes for the type `Std_ReturnType` are correctly defined according to Table B.3, "`SchM` error codes for `Std_ReturnType`".

Constant	Value
RTE_E_OK	0
RTE_E_INVALID	1
RTE_E_COMMS_ERROR	128
RTE_E_COM_STOPPED	128
RTE_E_TIMEOUT	129
RTE_E_LIMIT	130
RTE_E_NO_DATA	131
RTE_E_TRANSMIT_ACK	132
RTE_E_NEVER_RECEIVED	133
RTE_E_LOST_DATA	64
RTE_E_MAX_AGE_EXCEEDED	64
RTE_E_SHUTDOWN_NOTIFICATION	156
RTE_E_UNCONNECTED	134
RTE_E_UNKNOWN_ERROR	254

Table B.2. `RTE` error codes for `Std_ReturnType`

Constant	Value
SCHM_E_OK	0
SCHM_E_TIMEOUT	129
SCHM_E_LIMIT	130
SCHM_E_NO_DATA	131
SCHM_E_TRANSMIT_ACK	132
SCHM_E_IN_EXCLUSIVE_AREA	135

Table B.3. `SchM` error codes for `Std_ReturnType`

B.3. Lifecycle

B.3.1. Detailed review instructions

[SafetyRTE.Review.Lifecycle.PartitionState.Initialization]

Verify that the RTE initializes each partition state variable Rte_State_<partition> with RTE_PARTITION_UNINITIALIZED.

Verify that the RTE initializes the SchM partition state variable SchM_State_<partition> with SCHM_PARTITION_UNINITIALIZED.

[SafetyRTE.Review.Lifecycle.PartitionStates]

Verify that the partition state constants are correctly defined in each partition according to Table B.4, "RTE partition state constants".

Verify that the SchM state constants are correctly defined in each partition according to Table B.5, "SchM state constants".

Constant	Value
RTE_PARTITION_UNINITIALIZED	0x0U
RTE_PARTITION_ACTIVE	0x1U
RTE_PARTITION_STOPPED	0x2U
RTE_PARTITION_TERMINATED	0x3U
RTE_PARTITION_RESTARTING	0x4U
RTE_PARTITION_STARTING	0x5U

Table B.4. RTE partition state constants

Constant	Value
SCHM_PARTITION_UNINITIALIZED	0xAU
SCHM_PARTITION_ACTIVE	0xBU
SCHM_PARTITION_STOPPED	0xCU

Constant	Value
SCHM_PARTITION_STARTING	0xDU

Table B.5. SchM state constants

B.4. Tasks

B.4.1. Code listings

B.4.1.1. Rte_WaitGetClearEvent_<partition>()

This section shows the different implementations of Rte_WaitGetClearEvent_<partition>(). The placeholders used in the listings are defined as follows:

- <partition>

  The shortName of the /Os/OsApplication for which the Rte_WaitGetClearEvent_<partition>() function is implemented.

OS supports OS_WaitGetClearEvent():

```
#define Rte_WaitGetClearEvent_<partition> (e, ep, tid) (OS_WaitGetClearEvent (e, ep), \
    ( (*(ep) & (EventMaskType)~(EventMaskType)(e) ) != 0U) ? \
    SetEvent (tid, (*(ep) & (EventMaskType)~(EventMaskType)(e))), *(ep) &= (e), E_OK : \
    E_OK)
```

Figure B.1. Implementation of Rte_WaitGetClearEvent_<partition>() with OS support

OS does not support `OS_WaitGetClearEvent()`:

```
StatusType Rte_WaitGetClearEvent_<partition>
(
    EventMaskType eventToWait,
    EventMaskType * eventReceived,
    TaskType myTaskId
)
{
    Rte_Task_WaitEvent (myTaskId, eventToWait);
    (void) WaitEvent (eventToWait);
    Rte_Task_WaitEventRet (myTaskId, eventToWait);
    (void) GetEvent (myTaskId, eventReceived);
    *eventReceived &= eventToWait;
    (void) ClearEvent (*eventReceived);
    return E_OK;
}
```

Figure B.2. Implementation of `Rte_WaitGetClearEvent_<partition>()` without `OS` support

B.4.2. Detailed review instructions

[SafetyRTE.Review.Tasks.Existence]

Verify that the `RTE` does not implement any tasks that are not configured within the Os Ecu configuration (`/Os/OsTask`).

[SafetyRTE.Review.Tasks.ExecutionCounter.Initialization]

Verify that the `RTE` initializes each execution counter variable `Rte_ExecutionCounter_<taskName>` with `0`. The execution counter variable for a specific basic task is only generated if the following conditions are true:

- The task has mapped timing events.
- Not all of the timing events have the same period.

For more details about the implementation of basic tasks, see Section “ **[SafetyRTE.Review.Tasks.Implementation]** ”.

[SafetyRTE.Review.Tasks.Implementation]

Verify that the RTE implements a task correctly according to one of the following sections:

- Follow Section "SafetyRTE.Review.Tasks.BasicTask.Implementation" if the task type is BASIC (configuration parameter: /Os/OsTask/OsTaskType).
- Follow Section "SafetyRTE.Review.Tasks.ExtendedTask.Implementation" if both of the following conditions are true:
  - The task type is EXTENDED (configuration parameter: /Os/OsTask/OsTaskType).
  - The task is not the BSW OS task (configuration parameter: /Rte/RteGeneration/BswConfiguration/BswOsTaskRef).
- Follow Section "SafetyRTE.Review.SenderReceiver.BswOsTask.Implementation" if the task is the BSW OS task (configuration parameter: /Rte/RteGeneration/BswConfiguration/BswOsTaskRef).

SafetyRTE.Review.Tasks.BasicTask.Implementation

(part of Section " **[SafetyRTE.Review.Tasks.Implementation]** ")

Verify that the task dispatch and task end trace hooks are correctly implemented according to Section "SafetyRTE.Review.Tasks.TaskTraceHooks".

NOTE

Timing supervision for basic tasks

If there are safety requirements to the timing of executable entities which are called from a basic task, the integrator must use some kind of timing supervision to ensure that the basic task runs with the correct period. EB tresos Safety RTE does not cover the scheduling and therefore it does not cover task timings.

SafetyRTE.Review.Tasks.ExtendedTask.Implementation

(part of Section " **[SafetyRTE.Review.Tasks.Implementation]** ")

Verify that the RTE implements each extended task and does not perform any additional operations:

1. Verify that the task dispatch and task end trace hooks are correctly implemented according to Section "SafetyRTE.Review.Tasks.TaskTraceHooks".
2. Verify that the task waits for the correct events:
   - The task calls Rte_WaitGetClearEvent_<partition>() to wait for the events.
   - The task provides all OS events which are mapped to that task (configuration parameter: /Os/OsTask/OsTaskEventRef) as event mask to the wait call.
3. Verify that the OS event used for the execution condition of an executable entity is either unique or only shared with other executable entities which have the same period (system model attribute: peri-

od) and offset (configuration parameters: /Rte/RteSwComponentInstance/RteEventToTaskMapping/RteActivationOffset for RTE events, or /Rte/RteBswModuleInstance/RteBswEventToTaskMapping/RteBswActivationOffset for BSW events).

NOTE **Timing supervision for extended tasks**

If there are safety requirements to the timing of executable entities which are called from an extended task, the integrator must use some kind of timing supervision to ensure that the extended task and the called executable entities are running with the correct period. EB tresos Safety RTE does not cover the timing of those executable entities.

SafetyRTE.Review.Tasks.TaskTraceHooks

(part of Section " **[SafetyRTE.Review.Tasks.Implementation]** ")

Verify that the RTE implements the trace hooks correctly:

1. The RTE does not implement Rte_Task_Dispatch() if the trace hook is configured, i.e. an Rte_Task_Dispatch entry exists in the /Rte/RteGeneration/RteVfbTraceFunction list of the configuration. Otherwise the RTE must implement Rte_Task_Dispatch() as ((void)0).
2. The RTE does not implement Rte_Task_EndHook() if the trace hook is configured, i.e. there exists an entry Rte_Task_Endhook in list /Rte/RteGeneration/RteVfbTraceFunction in the configuration. Otherwise the RTE must implement Rte_Task_EndHook() as ((void)0).

SafetyRTE.Review.Tasks.ExecutableInvocation

(part of Section " **[SafetyRTE.Review.Tasks.Implementation]** " and its sub-reviews)

Verify that each executable entity is correctly called:

- The calls to the executable entity trace hooks are defined as ((void)0).
- If an executable entity references exclusive areas (system model reference: runsInsideExclusiveArea), verify that the used resource (if the OS resource API is used) is unique for that exclusive area, i.e. it is only used for that exclusive area and for nothing else. Note that the configuration parameter /Rte/RteBswModuleInstance/RteBswExclusiveAreaImpl/RteBswExclusiveAreaOsResourceRef is ignored so you may not use it to verify the resource name.

[SafetyRTE.Review.Tasks.EventDeclaration]

Verify that the RTE declares each OS event that is used in an extended task with the OS DeclareEvent() macro, i.e. each used event shall be declared as DeclareEvent(<event>) where <event> is the shortName of the event referenced by the configuration parameter /Os/OsTask/OsTaskEventRef.

[SafetyRTE.Review.Tasks.Rte_WaitGetClearEvent.Implementation]

Verify that the RTE implements Rte_WaitGetClearEvent_<partition>() as defined in Figure B.1, "Implementation of Rte_WaitGetClearEvent_<partition>() with OS support" if the C-macro OS_HAS_WAITGETCLEAREVENT is defined.

Verify that the RTE implements Rte_WaitGetClearEvent_<partition>() as defined in Figure B.2, "Implementation of Rte_WaitGetClearEvent_<partition>() without OS support" if the C-macro OS_HAS_WAITGETCLEAREVENT is not defined.

B.5. Partitioning

B.5.1. Detailed review instructions

[SafetyRTE.Review.Partitioning.SeparatedPartitions]

Verify that the RTE generates the following files, implementing the partitioning functionality:

- Rte_Partitioning.h

  See Section "SafetyRTE.Review.Partitioning.RtePartition.Header.Implementation" for more detailed review instructions.

- Rte_Main.h

  See Section "SafetyRTE.Review.Partitioning.RteMain.Header.Implementation" for more detailed review instructions.

- Rte_Main_<partition>.h

  See Section "SafetyRTE.Review.Partitioning.RteMainPartition.Header.Implementation" for more detailed review instructions.

- Rte_Main.c

  See Section "SafetyRTE.Review.Partitioning.RteMain.Source.Implementation" for more detailed review instructions.

- Rte_Intern_<partition>.h

  See Section "SafetyRTE.Review.Partitioning.RteIntern.Header.Implementation" for more detailed review instructions.

- `Rte_<partition>.c`

  See Section "SafetyRTE.Review.Partitioning.RtePartition.Source.Implementation" for more detailed review instructions.

- `Rte_Components_Shared.h`

  See Section "SafetyRTE.Review.Partitioning.RteComponentsShared.Header.Implementation" for more detailed review instructions.

Verify that the `RTE` generates the following files implementing the SMC functionality, which is used for the inter-partition communication:

- `Rte_Smc.h`

  See Section "SafetyRTE.Review.Partitioning.RteSmc.Header.Implementation" for more detailed review instructions.

- `Rte_Smc_<partition>.h`

  See Section "SafetyRTE.Review.Partitioning.RteSmcData.Header.Implementation" for more detailed review instructions.

- `Rte_Smc_Data_<partition>.c`

  See Section "SafetyRTE.Review.Partitioning.RteSmcData.Source.Implementation" for more detailed review instructions.

Verify that the `RTE` does not generate any additional files which implement the partitioning functionality.

SafetyRTE.Review.Partitioning.RtePartition.Header.Implementation

(part of Section " **[SafetyRTE.Review.Partitioning.SeparatedPartitions]** ")

Verify that the `RTE` generates the header file `Rte_Partitioning.h`, if both following conditions are true:

- More than one partition is configured (configuration parameter: `/Os/OsApplication`).
- The parameter `/Rte/RteGeneration/InterPartitionCommunication` is not set to `Disabled`.

  Note: The configuration parameter `/Rte/RteGeneration/InterPartitionCommunication` shall always be set to `SharedMemory` according to **[SafetyRTE.Assumption.Partitioning]**.

SafetyRTE.Review.Partitioning.RteMain.Header.Implementation

(part of Section " **[SafetyRTE.Review.Partitioning.SeparatedPartitions]** ")

Verify that the `RTE` generates a lifecycle header file `Rte_Main.h`.

SafetyRTE.Review.Partitioning.RteMainPartition.Header.Implementation

(part of Section " **[SafetyRTE.Review.Partitioning.SeparatedPartitions]** ")

Verify that the `RTE` generates a partition-specific lifecycle header file `Rte_Main_<partition>.h` for each configured partition (configuration parameter: `/Os/OsApplication`) which uses an `RTE` feature.

SafetyRTE.Review.Partitioning.RteMain.Source.Implementation

(part of Section " **[SafetyRTE.Review.Partitioning.SeparatedPartitions]** ")

Verify that the `RTE` generates a lifecycle source file `Rte_Main.c`.

SafetyRTE.Review.Partitioning.RteIntern.Header.Implementation

(part of Section " **[SafetyRTE.Review.Partitioning.SeparatedPartitions]** ")

Verify that the `RTE` generates a partition-specific internal header file `Rte_Intern_<partition>.h` for each configured partition (configuration parameter: `/Os/OsApplication`) which uses an `RTE` feature.

SafetyRTE.Review.Partitioning.RtePartition.Source.Implementation

(part of Section " **[SafetyRTE.Review.Partitioning.SeparatedPartitions]** ")

Verify that the `RTE` generates a partition-specific source file `Rte_<partition>.c` for each configured partition (configuration parameter: `/Os/OsApplication`) which uses an `RTE` feature.

SafetyRTE.Review.Partitioning.RteComponentsShared.Header.Implementation

(part of Section " **[SafetyRTE.Review.Partitioning.SeparatedPartitions]** ")

Verify that the `RTE` generates the header file `Rte_Components_Shared.h`.

SafetyRTE.Review.Partitioning.RteSmc.Header.Implementation

(part of Section " **[SafetyRTE.Review.Partitioning.SeparatedPartitions]** ")

Verify that the `RTE` generates the SMC header file `Rte_Smc.h`, if both of the following conditions are true:

- More than one partition is configured (configuration parameter: `/Os/OsApplication`).
- The parameter `/Rte/RteGeneration/InterPartitionCommunication` is set to `SharedMemory`.

SafetyRTE.Review.Partitioning.RteSmcData.Header.Implementation

(part of Section " **[SafetyRTE.Review.Partitioning.SeparatedPartitions]** ")

Verify that the `RTE` generates a header file `Rte_Smc_<partition>.h` for one or a set of partitions which shall have write access to their assigned shared buffers used by the SMC, if both of the following conditions are true:

- More than one partition is configured (configuration parameter: `/Os/OsApplication`).
- The parameter `/Rte/RteGeneration/InterPartitionCommunication` is set to `SharedMemory`.

SafetyRTE.Review.Partitioning.RteSmcData.Source.Implementation

(part of Section " **[SafetyRTE.Review.Partitioning.SeparatedPartitions]** ")

Verify that the `RTE` generates a source file `Rte_Smc_Data_<partition>.c` for one or a set of partitions which shall have write access to their assigned shared buffers used by the SMC, if both of the following conditions are true:

- More than one partition is configured (configuration parameter: `/Os/OsApplication`).
- The parameter `/Rte/RteGeneration/InterPartitionCommunication` is set to `SharedMemory`.

[SafetyRTE.Review.Partitioning.SuperfluousFiles]

Verify that the `RTE` does not generate additional files beyond those described in Section " **[SafetyRTE.Review.Partitioning.SeparatedPartitions]** " and the following:

- `Rte.h`
- `Rte_Cbk.h`
- `Rte_Cfg.h`
- `Rte_Hook.h`
- `Rte_MemMap.h`
- `Rte_NvMData.h`
- `Rte_Type.h`
- `Rte_<swc>.h`
- `Rte_<swc>_Hook.h`
- `Rte_<swc>_Type.h`
- `SchM_<bsnp>.h`
- `SchM_<bsnp>Hook.h`
- `SchM_<bsnp>Type.h`

<bsnp> is the *Basic Software Scheduler Name Prefix*, which is either the symbol attribute of the BswSchedulerNamePrefix of the BswSchedulableEntity or the shortName of the enclosing BswModuleDescription if no BswSchedulerNamePrefix is defined.

B.6. Sender/receiver communication

B.6.1. Detailed review instructions

[SafetyRTE.Review.SenderReceiver.Buffer.DataType]

- Verify for each sender/receiver connection that the receive buffer has the data type which is referenced by the VariableDataPrototype which is referenced by the VariableAccess of that sender/receiver connection.
- Verify for each implicit sender/receiver connection that the data handle buffers have the Rte_DE_<idt> data type, where <idt> is the data type referenced by the VariableDataPrototype which is referenced by the VariableAccess of that sender/receiver connection.

[SafetyRTE.Review.SenderReceiver.Buffer.InitialValue]

- Verify that all receive buffers of all sender/receiver connections are initialized in the Rte_Common_Init_<partition>() function to:
  - The initial value that is defined in the communication specification in the AUTOSAR model (i.e. NonqueuedSenderComSpec or NonqueuedReceiverComSpec), or
  - 0, if no initial value is defined.
- Verify that all shared data handle buffers of all implicit sender/receiver connections are initialized in the Rte_Common_Init_<partition>() function to the initial value that is defined in the communication specification in the AUTOSAR model (i.e. NonqueuedSenderComSpec or NonqueuedReceiverComSpec), if such an initial value is defined.

  A data handle buffer is shared, if:
  - The sender/receiver connection is intra-core, and
  - the involved sender and receiver tasks cannot interrupt each other, i.e. the tasks have the same priority or are configured as non-preemptive.

[SafetyRTE.Review.SenderReceiver.SuperfluousElements]

Verify that all of the variables named with prefix `Rte_Smc_` are actually used by at least one of the following: `RTE` API, task, or callback function.

Verify that there are no unused receive buffers.

Verify that there is no internal function or macro which accesses a variable with prefix `Rte_Smc_` beyond these listed below:

- API and internal functions and macros used for sender/receiver connections

SafetyRTE.Review.SenderReceiver.BswOsTask.Implementation

(part of Section " **[SafetyRTE.Review.Tasks.Implementation]** ")

- Verify that the task dispatch and task end trace hooks are correctly implemented according to Section "SafetyRTE.Review.Tasks.TaskTraceHooks" and that any additional trace hook function called by BSW `OS` task is implemented with `((void)(0))`.

[SafetyRTE.Review.SenderReceiver.Smc.ReceiveBuffer.InitialValue]

Verify that for each queue for `Com` signals and signal groups, the head and tail variables are both initialized with `0`.

[SafetyRTE.Review.SenderReceiver.DataHandleBuffer.InitialValue]

- Verify that all shared data handle buffers of all implicit sender/receiver connections are initialized in the `Rte_Common_Init_<partition>()` function to the initial value that is defined in the communication specification in the AUTOSAR model (i.e. `NonqueuedSenderComSpec` or `NonqueuedReceiverComSpec`), if such an initial value is defined.

  A data handle buffer is shared, if:

  - The sender/receiver connection is intra-core, and
  - the involved sender and receiver tasks cannot interrupt each other, i.e. the tasks have the same priority or are configured as non-preemptive.

[SafetyRTE.Review.SenderReceiver.ComSignalGroupLength]

Verify that the length of each Com signal group that has a data transformation chain configured, is equal to or less than the length of the `Rte_InternalBuffer` generated by the `RTE` in function `Rte_COMCbk_<sng>()`.

[SafetyRTE.Review.SenderReceiver.SpinLockResource]

If the RTE configuration parameter /Rte/RteGeneration/SpinlockAllocationStrategy is set to OnePerChannel then verify that the following condition is true:

- The connection is intra-ECU

  AND

- the same spinlock identifier is used for the Rte_Read() and Rte_Write() functions of a sender/receiver connection

  AND

- the spinlock identifier is not used anywhere else

  OR

- The connection is inter-ECU

  AND

- the same spinlock identifier is used for the Rte_Read() and the Rte_COMCbk() functions for a signal

  AND

- the same spinlock identifier is used for the Rte_Write() function and in the according signal case in the BswOsTask function

  AND

- the spinlock identifier is not used anywhere else

B.7. Inter-runnable variables

B.7.1. Detailed review instructions

[SafetyRTE.Review.InterRunnableVariables.SharedBuffer]

Verify that for each connection via an inter-runnable variable all following conditions are true:

- The Rte_IrvWrite_<re>_<o>() and Rte_IrvRead_<re>_<o>() API use the same shared buffer.
- The shared buffer is unique for that connection, i.e. is not shared with APIs of any other connection.

- The data type of the shared buffer is referenced by the `VariableDataPrototype` which is referenced by the `VariableAccess` of that connection.

[SafetyRTE.Review.InterRunnableVariables.SharedBuffer.InitialValue]

For each shared buffer of a connection via an inter-runnable variable apply one of the following initialization rules:

- If an initial value is defined in the `VariableDataPrototype` of the referred `explicitInterRunnableVariable`, then verify that the inter-runnable variable is initialized in the `Rte_Common_Init_<partition>()` function to the defined initial value.
- If no initial value is defined and the inter-runnable variable has a primitive data type, then verify that it is initialized to `0`.
- If no initial value is defined and the inter-runnable variable has a complex data type, then verify that each contained `ImplementationDataTypeElement` with a primitive data type is initialized to `0`.

[SafetyRTE.Review.InterRunnableVariables.SuperfluousElements]

Verify that all following conditions are true:

- The variables named with the `Rte_Irv_` prefix are actually used by either `Rte_IrvWrite_<re>_<o>()` or `Rte_IrvRead_<re>_<o>()` API.
- There are no unused shared buffers.
- There is no internal function or macro that accesses a variable with the `Rte_Irv_` prefix beyond the `RTE` APIs, internal functions, and macros used for that connection via an inter-runnable variable.

B.8. Exclusive areas

B.8.1. Detailed review instructions

[SafetyRTE.Review.ExplicitExclusiveAreas.Rte_Enter.Existence]

Verify that the `RTE` does not implement any additional `Rte_Enter_<name>()` API.

[SafetyRTE.Review.ExplicitExclusiveAreas.Rte_Enter.Implementation]

If the `GetResource() OS` service is used for this exclusive area API, verify that all of the following conditions are true:

- The same OsResource identifier is used for the matching `ReleaseResource()` call in the `Rte_Exit_<name>()` API.
- The OsResource identifier is not used by another `GetResource()` function.

[SafetyRTE.Review.ExplicitExclusiveAreas.Rte_Exit.Existence]

Verify that the `RTE` does not implement any additional `Rte_Exit_<name>()` API.

[SafetyRTE.Review.ExplicitExclusiveAreas.Rte_Exit.Implementation]

If the `GetResource() OS` service is used for this exclusive area API, verify that all of the following conditions are true:

- The same OsResource identifier is used for the matching `GetResource()` call in the `Rte_Enter_<name>()` API.
- The OsResource identifier is not used by another `ReleaseResource()` function.

[SafetyRTE.Review.ExplicitExclusiveAreas.SchM_Enter.Existence]

Verify that the `RTE` does not implement any additional `SchM_Enter_<name>()` API.

[SafetyRTE.Review.ExplicitExclusiveAreas.SchM_Enter.Implementation]

If `OS_RESOURCE` is configured as the `RteExclusiveAreaImplMechanism`, verify that all following conditions are true:

- The same OsResource identifier is used for the matching `SchM_Exit_<bsnp>[_<vi>_<ai>]_<name>()` function.
- The OsResource identifier is not used by another `SchM_Enter_<bsnp>[_<vi>_<ai>]_<name>()` function.

If `OS_SPINLOCK` is configured as the `RteExclusiveAreaImplMechanism`, verify that all following conditions are true:

- The same spinlock identifier is used for the matching `SchM_Exit_<bsnp>[_<vi>_<ai>]_<name>()` function.
- The spinlock identifier is not used by another `SchM_Enter_<bsnp>[_<vi>_<ai>]_<name>()` function.

[SafetyRTE.Review.ExplicitExclusiveAreas.SchM_Exit.Existence]

Verify that the `RTE` does not implement any additional `SchM_Exit_<name>()` API.

[SafetyRTE.Review.ExplicitExclusiveAreas.SchM_Exit.Implementation]

If `OS_RESOURCE` is configured as the `RteExclusiveAreaImplMechanism`, verify that all following conditions are true:

- The same `OsResource` identifier is used for the matching `SchM_Enter_<bsnp>[_<vi>_<ai>]_<name>()` function.
- The `OsResource` identifier is not used by another `SchM_Exit_<bsnp>[_<vi>_<ai>]_<name>()` function.

If `OS_SPINLOCK` is configured as the `RteExclusiveAreaImplMechanism`, verify that all following conditions are true:

- The same spinlock identifier is used for the matching `SchM_Enter_<bsnp>[_<vi>_<ai>]_<name>()` function.
- The spinlock identifier is not used by another `SchM_Exit_<bsnp>[_<vi>_<ai>]_<name>()` function.

Exclusive area implementation mechanism

Verify that the used resource (if the `OS` resource API is used) is unique for that exclusive area, i.e. it is only used for that exclusive area and for nothing else. Please note that the configuration parameters `/Rte/RteBswModuleInstance/RteBswExclusiveAreaImpl/RteBswExclusiveAreaOsResourceRef` and `/Rte/RteSwComponentInstance/RteExclusiveAreaImplMechanism/RteExclusiveAreaOsResourceRef` are ignored so you may not use them to verify the resource name.

Verify that within the Os ECU configuration, the OsTasks that activate the runnables that run inside or can enter the exclusive area reference the `OsResource`.

`RteExclusiveAreaImplMechanism` [a]	**Each executable entity**[b] **is assigned to a task**[c]	**The tasks**[d] **can interrupt each other**	**Exclusive Area APIs**
`NO_LOCK`	no	-	none

RteExclusiveAreaImplMechanism [a]	**Each executable entity**[b] **is assigned to a task**[c]	**The tasks**[d] **can interrupt each other**	**Exclusive Area APIs**
	yes	no	none
	yes	yes	`GetResource()`, `ReleaseResource()`
`OS_RESOURCE`	no	-	`SuspendAllInterrupts()`, `ResumeAllInterrupts()`
	yes	no	none
	yes	yes	`GetResource()`, `ReleaseResource()`
`OS_INTERRUPT_BLOCKING`	no	-	`SuspendOSInterrupts()`, `ResumeOSInterrupts()`
	yes	no	none
	yes	yes	`SuspendOSInterrupts()`, `ResumeOSInterrupts()`
`ALL_INTERRUPT_BLOCKING`	no	-	`SuspendAllInterrupts()`, `ResumeAllInterrupts()`
	yes	no	none
	yes	yes	`SuspendAllInterrupts()`, `ResumeAllInterrupts()`
`UNCONFIGURED` [e]	no	-	`SuspendAllInterrupts()`, `ResumeAllInterrupts()`
	yes	no	`SuspendAllInterrupts()`, `ResumeAllInterrupts()`
	yes	yes	`SuspendAllInterrupts()`, `ResumeAllInterrupts()`

[a] This parameter is restricted to the values `ALL_INTERRUPT_BLOCKING`, `OS_INTERRUPT_BLOCKING`, `OS_RESOURCE` and `NO_LOCK` in the current feature set.

[b] This refers to all executable entities which reference the corresponding exclusive area via a `runsInsideExclusiveArea` reference.

[c] An executable entity is assigned to a task if there is an `RteEventToTaskMapping` or a `RteBswEventToTaskMapping` for the `RTE`/BSW event which triggers that executable entity, for more information, see Section “SafetyRTE.Review.Tasks.BasicTask.Implementation”. An executable entity is implicitly mapped to a task if it is called by a client via direct function call.

[d] This refers to the tasks from the previous table column, i.e. all the tasks, to which the executable entities are mapped, which reference the corresponding exclusive area. **yes** means, that there is at least one task among all those tasks, which can interrupt another task from that group of tasks. **-** means that this column has no influence on the used API in this case.

[e] The `RteExclusiveAreaImplMechanism` parameter is considered `UNCONFIGURED`, if there is no `RteExclusiveAreaImplementation` container which references the corresponding exclusive area via a `RteExclusiveAreaRef` reference.

Table B.6. Exclusive area APIs

Appendix C. Document configuration information

This document was created by the DocBook engine using the source files and revisions listed below. All paths are relative to the directory https://subversion.ebgroup.elektrobit.com/svn/autosar/asc_RteCV/trunk/doc/public/safety_manual.

Filename	Revision / Hash
../fragments/assumptions/Assumptions.xml	13938
../fragments/assumptions/figures/scheme_assumption.svg	5968
../fragments/bibliography/Bibliography.xml	13947
../fragments/glossary/Glossary.xml	13938
../fragments/top_level_requirements/Definitions.xml	13938
../fragments/top_level_requirements/Level1_Requirements.xml	13938
About_the_Safety_RTE.xml	13938
appendix/Appendix_ExclusiveAreas.xml	11659
appendix/Appendix_General.xml	3302
appendix/Appendix_InterRunnableVariables.xml	10374
appendix/Appendix_Lifecycle.xml	5396
appendix/Appendix_Partitioning.xml	13597
appendix/Appendix_SenderReceiver.xml	13829
appendix/Appendix_Tasks.xml	13553
appendix/Appendix_Workflow.xml	13855
appendix/Index.xml	5029
constraints/Index.xml	11543
constraints/RteFeatureSetLimitations.xml	13899
constraints/SafetyElementOutOfContext.xml	3328
Document_information.xml	5943
History.xml	13938
ManualTestsAndReviews/Index.xml	5968
ManualTestsAndReviews/LimitationChecks.xml	13899
ManualTestsAndReviews/MTnR_ClientServer.xml	8383
ManualTestsAndReviews/MTnR_ExclusiveAreaAPI.xml	9528

Filename	Revision / Hash
ManualTestsAndReviews/MTnR_General.xml	12004
ManualTestsAndReviews/MTnR_InterRunnableVariables.xml	10225
ManualTestsAndReviews/MTnR_LifecycleAPI.xml	13829
ManualTestsAndReviews/MTnR_Partitioning.xml	12894
ManualTestsAndReviews/MTnR_SenderReceiver.xml	13829
ManualTestsAndReviews/MTnR_Tasks.xml	13387
Safety_RTE_safety_manual.xml	10822
UsingSafetyRteSafely/GettingStarted.xml	5968
UsingSafetyRteSafely/Main.xml	13938
UsingSafetyRteSafely/ValidatingRte/InterpretationValidationResults.xml	13855
UsingSafetyRteSafely/ValidatingRte/ReviewGeneratedRteCode.xml	5968
UsingSafetyRteSafely/ValidatingRte/ValidateRteDataModelReport.xml	11055
UsingSafetyRteSafely/ValidatingRte/ValidateUnsupportedRteFeatures.xml	13224

Glossary

EB tresos AutoCore Generic RTE	EB tresos AutoCore Generic RTE is the AUTOSAR RTE implementation. It provides the configuration interface and the code generator to generate the RTE code. EB tresos AutoCore Generic RTE generates the RTE target implementation based on a predefined input model. See *Runtime Environment (RTE)*.
Application software	The application software is the software implemented within the AUTOSAR application layer. Among others this includes the *AUTOSAR application software components*.
AUTOSAR model	An AUTOSAR model is an M1 AUTOSAR model, compare [ASRGENST], chapter 2.2. It can be e.g. an *AUTOSAR system description*, a *software component* description, or a *BSW module* configuration.
Basic software module (BSWM)	A basic software module is an AUTOSAR basic software module.
BSW event	A BSW event is an AUTOSAR `BswEvent`. BSW events are defined in the *AUTOSAR model*.
BSW partition	A BSW partition is a *partition* to which *basic software modules* are assigned.
BSW Scheduler Name Prefix	A prefix to be used in names of generated code artifacts which make up the interface of a BSW module to the BSW Scheduler (SchM).
EB tresos Safety RTE	The EB tresos Safety RTE is the implementation of an AUTOSAR *Runtime Environment*. It consists the *EB tresos AutoCore Generic RTE* to generate the RTE code. It consists also the *RteCV* with the safety manual to verify the generated RTE code.
Electronic control unit (ECU)	An electronic control unit is an AUTOSAR ECU (compare [ASRGLOSSARY]).
Executable entity	An executable entity is either a *runnable entity* or an AUTOSAR *BSW schedulable entity*. In contrast to an AUTOSAR executable entity this term does e.g. not cover AUTOSAR BSW interrupt entities.
Global buffer	A RTE global buffer is a C variable with external or internal linkage that is used to store and retrieve data between RTE API function calls. The name of a RTE global buffer always starts with the `Rte_` prefix.
Integrator	An integrator is a person who integrates the software on the ECU.

Inter-ECU communication	Inter-ECU communication refers to the communication between different ECUs, typically using the AUTOSAR Com module.
Logical program flow monitoring	Logical program flow monitoring is defined as mechanism to detect a wrong order of the executed program sequence or not completely executed program sequence.
Model object	A model object is an object or element of an *AUTOSAR model*. This can e.g. be a configuration parameter or an internal behavior of a *software component*.
OS-application	An OS-application is a group of AUTOSAR OS objects like tasks or interrupt service routines. It is configured in the OS configuration in container `/Os/OsApplication`.
Partition	A partition is the RTE term to represent an *AUTOSAR OS application*.
Runtime Environment (RTE)	The Runtime Environment (RTE) or also AUTOSAR Runtime Environment is an AUTOSAR module. It is specified by [ASRRTE422].
RTE data model	The RTE data model is a model-to-model transformation of the *RTE intermediate model*. It provides a simplified view of the RTE intermediate model that does not contain any information directly related to the generation of RTE artifacts (e.g. buffers, typdefs, etc.). The RTE intermediate model is exported into an XML file by the RTE Generator. This file is then imported into the RteCV tool, where it is deserialized and transformed into the RTE data model representation. As a result, the RTE data model is never accessed in the same execution context by the RTE Generator and RteCV tool.
RTE data model report	The RTE data model report presents the *RTE data model* in order to enable the user of the RteCV to recognize if an error is present in the RTE data model. As a general rule, the RTE data model presents this information from the perspective of the *software component*s.
RTE event	An RTE event is an AUTOSAR `RTEEvent`. RTE events are defined in the *AUTOSAR model*.
RTE intermediate model	The RTE intermediate model is a model-to-model transformation of the *AUTOSAR model*. It provides a simplified view of the AUTOSAR model that replaces artifacts such as assembly-connectors with direct references. In addition, new elements are created that represent implementation dependent and independent concepts that are the basis for code generation (e.g. API functions, buffers, typdefs, inter-partition communication (IP) channels, Os tasks, etc.).

Rte Code Verifier (RteCV)	The Rte Code Verifier (RteCV) is a development tool to verify the generated code from *EB tresos AutoCore Generic RTE*. In this way the confidence in the generated code is increased. The RteCV performs static code checks on the generated RTE code to verify that the generated code correctly implements the provided model.
Runnable entity (RE)	A runnable entity (RE) is an AUTOSAR runnable entity.
BSW schedulable entity	A BSW schedulable entity is a C-function of a *BSW module* which runs in the context of an OS task and is scheduled via an OS service, e.g. via a schedule table or an OS event.
Serialized RTE intermediate model	The serialized RTE intermediate model is a model-to-text transformation of the *RTE intermediate model*. The RTE intermediate model is automatically transformed to XML using the JAXB library.
Software component (SWC)	A software component (SWC) is an AUTOSAR software component.
System application	A system application is the software implementing the functionality of an *ECU*. In context of the EB tresos Safety RTE this includes the *application software*, the *AUTOSAR RTE*, the AUTOSAR basic software and the AUTOSAR OS.
System description	A system description is an AUTOSAR system description.
Temporal program flow monitoring	Temporal program flow monitoring is defined as mechanism to detect a violation of timing constraints between two checkpoints.
Transformer	A transformer is an AUTOSAR *BSW module*. It is specified by [ASRGENTRANSFORM422]. The transformers are used by the *AUTOSAR RTE* to serialize data (e.g. bring structured data into a linear form) and to transform them into a specified format (e.g. extend linear data with a checksum). There are several classes of transformers with different functionality. It is possible to connect transformers to a chain and the AUTOSAR RTE executes this transformer chain in the specified order to transform the data.

Bibliography

[ASRGENST] *Generic Structure Template*, Version 3.2.0 Final, 2011-10-31

[ASRGEN-TRANSFORM422] *General Specification of Transformers*, Version 4.2.2 Final

[ASRGLOSSARY] *Glossary*, Version 4.2.2 Final

[ASROS403] *Specification of Operating System*, Version 4.0.3 Final

[ASRRTE403] *Specification of RTE*, Version 4.0.3 Final, 2011-10-26

[ASRRTE422] *Specification of RTE*, Version 4.2.2 Final

[ASRSWCTR2011] *Software Component Template*, Version R20-11 published, 2020-11-30

[ASRSYST] *System Template*, Version 4.2.0 Final, 2011-11-08

[ASRWEB] *AUTOSAR website*, 2015-03-13
http://www.autosar.org

[AUTOCOR-ERTEDOC] *EB tresos® AutoCore Generic 8 RTE documentation*, product release 8.8.7

[AUTO-COREUSRDOC] *EB tresos® AutoCore Generic documentation*, product release 7.7

[ISO26262_1ST] *INTERNATIONAL STANDARD ISO 26262*: Road vehicles - Functional safety, 2011

[ISO26262-10_1ST] *INTERNATIONAL STANDARD ISO 26262-10*: Road vehicles - Functional safety - Part 10: Guideline on ISO 26262, 2012

[QSSAFERTE] *Quality Statement*

[RELNOTES] *EB tresos Safety RTE release notes*
Safety_RTE_release_notes.pdf

[RMP] *Requirement Management Plan*: EB tresos Safety RTE

[USERSGUIDE] *EB tresos Safety RTE User's Guide*
Safety_RTE_documentation.pdf