VECTOR

Safety Manual

CBD1700369 D04

ID	SPECDOC36099
Status	Not Released
Generated on	2018-01-30 06:55

Contents

Safety Manual

Version	Date	Author	Remarks
1.00.00	2015-11-13	Jonas Wolf	Initial creation
1.01.00	2015-12-18	Jonas Wolf	Information about ASIL added.
1.02.00	2016-01-29	Jonas Wolf	Improvements in formulation.
1.03.00	2016-02-25	Jonas Wolf	Improvements in formulation.
1.03.01	2016-03-30	Jonas Wolf	Review findings incorporated.
1.03.02	2016-05-13	Jonas Wolf	Added hint on hardware-software integration (SMI-4).
1.03.03	2016-07-20	Hartmut Hoerner	Added SMI related to interrupt handling.
1.04.00	2016-09-02	Jonas Wolf	Added TSR-101876 for data consistency.
1.04.01	2016-09-19	Jonas Wolf	Clarifications on SMI-100 and SMI-19.
1.04.02	2016-12-02	Jonas Wolf	Version of referenced document fixed.
1.05.00	2017-02-24	Jonas Wolf	Modified SMI-18: checks need to be enabled per component as well.
1.06.00	2017-05-05	Jonas Wolf	Added SMI-36039: Pointer validity passed to call-outs. Added SMI-36041: Information on Beta-ESCANs. Update of versions of referenced documents. Section 1.1.5 clarified.
1.07.00	2017-06-20	Jonas Wolf	Update of TSRs from TSC (update is backward compatible) Added info on this version history. Fixes of typos.
1.07.01	2017-08-18	Jonas Wolf	Added information on TCL for MSSV and RTE Analyzer.

1 General Part

1.1 Introduction

1.1.1 Purpose

This document describes the assumptions made by Vector during the development of MICROSAR Safe as Software Safety Element out of Context (SEooC). This document provides information on how to integrate MICROSAR Safe into a safety-related project.

This document is intended for the user of MICROSAR Safe. It shall be read by project managers, safety managers, and engineers to allow proper integration of MICROSAR Safe.

1.1.2 Scope

This document adds additional information to the components that are marked with an ASIL in the delivery description provided by Vector. Neither QM Vector components, nor components by other vendors are in the scope of this document.

Vector assumes that hardware and compiler manuals are correct and complete.
Vector uses the hardware reference manuals and compiler manuals for the development of MICROSAR Safe. Vector has no means to verify correctness or completeness of the hardware and compiler manuals.
Example information that may be critical from these manuals is the register assignment by compiler. This information is used to built up the context that is saved and restored by the operating system.
The compiler manual from the compiler version specified for the project is considered. The considered hardware manuals are documented in the Technical Reference of the hardware-specific component.

A general description of Vector's approach to ISO 26262 is described in [2]. This document is available on request.

1.1.3 Definitions

The words *shall*, *shall not*, *should*, *can* in this document are to be interpreted as described here:

Shall means that the definition is an absolute requirement of the specification.

Shall not means that the definition is an absolute prohibition of the specification.

Should means that there may exist valid reasons in particular circumstances to ignore a particular definition, but the full implications must be understood and carefully weighed before choosing a different course.

Can means that a definition is truly optional.

Each requirement in this specification has a unique identifier beginning with SMI. This identifier is semantically the same even for different Safety Manuals provided by Vector.

The user of MICROSAR Safe can deviate from all constraints and requirements in this Safety Manual in the responsibility of the user of MICROSAR Safe, if equivalent measures are used.
If a measure is equivalent can be decided in the responsibility of the user of MICROSAR Safe.

1.1.4 References

No.	Source	Title	Version
[1]	ISO	ISO 26262 Road vehicles — Functional safety (all parts)	2011/2012
[2]	Vector	ISO 26262 Compliance Document	1.3.2
[3]	Vector	MICROSAR Safe Product Information	As provided with the quote
[4]	Vector	MICROSAR Safe Silence Verifier Technical Reference	As provided with the delivery

1.1.5 Overview

This document is automatically generated for a delivery. The content of this document depends on the components (incl. their version) and used hardware (e.g. microcontroller or external hardware) in the delivery. This document is thus valid only for the delivery from Vector that it is included in. The version history above only reflects changes in the component independent part of the Safety Manual.

The structure of this document comprises:

- a general section that covers all assumptions and constraints that are always applicable, and
- at least one section for each ASIL component that covers its constraints and necessary verification steps. The section for a component is identified by its module abbreviation.

Vector's assumptions on the environment of the MICROSAR Safe components as well as the integration process are described.

Vector developed MICROSAR Safe as Safety Element out of Context for projects demanding ASIL D software. All requirements in this document apply independently from the actual highest ASIL of the project (unless otherwise stated).

1.2 Concept

MICROSAR Safe comprises a set of components developed according to ISO 26262. These components can be combined - together with other measures - to build a safe system according to ISO 26262.

Please read the Product Information MICROSAR Safe [3] first.

1.2.1 Safety Concept

It is assumed that the user of MICROSAR Safe is responsible for overall system safety. MICROSAR Safe can provide parts of safety mechanisms to its user. It is also assumed that the user of MICROSAR Safe verifies the robustness and effectiveness of the safety mechanisms within its system based on the configuration of MICROSAR Safe.

1.2.1.1 Technical Solution

Possible hardware faults (transient or permanent) are detected and handled by the application (user of MICROSAR Safe) according to the required ASIL necessary for the project. This also implies protection against Single Event Upsets (SEUs) in volatile and non-volatile memory, as well as faults in registers and processing logic.
The hardware manual is assumed to be complete and correct.

MICROSAR Safe does not implement mechanisms to mitigate random hardware faults. Instead, MICROSAR Safe relies on hardware mechanisms, such as ECC memory and lock-step cores.

Note that multi-core controllers with cores that provide different levels of diagnostic coverage are considered during development of the MICROSAR Safe operating system. However, they also require a detailed analysis of core interdependency by the user of MICROSAR Safe.

It is assumed that systematic hardware faults are handled on system or application level by the user of MICROSAR Safe.

Systematic faults in software are prevented by implementation of an ISO 26262-compliant development process. It is assumed that development according to an ISO 26262-compliant development process implicitly ensures freedom from interference with respect to memory.
MICROSAR Safe does not implement redundant (inverted) data storage to mitigate software faults.

It is assumed that software with a different quality level is separated using a memory protection unit (MPU) (see also TSR-7). The user of MICROSAR Safe has to ensure that write access to the same memory parts is only possible for software of the same or higher quality level.

Depending on the requirements of the item, timing and program flow must be monitored. MICROSAR Safe provide mechanisms to implement this monitoring (see also TSR-13, TSR-14 and TSR-15). MICROSAR Safe does not automatically monitor itself for intended timing and program flow.

MICROSAR Safe components without allocated safety requirements are developed according to an ISO 26262-compliant development process to provide an argument for coexistence.
If safety requirements are implemented in components of MICROSAR Safe, Vector

ensures that they are locally non-complex. For locally non-complex components no diverse implementation of algorithms is designed.

1.2.1.2 Tool Confidence

MICROSAR Safe must be configured using the DaVinci tools. MICROSAR Safe must be compiled by using a compiler for the target hardware. These tools must be evaluated with respect to their impact on functional safety by the user.

It is assumed that the user of MICROSAR Safe evaluates and qualifies the defined compiler (incl. options) and linker according to ISO 26262 Part 8 Clause 11.

The DaVinci Developer and DaVinci Configurator are assumed to have an impact on functional safety. In the first place, only a tool error detection level of two can be expected. The resulting tool confidence level (TCL) would require a qualification of those tools.
To ensure freedom from interference with respect to memory for the BSW, Vector provides the MICROSAR Safe Silence Verifier (MSSV).
To ensure freedom from interference with respect to memory and to detect additional faults that may have been introduced by the DaVinci tools for the RTE, Vector provides the RTE Analyzer.
If MSSV and RTE Analyzer are used according to their Technical References and the Safety Manual, the tool confidence level for the DaVinci tools can be reduced to TCL1 by the user of MICROSAR Safe. MSSV and RTE Analyzer can be assumed TCL2 and qualified for this TCL, since a safety-related development process at Vector is applied for those tools.

Tools by Vector do not implement mechanisms to handle hardware faults on the host development computer.

1.2.2 Technical Safety Requirements

The items listed in this section are the assumed technical safety requirements on the Safety Element out of Context MICROSAR Safe. These requirements are expected to match the requirements in the actual item development.

All technical safety requirements (TSRs) are assigned an ASIL D (unless otherwise stated) to service as many projects as possible.

No fault tolerant time intervals are given. Timing depends on the used hardware and its configuration. It is assumed that the user configures MICROSAR Safe adequately for the intended use.

No safe state is defined since MICROSAR Safe allows the user to define the desired behavior in case of a detected fault.

The Safety Manual provided with a delivery shows the details on the support of a safety feature on component level.

1.2.2.1 Initialization

TSR-1 MICROSAR Safe shall provide a mechanism to initialize itself and its controlled hardware.
It is sensible for safety-related systems to start performing potentially hazardous actions only in a defined and intended state. For example, adequate setup of clocks may be required to reach required fault tolerant times.
Components of MICROSAR Safe initialize their variables. For post build selectable and post build loadable configurations MICROSAR Safe components use a consistent and defined set of configuration data.
Hardware-specific components of MICROSAR Safe describe in their Technical References what hardware units are controlled.
The user of MICROSAR Safe shall ensure that initialization functions of MICROSAR Safe are called in the right order and at the right point in time.
The user of MICROSAR Safe is also responsible for the startup code and main function.

1.2.2.2 Self-test

TSR-2 MICROSAR Safe shall provide mechanisms to perform self-tests of hardware.
Self-tests of hardware may be required to achieve the required single-point or latent fault metrics. Usually these tests are performed cyclically or during start-up.
MICROSAR Safe provides mechanisms to perform self-tests of hardware. The details on the provided self-tests are described in the component-specific part of the Safety Manual provided with a delivery.

1.2.2.3 Reset of ECU

TSR-3 MICROSAR Safe shall provide a mechanism to reset the ECU.
Resetting the CPU of the ECU is in most cases an appropriate measure to achieve a safe state in case of a detected fault.
It is assumed that the reset (and powerless) state of a microcontroller leads to the safe state of the ECU and system, since a reset may occur at any time due to e.g. random hardware fault.
MICROSAR Safe does not reset the CPU without request.

1.2.2.4 Data Consistency

TSR-101876 MICROSAR Safe shall provide mechanisms to ensure data consistency.
Concurrent access to resources (e.g. variables) from e.g. task and interrupt level, may lead to data inconsistencies.
MICROSAR Safe provides functions to enable or disable interrupts, spin-locks, resources or abstractions (i.e. exclusive areas) to enable the user of MICROSAR Safe to ensure data consistency.
MICROSAR Safe will not unintentionally enable or disable the mechanisms to ensure data consistency.
MICROSAR Safe also uses this functionality in its own components to ensure data consistency.

1.2.2.5 Non-volatile Memory

TSR-4 MICROSAR Safe shall provide a mechanism to store data in non-volatile memory.
Calibration or other application data may be safety-related, i.e. if there is data available from non-volatile memory it must not be corrupted by either software or hardware.
MICROSAR Safe implements an ""end-to-end protection"" in the Non-volatile RAM Manager (NvM) to ensure that data is neither corrupted nor masqueraded in either software or hardware.
Note that hardware may not even start storing data in non-volatile memory or loose it at any time.
Availability of data is, thus, not guaranteed. Availability may be increased by redundantly storing data in non-volatile memory. The user of MICROSAR Safe must handle unavailability of data on application level.

TSR-5 MICROSAR Safe shall provide a mechanism to retrieve data from non-volatile memory.
Calibration or other application data may be safety-related, i.e. if there is data available from non-volatile memory it must not be corrupted by either software or hardware.
MICROSAR Safe implements an ""end-to-end protection"" in the Non-volatile RAM Manager (NvM) to ensure that data is neither corrupted nor masqueraded in either software or hardware.
Note that hardware may not even start storing data in non-volatile memory or loose it at any time.
Availability of data is, thus, not guaranteed. Availability may be increased by redundantly storing data in non-volatile memory. The user of MICROSAR Safe must handle unavailability of data on application level.
Note that if data is written more than once and the latest data in the non-volatile memory gets corrupted, older (uncorrupted) data may be returned to the user of MICROSAR Safe.

1.2.2.6 Hard Real-time Scheduling

TSR-6 MICROSAR Safe shall provide hard real-time scheduling properties.
Hard real-time scheduling is not required for safety in the first place, since deadline violations can usually be detected and a safe state entered. However, hard real-time scheduling may support argumentation in some cases.
For fail-operational systems hard real-time scheduling properties are essential for the software that needs to stay operational.
The operating system of MICROSAR Safe implements fixed priority scheduling and the immediate priority ceiling protocol specified by AUTOSAR to guarantee hard real-time scheduling properties.
The operating system of MICROSAR Safe also provides an upper bound for the execution time of its functionality.
Note that fail-operational requirements on a system usually require a second independent channel of control.
Note that fail-operational requirements require analyses of the worst-case execution time (WCET) of the software that needs to stay operational.

1.2.2.7 Memory Protection

TSR-7 MICROSAR Safe shall provide mechanisms to protect software applications from unspecified memory access.
Partitioning in software is often introduced because of different quality levels of software and different responsibilities of software development on one ECU.
Memory partitioning relies on the memory protection unit (MPU) in hardware for the effectiveness of the mechanism.
MICROSAR Safe configures exactly the configured memory access rights for each task and ISR at the hardware interface of the MPU.
Note that MICROSAR Safe does not necessarily control all available protection units (e.g. system MPUs or peripheral protection units).
The user of MICROSAR Safe is responsible for adequate configuration of the memory partitions.

1.2.2.8 Communication

1.2.2.8.1 Inter ECU Communication

TSR-10 MICROSAR Safe shall provide mechanisms to protect communication between ECUs.
Communication between ECUs may be corrupted, unintentionally replayed, lost or masked. To protect against these failure modes MICROSAR Safe provides the end-to-end (E2E) protection mechanism defined by AUTOSAR.
MICROSAR Safe detects repetition, loss, insertion, masking, reodering and corruption of messages between ECUs.
MICROSAR Safe allows the usage of cyclic redundancy checks (CRCs) and cryptographic algorithms to protect the integrity of messages between ECUs.
CRCs provide a certain hamming distance given a polynomial and maximum data block size.
Cryptographic hash functions or message authentication codes (MACs) provide a probabilistic statement on data corruption detection depending on the hash function or MAC, data block size and hash value size.
The user of MICROSAR Safe is responsible for the selection of the E2E profile or algorithm that is adequate to the requirements of the item development.

TSR-104422 MICROSAR Safe shall provide mechanisms to communicate between ECUs.
The use of end-to-end protection mechanisms requires additional data to be computed in software and sent over the network. For ASIL A it might be acceptable to reduce fault detection capabilities to reduce this overhead.
MICROSAR Safe detects if there are no incoming messages in a configured time frame and if messages have been corrupted on the bus.
MICROSAR Safe does not unintentionally repeat, discard, delay, insert, mask, reorder or corrupt messages in software.
MICROSAR Safe cannot detect if old, repeated or masked data was received. MICROSAR Safe cannot detect if messages are unintentionally reordered or if some messages have been lost.
MICROSAR Safe does not provide mechanisms to detect latent faults in the communication hardware, e.g. runtime check of the CRC check in the CAN controller.

The user of MICROSAR Safe is responsible to decide whether reduced fault detection capabilities are acceptable. This should include an analysis on a per signal base of the underlying system requirements of the item.
Note this TSR is only assigned an ASIL A. For higher ASILs Vector requires the usage of an end-to-end protection mechanism (see TSR-10).

1.2.2.8.2 Intra ECU Communication

TSR-16 MICROSAR Safe shall provide mechanisms to communicate within its applications.
Software components need to communicate in order to fulfill their task.
MICROSAR Safe provides mechanisms to communicate within OS applications without end-to-end protection.
MICROSAR Safe does not unintentionally repeat, delay, insert, mask, corrupt or loose data communicated within OS applications.
MICROSAR Safe assumes protection of the memory against random hardware faults by the system, e.g. via ECC RAM and lock-step mode.

TSR-12 MICROSAR Safe shall provide mechanisms to communicate between its applications.
Mixed ASIL or multi-core systems need to exchange safety-related information between applications.
MICROSAR Safe provides mechanisms to communicate between OS applications without end-to-end protection.
MICROSAR Safe does not unintentionally repeat, delay, insert, mask, corrupt or loose data communicated between OS applications.
MICROSAR Safe assumes protection of the memory against random hardware faults by the system, e.g. via ECC RAM and lock-step mode.

1.2.2.9 Monitoring of Software

MICROSAR Safe provides monitoring mechanisms to supervise correct execution of software. These mechanisms must be configured by the user of MICROSAR Safe according to the requirements within the context of the item development.

TSR-13 MICROSAR Safe shall provide a mechanism to detect faults in program flow.
Program flow can be corrupted by random hardware faults or software faults.
The user of MICROSAR Safe can configure a graph of the logical program flow that is then supervised by MICROSAR Safe. A transition from one checkpoint to another that is not allowed is detected.

TSR-14 MICROSAR Safe shall provide a mechanism to detect stuck software.
Alive monitoring is used to reset the software or controller in case it is unresponsive.
The user of MICROSAR Safe can configure entities that need to regularly report their alive status that is then supervised by MICROSAR Safe. Omission of a regular report is detected.

TSR-15 MICROSAR Safe shall provide a mechanism to detect deadline violations.
Deadlines are important to reach a safe state within the defined fault tolerant time interval.
The user of MICROSAR Safe can configure a tolerable time interval for each transition in a

logical program flow graph that is then supervised by MICROSAR Safe. Violation of a deadline is detected.

TSR-8 MICROSAR Safe shall provide a mechanism to detect time budget violations.
Budgets for task execution times, inter-arrival times and locking times allow to identify the originator of a deadline violation.
The user of MICROSAR Safe can configure budgets for execution times of tasks and category 2 ISRs, inter-arrival times and duration of locks (e.g. resource or interrupt locks) that are then supervised by MICROSAR Safe. Exhaustion of budgets is detected.

TSR-9 MICROSAR Safe shall provide a mechanism to terminate software applications.
Terminating a complete OS application may be used as a safety mechanisms to mitigate a software fault within this OS application.
MICROSAR Safe terminates exactly the requested OS application and only upon request.
Note that MICROSAR Safe will not terminate OS applications automatically without configuration or request.

1.2.2.10 Operating Modes

TSR-100551 MICROSAR Safe shall provide a mechanism to switch between operating modes.
Fault detection and mitigation often result in a degraded or safe state. These modes can be configured and switched in BswM and Rte.
MICROSAR Safe does not unintentionally switch between operating modes. MICROSAR Safe switches to exactly the requested operating mode.

1.2.2.11 Peripheral In- and output

TSR-17 MICROSAR Safe shall provide a mechanism to read input data from peripheral units.
Input of data from microcontroller peripheral units is required to implement almost any safety-related system.
MICROSAR Safe provides the data read from peripheral units without corruption or unintentional delay.
MICROSAR Safe does not unintentionally reorder nor masquerade data read from peripheral units.
Note that the peripheral hardware may not provide sufficient diagnostic coverage without redundant input.
For example, DIO and SPI drivers provided by Vector support the reading input data from peripheral units as safety feature.

TSR-18 MICROSAR Safe shall provide a mechanism to write output data to peripheral units.
Output of data from microcontroller peripheral units is required to implement almost any safety-related system.
MICROSAR Safe writes the data provided at the software interfaces without corruption or unintentional delay.
MICROSAR Safe does not unintentionally trigger peripheral output units.
MICROSAR Safe does not unintentionally reorder nor masquerade data read from

peripheral units.
Note that the peripheral hardware may require additional mechanisms on application or system level to ensure sufficient safety.
For example, DIO and SPI drivers provided by Vector support the output of data to peripheral units as safety feature (e.g. to trigger external watchdogs or switch actuation paths).

1.2.3 Environment

1.2.3.1 Safety Concept

SMI-14
The user of MICROSAR Safe shall be responsible for the functional safety concept.

The overall functional safety concept is in the responsibility of the user of MICROSAR Safe. MICROSAR Safe can only provide parts that can be used to implement the functional safety concept of the item.
It is also the responsibility of the user of MICROSAR Safe to configure MICROSAR Safe as intended by the user's safety concept.
The safety concept shall only rely on safety features explicitly described in this safety manual. If a component from MICROSAR Safe does not explicitly describe safety features in this safety manual, this component has been developed according to the methods for ASIL D to provide coexistence with other ASIL components.

- Example: NvM provides safety features for writing and reading of data, the lower layers, i.e. MemIf, Ea, Fee and drivers, only provide the ASIL for coexistence.

The safety concept shall **not** rely on functionality that is **not** explicitly described as safety feature in this safety manual. This functionality may fail silently in case of a detected fault.

- Example: If a potential out-of-bounds memory access, e.g. due to invalid input or misconfiguration, is detected the requested function will not be performed. An error via DET is only reported if error reporting is enabled.

SMI-1
The user of MICROSAR Safe shall adequately address hardware faults.

The components of MICROSAR Safe can support in the detection and handling of some hardware faults (e.g. using watchdog).
MICROSAR Safe does not provide redundant data storage.
The user of MICROSAR Safe especially has to address faults in volatile random access memory, non-volatile memory, e.g. flash or EEPROM, and the CPU.
MICROSAR Safe relies on the adequate detection of faults in memory and the CPU by other means, e.g. hardware. Thus, Vector recommends using lock-step CPUs together with ECC memory.
See also SMI-14.

SMI-10
The user of MICROSAR Safe shall ensure that the reset or powerless state is a safe state of the system.

Vector uses this assumption in its safety analyses and development process.

SMI-20
The user of MICROSAR Safe shall implement a timing monitoring using e.g. a watchdog.

The components of MICROSAR Safe do not provide mechanisms to monitor their own timing behavior.
The watchdog stack that is part of MICROSAR Safe can be used to fulfill this assumption.
If the functional safety concept also requires a logic monitoring, The watchdog stack that is part of MICROSAR Safe can be used to implement it.
The watchdog is one way to perform timing monitoring. Today the watchdog is the most common approach. In future there may be different approaches e.g. by monitoring using a different ECU.
See also SMI-14.

SMI-98
The user of MICROSAR Safe shall ensure an end-to-end protection for safety-related communication between ECUs.

The communication components of MICROSAR Safe do not assume sending or receiving as a safety requirement, because considered faults can only be detected using additional information like a cycle counter. Vector always assumes that an end-to-end protection or equivalent mechanism is implemented on application level.
Considered faults in communication are:

- Failure of communication peer
- Message masquerading
- Message corruption
- Unintended message repetition
- Insertion of messages
- Re-sequencing
- Message loss
- Message delay

This requirement can be fulfilled by e.g. using the end-to-end protection wrapper for safety related communication.

SMI-11
The user of MICROSAR Safe shall ensure data consistency for its application.

Data consistency is not automatically provided when using MICROSAR Safe. MICROSAR Safe only provides services to support enforcement of data consistency. Their application is in the responsibility of the user of MICROSAR Safe.

To ensure data consistency in an application, critical sections need to be identified and protected.
To identify critical sections in the code, e.g. review or static code analysis can be used.
To protect critical sections, e.g. the services to disable and enable interrupts provided by the MICROSAR Safe operating system can be used.
To verify correctly implemented protection, e.g. stress testing or review can be used.
Note the AUTOSAR specification with respect to nesting and sequence of calls to interrupt enabling and disabling functions.
See also TSR-101876.

1.2.3.2 Use of MICROSAR Safe Components

SMI-2
The user of MICROSAR Safe shall adequately select the type definitions to reflect the hardware platform and compiler environment.

The user of MICROSAR Safe is responsible for selecting the correct platform types (PlatformTypes.h) and compiler abstraction (Compiler.h). Especially the size of the predefined types must match the target environment.
Example: A uint32 must be mapped to an unsigned integer type with a size of 32 bits.
The user of MICROSAR Safe can use the platform types provided by Vector. Vector has created and verified the platform types mapping according to the information provided by the user of MICROSAR Safe.

SMI-12
The user of MICROSAR Safe shall initialize all components of MICROSAR Safe prior to using them.

This constraint is required by AUTOSAR anyway. Vector uses this assumption in its safety analyses and development process.
Correct initialization can be verified, e.g. during integration testing.

SMI-16
The user of MICROSAR Safe shall only pass valid pointers at all interfaces to MICROSAR Safe components.

Plausibility checks on pointers are performed by MICROSAR Safe (see also SMI-18), but they are limited. MICROSAR Safe components potentially use provided pointers to write to the location in memory.
Also the length and pointer of a buffer provided to a MICROSAR Safe component need to be consistent.
This assumption also applies to QM as well as ASIL components.
This can e.g. be verified using static code analysis tools, reviews and integration testing.

SMI-36039
The user of MICROSAR Safe shall only retain pointers provided by MICROSAR Safe where explicitly stated.

Some MICROSAR Safe components provide call-outs to user or application code. The pointers passed to these call-outs may have limited validity.

The user of MICROSAR Safe shall ensure that pointers are not used neither for reading nor for writing after returning from the call-out. Exceptions of this rule are explicitly documented for the respective call-out.

SMI-18
The user of MICROSAR Safe shall enable plausibility checks for the MICROSAR Safe components.

This setting is necessary to introduce defensive programming and increase robustness at the interfaces as required by ISO 26262.
This setting has to be configured at `/MICROSAR/EcuC/EcucGeneral/EcuCSafeBswChecks` and `<Component-specific path>/<Ma>SafeBswChecks` for all components that are intended to be ASIL.
Ma is the Module Abbreviation.
This setting is enforced by an MSSV plug-in.
This setting does not enable error reporting to the DET component.

SMI-1725
The user of MICROSAR Safe shall configure and use the interrupt system correctly.

The user of MICROSAR Safe is responsible for a correct and consistent configuration and usage of the interrupt system.
Especially the following topics shall be verified:

- Consistent configuration of interrupt category, level and priority in OS and MCAL modules
- Correct assignment of logical channels/instances to interrupt vectors in case of MCAL modules with multiple channels/instances
- The interrupt controller is configured in a mode which processes interrupts of the same level sequentially to avoid unbounded interrupt nesting

SMI-36041
The user of MICROSAR Safe shall not use functionality of MICROSAR Safe components marked with a BETA-ESCAN.

Functionality marked with a BETA-ESCAN is not thoroughly tested by Vector.
Vector ensures that functionality marked with a BETA-ESCAN can be completely disabled.

The list of ESCANs (incl. BETA-ESCANs) is provided with a delivery.

1.2.3.3 Partitioning

SMI-9
The user of MICROSAR Safe shall ensure that for one AUTOSAR functional cluster (e.g. System Services, Operating System, CAN, COM, etc.) only components from Vector are used.

This assumption is required because of dependencies within the development process of Vector.

This assumption does not apply to the MCAL or the EXT cluster.
Vector may have requirements on MCAL or EXT components depending on the upper layers that are used and provided by Vector. For example, the watchdog driver is considered to have safety requirements allocated to its initialization and triggering services. Details are described in the component specific parts of this safety manual.
This assumption does not apply to components that are not provided by Vector.
In case the partitioning solution is used, this assumption only partially applies to the System Services cluster. Only the Watchdog Manager and Watchdog Interface need to be used from Vector then, because the Watchdog Manager and Watchdog Interface will be placed in separate memory partitions apart from the other System Services components.

SMI-32
The user of MICROSAR Safe shall provide an argument for coexistence for software that resides in the same partition as components from MICROSAR Safe.

Vector considers an ISO 26262-compliant development process for the software as an argument for coexistence (see [1] Part 9 Clause 6). Vector assumes that especially freedom from interference with respect to memory is provided by an ISO 26262-compliant development process.
Redundant data storage as the only measure by the other software is not considered a sufficient measure.
If ASIL components provided by Vector are used, this requirement is fulfilled.
In general Vector components do not implement methods to interface with other software (e.g. components, hooks, callouts) in other partitions. They assume that all interfacing components reside in the same partition. Interfacing components are described in the respective technical reference.
If an argument for coexistence cannot be provided, other means of separation have to be implemented (e.g. trusted or non-trusted function calls).

SMI-99
The user of MICROSAR Safe shall verify that the memory mapping is consistent with the partitioning concept.
The volatile data of every component shall be placed in the associated memory partition. This can be verified e.g. by review of the linker map file.
The memory sections for each component placed in RAM can be identified <MIP>_START_SEC_VAR[_<xxx>], where <MIP> is the Module Implementation Prefix of the component.

1.2.3.4 Resources

SMI-33
The user of MICROSAR Safe shall provide sufficient resources in RAM, ROM, stack and CPU runtime for MICROSAR Safe.

Selection of the microcontroller and memory capacities as well as dimensioning of the stacks is in the responsibility of the user of MICROSAR Safe.
If MICROSAR Safe components have specific requirements, these are documented in the respective Technical Reference document.

1.2.4 Process

SMI-15

The user of MICROSAR Safe shall follow the instructions of the corresponding Technical Reference of the components.

Especially deviations from AUTOSAR specifications are described in the Technical References.
If there are constraints for the implementation of an exclusive area, these are described in the Technical References.

SMI-5

The user of MICROSAR Safe shall verify all code that is modified during integration of MICROSAR Safe.

Code that is typically modified by the user of MICROSAR Safe during integration comprises generated templates, hooks, callouts, or similar.
This assumption also applies if interfaces between components are looped through user-defined functions.
Vector assumes that this verification also covers ISO 26262:6-9. Vector assumes that modified code that belongs to a Vector component, e.g. EcuM callouts or OS trace hooks can at least coexist with this component, because no separation in memory or time is implemented.
Example: Callouts of the EcuM are executed in the context of the EcuM.
Non-trusted functions provided by the Vector operating system can be used to implement a separation in memory in code modified by the user of MICROSAR Safe.
Support by Vector can be requested on a per-project basis.

SMI-30

The user of MICROSAR Safe shall only modify source code of MICROSAR Safe that is explicitly allowed to be changed.

Usually no source code of MICROSAR Safe is allowed to be changed by the user of MICROSAR Safe.
The user of MICROSAR Safe can check if the source code was modified by e.g, comparing it to the original delivery.

SMI-8

The user of MICROSAR Safe shall verify generated functions according to ISO 26262:6-9.

Generated functions can be identified when searching through the generated code.
Support by Vector can be requested on a per-project basis.
An example of generated functions is the configured rules of the Basic Software Manager (BSWM). Their correctness can only be verified by the user of MICROSAR Safe. Please note, however, that BSWM does not provide safety features.
This requirement does not apply to MICROSAR SafeRTE.

SMI-19

The user of MICROSAR Safe shall execute the MICROSAR Safe Silence Verifier (MSSV).

MSSV is used to detect potential out-of-bounds accesses by Vector's basic software based on inconsistent configuration.
Details on the required command line arguments and integration into the tool chain can be found in [4].
If the report shows “Overall Check Result: Fail", please contact the Safety Manager at Vector. See the Product Information MICROSAR Safe for contact details.

SMI-4
The user of MICROSAR Safe shall perform the integration (ISO 26262:6-10) and verification (ISO 26262:6-11) processes as required by ISO 26262.

Especially the safety mechanisms must be verified in the final target ECU.
Vector assumes that by performing the integration and verification processes as required by ISO 26262 the generated configuration data, e.g. data tables, task priorities or PDU handles, are sufficiently checked. An additional review of the configuration data is then considered not necessary.
Integration does not apply to a MICROSAR Safe component that consists of several subcomponents. This integration is already performed by Vector. The integration of subcomponents is validated during creation of the safety case by Vector based on the configuration handed in by the user of MICROSAR Safe.
However, integration of all MICROSAR Safe components in the specific use-case of the user of MICROSAR Safe is the responsibility of the user of MICROSAR Safe. This also includes the hardware-software integration in the context of the target ECU.
Support by Vector can be requested on a per-project basis.

SMI-100
The user of MICROSAR Safe shall ensure that a consistent set of generated configuration is used for verification and production.
Make sure that the same generated files are used for testing and production code, i.e. be aware that configuration can be changed without generating the code again.
Make sure that all generated files have the same configuration basis, i.e. always generate the MICROSAR Safe configuration for all components for a relevant release of the ECU software.
The use of post build loadable is supported but not recommended by Vector.

SMI-176
The user of MICROSAR Safe shall verify the integrity of the delivery by Vector.
Run the SIPModificationChecker.exe and verify that the source code, BSWMD and safety manual files are unchanged.

SMI-31
The user of MICROSAR Safe shall verify the consistency of the binary downloaded into the ECU's flash memory.

This also includes re-programming of flash memory via a diagnostics service. The consistency of the downloaded binary can be checked by the bootloader or the application. MICROSAR Safe assumes a correct program image.

SMI-3

The user of MICROSAR Safe shall evaluate all tools (incl. compiler) that are used by the user of MICROSAR Safe according to ISO 26262:8-11.

Evaluation especially has to be performed for the compiler, linker, debugging and test tools.
Vector provides a guideline for the evaluation of the Tool Confidence Level (TCL) for the tools provided by Vector (e.g. DaVinci Configurator PRO).
MSSV and RTE Analyzer can be assumed TCL2 and qualified for this TCL, since a safety-related development process at Vector is applied for those tools. Confirmation that this safety-related development process was applied is part of the safety case.
Vector has only evaluated the tools used by Vector during the development of MICROSAR Safe. Tool evaluation, for tools used by the user of MICROSAR Safe, is in the responsibility of the user of MICROSAR Safe.

2 Safety Manual FrTrans

2.1 Safety features

This component does not provide safety features.

2.2 Configuration constraints

This component does not have configuration constraints.

2.3 Additional verification measures

This component does not require additional verification measures.

2.4 Dependencies to other components

2.4.1 Safety features required from other components

This component does not require safety features from other components.

2.4.2 Coexistence with other components

SMI-376
This component requires coexistence with FrIf, EcuM, Dem, Det, Dio, SPI and ICU components if the interface for those components is configured.

2.5 Dependencies to hardware

This component does not use a direct hardware interface.

3 Safety Manual BswM

3.1 Safety features

This component does not provide safety features.

3.2 Configuration constraints

SMI-3528
The user of MICROSAR Safe shall configure the following attribute:

- Set */MICROSAR/BswM/BswMGeneral/BswMEthIfEnabled* to *FALSE*.

This setting is enforced by an MSSV plugin.

3.3 Additional Verification measures

This component does not require additional verification measures.

3.4 Safety features required from other components

This component does not require safety features from other components.

3.5 Dependencies to hardware

This component does not use a direct hardware interface.

4 Safety Manual Com

4.1 Safety features

This component does not provide safety features.

4.2 Configuration constraints

SMI-314
The user of MICROSAR Safe shall configure the following parameters:

- Set /MICROSAR/Com/ComGeneral/ComReceiveSignalMacroAPI to FALSE.
- Set /MICROSAR/Com/ComGeneral/ComMetaDataSupport to FALSE.
- Set /MICROSAR/Com/ComGeneral/ComDescriptionRoutingCodeGeneration to FALSE.

This setting is enforced by a MSSV plugin.

SMI-315
The user of MICROSAR Safe shall configure the following:
A container in /MICROSAR/PduR/PduRBswModules with a reference (/MICROSAR/PduR/PduRBswModules/PduRBswModuleRef) to /ActiveEcuC/Com and shall set the following parameters:

- /MICROSAR/PduR/PduRBswModules/PduRCommunicationInterface to TRUE
- /MICROSAR/PduR/PduRBswModules/PduRTransportProtocol to FALSE
- /MICROSAR/PduR/PduRBswModules/PduRCancelTransmit to FALSE

This setting is enforced by a MSSV plugin.

4.3 Additional verification measures

SMI-1104
The user of MICROSAR Safe shall verify that the `SignalDataPtr` passed to `Com_ReceiveSignal` and `Com_ReceiveShadowSignal` points to a valid buffer which matches the configured /MICROSAR/Com/ComConfig/ComSignal/ComSignalType or /MICROSAR/Com/ComConfig/ComGroupSignal/ComSignalType. In case of the ComSignalType UINT8_N the caller must ensure that the array size matches to the configured /MICROSAR/Com/ComConfig/ComSignal/ComSignalLength or /MICROSAR/Com/ComConfig/ComGroupSignal/ComSignalLength.

This can be verified by comparing the type of the pointer passed to `SignalDataPtr` to the `ApplType` returned by `Com_GetApplTypeOfRxAccessInfo(Index)`.

If the `ApplType` is set to `COM_UINT8_N_APPLTYPEOFRXACCESSINFO` additionally verify that the value of `Com_GetRxSigBufferArrayBasedBufferLengthOfRxAccessInfo(Index)` is less or equal to the size (in bytes) of the array passed to `SignalDataPtr`.

The parameter of the macros `Com_GetApplTypeOfRxAccessInfo(Index)` and `Com_GetRxSigBufferArrayBasedBufferLengthOfRxAccessInfo(Index)` is the `SignalId` and can be found in the generated header files.

4.4 Safety features required from other components

This component does not require safety features from other components.

4.5 Dependencies to hardware

This component does not use a direct hardware interface.

5 Safety Manual ComM

5.1 Safety features

This component does not provide safety features.

5.2 Configuration constraints

This component does not have configuration constraints.

5.3 Additional Verification measures

SMI-94
The user of MICROSAR Safe shall verify that the array size generated by ComM matches to the array size in the type definition of RTE. The following procedure shall be applied to each channel that has activated the ComM parameter 'Full Comm Request Notification Enabled'.

ComM_Cfg.h contains array size definition in the format COMM_MAX_CR_<ShortNameOfChannel>

rte_type.h contains the definition of the corresponding structure type in the format ComM_UserHandleArrayType_<ShortNameOfChannel>

Verify that the structure member 'handleArray' has the same size as the corresponding define value of ComM in 1).

Verify the content of the generated functions ComM_CurrentChannelRequestInit and ComM_CurrentChannelRequestNotification to ensure that the proper define COMM_MAX_CR_<ShortNameOfChannel> is used to limit the array index when writing to ComM_UserHandleArrayType_<ShortNameOfChannel>.handleArray[].

SMI-95
The user of MICROSAR Safe shall verify that the value of ComSignalLength (byte) in Com module is smaller or equal to the value of COMM_PNC_SIGNAL_LENGTH (can be found in ComM_Cfg.h).
This shall be verified for each ComPncSignal referenced by Partial Network Clusters and having ComMPncComSignalDirection = RX.

SMI-1046
The user of MICROSAR Safe shall verify that each element of table `ComM_UserModeNotiFunc` refers either a `NULL_PTR` or a function that has the following signature (the placeholder `<name>` represents the function's name):

```
Std_ReturnType <name>(uint8 nextMode)
```

The table `ComM_UserModeNotiFunc` can be found in ComM_Lcfg.c. This measure is only needed if at least one ComM user has enabled the parameter 'User Mode Notification'.

5.4 Safety features required from other components

This component does not require safety features from other components.

5.5 Dependencies to hardware

This component does not use a direct hardware interface.

6 Safety Manual Crc

6.1 Safety features

SMI-344
Crc provides the following safety features:

ID	Safety feature
CREQ-858	Crc shall provide a service to calculate 8-bit SAE-J1850 CRC.
CREQ-859	Crc shall provide a service to calculate 8-bit 0x2F CRC.
CREQ-860	Crc shall provide a service to calculate 16-bit CCITT CRC.
CREQ-861	Crc shall provide a service to calculate 32-bit IEEE-802.3 CRC.
CREQ-862	Crc shall provide a service to calculate 32-bit E2E Profile 4 CRC.
CREQ-117997	Crc shall provide a service to calculate 64-bit ECMA CRC.

6.2 Configuration constraints

This component has no configuration constraints.

6.3 Additional verification measures

SMI-49
The user of MICROSAR Safe shall verify that the CRC is calculated for the intended data.

This includes the intended buffer and its size (see also SMI-16), start value and if it is the first call to the service.
Verification can be performed by the "magic check" (see AUTOSAR SWS Crc).

If Crc is used by a MICROSAR Safe component (e.g. E2E, NvM), this requirement is fulfilled for the MICROSAR Safe component.

6.4 Dependencies to other components

6.4.1 Safety features required from other components

This component does not require safety features from other components.

6.4.2 Coexistence with other components

This component does not require coexistence with other components.

It is assumed that the user of Crc has the adequate ASIL.

6.5 Dependencies to hardware

This component does not use a direct hardware interface.

7 Safety Manual CryIf

7.1 Safety features

This component does not provide safety features.

7.2 Configuration constraints

This component does not have configuration constraints.

7.3 Additional verification measures

This component does not require additional verification measures.

7.4 Safety features required from other components

This component does not require safety features from other components.

7.5 Dependencies to hardware

This component does not use a direct hardware interface.

8 Safety Manual Cry

8.1 Safety Features

SMI-37419
Cry provides the following safety features:

ID	Safety feature
CREQ-121598	Cry shall provide services to compute MAC functions.
CREQ-121582	Cry shall provide services to verify MAC functions.
CREQ-139835	Cry shall provide a service to perform a self check which ensures valid CMAC Verification.

Note: It is assumed that cryptographic algorithms yield different results for different input parameters.

8.2 Configuration constraints

SMI-69642
The user of MICROSAR Safe shall configure the following attributes:

- Set /MICROSAR/Cry_30_ai/Cry/CryGeneral/CrySelfTest to TRUE.

The user of MICROSAR Safe shall ensure that the switch is correctly generated.

The define `CRY_30_<AI>_SELF_TEST` which can be found in file Cry_30_<ai>_Cfg.h needs to be set to STD_ON.

This requirement only applies if TSR-2 is considered a safety requirement.

8.3 Additional verification measures

SMI-37421
The user of MICROSAR Safe shall verify that the intended Cry service is called during integration testing.

This requirement only applies if TSR-10 is considered a safety requirement.

SMI-69639
The used Cry shall perform a self check of the CMAC verify functionality.

This requirement only applies if TSR-10 is considered a safety requirement.

8.4 Safety features required from other components

This component does not require safety features from other components.

8.5 Dependencies to hardware

This component does not use a direct hardware interface.

9 Safety Manual Cry (RH850 ICU-S)

9.1 Safety Features

No additional safety features are provided.

9.2 Configuration constraints

No additional configuration constraints are required.

9.3 Additional verification measures

No additional verification measures are required.

9.4 Safety features required from other components

No additional safety features are required from other components.

9.5 Dependencies to hardware

The safety manual Renesas Safety Application Note RH850/F1L Group, Revision 2.11 was used during development. This component does not implement any requirement from that safety manual.

This component implements SAN-F1L-1001

This component requires exclusive access to the hardware registers of the unit it is intended to control. See the technical reference for the hardware register names and used hardware manuals.

10 Safety Manual Csm

10.1 Safety features

This component does not provide safety features.

10.2 Configuration constraints

This component does not have configuration constraints.

10.3 Additional verification measures

This component does not require additional verification measures.

10.4 Safety features required from other components

This component does not require safety features from other components.

10.5 Dependencies to hardware

This component does not use a direct hardware interface.

11 Safety Manual Dcm

SMI-4672
Please note: This version is not fully developed according to ISO 26262 ASIL D. Vector provides an alternative argument for freedom from interference with respect to memory though. The Silent methodology developed by Vector has been completely implemented for this component.

11.1 Safety features

This component does not provide safety features.

11.2 Configuration constraints

The user of MICROSAR Safe shall configure the following attributes:

SMI-37385
Set */MICROSAR/Dcm/DcmConfigSet/DcmDsd/DcmDsdServiceTable/DcmDsdService/DcmDsdSidTabFnc* to contain the corresponding diagnostic service handler in your application. This has to be done for all diagnostic services, except the ones with SID: 0x10, 0x11, 0x14, 0x19, 0x27, 0x28, 0x31, 0x3E and 0x85.
– Note: Since all other diagnostic services are not handled by DCM, all their related configuration parameters will be ignored during the code generation (e.g. DIDs, etc.)

SMI-64979
Set */MICROSAR/Dcm/DcmConfigSet/DcmGeneral/DcmDemApiVersion* to *DCM_DEM_API_4_01_02*, *DCM_DEM_API_4_02_01* or *DCM_DEM_API_4_03_00*.

SMI-64980
If the ECU shall support OBD communications, a dedicated connection to an OBD tester has to be configured: */MICROSAR/Dcm/DcmConfigSet/DcmDsl/DcmDslProtocol/DcmDslProtocolRow/DcmDslConnection*.

SMI-64981
Verify that the container */MICROSAR/Dcm/DcmConfigSet/DcmDsp/DcmDspVehicleSystemGroups* does not exist.

SMI-64982
Set */MICROSAR/Dcm/DcmConfigSet/DcmGeneral/DcmStateRecoveryAfterResetEnabled* to *false*.

SMI-64983
Set */MICROSAR/Dcm/DcmConfigSet/DcmGeneral/DcmSupportedIDCalculationSuppressionEnabled* to *false*.

These settings are enforced by an MSSV plugin.

11.3 Additional verification measures

SMI-37386
The user of MICROSAR Safe shall verify that none of the generated functions in *Dcm_Lcfg.c* modifies the pointer passed as an argument, but only writes a new value to it.

SMI-37387
The user of MICROSAR Safe shall verify that none of the generated functions in *Dcm_Lcfg.c* modifies the pointer passed as an argument, but only forwards it to another function.

SMI-37388
The user of MICROSAR Safe shall verify that none of the generated functions in *Dcm_Lcfg.c* modifies the pointer taken from a local variable, and that the function receiving those pointer parameter(s) does not store the addresses after return.

SMI-40607
The user of MICROSAR Safe shall verify that all generated *GetSeedFunc* functions in table *Dcm_CfgSvc27SecLevelInfo* located in *Dcm_Lcfg.c* match the signature indicated by the *GetSeedFuncClass* setting.

GetSeedFuncClass	Expected Signature
0	`Std_ReturnType <name>(Dcm_OpStatusType OpStatus, uint8 *Seed, Dcm_NegativeResponseCodeType *ErrorCode)`
1	`Std_ReturnType <name>(const uint8 *SecurityAccessDataRecord, Dcm_OpStatusType OpStatus, uint8 *Seed, Dcm_NegativeResponseCodeType *ErrorCode)`

SMI-49478
The user of MICROSAR Safe shall verify that for all generated *GetSeedFunc* functions in table *Dcm_CfgSvc27SecLevelInfo* located in *Dcm_Lcfg.c* no more data will be written than specified by the corresponding *SeedResLength* setting.

SMI-64952
The user of MICROSAR Safe shall verify that all generated *OpFunc* functions in table *Dcm_CfgRidMgrOpInfo* located in *Dcm_Lcfg.c* match the signature indicated by the *OpType* setting.

OpType	Expected Signature
0	`Std_ReturnType <name>(Dcm_OpStatusType OpStatus, Dcm_NegativeResponseCodeType *ErrorCode)`
1	`Std_ReturnType <name>(const uint8 *<InSignal>, Dcm_OpStatusType OpStatus, Dcm_NegativeResponseCodeType *ErrorCode)`
2	`Std_ReturnType <name>(const uint8 *<InSignal>, Dcm_OpStatusType OpStatus, uint8 *<OutSignal>, Dcm_NegativeResponseCodeType *ErrorCode)`

3	`Std_ReturnType <name>(Dcm_OpStatusType OpStatus, uint8 *<OutSignal>, Dcm_NegativeResponseCodeType *ErrorCode)`
4	`Std_ReturnType <name>(const uint8 *<InSignal>, Dcm_OpStatusType OpStatus, uint16 DataLength, Dcm_NegativeResponseCodeType *ErrorCode)`
5	`Std_ReturnType <name>(const uint8 *<InSignal>, Dcm_OpStatusType OpStatus, uint8 *<OutSignal>, uint16 DataLength, Dcm_NegativeResponseCodeType *ErrorCode)`
6	`Std_ReturnType <name>(Dcm_OpStatusType OpStatus, uint8 *<OutSignal>, uint16 *DataLength, Dcm_NegativeResponseCodeType *ErrorCode)`
7	`Std_ReturnType <name>(const uint8 *<InSignal>, Dcm_OpStatusType OpStatus, uint8 *<OutSignal>, uint16 *DataLength, Dcm_NegativeResponseCodeType *ErrorCode)`
8	`Same as OpType 7`
9	`Std_ReturnType <name>(Dcm_OpStatusType OpStatus, Dcm_MsgContextPtrType pMsgContext, Dcm_RidMgrRidLengthPtrType DataLength, Dcm_NegativeResponseCodePtrType ErrorCode)`

SMI-64953
The user of MICROSAR Safe shall verify that for all generated *OpFunc* functions in table *Dcm_CfgRidMgrOpInfo* located in *Dcm_Lcfg.c* no more data will be written than specified by the corresponding *ResMaxLength* setting.

SMI-65597
The user of MICROSAR Safe shall verify all generated Dcm_RidMgr_<RID>_<Start|Stop|RequestResults> functions implemented in *Dcm_Lcfg.c* that for each Dcm_DiagGetResDataRel(pMsgContext, **<offset>**) usage on out signals the application does not write more data than (*ResMaxLength* - **<offset>**), specified by the corresponding *ResMaxLength* setting in table *Dcm_CfgRidMgrOpInfo* located in _Dcm_Lcfg.c.

11.4 Safety features required from other components

This component does not require safety features from other components.

11.5 Dependencies to hardware

This component does not use a direct hardware interface.

12 Safety Manual Det

12.1 Safety features

This component does not provide safety features.

12.2 Configuration constraints

This component does not have configuration constraints.

SMI-60
If the DET is used in series production the extended debug features shall be switched off, because they are only relevant if a debugger is attached.

The user of MICROSAR Safe shall configure and verify the following attribute:

- /MICROSAR/Det/DetGeneral/DetExtDebugSupport = False

12.3 Additional Verification measures

SMI-4671
The user of MICROSAR Safe shall verify that the enter and exit functions of the DET's exclusive area do not produce DET errors.

Verification can e.g. be performed by review. If these functions are mapped to OS services it has to be checked that from the ErrorHook of the OS no DET error reporting functions are called if the ErrorHook has been called due to an error in the OS service used for the DET's exclusive area.

12.4 Safety features required from other components

This component does not require safety features from other components.

12.5 Dependencies to hardware

This component does not use a direct hardware interface.

13 Safety Manual E2E

13.1 Safety features

SMI-51
E2E provides the following safety features:

ID	Safety feature
CREQ-1086	E2E shall provide services to protect data.
CREQ-1087	E2E shall provide services to initialize the protection state.
CREQ-1088	E2E shall provide services to check protected data.
CREQ-1089	E2E shall provide services to initialize the check state.
CREQ-1175	E2E shall provide a service to map the check result to an adequate state machine status
CREQ-1091	E2E shall provide a service to check data within a reception window.
CREQ-1092	E2E shall provide a service to initialize the state machine.

13.2 Configuration constraints

This component has no configuration constraints.

13.3 Additional verification measures

SMI-52
The user of MICROSAR Safe shall verify that the E2E is used as intended.

This includes the verification of the parameters for end-to-end protection, e.g. data ID, and the order of calls as described in the Technical Reference for creating and checking the end-to-end protection.

The service to protect shall be called exactly once before sending. The service to check shall be called exactly once for reception.

This requirement is fulfilled for the MICROSAR Safe E2E Protection Wrapper and MICROSAR Safe E2E Transformer if used as specified in their respective safety manual section.

13.4 Dependencies to other components

13.4.1 Safety features required from other components

SMI-50
The used Crc library shall provide the CRC calculation routines as safety feature.
If the Crc library from MICROSAR Safe is used, this dependency is fulfilled.

13.4.2 Coexistence with other compoents

This component does not require coexistence with other components.

It is assumed that the user of E2E has the adequate ASIL.

13.5 Dependencies to hardware

This component does not use a direct hardware interface.

14 Safety Manual E2E Protection Wrapper

Term/Abbreviation	Definition
E2EConfig	E2E Protection Wrapper configuration file. The E2EConfig file is a data file containing information about the protected areas within I-PDUs of a given AUTOSAR system that are to be protected by the E2Elib and the E2EPW and how they are to be protected. It serves as input for the E2EPWG.
E2EPWG	E2E Protection Wrapper Generator. It takes an E2EConfig file as input and generates the E2E Protection Wraper code.
node	node name (also SWC)
port or p	port
pde or o	protected data element (PDE)
direc	direction for which a PDE-type is configured (*rx* or *tx*)

The shortcuts *p* and *o* are used for the E2EPW API functions. For other identifiers (like filenames) the notation *port* and *pde* is preferred.

Some identifiers in this document include concatenations of these terms.
E.g., *E2EPW_Write_p_o_direc ()* (i.e., ".. for a combination of some port *p*, some data element *o* and some direction *direc*).
A protected area (PA) is the container of configuration data for a certain protected data element (PDE).

14.1 Safety features

SMI-143
This component provides the following safety features:

- Creation of end-to-end protection for data
- Check of end-to-end protection for data

14.2 Configuration constraints

SMI-124
The user of MICROSAR Safe shall correctly select and configure I-PDUs and their signal groups (protected areas) that are to be protected by the end-to-end protection. That is, the selection and configuration shall be made according to the communication safety requirements of the system. This includes also the selection of the E2E profile.

SMI-125
The user of MICROSAR Safe shall only configure PAs for the same ECU in one E2EConfig.
If source code needs to be generated for different ECUs, then an individual E2EConfig file must be defined for each ECU.

14.3 Additional verification measures

14.3.1 Additional verification of E2EConfig file

SMI-126
The user of MICROSAR Safe shall ensure that the length of the following identifiers can be uniquely identified by the compiler and linker.

The following strings are internal identifiers:

- E2EPW_Marshal_pde_h
- E2EPW_CheckDeserial_pde_h
- E2EPW_node_port_pde_RX_H
- E2EPW_node_port_pde_TX_H

The following strings are external identifiers:

- E2EPW_Marshal_pde
- E2EPW_CheckDeserial_pde
- E2EPW_Init_p_o_rx
- E2EPW_Init_p_o_tx
- E2EPW_Get_SenderState_p_o
- E2EPW_Get_ReceiverState_p_o
- E2EPW_Write_p_o
- E2EPW_Read_p_o

This requirement applies for all PAs in the E2EConfig with *node*, *p(ort)* and *pde/o*. The number of significant initial characters of the compiler and linker can usually be found in their documentation.

SMI-127
The user of MICROSAR Safe shall verify that all PAs in an E2EConfig with the same *PDE_Name* have the same values in the following fields:

- PDE_Type
- Byte_Order_CPU
- Bit_Order
- Bit_Counting

- Unused_Bit_Value
- Category
- Data_Length
- Is_Opaque

SMI-128
The user of MICROSAR Safe shall verify that each signal in a PA is also be defined in all other PAs with same *PDE_Name*, and the signals must also have the same values in the following fields:

- Signal_Name
- Signal_Type
- Signal_Property
- Byte_Order
- Bit_Length
- Bit_Position

SMI-131
The user of MICROSAR Safe shall verify that the attributes of each PA are defined in the E2EConfig as intended.

Attribute	**Requirement**
PDE_Type	*PDE_Type* shall equal the data type of the corresponding PDE in the AUTOSAR application and the RTE functions that are invoked by the E2EPW code.
Node_Name	*Node_Name* shall equal the name of the SWC. The preprocessor sets *Node_Name* to the SWC's name. If you do not use the preprocessor or manipulate the E2EConfig file, then make sure that the requirement is still met.
Direction	*Direction* shall equal the direction expected for the corresponding PDE in the RTE.
Byte_Order_CPU	*Byte_Order_CPU* shall equal the byte order of the used CPU.
Unused_Bit_Value	*Unused_Bit_Value* shall equal the fillbit configured in the COM layer for the I-PDU representation of the PDE.
Check_DeSerial	*Check_DeSerial* shall be set to YES if it is required that the E2EPW code checks for deserialization errors on the receiver side.
Includes_H	*Includes_H* shall list the header file that defines the data type of the PDE, the header file that defines the type Rte_Instance (if used) and the header file that defines RTE_E_OK.
Includes_C	*Includes_C* shall list the header file(s) that declare(s) the

	functions Rte_Read_p_o (), Rte_Write_p_o () and Rte_IsUpdated_p_o () (if used). This requirement only applies if the header files are not already included in the field "Includes_H".
Category	*Category* shall equal the short name of the E2Elib profile selected for the PDE.
Data_Length	*Data_Length* shall equal the bit-length of the I-PDU representation of the PDE.
Data_ID and *Data_ID_List*	*Data_IDs and _Data_ID_List* shall fulfill the requirements of the respective E2E profile.
Max_Delta_Counter_Init	*Max_Delta_Counter_Init* shall equal the required initial value of MaxDeltaCounter of the E2E library communication state for the PDE.
Data_Id_Mode	*Data_Id_Mode* shall equal the requirements from the respective E2E profile.
Offset	*Offset* shall equal the communication specification by the OEM.
Max_No_New_Or_Repeated_Data	*Max_No_New_Or_Repeated_Data* shall equal the requirements from the respective E2E profile.

SMI-130
The user of MICROSAR Safe shall verify that each signal that is configured in a PA shall be assigned to a signal in the corresponding PDE in the RTE. Vice versa, each signal in the PDE in the RTE shall be assigned to a signal in the corresponding PA.

If a signal in the RTE is not configured for protection, it will not be protected.
Note that the set of signals in the PA represents the corresponding signal group in the PDE.

Attribute	**Requirement**
Port_Name	*Port_Name* shall equal the Port Prototype name that is used by the RTE for the related data element in the RTE.
VDP_Name	*VDP_Name* shall always be equal to *PDE_Name*. *VDP_Name* shall equal the VDP name that is used by the RTE for the related data element in the RTE.
Is_Opaque	*Is_Opaque* shall always be FALSE. The option *Is_Opaque* is for future versions of the E2EPW. It allows to pass PAs as UINT8-arrays without the necessity of marshaling. The E2EPW then treats the data element as opaque (no marshaling is performed).
Use_Call_By_Ref	*Use_Call_By_Ref* shall be YES if the data element is not a primitive data type or configured for reception. If YES, then a pointer to the data element is passed as an argument to *Rte_Read_p_o()* and *Rte_Write_p_o()*, respectively ("call by reference"). If NO, then the value of data element is passed ("call by value"). Note that for *Rte_Read_p_o()*, *Use_Call_By_Ref* is always YES.
Use_Rte_Update	*Use_Rte_Update* shall be YES if and only if the protection wrapper shall check for new received data elements on the RTE level before calling *Rte_Read_p_o()*. If YES, then *Rte_Read_p_o()* is only called if

	Rte_IsUpdated_p_o() returns TRUE before. If NO, then *Rte_Read_p_o()* is always called. Note that *Rte_IsUpdated_p_o()* is only available in AUTOSAR 4.0 or higher.
Use_Rte_Instance	*Use_Rte_Instance* shall be YES if and only if the *instance* parameter is required as an argument for *Rte_Write_p_o()*, *Rte_Read_p_o()* and *Rte_IsUpdated_p_o()*. Note that the *instance* parameter is currently not interpreted by the E2EPW, hence different instances share the same E2EPW and E2E library code and data. Therefore, *Use_Rte_Instance* may only be YES if each port *p* and data element *o* is not used by more than one instance.

The user of MICROSAR Safe shall verify that the combination of *Port_Name* and *VDP_Name* is unique over all PDEs for the same node.

SMI-132

The user of MICROSAR Safe shall verify that the attributes of each signal in a PA are defined in the E2EConfig as intended.

Attribute	**Requirement**
Signal_Name	*Signal_Name* shall equal the name of the corresponding signal in the data element in the RTE.
Signal_Type	*Signal_Type* shall equal the type of the corresponding signal in the data element in the RTE.
Byte_Order	*Byte_Order* shall equal the byte order that is defined for the corresponding signal in the I-PDU for the I-PDU mapping in the COM layer.
Bit_Length	*Bit_Length* shall equal the bit length of the corresponding signal in the I-PDU.
Bit_Position	*Bit_Position* shall equal the start-bit position of the corresponding signal relative to the protected signal group (protected area) and depending on the endianess. If the signal is mapped with Little Endian in the I-PDU, then *Bit_Position* is the least significant bit's position in the least significant byte. If the signal is mapped with Big Endian in the I-PDU, then *Bit_Position* is the most significant bit's position in the most significant byte. For a signal of type UINT8N, the *Bit_Position* is the least significant bit's position in the byte with the lowest address.

SMI-133

The user of MICROSAR Safe shall verify that for all instances of a PDE of the corresponding PA in the various E2EConfig files,

- the profile configurations are equal throughout all PA instances (except *Max_Delta_Counter_Init*),
- all signal configurations are equal throughout all PA instances (except *Signal_Name* and *Signal_ID*) and
- the following field values are equal throughout all PA instances:
  - *Bit_Order*
  - *Bit_Counting*
  - *Unused_Bit_Value*

For example, a sender PDE and a receiver PDE on different ECUs must be configured in different E2EConfig files.

14.3.2 Additional verification of generator execution

SMI-134
The user of MICROSAR Safe shall ensure that the output path for the generated E2EPW code (command-line argument "outpath-path") shall be empty before the generator is started.
If the output path is not empty, code from previous generation runs may be accidentally integrated into the system.
This requirement is fulfilled if generation is performed using DaVinci Configurator PRO.

SMI-135
The user of MICROSAR Safe shall inspect the messages of the E2EPW generator execution.
If the generator aborts the generation process with an error message, the (partially) generated output files shall not be used in the system.
If the generator detects an error, a message starting with "ERROR" is displayed on the standard output.
If the generator shows a warning message starting with "WARNING", the user of MICROSAR Safe shall ensure that the cause of the warning does not invalidate the generated output files.
The generator shows a warning message in the following cases:
1. The E2EConfig file ends with some text after the configuration definitions of all protected areas.
2. The E2EConfig file has at least two PDEs defined with the same *PDE_Name*.
It is strongly recommended to analyze these cases.

SMI-142
The user of MICROSAR Safe shall verify that each generated file is complete, i.e. each file must end with an "EOF" comment.

14.3.3 Additional verification for application

SMI-136
The user of MICROSAR Safe shall verify that the E2E library communication states are not modified by the application while an E2EPW API function is running.
If a pointer to the senders/receivers E2E library communication state is required, it shall be queried with *E2EPW_GetProtectState_p_o() / E2EPW_GetCheckState_p_o()*.
Make sure that the values of the E2E library communication state are only read and not altered by the application or a module other than the E2EPW or the E2E library unless intended so.
For special purposes, modification of the sequence counter by the application may be useful or required. Be aware that this is done in the responsibility of the user of MICROSAR Safe.

SMI-137

The user of MICROSAR Safe shall verify that, when a data element is passed to *E2EPW_Write_p_o()* or *E2EPW_Read_p_o()* using call by reference, then the data element is not altered while *E2EPW_Write_p_o()* or *E2EPW_Read_p_o()* is running.

SMI-141
The user of MICROSAR Safe shall verify that an E2EPW API function for a certain *port / pde* combination is not called while another or the same E2EPW API function is running for the same combination.
This applies to the following functions:

- *E2EPW_WriteInit_p_o()*, Note: *E2EPW_WriteInit_p_o()* is not reentrant
- *E2EPW_Get_ProtectState_p_o()*
- *E2EPW_Write_p_o()*
- *E2EPW_ReadInit_p_o()*, Note: *E2EPW_ReadInit_p_o()* is not reentrant
- *E2EPW_Get_CheckState_p_o()*
- *E2EPW_Read_p_o()*

SMI-138
The user of MICROSAR Safe shall verify that the E2E library communication state is checked for plausibility with *E2EPW_Get[Protect|Check]State_p_o()* after a startup/restart.

SMI-139
The user of MICROSAR Safe shall verify that that *E2EPW_ReadInit_p_o()* and *E2EPW_WriteInit_p_o()* is only called at partition startup/restart.
Note: *E2EPW_ReadInit_p_o()* and *E2EPW_WriteInit_p_o()* have to be called prior to any other E2EPW API function call.
This initializes the communication state on sender and receiver side. The E2E library communication state should be initialized when the application is initialized or reset.

SMI-140
The user of MICROSAR Safe shall verify that the intended parameters are passed. This especially applies to:

- instance IDs. Note: only one instance ID is used for each *p*, *o* combination.
- AppData parameter for *E2EPW_Write_p_o()* and *E2EPW_Read_p_o()*

14.4 Dependencies to other components

14.4.1 Safety features required from other components

SMI-129
This component requires the initialization, protection, mapping and check features from the E2E library as safety features.

14.4.2 Coexistence with other components

SMI-144
This component requires coexistence with the used Rte.
It is assumed that the user of E2EPW has the required ASIL.

14.5 Dependencies to hardware

This component does not use a direct hardware interface.

15 Safety Manual EcuM

15.1 Safety features

SMI-34
EcuM Flex provides the following safety features:

ID	Safety feature
CREQ-470	EcuM shall provide a service to initialize the ECU management.
CREQ-454	EcuM shall provide ECU initialization on multiple cores.
CREQ-543	EcuM shall perform validation of all postbuild configurable BSW module configuration parameters.
CREQ-375	EcuM shall provide a callout to set programmable interrupts during the startup phase.
CREQ-525	EcuM shall provide a callout to initialize driver prior the start of the OS.
CREQ-488	EcuM shall provide a callout to determine the Postbuild configuration data.
CREQ-505	EcuM shall provide a callout to initialize drivers prior Postbuild data is available.
CREQ-391	EcuM shall provide a callout to reset the ECU.
CREQ-381	EcuM shall provide a callout to generate a RAM Hash.
CREQ-440	EcuM shall provide a callout to check the RAM Hash.
CREQ-509	EcuM shall provide a callout to re-initialize drivers during a restart.
CREQ-445	EcuM shall provide a service to set the current shutdown target of the ECU.
CREQ-483	EcuM shall provide a service to get the shutdown target of the ECU.
CREQ-372	EcuM shall provide a service to initiate the ECU shutdown depending on the shutdown target.
CREQ-431	EcuM shall provide a callout to notify Errors from the ECU management.
CREQ-421	EcuM shall provide a service to complete the ECU shutdown process.
CREQ-535	EcuM shall provide a service to initiate an ECU shutdown.
CREQ-699	EcuM shall indicate mode changes to the RTE.
CREQ-	EcuM shall provide a service to request the run state.

691	
CREQ-692	EcuM shall provide a service to release the run state.
CREQ-693	EcuM shall provide a service to release the post run state.
CREQ-690	EcuM shall provide a service to request the post run state.

Note: RAM Hash generation and checking callouts are only available when sleep modes are configured.

SMI-286
EcuM Fix provides the following safety features:

ID	Safety feature
CREQ-470	EcuM shall provide a service to initialize the ECU management.
CREQ-454	EcuM shall provide ECU initialization on multiple cores.
CREQ-543	EcuM shall perform validation of all postbuild configurable BSW module configuration parameters.
CREQ-375	EcuM shall provide a callout to set programmable interrupts during the startup phase.
CREQ-525	EcuM shall provide a callout to initialize driver prior the start of the OS.
CREQ-488	EcuM shall provide a callout to determine the Postbuild configuration data.
CREQ-505	EcuM shall provide a callout to initialize drivers prior Postbuild data is available.
CREQ-391	EcuM shall provide a callout to reset the ECU.
CREQ-381	EcuM shall provide a callout to generate a RAM Hash.
CREQ-440	EcuM shall provide a callout to check the RAM Hash.
CREQ-509	EcuM shall provide a callout to re-initialize drivers during a restart.
CREQ-445	EcuM shall provide a service to set the current shutdown target of the ECU.
CREQ-372	EcuM shall provide a service to initiate the ECU shutdown depending on the shutdown target.
CREQ-431	EcuM shall provide a callout to notify Errors from the ECU management.
CREQ-421	EcuM shall provide a service to complete the ECU shutdown process.
CREQ-699	EcuM shall indicate mode changes to the RTE.

CREQ-703	EcuM shall provide the ECU state machine.
CREQ-707	EcuM shall trigger the NvM write job in shutdown path.
CREQ-668	EcuM shall provide a callback which is called by the NvM to notify the end of the write all job.
CREQ-691	EcuM shall provide a service to request the run state.
CREQ-692	EcuM shall provide a service to release the run state.
CREQ-693	EcuM shall provide a service to release the post run state.
CREQ-690	EcuM shall provide a service to request the post run state.
CREQ-694	EcuM shall provide a service to kill all post run requests.
CREQ-695	EcuM shall provide a service to kill all run requests.

Note: RAM Hash generation and checking callouts are only available when sleep modes are configured.

SMI-38
If EcuM service to complete the shutdown process is called prior to initiate the shutdown process, no shutdown will be performed.

15.2 Configuration constraints

SMI-36
The user of MICROSAR Safe shall configure the following attribute:

- Set /MICROSAR/EcuM/EcuMFlexGeneral/EcuMEnableDefBehaviour to FALSE.
- Do not configure any reference in /MICROSAR/EcuM/EcuMConfiguration/EcuMCommonConfiguration/EcuMWakeupSource/EcuMComMPNCRef to a PNC for a wakeup source

These settings are enforced by MSSV plugins.

15.3 Additional verification measures

SMI-39
The user of MICROSAR Safe shall verify the intended initialization procedure during integration testing.

The user of MICROSAR Safe can verify the intended initialization procedure by performing the following tests:

- Start the ECU and verify that the intended initalization routines are called. This needs to be verified for each Postbuild-selectable configuration.
- Corrupt the Postbuild data (but not corresponding CRC) in non-volatile memory and start the ECU. Then verify that the corruption is detected by EcuM.
- Start the ECU and verify that the intended Postbuild-selectable configuration is used by the EcuM. This needs to be verified for each Postbuild-selectable configuration.

A start of the ECU includes a "cold-start", reset as well as wake-up from sleep if applicable.

This requirement only applies if TSR-1 is considered a safety requirement.

SMI-35
The user of MICROSAR Safe shall verify the intended shutdown procedure during integration testing.

The user of MICROSAR Safe can verify the intended shutdown procedure by shutting down the ECU with all configured shutdown paths. A shutdown path is a call to the service that sets the current shutdown target with a relevant (e.g. combination used to achieve safe state) combination of its parameters. For each shutdown path the intended final state of the ECU (e.g. sleep, shutdown, reset) and the method of reset (e.g. using MCU or Watchdog) is used.

The user of MICROSAR Safe shall also consider the service to initiate the ECU shutdown depending on the shutdown target as a possible shutdown path.

The user of MICROSAR Safe shall also verify the default shutdown target.

This requirement only applies if TSR-3 is considered as a safety requirement.

SMI-40
The user of MICROSAR Safe shall verify that the memory region used for RAM hash generation and verification is as intended.

Verification can be e.g. performed by review.

15.4 Safety features required from other components

SMI-42
The used operating system shall provide the service to get the core ID as safety feature.

If the operating system from MICROSAR Safe is used, this dependency is fulfilled.

This requirement only applies if TSR-1 is considered a safety requirement.

15.5 Dependencies to hardware

This component does not use a direct hardware interface.

16 Safety Manual Fee

16.1 Safety features

This component does not provide safety features.

16.2 Configuration constraints

This component does not have configuration constraints.

16.3 Additional Verification measures

SMI-1292
The user of MICROSAR Safe shall verify that valid notification routines are provided to FEE via configuration.
'FeeNvmJobEndNotification' and 'FeeNvmJobErrorNotification' callbacks are called by FEE using a function pointer.

16.4 Safety features required from other components

This component does not require safety features from other components.

SMI-1643
This component requires coexistence with MemIf, NvM, FlsDrv and Det components if the interface for those components is configured.

16.5 Dependencies to hardware

This component does not use a direct hardware interface.

17 Safety Manual Fr

17.1 Safety features

This component does not provide safety features.

17.2 Configuration constraints

This component does not have configuration constraints.

17.3 Additional verification measures

SMI-345
The user of MICROSAR Safe shall verify that the pointer (*Fr_NmVectorPtr*) passed to the function *Fr_GetNMVector* references a valid memory location and that the size of the array referenced by parameter *Fr_NmVectorPtr* is greater than or equal to the value of the BSWMD parameter
/MICROSAR/FrIf/FrIfConfig/FrIfCluster/FrIfGNetworkManagementVectorLength.
The generated value of the BSWMD parameter can be found in the files:

- Fr_Cfg.h: *FR_NM_VECTOR_LENGTH* and *NmVectorLength* in struct *Fr_VConfigType*
- Fr_LCfg.c: *Fr_NmVectorLength*

This function is called indirectly by the FrNm via *FrIf_GetNMVector*.
This interface only passes a pointer without a length. The length is statically configured in the FrNm.

17.4 Safety features required from other components

This component does not require safety features from other components.

17.5 Dependencies to hardware

The dependencies of this component to hardware is described in the platform specific part of the Safety Manual.

1 Safety Manual Fr (V85x)

1.1 Safety features

This component does not provide safety features.

1.2 Configuration constraints

SMI-1037
The user of MICROSAR Safe shall configure the following attributes:

- Set /MICROSAR/Fr/FrGeneral/FrDirectBufferAccessEnable to FALSE.
- Just a single /MICROSAR/FrIf/FrIfConfig/FrIfCluster container exists.

These settings are enforced by an MSSV plugin.

1.3 Additional verification measures

This component does not require additional verification measures.

1.4 Safety features required from other components

This component does not require safety features from other components.

1.5 Dependencies to hardware

SMI-1034
This component requires exclusive access to the hardware registers of the unit it is intended to control. See the technical reference for the hardware register names and used hardware manuals.

19 Safety Manual FrIf

19.1 Safety features

This component does not provide safety features.

19.2 Configuration constraints

SMI-1103
The user of MICROSAR Safe shall configure the following attributes:

- Set /MICROSAR/FrIf/FrIfGeneral/FrIf3rdPartyDriverSupport to FALSE.
- Set /MICROSAR/FrIf/FrIfGeneral/FrIfWrapperAPIsAsMacro to FALSE.
- Just a single /MICROSAR/FrIf/FrIfConfig/FrIfCluster container exists.

These settings are enforced by a MSSV plugin.

19.3 Additional verification measures

SMI-1102
The user of MICROSAR Safe shall verify that the function referenced by define `FrIf_RxVotingFunction` returns `E_NOT_OK` if all the reduntant Rx PDUs are invalid. A PDU is invalid if `PduInfo->SduDataPtr` is `NULL_PTR` or `PduInfo->SduLength` is `0`.

This measure applies only if define `FRIF_DUALCHANNELREDUNDANCYSUPPORT` is `STD_ON`.

Both define `FrIf_RxVotingFunction` and `FRIF_DUALCHANNELREDUNDANCYSUPPORT` can be found in `FrIf_Cfg.h`.

The values of the defines are derived from `/MICROSAR/FrIf/FrIfGeneral/FrIfRxVotingFunction` and `/MICROSAR/FrIf/FrIfGeneral/FrIfDualChannelRedundancySupport`.

19.4 Safety features required from other components

This component does not require safety features from other components.

19.5 Dependencies to hardware

This component does not use a direct hardware interface.

20 Safety Manual FrSM

20.1 Safety features

This component does not provide safety features.

20.2 Configuration constraints

The functionality "Allow Wake Up Attempts On Both Channels" is in BETA state, see ESCAN00083894.

20.3 Additional verification measures

SMI-332
The user of MICROSAR Safe shall verify that only existing functions with the correct prototype are referred by the following function pointer if the attribute */MICROSAR/FrSM/FrSMGeneral/FrSMSyncLossErrorIndicationName* is configured to an non-empty value

- FrSMSyncLossErrorIndicationFctPtr

The function pointers shall especially not contain numeric values of memory addresses.

All function pointers can be found in *FrSM_Lcfg.c*.

20.4 Dependencies to other components

20.4.1 Safety features required from other components

This component does not require safety features from other components.

20.4.2 Coexistence with other components

SMI-331
This component requires coexistence with Det, Dem, BswM, FrIf, ComM and complex driver components if the interface for those components is configured.

20.5 Dependencies to hardware

This component does not use a direct hardware interface.

21 Safety Manual FrTrans (TJA1082)

21.1 Safety features

No additional safety features are provided.

21.2 Configuration constraints

No additional configuration constraints are required.

21.3 Additional verification measures

No additional verification measures are required.

21.4 Safety features required from other components

No additional require safety features are required from other components.

21.5 Dependencies to hardware

This component does not use a direct hardware interface. A hardware abstraction (Dio driver or user callbacks) is used to access the transceiver hardware.

22 Safety Manual IpduM

22.1 Safety features

This component does not provide safety features.

22.2 Configuration constraints

This component does not have configuration constraints.

22.3 Additional verification measures

This component does not require additional verification measures.

22.4 Safety features required from other components

This component does not require safety features from other components.

22.5 Dependencies to hardware

This component does not use a direct hardware interface.

23 Safety Manual MemIf

23.1 Safety features

This component does not provide safety features.

23.2 Configuration constraints

This component does not have configuration constraints.

23.3 Additional verification measures

This component does not require additional verification measures.

23.4 Dependencies to other components

23.4.1 Safety features required from other components

This component does not require safety features from other components.

23.4.2 Coexistence with other components

SMI-311
This component requires coexistence with NvM, Det, Fee, and Ea components if the interface for those components is configured.

23.5 Dependencies to hardware

This component does not use a direct hardware interface.

24 Safety Manual NvM

24.1 Safety features

SMI-21
This component provides the following safety features:

ID	Safety feature
CREQ-724	NvM shall provide a service to read a single NvM block from NVRAM.
CREQ-725	NvM shall provide a service to write a single NvM block to NVRAM.
CREQ-730	NvM shall provide a service to read all possbile NvM blocks from NVRAM.
CREQ-731	NvM shall provide a service to write all possible NvM blocks to NVRAM.
CREQ-746	NvM shall provide configurable callbacks to synchronize block data.

NvM can detect missing, corruption and masquerading (lower layers provide the wrong block) of NvM blocks.

SMI-29
The user of MICROSAR Safe must design the system in a way that in case of the absence of non-volatile data it is still safe (e.g. safe state or degradation). It cannot be assured by the memory stack that data is saved completely or at all because a reset or loss of energy might happen at any time, e.g. brown-out, black-out.
This also implies that it is in general impossible to guarantee that the latest information is available in the non-volatile memory, e.g. the system is reset before memory stack is even notified to write data to non-volatile memory.
Thus, safety-related functionality may not rely on the availability of data in non-volatile memory.
Since the availability of data in non-volatile memory cannot be guaranteed in any case, the only sensible use-case is reading safety-related calibration data.
Writing of data into non-volatile memory must be verified to assure that the information is available in non-volatile memory. Verification can only be done manually in a protected environment, e.g. at end of line, in a workshop, etc.
ECU software cannot verify if data was written, since at any time a reset could occur and the information that had to be written is lost immediately.
Reading of data does not modify data stored in non-volatile memory. Thus, reading can be used by safety-related functionality. The memory stack assures that the read data is identical to the data stored in non-volatile memory.
The availability may be increased by e.g. redundant storage.

SMI-51478
The user of MICROSAR Safe must ensure that the selected CRC is suitable for the ASIL and data size.

The NvM uses a CRC32 to ensure integrity of data (see also SMI-25). Since fault detection capabilities depend on the CRC polynomial and maximum data size, this needs to be considered during definition of safety-related NvM blocks.

Please note that also the block ID with 3 bytes is taken into account for CRC calculation (see SMI-26).

The NvM may also use the CRC to decide whether data in RAM is modified and needs to be written.

24.2 Configuration constraints

SMI-25
The user of MICROSAR Safe shall configure and verify the following attributes for **each NvM block that contains safety-related data**:

- Set /MICROSAR/NvM/NvMBlockDescriptor/NvMBlockUseCrc to TRUE.
- Set /MICROSAR/NvM/NvMBlockDescriptor/NvMBlockCrcType to NVM_CRC32.

Search for the NvM block to verify in NvM_Cfg.c (use the NvMBlockDescriptor short name for textual search, or the NvMNvramBlockIdentifier as index in NvM block array NvM_BlockDescriptorTable_at) - the block structure shall have NVM_BLOCK_CRC_32_ON set.

SMI-26
The user of MICROSAR Safe shall configure and verify the following attribute:

- Set /MICROSAR/NvM/NvMCommon/NvMUseBlockIdCheck to TRUE.

This setting is enforced by MSSV plugin.

24.3 Additional verification measures

SMI-22
The user of MICROSAR Safe shall pass the intended block ID for reading and writing of a single NvM block. NvM cannot detect if an unintended block that is configured is provided by the user.
Verification can be performed during integration testing.

SMI-23
The user of MICROSAR Safe shall verify that the buffer passed for reading and writing of a single NvM block is valid and sufficiently large for the passed block ID.
Verification can be performed by a review of the generated configuration and the code passing the pointer and block ID to the NvM.

SMI-48
The user of MICROSAR Safe shall verify the size of the internal NvM buffer.

The buffer has the symbol name NvM_InternalBuffer_au8.
The buffer is generated when at least one of the following features is used:

- at least one block with explicit synchronization is configured

- repair of redundant blocks is enabled
- NvM internal CRC buffer is enabled

The buffer size shall be at least the size of the largest NVM block plus the size of the configured CRC value.

Verification can be performed e.g. by review.

24.4 Safety features required from other components

SMI-28
The used Crc library shall provide the CRC calculation routines as safety feature.
If the Crc library from MICROSAR Safe is used, this dependency is fulfilled.

24.5 Dependencies to hardware

This component does not use a direct hardware interface.

25 Safety Manual OS

25.1 Safety features

SMI-1259
This component provides the following safety features:

ID	Safety feature
CREQ-63	OS shall provide a service to start the OS.
CREQ-162	OS shall provide a service to initialize itself.
CREQ-51	OS shall automatically start a subset of alarms for an application mode.
CREQ-146	OS shall automatically start a subset of tasks for an application mode.
CREQ-72	OS shall automatically start a subset of schedule tables for an application mode.
CREQ-117	OS shall provide a service to get the current application mode.
CREQ-45	OS shall provide a global callback during OS startup.
CREQ-299	Os shall synchronize the startup in multicore systems.
CREQ-153	OS shall provide a service to shutdown the OS.
CREQ-95	OS shall provide a service to shutdown all cores synchronously.
CREQ-161	OS shall provide a global callback upon shutdown.
CREQ-71	OS shall provide a global callback directly before a task starts its execution.
CREQ-165	OS shall provide a global callback directly before a task finishes its execution.
CREQ-42	OS shall provide a service to activate a task.
CREQ-28	OS shall handle multiple activations of basic tasks.
CREQ-101	OS shall provide a service to terminate the calling task.
CREQ-121	OS shall provide a service to define the next activated task.
CREQ-126	OS shall provide a service to explicitly invoke the scheduler.
CREQ-80	OS shall provide a service to get the ID of the current task
CREQ-74	OS shall provide a service to get the state of a given task.
CREQ-135	OS shall provide a service to declare a task.
CREQ-115	OS shall provide a service to execute a callback in category 2 ISRs.
CREQ-16	OS shall provide a service to get the ID of the currently executing category 2 ISR.
CREQ-24	OS shall handle unconfigured interrupt sources.
CREQ-78	OS shall provide a service to determine the interrupt source of a non-configured interrupt upon handling of such interrupt.
CREQ-154	OS shall provide a nestable service to disable all interrupts.
CREQ-98	OS shall provide a nestable service to enable all interrupts.
CREQ-151	OS shall provide a nestable service to disable all category 2 interrupts.
CREQ-82	OS shall provide a nestable service to enable all category 2 interrupts.
CREQ-111	OS shall provide a non nestable service to disable all interrupts.
CREQ-43	OS shall provide a non nestable service to enable all interrupts.

CREQ-1257	OS shall provide a non nestable service to disable all interrupts callable from kernel mode
CREQ-1259	OS shall provide a non nestable service to enable all interrupts callable from kernel mode
CREQ-1256	OS shall provide a non nestable service to disable all interrupts callable from user mode
CREQ-1258	OS shall provide a non nestable service to enable all interrupts callable from user mode
CREQ-108741	OS shall provide a non nestable service to disable all interrupts callable from any mode.
CREQ-108742	OS shall provide a non nestable service to enable all interrupts callable from any mode.
CREQ-108744	OS shall provide a non nestable service to disable all category 2 interrupts callable from kernel mode.
CREQ-108747	OS shall provide a non nestable service to enable all category 2 interrupts callable from kernel mode.
CREQ-108743	OS shall provide a non nestable service to disable all category 2 interrupts callable from user mode.
CREQ-108748	OS shall provide a non nestable service to enable all category 2 interrupts callable from user mode.
CREQ-108745	OS shall provide a non nestable service to disable all category 2 interrupts callable from any mode.
CREQ-108746	OS shall provide a non nestable service to enable all category 2 interrupts callable from any mode.
CREQ-106181	OS shall provide a service to disable a specific interrupt source.
CREQ-106182	OS shall provide a service to enable a specific interrupt source.
CREQ-106183	OS shall provide a service to clear pending interrupts
CREQ-114872	OS shall provide a service to check whether or not the source of the given ISR is enabled
CREQ-114873	OS shall provide a service to check whether or not the given ISR has been requested
CREQ-68	OS shall provide a service to wait for the occurrence of events.
CREQ-155	OS shall provide a service to signal the occurrence of events to a task.
CREQ-53	OS shall provide a service to acknowledge the occurrence of events.
CREQ-129	OS shall provide a service to get the event states of a given task.
CREQ-133	OS shall provide a service to declare an event.
CREQ-22	OS shall provide a service to increment a counter.
CREQ-156	OS shall provide a service to get the current value of a counter.
CREQ-39	OS shall provide a service to get the difference between a given and the current counter value.
CREQ-44	OS shall provide a service for each hardware counter to translate a given period of

	time into number of ticks.
CREQ-297	OS shall provide a service for each hardware counter to translate number of counter ticks into a period of time.
CREQ-1260	OS shall provide a service to get the maximum possible value of a counter
CREQ-1261	OS shall provide a service to get the number of underlying driver ticks required to reach a specific unit
CREQ-1262	OS shall provide a service to get the minumum allowed number of ticks for a cyclic alarm of a counter
CREQ-116	OS shall provide a service to set a relative alarm.
CREQ-29	OS shall provide a service to set an absolute alarm.
CREQ-93	OS shall provide a service to get an alarm.
CREQ-164	OS shall provide a service to cancel an alarm.
CREQ-19	OS shall provide a service to get the alarm base.
CREQ-142	OS shall provide alarm callbacks.
CREQ-32	OS shall provide a service to declare an alarm.
CREQ-61	OS shall provide a service to start a schedule table at a relative value.
CREQ-136	OS shall provide a service to start a schedule table at an absolute value.
CREQ-96	OS shall provide a service to stop the processing of a schedule table.
CREQ-112	OS shall provide a service to switch the processing between different schedule tables.
CREQ-100	OS shall provide a service to start an explicitly synchronized schedule table synchronously.
CREQ-152	OS shall provide a service to synchronize a schedule table with a synchronization counter.
CREQ-25	OS shall provide a service to stop synchronization of a schedule table.
CREQ-108	OS shall provide a service to query the state of a schedule table.
CREQ-36	OS shall provide a mechanism to coordinate concurrent access to shared resources.
CREQ-56	OS shall provide a service to acquire a resource.
CREQ-107	OS shall provide a service to release a resource.
CREQ-163	OS shall provide a service to declare a resource.
CREQ-17	OS shall provide a service to acquire a spinlock.
CREQ-139	OS shall provide a service to asynchronously acquire a spinlock.
CREQ-167	OS shall provide a service to release a spinlock.
CREQ-172	OS shall provide a service to determine the application ID to which the current execution context was configured.
CREQ-60	OS shall provide a service to determine the application ID in which the current execution context is executed.
CREQ-114	OS shall provide a service to make an application accessible.
CREQ-109	OS shall provide a service to identify accessibility of OS objects .
CREQ-18	OS shall provide a service to identify object ownership.
CREQ-110	OS shall provide a service to terminate an application.

CREQ-170	OS shall provide restart tasks.
CREQ-104	OS shall provide a service to get the state of a given application.
CREQ-34	OS shall provide a service to call exported services from trusted applications.
CREQ-115372	OS shall allow usage of exported services from trusted applications before start of the OS.
CREQ-105586	OS shall provide a service to call exported services from non-trusted applications.
CREQ-105587	OS shall allow usage of exported services from non-trusted applications before start of the OS.
CREQ-48	OS shall provide an application specific callback during OS startup.
CREQ-76	OS shall provide an application specific callback during OS shutdown.
CREQ-54	OS shall provide an application specific callback if an error occurs.
CREQ-73	OS shall provide a service to return the access rights of a memory access of a task.
CREQ-13	OS shall provide a service to return the access rights of a memory access of a category 2 ISR.
CREQ-49	OS shall provide execution time protection.
CREQ-85	OS shall provide inter-arrival time protection.
CREQ-31	OS shall provide locking time protection.
CREQ-845	OS shall monitor execution times.
CREQ-846	OS shall monitor inter arrival time frames.
CREQ-847	OS shall monitor locking times.
CREQ-35	OS shall provide a service to modify a value in a peripheral region.
CREQ-79	OS shall provide a service to read a value from a peripheral region.
CREQ-145	OS shall provide a service to write a value in a peripheral region.
CREQ-115373	OS shall allow usage of services for peripheral regions before start of the OS.
CREQ-26	OS shall be able to call a global callback function if an error occurs.
CREQ-38	OS shall be able to call a global callback function if a fatal error occurs
CREQ-97	OS shall provide a service to all configured error callbacks, which return the parameters of the system service which triggered error handling.
CREQ-23	OS shall provide a service to all configured error callbacks, which returns the service identifier where the error has been risen.
CREQ-102	OS shall provide information to determine the service and the cause of a reported error.
CREQ-129663	OS shall provide a service, which writes the context of the thread to which the system returns after error handling.
CREQ-129664	OS shall provide a service, which returns the context of the thread which triggered a fatal error.
CREQ-70	OS shall provide a service to forcibly terminate a task.
CREQ-21	OS shall provide a service to forcibly terminate a category 2 ISR.
CREQ-168	OS shall provide a service to select the idle mode action.
CREQ-150	OS shall provide a service to write data to an unqueued IOC channel.

CREQ-55	OS shall provide a service to read data from a unqueued IOC channel.
CREQ-91	OS shall provide a service to send data to a queued IOC channel.
CREQ-160	OS shall provide a service to receive data from a queued IOC channel.
CREQ-90	OS shall provide a service to write multiple data to an unqueued IOC channel.
CREQ-147	OS shall provide a service to read multiple data from an unqueued IOC channel.
CREQ-119	OS shall provide a service to send multiple data to a queued IOC channel.
CREQ-113	OS shall service a method to receive multiple data from a queued IOC channel.
CREQ-128	OS shall provide a service to clear all data from a queued IOC channel.
CREQ-141	OS shall be able to call a callback function upon IOC data reception.
CREQ-149	OS shall provide a service to identify the local core.
CREQ-148	OS shall provide a service to get the number of cores controlled by OS.
CREQ-37	OS shall provide a service to start a core for usage of AUTOSAR OS software.
CREQ-120	OS shall provide a service to start a core for usage of non AUTOSAR OS software.
CREQ-115996	OS shall be able to initialize itself and the hareware on any of the available cores.
CREQ-115010	OS shall provide a callback for signalling a task activation.
CREQ-115028	OS shall provide a callback for signalling the setting of an event.
CREQ-115029	OS shall provide a callback for signalling a thread switch.
CREQ-115030	OS shall provide a callback for signalling forcible termination of a thread.
CREQ-115031	OS shall provide a callback for signalling the acquirement of a resource.
CREQ-115032	OS shall provide a callback for signalling the release of a resource.
CREQ-115033	OS shall provide a callback for signalling the attempt to acquire a spinlock.
CREQ-115034	OS shall provide a callback for signalling the acquirement of a spinlock
CREQ-115035	OS shall provide a callback for signalling the release of a spinlock.
CREQ-115036	OS shall provide a callback for signalling the attempt to internally acquire a spinlock.
CREQ-115037	OS shall provide a callback for signalling the internal acquirement of a spinlock
CREQ-115038	OS shall provide a callback for signalling the internal release of a spinlock.
CREQ-115039	OS shall provide a callback for signalling the locking of interrupts.
CREQ-115040	OS shall provide a callback for signalling the release of an interrupt lock.

CREQ-140268	OS shall provide a callback for signalling failed task activation because the number of activations exceeds the limit.
CREQ-140269	OS shall provide a callback for signalling that event is already set when WaitEvent is called.

25.2 Configuration constraints

SMI-378
The user of MICROSAR Safe shall configure and verify the extended OS status of APIs.

The attribute */MICROSAR/Os_Core/Os/OsOs/OsStatus* shall equal to *EXTENDED*.

The OS safety measures rely on the validity checks defined for EXTENDED status of OS API calls. Without these checks invalid calls might destroy the system integrity and violate safety requirements. Ensuring the validity of API calls and arguments in STANDARD status for any caller (which e.g. might be QM software) is considered to be infeasible.

SMI-377
The user of MICROSAR Safe shall configure and verify the service protection.

The attribute */MICROSAR/Os_Core/Os/OsOs/OsServiceProtection* shall equal to *TRUE*.

The OS safety measures rely on the validity checks defined for OsServiceProtection enabled. Without these checks API invalid calls might destroy the system integrity and violate safety requirements. Ensuring the validity of API calls with OsServiceProtectiondisabled for any caller (which e.g. might be QM software) is considered to be infeasible.

SMI-379
The user of MICROSAR Safe shall configure and verify the scalability class 3 or 4.

The attribute */MICROSAR/Os_Core/Os/OsOs/OsScalabilityClass* shall equal to *SC3* or *SC4*.

The OS safety measures rely on memory protection and service protection provided by the scalability classes SC3 and SC4. Without memory protection, all software parts (even QM parts) would have to ensure freedom from interference regarding memory (including absence of stack overflow). Without service protection, all software parts (even QM parts) would have to ensure only calls with valid access rights.

SMI-385
The user of MICROSAR Safe shall not use ISRs of category 1 if timing protection is configured.

If a thread is killed because of timing protection, ISRs of category 1 might be aborted.

A possible workaround is using ISRs of category 2 instead of category 1.

25.3 Additional verification measures

SMI-380
The user of MICROSAR Safe shall ensure the correct usage of the OS regarding program flow. The correct program flow is ensured only if all OS API functions are correctly used according to the AUTOSAR OS specification, according to the technical reference and according to the requirements of the user application.

SMI-3732
The user of MICROSAR Safe shall ensure the correct usage of the hardware.
It is assumed that user software uses the microcontroller exactly as specified in the vendors hardware documentation.

SMI-383
The user of MICROSAR Safe shall not call OS hook functions. The OS hook functions shall be called by the OS only. This applies to the following hook functions:

- *StartupHook*
- *ShutdownHook*
- *ProtectionHook*
- *PreTaskHook*
- *PostTaskHook*
- *ErrorHook*

The OS makes assumptions which are valid if these hook functions are called by the OS (e.g. set a hook context). These assumptions might be violated if the hook functions are called directly by the user. As a hook may expect, that it is called within a specific context, hooks shall not be called from directly from user code.

SMI-1047
The user of MICROSAR Safe shall ensure that the context definition as described in the Technical Reference is complete for his application. Only this context is preserved on context switches.

25.3.1 Interrupt handling

SMI-381
The user of MICROSAR Safe shall ensure the correct usage of the OS regarding interrupt disabling. Unintended disabling of interrupts may lead to timing inconsistency because pending interrupts might be delayed.

The following interrupt disabling API functions shall be used correct according to the AUTOSAR OS specification and according to the requirements of the user application, otherwise the correct functionality is not ensured:

- *DisableAllInterrupts*

- *SuspendAllInterrupts*
- *SuspendOSInterrupts*
- *DisableLevel*

SMI-382
The user of MICROSAR Safe shall ensure the correct usage of the OS regarding interrupt enabling. Unintended enabling of interrupts may lead to timing inconsistency (because interrupts might occur which should be disabled) and data inconsistency (see also SMI-11). The user shall ensure that timing inconsistencies are detected or avoided.

The following interrupt enabling API functions shall be used correct according to the AUTOSAR OS specification and according to the requirements of the user application, otherwise the correct functionality is not ensured:

- *EnableAllInterrupts*
- *ResumeAllInterrupts*
- *ResumeOSInterrupts*
- *EnableLevel*

SMI-482
The user of MICROSAR Safe shall verify that API functions *DisableLevel*, *EnableLevel*, *DisableGlobal* and *EnableGlobal* are never called in the following cases:

- if interrupts are disabled
- within critical sections
- nested within other interrupt APIs
- within interrupt resources
- within interrupt locking spinlocks
- within ISRs, Hook functions, non-trusted functions, trusted functions and alarm callbacks

SMI-488
The user of MICROSAR Safe shall verify that the following API functions are called from privileged mode only:

- *DisableLevelKM*
- *EnableLevelKM*
- *DisableGlobalKM*
- *EnableGlobalKM*

SMI-489
The user of MICROSAR Safe shall verify that the API function *EnableGlobal* is called only if interrupts are disabled before by call of *DisableGlobal*.

SMI-490
The user of MICROSAR Safe shall verify that the API function *EnableLevel* is called only if interrupts are disabled before by call of *DisableLevel*.

SMI-44677
The user of MICROSAR Safe shall verify that all APIs called in ISRs of category 0 are allowed to be called in this context by the AUTOSAR specification or technical reference.

ISRs of category 0 are transparent to the OS. Therefore the call context „inside category 0 ISR“ cannot be checked by the API functions. Erroneous calls are not detected.

SMI-590
The user of MICROSAR Safe shall verify that all APIs called in ISRs of category 1 are allowed to be called in this context by the AUTOSAR specification or technical reference.

ISRs of category 1 are transparent to the OS. Therefore the call context „inside category 1 ISR“ cannot be checked by the API functions. Erroneous calls are not detected.

SMI-44676
The user of MICROSAR Safe shall verify that all ISRs of category 0 are implemented transparent with respect to the processor state for the interrupted code. This includes core registers, MPU settings and the current interrupt priority.

SMI-541
The user of MICROSAR Safe shall verify that all ISRs of category 1 are implemented transparent with respect to the processor state for the interrupted code. This includes core registers, MPU settings and the current interrupt priority.

SMI-491
The user of MICROSAR Safe shall verify the functionality of each configured ISR.

This includes the correct call of the ISR handler with the correct priority (level) as well as enabling, disabling, reading the enable state, reading the pending state and clearing of the pending information of the corresponding ISR sources.

SMI-44675
The user of MICROSAR Safe shall be aware that category 0 ISRs can be interrupted by the OS in case of ECC violations.
In case that ECC violations are handled by the MICROSAR OS, the *ProtectionHook* is called for ECC violations.
The protection handling can be interrupted by a category 0 ISR. This also applies if the protection handling was triggerd by the same category 0 ISR.
If the protection reaction is terminate Task, ISR or Application the category 0 ISR will be terminated by the OS, as well.

SMI-44678

The user of MICROSAR Safe shall be aware that category 1 ISRs can be interrupted by the OS in case of ECC violations.
In case that ECC violations are handled by the MICROSAR OS, the *ProtectionHook* is called for ECC violations.
If the protection reaction is terminate Task, ISR or Application the category 1 ISR will be terminated by the OS, as well.

SMI-44680
The user of MICROSAR Safe shall be aware that category 0 ISRs can be interrupted by the OS in case of exceptions.
In case that unhandled or handled exceptions are managed by the MICROSAR OS, the *ProtectionHook* is called.
The protection handling can be interrupted by a category 0 ISR. This also applies if the protection handling was triggerd by the same category 0 ISR.
If the protection reaction is terminate Task, ISR or Application the category 0 ISR will be terminated by the OS, as well.

SMI-44681
The user of MICROSAR Safe shall be aware that category 0 ISRs cannot be disabled or suspended by the OS interrupt APIs.
The following APIs have no effect on category 0 ISRs:

- *DisableAllInterrupts*
- *EnableAllInterrupts*
- *SuspendAllInterrupts*
- *ResumeAllInterrupts*
- *SuspendOSInterrupts*
- *ResumeOSInterrupts*
- *Os_DisableLevelAM*
- *Os_EnableLevelAM*
- *Os_DisableLevelKM*
- *Os_EnableLevelKM*
- *Os_DisableLevelUM*
- *Os_EnableLevelUM*
- *Os_DisableGlobalAM*
- *Os_EnableGlobalAM*
- *Os_DisableGlobalKM*

- *Os_EnableGlobalKM*
- *Os_DisableGlobalUM*
- *Os_EnableGlobalUM*

25.3.2 Memory mapping and linking

SMI-340
The user of MICROSAR Safe shall verify that the complete range specified by each *Os_PeripheralConfigType* object in *Os_Peripheral_Lcfg.c* is either part of the writable address space or that there are no write accesses to that region via the Peripheral API. The first writable address is denoted as *AddressStart* and the last writable address is denoted as *AddressEnd*.

If the addresses do not fit the intended/configured addresses, illegal write accesses would be possible.

SMI-494
The user of MICROSAR Safe shall verify for IOC functions that the configured access rights and linker configuration allow only valid callers to write the corresponding IOC data.

SMI-495
The user of MICROSAR Safe shall ensure by linkage for each optimized spinlock that only intended tasks and ISRs have write access to the corresponding spinlock data (or at least only tasks and ISRs of partitions with the same ASIL levels).

The user of MICROSAR Safe shall verify that no unintended task or ISR has access to data of optimized spinlocks.

SMI-539
The user of MICROSAR Safe shall verify that none of the configured MPU regions allows write access to OS variables from non-trusted software.

SMI-549
The user of MICROSAR Safe shall verify the linkage of stack sections and MPU configuration that none of the configured MPU regions grants write access to any OS stack.

The MPU setting for stacks is internally done by the OS and granting write access might prevent from memory protection of stacks.

SMI-1044
The user of MICROSAR Safe shall verify that additional configured MPU regions shall never overlap with any OS stack sections.

Overlapping MPU regions might provide illegal write access to OS stack sections. By using an OS generated linker command file (see technical reference) it is assured that the OS stacks are linked consecutively into the RAM.

SMI-1045

The user of MICROSAR Safe shall verify that the linkage scheme includes a stack safety gap linked adjacent to the stack section (in dependency of the stack growth direction, see technical reference). No software parts shall have write access to the stack safety gap.

This measure enables to detect stack overflows by MPU even if the owner of the stack has also write access to data linked adjacent to the stack section.

SMI-562
The user of MICROSAR Safe shall verify that all user data are linked into the intended sections.

SMI-564
The user of MICROSAR Safe shall verify the configuration of access rights to sections. Software with lower diagnostic coverage shall not be able to destroy data of software with higher diagnostic coverage. This applies to memory access within one core as well as memory access across cores.

See Technical Reference, chapter "Memory Protection" for details.

Note that OSApplications do not need access to other OS Applications memory.

25.3.3 Stack

SMI-565
The user of MICROSAR Safe shall ensure that sufficient stack is available for call/execution of StartOS.

StartOS performs some initializations before switching to an internal stack and enabling the memory protection. The active stack at call of StartOS shall provide sufficient space to execute this code. Because the stack consumtion is depending on compiler and compiler options it is recommended to switch to a stack provided by the OS before calling StartOS and to use the stack usage measurement API of the OS to determine the necessary stack size.

SMI-4452
If the attribute */MICROSAR/Os/OsOS/OsGenerateMemMap* is equal to *USERCODE_AND_STACKS_GROUPED_PER_CORE*, the user of MICROSAR Safe shall ensure that all configured stack sizes match the MPU granularity and alignment. Otherwise stack protection cannot be ensured.

25.3.4 Multicore systems with mixed diagnostic coverage capability

SMI-592
The user of MICROSAR Safe shall verify that software with higher diagnostic coverage does not rely on the results of APIs with lower diagnostic coverage.

Note that only APIs listed in section "Safety Features" provide functionality on ASIL level.

SMI-483
The user of MICROSAR Safe shall ensure that cross core API calls with high frequency from cores with lower diagnostic coverage to cores with higher diagnostic coverage do not

interfere with the requirements. Excessive runtime consumption of cores with lower diagnostic coverage shall not prevent cores with higher diagnostic coverage form keeping the timing constraints.

One possible measure is using timing protection.

SMI-484
The user of MICROSAR Safe shall ensure that synchronous cross core API calls from a core with higher diagnostic coverage to a core with lower diagnostic coverage do not violate the safety goals if the API calls never return.

Synchronous calls block the caller until the return result is received. If for any reason a core with lower diagnostic coverage does not return the result or does not return the result in time, the caller has to deal with this situation.

SMI-485
The user of MICROSAR Safe shall call *ShutdownAllCores* only on cores with the highest diagnostic coverage.

SMI-486
The user of MICROSAR Safe shall note that the *ShutdownHook* might not be called on Shutdown for multicore systems with mixed diagnostic coverage capability.

Errors caused by cores of lower diagnostic coverage (data overwrite) might prevent the call of ShutdownHook by cores with higher diagnostic coverage.

SMI-487
The user of MICROSAR Safe shall configure and verify that the core with the highest diagnostic coverage initializes the peripheral moduls used by the OS (e.g. MPU, Interrupt Controller).
The attribute */MICROSAR/Os/OsOS/OsHardwareInitCore* shall be set to the core reference with the highest diagnostic coverage.
The user of MICROSAR Safe shall ensure that if a core with lower diagnostic coverage initializes peripherals or hardware components (like e.g. a system MPU), the core with higher diagnostic coverage does not rely on this initialization.

SMI-493
The user of MICROSAR Safe shall verify that the configuration of cross core API calls prevents cores with lower diagnostic coverage from shutdown of cores with higher diagnostic coverage.

SMI-25766
The user of MICROSAR Safe shall ensure that receiver core of cross core API calls is able to handle unintended calls of APIs. This applies only to APIs which are allowed to be called between two cores by configuration.

25.3.5 Miscellaneous

SMI-480

The user of MICROSAR Safe shall not rely on the error parameter macros (starting with *Os_ErrorGetParameter_*...).
They are not assumed as safety features by Vector.

SMI-481
The user of MICROSAR Safe shall notify that the *PanicHook* might not be called if the active thread is not allowed to modify the interrupt enable/disable state.

Before calling the *PanicHook* the OS disables all interrupts. If this fails, the *ProtectionHook* might be called, caused by the illegal access (depending on hardware platform).

SMI-492
The user of MICROSAR Safe shall verify for cross core API calls that for each pair of sender/receiver cores at least one API call is tested and verified across these cores.

SMI-496
The user of MICROSAR Safe shall verify that calls of the optimized spinlock API don't violate any of the spinlock API constraints (e.g. the order of locking). The optimized spinlock API skips any checks and therefore does not prevent from wrong calls.

SMI-497
The user of MICROSAR Safe shall verify that if a trusted or non-trusted function uses the passed argument, the trusted function validates these data before usage to prevent from any violation of safety goals. The caller of CallTrustedFunction or Os_CallNonTrustedFunction and therefore the passed data might be non-trusted.

SMI-542
The user of MICROSAR Safe shall verify that each caller of a trusted or non-trusted function is allowed to call the function, or the function validates the caller before performing its functionality to prevent from any violation of safety goals.

SMI-538
The user of MICROSAR Safe shall verify that the context described in the Hardware Software Interface (HSI) of the used platform is sufficient for the requirements his application.

SMI-591
The user of MICROSAR Safe shall verify the correct usage of IOC API functions. Some of these functions don't call the ErrorHook if the called function does not return a valid result. It is recommended to check the return code of these functions:

- IocSend/IocWrite
- IocSendGroup/IocWriteGroup
- IocReceive/IocRead
- IocReceiveGroup/IocReadGroup

SMI-540

The user of MICROSAR Safe shall verify that the user software does not contain system call instructions.

Any system call instruction will result in an OS API or in an OS Error. If the user code directly uses a system call instruction it is likely that the triggered OS API does not work as expected. Instead, system calls shall only be used by using calls to OS APIs.

A possible verification method might be reviewing the code for (inline) assembler statements, pragmas or intrinsic functions containing system call instructions.

SMI-2900
The user of MICROSAR Safe shall verify that the array OsCfg_CorePhysicalRefs contains all physical cores.
For each existing physical hardware core identifier there shall be one corresponding entry inside the array which is indexed by the physical hardware core identifier provided directly by the hardware registers.

SMI-39288
The user of MICROSAR Safe shall verify that the array OsCfg_Hal_Context_ExceptionContextRef contains all physical cores.
For each existing physical hardware core identifier, which is also an Autosar core, there shall be one corresponding entry (not NULL_PTR) inside the array which is indexed by the physical hardware core identifier provided directly by the hardware registers.

SMI-44342
The user of MICROSAR Safe shall verify that peripheral APIs

- Os_ReadPeripheral8Legacy
- Os_ReadPeripheral16Legacy
- Os_ReadPeripheral32Legacy
- Os_WritePeripheral8Legacy
- Os_WritePeripheral16Legacy
- Os_WritePeripheral32Legacy
- Os_ModifyPeripheral8Legacy
- Os_ModifyPeripheral16Legacy
- Os_ModifyPeripheral32Legacy

are not used on platforms with address width greater than 32 bits.

SMI-44679
The user shall ensure real time behavior of the system, even in case of delayed calls of *ProtectionHook*.
ProtectionHook may be delayed by the execution of Cat 0 ISRs.

25.3.6 Tracing

SMI-69754
The user of MICROSAR Safe shall be aware that user timing hook implementation influences runtime behaviour of the system.

SMI-69755
The user of MICROSAR Safe shall not use any OS API within TimingHooks.

25.4 Safety features required from other components

This component does not require safety features from other components.

25.5 Dependencies to hardware

The dependencies of this component to hardware is described in the platform specific part of the Safety Manual.

26 Safety Manual OS (RH850)

26.1 Safety features

No additional safety features are provided.

26.2 Configuration constraints

SMI-3167
The user of MICROSAR Safe shall enable software stack checks for RH850 derivatives which are based on G3K core (e.g. RH850 F1L) in order to ensure stack protection also in supervisor mode as monitoring by MPU in supervisor mode is not supported.

26.3 Additional verification measures

SMI-3310
The user of MICROSAR Safe shall verify the PIT timer configuration of type *Os_Hal_TimerPitConfigType* in *Os_Hal_Timer_Lcfg.c* for its correctness.

If the OSTM unit <X> (0,1,2,3,5) is configured, the following attributes must be generated as follows:
Timer Base Address = OS_HAL_TIMER_OSTM<X>_BASE
Timer Hardware Type = OS_HAL_TIMER_OSTM
Timer Channel Index = OS_HAL_TIMER_CH0

If the TAUJ unit <X> (0,1,2,3) channel <Y> (0,1,2,3) is configured, the following attributes must be generated as follows:
Timer Base Address = OS_HAL_TIMER_TAUJ<X>_BASE
Timer Hardware Type = OS_HAL_TIMER_TAUJ
Timer Channel Index = OS_HAL_TIMER_CH<Y>

SMI-3311
The user of MICROSAR Safe shall verify the FRT timer configuration of type *Os_Hal_TimerFrtConfigType* in *Os_Hal_Timer_Lcfg.c* for its correctness.

If the OSTM unit <X> (0,1,2,3,5) is configured, the following attributes must be generated as follows:
Timer Base Address = OS_HAL_TIMER_OSTM<X>_BASE
Timer Hardware Type = OS_HAL_TIMER_OSTM
Timer Channel Index = OS_HAL_TIMER_CH0
Timer Unit Index = OS_HAL_TIMER_OSTM<X>

If the STM unit <X> (0,1) channel <Y> (1,2,3) is configured, the following attributes must be generated as follows:
Timer Base Address = OS_HAL_TIMER_STM<X>_BASE
Timer Hardware Type = OS_HAL_TIMER_STM
Timer Channel Index = OS_HAL_TIMER_CH<Y>
Timer Unit Index = OS_HAL_TIMER_STM<X>

26.4 Safety features required from other components

No additional safety features are required from other components.

26.5 Dependencies to hardware

The following Safety Applications Notes (SAN) have been taken into consideration:

- RH850/P1M Safety Application Note R01AN2118EJ0300 Rev.3.00 Mar 29, 2017
- RH850/F1L Safety Application Note R01AN2152EJ0211 Rev.2.11 October 17, 2016
- RH850/F1H Safety Application Note R01AN2886EJ0110 Rev.1.10 July 27, 2016
- RH850/F1K Safety Application Note R01AN3578EJ0100 Rev.1.00 Dec 22, 2016
- RH850/D1L/D1M Safety Application Note LLWEB-10000950 Rev.2.00 Dec 1, 2016

This OS does not implement any recommended usage from safety application notes except for the memory protecion unit (SAN-P1x-0405, SAN-F1L-2201, SAN-F1H-2201, SAN-F1K2M-0190, SAN-D1x-1801).
It is assumed that the recommended usage related to OS functionality are related to latent faults only and ASIL D is still achievable without implementation of the recommended usage by the OS. Such implementation would cause significant runtime overhead.

SMI-3168
The user of MICROSAR Safe has to consider DMA controller usage if the used RH850 derivative incorporates a DMA controller (DMAC).
The DMA controller has direct access to the data bus. Therefore DMA access to memory is not controlled by MPU protection.

27 Safety Manual PduR

27.1 Safety features

This component does not provide safety features.

27.2 Configuration constraints

This component does not have configuration constraints.

27.3 Additional verification measures

This component does not require additional verification measures.

27.4 Safety features required from other components

This component does not require safety features from other components.

27.5 Dependencies to hardware

This component does not use a direct hardware interface.

28 Safety Manual Rte

28.1 Safety features

SMI-323
This component provides the following safety features:

ID	Safety feature
CREQ-1024	Rte shall provide a service to initiate the transmission of data elements with last-is-best semantic for explicit S/R communication.
CREQ-1021	Rte shall provide a service to copy the received data element to a buffer with last-is-best semantic for explicit S/R communication.
CREQ-1022	Rte shall provide a service to get the value of the received data element with last-is-best semantic for explicit S/R communication.
CREQ-1031	Rte shall provide a service to read a data element for implicit S/R communication.
CREQ-1029	Rte shall provide a service to write a data element for implicit S/R communication.
CREQ-1041	Rte shall provide a service to get the reference of a data element to be written for implicit S/R communication.
CREQ-1037	Rte shall provide a service to get the status of a data element for implicit S/R communication.
CREQ-1033	Rte shall provide a service to access the update flag for a data element for explicit S/R communication.
CREQ-1036	Rte shall provide a "Never-received" status of a data element for S/R communication.
CREQ-1023	Rte shall provide a service to initiate the transmission of a data element with queued semantic for explicit S/R communication.
CREQ-1025	Rte shall provide a service to initiate the reception of a data element with queued semantic for explicit S/R communication.
CREQ-1042	Rte shall provide a service to initiate a client-server communication.
CREQ-1043	Rte shall provide a service to get the result of an asynchronous client-server call.
CREQ-1109	Rte shall provide mode management.
CREQ-1055	Rte shall provide a service to get the currently active mode.
CREQ-1052	Rte shall provide a service to get the currently active, previous and next mode.
CREQ-1053	Rte shall provide a service to initiate a mode switch.
CREQ-1299	Rte shall provide Nv data communication.
CREQ-	Rte shall provide a callback to copy data from a NVM buffer to RTE.

1150	
CREQ-1148	Rte shall provide a callback to copy data from RTE to a NVM buffer.
CREQ-1144	Rte shall provide a callback to get notified about a finished NVM job.
CREQ-1147	Rte shall provide a callback to get notified about a requested mirror initialization.
CREQ-1046	Rte shall provide a service to read Inter-Runnable Variables with explicit behavior.
CREQ-1048	Rte shall provide a service to write Inter-Runnable Variables with explicit behavior.
CREQ-1047	Rte shall provide a service to read Inter-Runnable Variables with implicit behavior.
CREQ-1044	Rte shall provide a service to write Inter-Runnable Variables with implicit behavior.
CREQ-1045	Rte shall provide a service to access per-instance memory.
CREQ-1051	Rte shall provide a service to enter an exclusive area.
CREQ-1050	Rte shall provide a service to leave an exclusive area.
CREQ-1056	Rte's Basic Software Scheduler shall provide a service to enter an exclusive area of a BSW Module.
CREQ-1049	Rte's Basic Software Scheduler shall provide a service to leave an exclusive area of a BSW Module.
CREQ-1068	Rte shall provide a service to access internal calibration parameters.
CREQ-1075	Rte shall provide a service to access calibration parameters accessible via ports.
CREQ-1059	Rte shall provide a service to initialize itself.
CREQ-1073	Rte's Basic Software Scheduler shall provide a service to initialize itself.
CREQ-1161	Rte shall provide a service to trigger executable entities.
CREQ-1165	Rte shall use schedule points to invoke the scheduler of the OS.
CREQ-1129	Rte shall provide the event handling of TimingEvents to trigger a runnable.
CREQ-1112	Rte shall provide the event handling of SwcModeSwitchEvents to trigger a runnable.
CREQ-1124	Rte shall provide the event handling of AsynchronousServerCallReturnsEvents to trigger a runnable.
CREQ-1126	Rte shall provide the event handling of OperationInvokedEvents to trigger a runnable.

CREQ-1118	Rte shall provide the event handling of DataReceivedEvents to trigger a runnable.
CREQ-1132	Rte shall provide the event handling of ModeSwitchedAckEvents to trigger a runnable.
CREQ-1114	Rte shall provide the event handling of the InitEvents to trigger a runnable.
CREQ-1295	Rte shall provide the event handling of TimingEvents to trigger a schedulable entity.
CREQ-1152	Rte shall provide a "RTE_AND_SCHM_UNINIT" state.
CREQ-1164	Rte shall provide a "RTE_UNINT_SCHM_INIT" state.
CREQ-1166	Rte shall provide a "INIT" state.

28.2 Configuration constraints

SMI-2066
The user of MICROSAR Safe shall disable online calibration and measurement during series production.

The RTE can be generated with online calibration and measurement enabled for series production, but they shall be made inoperable for normal operation. Vector's XCP can e.g. be disabled safely during runtime by ASIL software.

28.3 Additional verification measures

Please note that the RTE Generator and RTE Analyzer only implement measures to detect systematic faults by the software. No measures are implemented to detect or mitigate random faults on the computer used for generation.

SMI-322
The user of MICROSAR Safe shall execute the RTE Analyzer.

The RTE Analyzer performs checks to identify faults in the generated RTE. Especially out-of-bounds accesses within the RTE are detected. If RTE Analyzer reports a fault, the generated RTE cannot be used. Moreover it provides the user of MICROSAR Safe with feedback what was generated. This feedback shall be reviewed during integration testing against the intended software design and its configuration.

Please see the Technical Reference of the RTE Analyzer how to execute it.

SMI-36067
The user of MICROSAR Safe shall ensure that the RTE Analyzer does not report unsupported templates.

The generated RTE code is based on templates. The templates are instantiated by the RTE Generator in different variants. The RTE Analyzer verifies that the analyzed template variants have been tested during the development of the RTE Generator according to ISO 26262.

The last section of the RTE Analyzer configuration feedback report provides the information about the template variants.
The report must show that no unsupported templates have been found.

28.3.1 Guided integration testing

Residual faults in the RTE Generator can only be found during integration testing. Vector assumes that the user of MICROSAR Safe performs an integration testing and verification of software safety requirements according to ISO 26262 Part 6 Clauses 10 and 11 (see also SMI-4). To support this integration testing the RTE Analyzer produces a configuration feedback report.

Please refer to the Technical Reference of the RTE Analyzer for a description of the configuration feedback report.

The following subsections describe the requirements that must be fulfilled during integration testing and verification of software safety requirements.

Each Safety Manual Item (SMI) is structured in the following way:

- The requirement that must be fulfilled
- Explanation of the requirement and a rationale
- Recommended configuration constraints (optional)
- Recommended means of complying with the requirement (optional)
- Details on the information provided by the RTE Analyzer supporting this requirement

28.3.1.1 BSW configuration

SMI-2124
The user of MICROSAR Safe shall ensure that the RTE and the operating system assume the same scheduling properties.

The scheduling properties of the RTE tasks depend on the configuration of the operating system. The scheduling properties are e.g. preemptability, core assignment or task priority.

The RTE Analyzer lists the scheduling properties in the configuration feedback report to assist in this integration step. The scheduling properties listed in the feedback report shall be verified.

SMI-2129

The user of MICROSAR Safe shall ensure that the assumptions of the operating system and the RTE are the same with regards to the locking behavior of the spinlocks.

The RTE generator uses spinlocks from the operating system to protect inter-core communication. Spinlocks must not be called concurrently on the same core. The operating system optionally provides spinlocks that can prevent these concurrent accesses on the same core. If this protection by the operating system is not used, the RTE generator has to prevent concurrent calls to the spinlock APIs on the same core.

Verification can e.g. be performed by review of the configuration feedback report.

The RTE Analyzer lists spinlocks that are not protected by the RTE to assist in this integration step.

SMI-684
The user of MICROSAR Safe shall ensure that the configuration of COM and RTE are consistent.

The interfaces to the COM that are used for signal reception use *void* pointers as parameter. Inconsistencies between the configuration of the COM and the RTE might lead to memory corruption by the COM. During integration the size assumptions between the COM and the RTE shall be compared.

Verification can be performed by review of the generated configuration and/or static code analysis.

The RTE Analyzer lists relevant calls to assist in this integration step.
The RTE Analyzer listing includes the number of written bytes for MICROSAR COM.

SMI-685
The user of MICROSAR Safe shall ensure that the configuration of NVM and RTE are consistent.

The interfaces to the NVM that are used to handle NV Block SWCs use *void* pointers as parameters. Inconsistencies between the configuration of the NVM and the RTE might lead to memory corruption by the RTE. During integration the size assumptions between the NVM and the RTE shall be compared.

Verification can be performed by review of the generated configuration and/or static code analysis.

The RTE Analyzer lists relevant calls to assist in this integration step.

28.3.1.2 Executable Entity Scheduling

SMI-2063
The user of MICROSAR Safe shall ensure that all safety-related executable entities are triggered with their correct conditions.

These conditions are:

- cylic triggers with cycle time and offset
- init triggers
- background triggers
- triggers fired by RTE APIs

If triggers have a dependency on modes, the scheduling has to be verified for all modes. Modes can be switched with the *Rte_Switch* API.
Triggers can be decoupled by the minimum start interval functionality and the data reception filter functionality.
The scheduling of the executable entities also depends on the configuration of the operating system, the used controller, other running tasks and interrupt service routines and the resource usage of the entities that are implemented by the user.

Vector recommends not using the minimum start interval and the data reception filter functionality for safety-related runnables.
Vector recommends not using background triggers for safety-related functionality.
Vector recommends using cyclic scheduling without mode dependencies and using of a watchdog as safety mechanism for safety-related entities where possible.

Cyclic triggers are e.g. scheduled deterministically. Thus, an integration test verifying that safety-related functionality is scheduled at the expected times may be sufficient.

The RTE Analyzer lists the executable entities of SWCs and the tasks in which they are executed to assist in this integration step.

SMI-2128
The user of MICROSAR Safe shall ensure that reentrant runnables are reentrant.

Runnables can be called reentrantly from multiple tasks. Their implementation needs to support this use case.

Verification can e.g. be performed by review, static code analysis and/or integration testing.

The RTE Analyzer lists all runnables of SWCs that are called from concurrent tasks to assist in this integration step.
Implicit exclusive areas and nonpreemptive tasks can be configured to prevent concurrent execution.

SMI-2064
The user of MICROSAR Safe shall ensure that the timeouts configured for blocking APIs that are used in safety-related executable entities are adequately addressed.

If a timeout for a blocking API is used as a safety mechanism (E.g. no checkpoint with deadline monitoring in the task), the user of MICROSAR Safe shall also ensure that the timeout value is adequate.

Relevant timeouts are:

- timeouts of *Rte_Call* APIs
- timeouts of *Rte_Result* APIs
- timeouts of *Rte_Receive* APIs
- timeouts of *Rte_Feedback* APIs
- timeouts of *Rte_SwitchAck* APIs

The timeouts also depend on the configuration of the operating system, the used controller, other running tasks and interrupt service routines and the resource usage of entities that are implemented by the user.

Vector recommends not using blocking APIs in safety-related entities except for cross partition client-/server communication.
Vector strongly recommends not using blocking APIs without timeout.

A review may be sufficient to verify that timeout handling is implemented properly by the SWC.
If no other safety mechanism is in place, a test that the timeout is notified at the expected time by the RTE can be used as means of verification.

The RTE Analyzer lists the blocking APIs of SWCs to assist in this integration step.

SMI-2122
The user of MICROSAR Safe shall ensure that the correct implementation method has been chosen for every exclusive area.

Exclusive areas can be used to ensure data consistency (see SMI-11).
The implementation depends on the requirements of the application and on other factors like the expected duration of the exclusive area. Interrupt locks are typically faster than resources but can only be used for short sequences due to the blocking of interrupts. Operating system interrupts only block the interrupts of the operating system whereas **all** interrupts blocks all interrupts.

Verification can e.g. be performed by review, static code analysis and/or integration testing.

The RTE Analyzer lists the exclusive areas and their implementation method to assist in this integration step.

28.3.1.3 SWC Communication

SMI-41492
The user of MICROSAR SafeBRE shall provide the RTE APIs for systems without RTE.

MICROSAR Basic Runtime Environment (BRE) only provides the BSW scheduler functionality of the RTE and does not support application SWCs. The implementation of the interface from the application to BSW modules must be developed according to ISO

26262. The Technical References of the BSW modules as well as the AUTOSAR standard define the semantics and APIs that will have to be implemented while integrating the BRE.

The RTE Analyzer does not assist in this integration step.

SMI-324, SMI-2065 and SMI-2123 do not apply to MICROSAR SafeBRE.

SMI-324
The user of MICROSAR Safe shall ensure that the connections between SWCs are as intended.

Many types of faults can lead to a mix of connections between SWCs. These are unlikely and usually already addressed by straight forward integration testing.
The list of senders needs to be correct for every receiver and the subset of the received data needs to be correct.

Vector recommends the following RTE subset for safety-related SWCs:

- use only 1:1 or 1:N connections.
- use the same datatype on both sides of a connection
- avoid data conversion

Information that is used from non-safety-related SWCs has to be checked for plausibility. If such a data path is found during integration this is an indicator that your safety analysis has to be reconsidered. Please note that also other code in the same partition as the non-safety-related SWCs can corrupt the communication if freedom from interference with regards to memory is not ensured.

Verification can be performed by review and/or an integration test testing the normal operation.

The RTE Analyzer list the connections between SWCs to assist in this integration step.

SMI-2065
The user of MICROSAR Safe shall ensure that inter-ECU sender-/receiver and inter-ECU client-/server communication work as expected.

This requires verification of:

- data needs to be routed to the correct ECU by the underlying communication stack. This includes 1:1, 1:N, N:1 and reception of partial signal data.
- both ECUs need to use the same data representation (datatypes, endianness, serialization)

Vector requires using E2E protection for safety-related signals.
Vector recommends using 1:1 connections.
Vector recommends always sending and receiving complete data elements.

Integration testing on vehicle network level using fault-injection can be used. Vector assumes that this is normally done to verify the effectiveness of the E2E protection.

The RTE Analyzer lists the APIs of SWCs with inter-ECU communication to assist in this integration step.

SMI-2123
The user of MICROSAR Safe shall ensure that all connected SWCs expect the same converted data.

The RTE offers conversions that can be applied to specific connections.

Vector recommends not using data conversion for safety-related connections.

Verification can e.g. be performed by review or integration testing of all data conversions.

The RTE Analyzer lists all APIs of SWCs that perform data conversion to assist in this integration step.

28.3.1.4 Usage of RTE Headers

SMI-2067
The user of MICROSAR Safe shall ensure that the *defines* and *typedefs* that are generated by the RTE match the expectations of the SWCs that use them.

Inconsistencies may lead to e.g, memory corruption when a runnable uses an RTE array datatype within its implementation and writes beyond the bounds of this array.
Moreover, different SWCs may have different assumptions with regards to the meaning of communicated values, e.g. if one SWC uses the symbolic name, another SWC the integral value of an enumerated type.

The following code is provided:

- Configured Application Error Defines for Client-/Server Communication
- Configured AUTOSAR Datatypes
- Configured Upper and Lower Limit Defines for Primitive Application Data Types
- Configured Init Value Defines for Sender-/Receiver Communication
- Configured InvalidValue Defines for Sender-/Receiver Communication
- Configured Enumeration Defines for CompuMethods
- Configured Mask Defines for CompuMethods
- Configured Mode Defines for Mode Communication
- Configured ActivationReasons

The defines and typedefs are part of the *Rte_Type.h*, *Rte_*<SWC>*.h* and *Rte_*<SWC>_Type*.h* headers.

Vector recommends using the headers generated by the RTE when a static code analysis is performed on the application code.
Vector recommends only using the *defines* instead of the defined values in the code.
Vector recommends using the *defines* only when needed (Mode, Application Error Defines).
Vector recommends reviewing the used *defines* for safety-related SWCs.
Vector recommends not using union types in the SWCs.

Verification for correct usage of datatypes may be performed by review and/or static code analysis. Consistent usage of *defines* can be verified by review and/or integration testing with all used values.

The RTE Analyzer verifies that all accesses within the RTE do not lead to memory corruption.

SMI-2071
The user of MICROSAR Safe shall ensure that the indirect API is used consistently.

Indirect API functionality consists of the APIs:

- *Rte_Port*
- *Rte_Ports*
- *Rte_NPorts*

The indirect API makes it possible to call different APIs through an array access. The indirect API functionality can be enabled individually per port.
A wrong configuration switch can easily lead to a call outside of the array returned by the Rte_Ports API.

Vector recommends not using the indirect API.

Verification can e.g. be performed by review that the intended APIs are returned.

The RTE Analyzer does not assist in this integration step.

28.3.1.5 Usage of RTE APIs

SMI-2072
The user of MICROSAR Safe shall ensure that the RTE and all of its users have the same assumptions with regards to the sizes of the datatypes.

The RTE supports the configuration of custom datatypes for its APIs. The RTE specification mandates that arrays are passed as pointer to the array base type.
The RTE does not enforce that both sides of a connection use arrays of the same size.

No NULL pointers or invalid pointers shall be passed to RTE APIs.
The object to which the pointer points needs to have at least the size of the pointer base type.

Vector recommends using the headers generated by the RTE when static code analysis is performed on the application code.
Vector recommends using the same datatypes on both sides of a connection.
Arrays and void pointers on interfaces where the called function writes to them are considered especially relevant.

Verification can e.g. be performed by review and/or static code analysis.

The RTE Analyzer lists APIs and runnables that use such parameters to assist in this integration step.

SMI-2073
The user of MICROSAR Safe shall ensure that RTE APIs are only called from their configured contexts.

Fast response times are crucial in embedded systems. Therefore, the RTE generator analyzes the call contexts of all APIs in order to optimize away unneeded interrupt locks. When the application calls the APIs from a different context than the RTE assumes, data consistency problems may arise.

In systems with ASIL partitions, the RTE generator uses a conservative locking strategy. Locks are only optimized away if all accesses are done within the same task.

Verification can e.g. be performed by review and/or static code analysis.

The RTE Analyzer lists the optimized APIs of SWCs to assist in this integration step.
The RTE Analyzer lists APIs that must not be called by the application because they are considered unreachable due to the RTE configuration.

28.3.1.6 Configuration of RTE APIs

SMI-2074
The user of MICROSAR Safe shall ensure that the receivers can handle the initial value provided by the RTE if no write or calibration occurred.

For implicit accesses the user of MICROSAR Safe shall assure that the correct initial value is sent when *Rte_IWrite* or *Rte_IWriteRef* were not called in the runnable.

Initial values can be configured for:

- non-queued sender-/receiver communication
- inter-runnable variables
- mode ports
- calibration parameters

The initial value is returned when no sending API was called before the first read or no calibration was performed before the first read. The initial values depend on the connected components.

Vector recommends using the same initial value on all port data elements that are connected with each other.

The RTE Analyzer lists all APIs of SWCs that may provide an initial value to assist in this integration step. If possible, the RTE Analyzer reports the initial value generated by the RTE into the configuration feedback report.

SMI-2075
The user of MICROSAR Safe shall ensure that the alive timeout by the COM is not used for safety-related inter-ECU sender/receiver communication.

Safety-related communication must be protected by E2E protection. The decision if new data is available (alive) can only be made by the E2E mechanism. Data is not interpreted by the COM. For example the sending ECU might repeat old data. This is only detected by the cycle counter that is part of the E2E protection.

The RTE Analyzer lists the reading APIs that provide the alive timeout status to assist in this integration step.

SMI-2126
The user of MICROSAR Safe shall ensure that SWCs handle the *RTE_E_INVALID* return code properly.

The RTE offers a functionality to invalidate signals.

Vector requires using end-to-end protection for safety-related inter-ECU communication. Relying on the invalidation mechanism for safety-related signals is not an option. The user of MICROSAR Safe shall not use invalidation for inter-ECU communication.

Verification can e.g. be performed by review and/or integration testing.

The RTE Analyzer lists RTE APIs that return the *RTE_E_INVALID* return code to assist in this integration step.

SMI-2127
The user of MICROSAR Safe shall ensure that SWCs handle the *RTE_E_NEVER_RECEIVED* return code properly.

The RTE offers a functionality to report if a signals was received after the ECU was started.

Vector requires using end-to-end protection for safety-related signals. Relying on the never received mechanism for safety-related signals is not an option. Especially, when using the E2E transformer, its return value shall be evaluated.

Verification can e.g. be performed by review or integration testing.

The RTE Analyzer lists RTE APIs that return the *RTE_E_NEVER_RECEIVED* return code to assist in this integration step.

SMI-2125
The user of MICROSAR Safe shall ensure that the queue sizes that were chosen during the configuration are sufficient for the integrated system.

The RTE uses queues for mode communication, sender-/receiver communication and mapped client-/server communication. The queue sizes depend on the scheduling of entities and the call sequences of the APIs.

Vector recommends not using APIs with queues for safety-related functionality.

Verification can e.g. be performed by stress testing.

The RTE Analyzer lists the queue sizes to assist in this integration step.

SMI-36068
The user of MICROSAR Safe shall ensure that all connections with end-to-end (E2E) protection are generated.

The RTE Analyzer lists RTE APIs that read or write end-to-end protected data (see SMI-98).

28.4 Safety features required from other components

SMI-2121
RTE requires the following functionality as safety feature from the operating system:

- Interrupt enabling/disabling
- Resource handling
- Inter OS Application Communicator (IOC) sending and receiving functionality
- Spin-lock functionality
- Alarm handling
- Schedule table handling
- Activation of tasks
- Event handling

This requirement is fulfilled if an ASIL operating system by Vector is used.

SMI-2978
RTE requires the following functionality as safety feature from the NvM:

- Reading blocks

- Writing blocks

This requirement is fulfilled if an ASIL NvM by Vector is used.

28.5 Dependencies to hardware

This component does not use a direct hardware interface.

29 Safety Manual SecOC

29.1 Safety features

This component does not provide safety features.

29.2 Configuration constraints

This component does not have configuration constraints.

29.3 Additional verification measures

SMI-41233
The user of MICROSAR Safe shall verify for each entry of table `SecOC_VerificationStatusCallout` that function referred by member `SecOCVerificationStatusCalloutType` has the following signature (the placeholder `<name>` represents the function's name):

```
void <name>(SecOC_VerificationStatusType verificationStatus)
```

The table `SecOC_VerificationStatusCallout` can be found in SecOC_Lcfg.c.

29.4 Safety features required from other components

This component does not require safety features from other components.

29.5 Dependencies to hardware

This component does not use a direct hardware interface.

30 Safety Manual FrTp

30.1 Safety features

This component does not provide safety features.

30.2 Configuration constraints

SMI-32326
The user of MICROSAR Safe shall configure the following attribute:

- Set /MICROSAR/FrTp/FrTpGeneral/FrTpRuntimeMeasurementSupport to FALSE.

This setting is enforced by a MSSV plugin.

30.3 Additional verification measures

This component does not require additional verification measures.

30.4 Safety features required from other components

This component does not require safety features from other components.

30.5 Dependencies to hardware

This component does not use a direct hardware interface.

31 Safety Manual WdgIf

31.1 Safety features

SMI-519
This component provides the following safety features:

ID	Safety feature
CREQ-107414	WdgIf shall provide a service to set the mode of a watchdog device
CREQ-107415	WdgIf shall provide a service to set the trigger condition for a watchdog device
CREQ-107416	WdgIf shall support a mechanism to combine statuses of different cores and handle one watchdog for different cores

31.2 Configuration constraints

SMI-522
If a state combiner is used, the user of MICROSAR Safe shall configure

- *WdgIfStateCombinerSpinlockID* and
- *WdgIfStateCombinerStartUpSyncCycles*

for each core that is used by the state combiner.

SMI-523
If a state combiner is used and *WdgIfStateCombinerStartUpSyncCycles* is set to a value s, the user of MICROSAR Safe shall consider that for the first s SupervisionCycles of the master, the master does not monitor the slave triggers.
However, reset requests from a slave within the first s SupervisionCycles, are escalated by the master with the next call of the master's *WdgM_MainFunction()*.

31.3 Additional verification measures

SMI-525
The user of MICROSAR Safe shall verify that the output path of the generator is empty before the generator is started.
The output path can be defined by the command line argument *OUTPUT-DIRECTORY*.

SMI-526
The user of MICROSAR Safe shall inspect the messages of the generator execution.
If the generator aborts the generation process with an error message, the (partially) generated output files shall not be used in the system.
If the generator detects an error, a message starting with "ERROR" is displayed on the standard output.
If the generator shows a warning message starting with "WARNING", the user of

MICROSAR Safe shall ensure that the cause of the warning does not invalidate the generated output files.

SMI-527
The user of MICROSAR Safe shall verify that the following preprocessor directives are defined with the correct value independent from whether a state combiner is configured or not:

Preprocessor Directive	Value
WDGIF_VERSION_INFO_API	*STD_ON* if *WdgIfVersionInfoApi* is *TRUE*, otherwise *STD_OFF*.
WDGIF_USE_AUTOSAR_DRV_API	*STD_ON*
WDGIF_DEV_ERROR_DETECT	*STD_ON* if *WdgIfDevErrorDetect* is *TRUE*, otherwise *STD_OFF*.
WDGIF_INTERNAL_TICK_COUNTER	*STD_OFF*
WDGIF_USE_STATECOMBINER	*STD_ON* if *WdgIfUseStateCombiner* is *TRUE*, otherwise *STD_OFF*.

The defines can be found in *WdgIf_Cfg_Features.h*.

SMI-528
The user of MICROSAR Safe shall verify that the following preprocessor directives are defined with the correct value only if a state combiner is configured:

Preprocessor Directive	Value
WDGIF_STATECOMBINER_USE_OS_SPIN_LOCK	*STD_ON* if *WdgIfStateCombinerUseOsSpinlock* is *TRUE*, otherwise *STD_OFF*.
WDGIF_STATECOMBINER_MANUAL_MODE	*STD_ON* if *WdgIfStateCombinerUseManualMode* is *TRUE*, otherwise *STD_OFF*.

The defines can be found in *WdgIf_Cfg_Features.h*.

SMI-529
The user of MICROSAR Safe shall verify that the following preprocessor directives are defined with the correct value independent from whether a state combiner is configured or not:

Preprocessor Directive	Value
WDGIF_NUMBER_OF_WDGIFDEVICES	The number of configured WD Interface Devices.

The define can be found in *WdgIf_LCfg.h*.

SMI-530
The user of MICROSAR Safe shall verify that the following preprocessor directives are defined with the correct value only if a state combiner is configured:

Preprocessor Directive	Value
WDGIF_NUMBER_OF_SLAVES	The configured number of slave cores. Cores that are not attached to a State Combiner (i.e. they run independent from

	other cores) do not count as slave.

The define can be found in *WdgIf_LCfg.h*.

SMI-531

The user of MICROSAR Safe shall verify that the C-struct *const WdgIf_InterfaceType WdgIf_Interface* is defined in *WdgIf_LCfg.c*.

WdgIf_Interface shall contain the following fields:

Field	**Value**	**Description**
1st	*WDGIF_NUMBER_OF_WDGIFDEVICES*	The number of WD Interface Devices.
2nd	*WdgIf_FunctionsPerWdg*	A reference to the list of WD Interface Devices.

If a state combiner is configured, *WdgIf_Interface* shall also contain the following fields:

Field	**Value**	**Description**
3rd	*&wdgif_statecombiner_common_config*	A reference to the state combiner data structure.
4th	*wdgif_statecombiner_manual_config*	In case of manual mode.

SMI-532

The user of MICROSAR Safe shall verify that the array *static const WdgIf_InterfaceFunctionsPerWdgDeviceType WdgIf_FunctionsPerWdg* [WDGIF_NUMBER_OF_WDGIFDEVICES] is defined in *WdgIf_LCfg.c*.
WdgIf_FunctionsPerWdg shall refer all – and only - the WD Interface Devices that are configured in the EDF.
WdgIf_FunctionsPerWdg[i] shall refer the underlying WD Interface Device in the EDF with *WdgIfDeviceIndex = i*.
The fields in an array element in *WdgIf_FunctionsPerWdg* shall be set as follows:

Field	**Value**	**Description**
1st	*&device_functions*	If the underlying WD Interface Device is directly linked to a WD driver for device, then the field refers to the C-struct device_functions.
1st	NULL_PTR	If the underlying WD Interface Device shares the WD driver with other WD Interface Devices using a state combiner, then the linked WD device is referred in *wdgif_statecombiner_common_config*.

If a state combiner is configured, an array element in *WdgIf_FunctionsPerWdg* shall also contain the following field:

Field	**Value**	**Description**
2nd	*WdgInstance*	A number that uniquely identifies the WD Interface Device instance for the underlying platform. All instances on the same platform are numbered consecutively starting with 0. Example: One platform has instances A and B, another platform has instances C and D. Then A, B, C, and D have *WdgInstance* set (in this order) as: 0,1,0,1.

SMI-533

If a state combiner is configured, the user of MICROSAR Safe shall verify that the array *static const WdgIf_StateCombinerCommonConfigType wdgif_statecombiner_common_config* is defined in *WdgIf_LCfg.c*.
The fields in *wdgif_statecombiner_common_config* shall be set as follows:

Field	Value	Description
1st	*WDGIF_NUMBER_OF_SLAVES*	The number of configured slaves.
2nd	*WdgIfStateCombinerSpinlockID*	The value of field *WdgIfStateCombinerSpinlockID* in the EDF.
3rd	*WdgIfStateCombinerStartUpSyncCycles*	The value of field *WdgIfStateCombinerStartUpSyncCycles* in the EDF.
4th	*&device_functions*	A reference to the WD driver API functions for device.
5th	*wdgif_statecombiner_shared_memory*	A reference to the shared memory for the state combiners of the Watchdog Interfaces.

SMI-534
If a state combiner is configured, the user of MICROSAR Safe shall verify that the array *WdgIf_StateCombinerSharedMemory wdgif_statecombiner_shared_memory* [WDGIF_NUMBER_OF_SLAVES] is defined in *WdgIf_LCfg.c*.
The WdgIf writes to this array, hence it is not *const*.
wdgif_statecombiner_shared_memory shall contain one array element for each configured slave and the fields in the element shall be set as shown in the following table:

Field	Value	Description
1st	*0u*	Initial value for the slave's WindowStart.
2nd	*(uint16)~0u*	Inverse initial value for the slave's WindowStart.
3rd	*0u*	Initial value for the slave's Timeout.
4th	*(uint16)~0u*	Inverse initial value for the slave's Timeout.
5th	*0u*	Initial value for the slave's counter.
6th	*(uint16)~0u*	Inverse initial value for the slave's counter.

SMI-535
If the state combiner is configured in manual mode, the user of MICROSAR Safe shall verify that the array *static const WdgIf_StateCombinerManualConfigType* wdgif_statecombiner_manual_config* [WDGIF_NUMBER_OF_SLAVES] is defined in *WdgIf_LCfg.c*.
wdgif_statecombiner_manual_config shall list the references to all configured slaves. *wdgif_statecombiner_manual_config[i] = &wdgif_statecombiner_config_slave<ID>*, where ID is the slave's consecutive number starting with 1.

SMI-536
If the state combiner is configured in manual mode, the user of MICROSAR Safe shall verify that for a configured slave with ID the C-struct *static const*

WdgIf_StateCombinerManualConfigType wdgif_statecombiner_config_slave<ID> is defined in *WdgIf_LCfg.c*.
wdgif_statecombiner_config_slave<ID> shall be set as follows:

Field	Value	Description
1st	*WdgIfStateCombinerReferenceCycle*	The value of field *WdgIfStateCombinerReferenceCycle* in the EDF.
2nd	*WdgIfStateCombinerSlaveIncrementsMin*	The value of field *WdgIfStateCombinerSlaveIncrementsMin* in the EDF.
3rd	*WdgIfStateCombinerSlaveIncrementsMax*	The value of field *WdgIfStateCombinerSlaveIncrementsMax* in the EDF.

SMI-537
The user of MICROSAR Safe shall verify that for each configured platform the C-struct *static const WdgIf_InterfaceFunctionsType <platform>_functions* is defined in WdgIf_LCfg.c.
The fields for each C-struct <platform>_functions shall be set as follows:

Field	Value	Description
1st	*Wdg_<infix>_SetMode_*	A reference to the WD driver's API function to set the mode of the device referred to by infix.
2nd	*Wdg_<infix>_SetTriggerWindow_* and *Wdg_<infix>_SetTriggerCondition_*	A reference to the WD driver's API function to set the trigger window of the device referred to by infix.

31.4 Safety features required from other components

SMI-520
This component requires the triggering of the watchdog and setting the triggering mode as a safety feature from the watchdog driver.
This requirement is fulfilled if a watchdog driver by Vector is used.

SMI-3085
The WdgIf shall be used in order to call the services of underlying Watchdog driver(/s) to set the mode and the trigger condition as expected by the drivers.
The user of MICROSAR Safe shall verify that the WdgIf services are only called by the WdgM.
This requirement is fulfilled for all components of MICROSAR (if Vector's WdgM is used).
If e.g. the trigger condition is called by another component, this may lead to unintended triggering of the watchdog.
If the watchdog stack is not properly set up, it may not provide the expected protection.

31.5 Dependencies to hardware

This component does not use a direct hardware interface.

32 Safety Manual WdgM

32.1 Safety features

SMI-373
This component provides the following safety features:

ID	Safety feature
CREQ-102746	WdgM shall provide a mechanism for Alive Supervision.
CREQ-102745	WdgM shall provide a mechanism for Deadline Supervision.
CREQ-102744	WdgM shall provide a mechanism for Logical Supervision.
CREQ-102749	WdgM shall provide a service to trigger a checkpoint.
CREQ-102752	WdgM shall provide a service to initiate a reset triggered by the hardware watchdog.
CREQ-102754	WdgM shall provide a service to activate the supervision of a Supervised Entity.
CREQ-102755	WdgM shall provide a service to deactivate the supervision of a Supervised Entity.
CREQ-102763	WdgM shall provide a service to cyclically update the global supervision status.
CREQ-102758	WdgM shall provide a service to set a new trigger mode.

The watchdog manager is able to detect program flow violations, alive counter violations and deadline violations.
The following types of faults can be detected:

- omission of an operation (program flow, alive counter),
- unrequested execution of an operation (program flow, alive counter),
- operation executed too early (alive counter, deadline),
- operation executed too late (alive counter, deadline), and
- operations executed in the wrong sequence (program flow).

32.2 Configuration constraints

SMI-501
The user of MICROSAR Safe shall use the WdgM only on 32-bit microcontroller platforms.

SMI-499
The user of MICROSAR Safe shall configure and verify alive supervision for a supervised entity.
If no alive supervision is configured for a supervised entity, WdgM cannot detect if the corresponding checkpoint is reached at least once.
Please note that for non-periodic supervised entities alive supervision is not possible.

A value of a pre-compile configuration parameter is valid for every core. A different value cannot be set for the same pre-compile parameter and different cores.

SMI-498
The user of MICROSAR Safe shall set the value of *WdgMTimebaseSource* according to the required source of time ticks:

WdgMTimebaseSource	Description
WDGM_INTERNAL_SOFTWARE_TICK	An internal time source for Deadline Monitoring is selected. The tick counter is incremented each time the *WdgM_MainFunction()* is invoked.
WDGM_OS_COUNTER_TICK	An OsCounter is selected for Deadline Monitoring as timebase. The user of MICROSAR Safe is responsible for configuring the OsCounter accurately.
WDGM_EXTERNAL_TICK	An external time source for Deadline Monitoring is selected. The tick counter is incremented each time the *WdgM_UpdateTickCount()* function is invoked. The function is implemented in the WdgM. The user of MICROSAR Safe is responsible for calling WdgM_UpdateTickCount() accurately.

SMI-560
The user of MICROSAR Safe shall use the functions *WdgM_ActivateSupervisionEntity()* and *WdgM_DeactivateSupervisionEntity()* to activate and deactivate the supervision of supervised entities.

Activation and deactivation shall only be performed by a software component that is developed according to the highest ASIL that is allocated to the ECU.

The functions are only available if *WdgMEntityDeactivationEnabled* is set to *TRUE*. Vector recommends setting *WdgMEntityDeactivationEnabled* to *FALSE* to prevent that faults are not detected.

32.3 Additional verification measures

SMI-3072
The user of MICROSAR Safe shall verify that the WdgM is initialized only at intended points in time, e.g. during initialization.
Unintended re-initialization may lead to a incorrect monitoring.

SMI-502

The user of MICROSAR Safe shall verify that the output path of the generator is empty before the generator is started.
The output path can be defined by the command line argument *OUTPUT-DIRECTORY*.

SMI-524
The user of MICROSAR Safe shall inspect the messages of the generator execution.
If the generator aborts the generation process with an error message, the (partially) generated output files shall not be used in the system.
If the generator detects an error, a message starting with "ERROR" is displayed on the standard output.
If the generator shows a warning message starting with "WARNING", the user of MICROSAR Safe shall ensure that the cause of the warning does not invalidate the generated output files.

32.3.1 Additional verification using WdgM Verifier

SMI-503
The user of MICROSAR Safe shall execute the supplied WdgM Verifier.
Instructions can be found in the technical reference of WdgM on how to run the WdgM Verifier.

SMI-504
The user of MICROSAR Safe shall verify that the report of the WdgM Verifier

- comprises "All tests passed" in the last line. and
- that all tests in the *Summary* of the report are marked with *PASSED*.

SMI-1578
If an AUTOSAR OS with a version higher than 4.0 is used the verification test with id '109' is marked as "NOT PASSED".

The user of MICROSAR Safe shall verify whether the assigned core id to a WdgMMode is generated correctly if an AUTOSAR OS with a version higher than 4.0 is used. The WdgM Verifier cannot not verify this step due to an incompatible change of the Os's description file.

SMI-512
The user of MICROSAR Safe shall verify the generated local transitions in *wdgm_verifier_info.c*.

Generated local transitions are defined by the C-struct array with the name *local_transitions*.
The array holds all local transitions of all supervised entities.
Each local transition lt contains the names of:

- the source entity (SE) of lt,
- the source checkpoint of lt,
- the destination entity (SE) of lt and

- the destination checkpoint of lt.

Verification shall include that

- each local transition is defined as stated in the System Specification, and
- no local transition in the System Specification is missing.

SMI-513
The user of MICROSAR Safe shall verify the generated global transitions in *wdgm_verifier_info.c*.

Generated global transitions are defined by the C-struct array with the name *global_transitions*.
The array holds all global transitions of all supervised entities.
Each global transition gt contains the names of:

- the source entity (SE) of gt,
- the source checkpoint of gt,
- the destination entity (SE) of gt, and
- the destination checkpoint of gt.

Verification shall include that

- each global transition is defined as stated in the System Specification, and
- no global transition in the System Specification is missing.

SMI-514
The user of MICROSAR Safe shall verify the checkpoints in *wdgm_verifier_info.c*.

Supervised entities named se are defined by a C-struct array with the name *se_*<se>_cp_list_.
The array *se_*<se>_cp_list_ holds information about all checkpoints configured for se.
Each array item contains information about one checkpoint cp of the supervised entity se:

- the supervised entity's ID (for this cp of se),
- the checkpoint's ID (for this cp of se),
- the supervised entity name (for this cp of se), and
- the checkpoint name (for this cp of se).

Verification shall include that

- each checkpoint is configured as stated in the System Specification, and
- no checkpoint for the actual supervised entity is missing.

SMI-515
The user of MICROSAR Safe shall verify the supervised entities in *wdgm_verifier_info.c*.

Supervised entities named se are defined by a C-struct array with the name *entities*.
The array *entities* holds information about a onfigured supervised entity.
The fields in an array item for a supervised entity se shall have the following values (in this order):

Value	Source Line Comment
The entity ID (of se).	enitity id
The entity name (of se).	entity name
The number of checkpoints configured for se.	number of checkpoints
A reference *se_<se>_cp_list_*, which refers to the list of CPs for se	this entity's checkpoints
A reference to the callback function for se as configured in *WdgMLocalStateChangeCbk* (or NULL_PTR if no callback function is configured).	*WdgMLocalStateChangeCbk*
false	autosar_3_1_x_compatibility
The application task for se as configured in field *WdgMAppTaskRef*	*WdgMAppTaskRef*

Verification shall include that

- each supervised entity is configured as stated in the System Specification, and
- no supervised entity is missing.

SMI-516
The user of MICROSAR Safe shall verify the deadline supervisions in *wdgm_verifier_info.c*.

Deadline supervisions are defined by a C-struct array with the name *deadline_supervisions*.
The array *deadline_supervisions* holds information about all transitions with deadline supervision.
Each deadline supervision dl contains the following values:

- the source entity name (SE) of dl,
- the source checkpoint name of dl,
- the destination entity name (SE) of dl,
- the destination checkpoint name of dl,
- the minimum deadline for dl (in seconds), and
- the maximum deadline of dl (in seconds).

Verification shall include that

- each deadline supervision is configured as stated in the System Specification, and

- no deadline supervision is missing.

SMI-517
The user of MICROSAR Safe shall verify the alive supervisions in *wdgm_verifier_info.c*.

Alive supervisions are defined by a C-struct array with the name *alive_supervisions*.
The array *alive_supervisions* holds information about all transitions with alive supervision.
Each alive supervision al contains the following values:

- the supervised entity name (SE) of al,
- the checkpoint name of that se of al,
- the number of expected alive indications per reference cycle of al,
- the minimum margin for alive indications of al,
- the maximum margin for alive indications of al, and
- the number of supervision reference cycle of al.

Verification shall include that

- each alive supervision is configured as stated in the System Specification, and
- no alive supervision is missing.

SMI-518
The user of MICROSAR Safe shall verify the configured cores in *wdgm_verifier_info.c*.

The configured cores are defined in the *main()* function.
For each configured core with ID, the following line shall be present:

```
result += verify (&WdgMConfig_Modem_core<ID>, &verifier_info);
```

where ID is the core ID and m is the ID of the WdgM mode.

Verification shall include that for each core there is a corresponding line in the file.

32.3.2 Additional verification of generator execution

SMI-506
The user of MICROSAR Safe shall verify that for the following arrays in *WdgM_PBcfg.c*, the array length matches the number of items in the array:

- *WdgMTransition*
- *WdgMGlobalTransition*
- all arrays named *StartsGlobalTransition*<se>_<cp>_<i>_ (for a supervised entity se, a checkpoint cp and an integer i)
- *WdgMCheckPoint*
- *WdgMSupervisedEntity*
- all arrays named *WdgMTriggerMode_core<ID>* (for each core with ID)
- *WdgMWatchdogDevice<ID>* (for each core with ID)
- *WdgMAllowedCallers*

Some of these arrays have preprocessor defines for their size, e.g. *WdgMCheckPoint [WDGM_NR_OF_CHECKPOINTS]*. These defines can be found in *WdgM_PBcfg.h*.

SMI-507
The user of MICROSAR Safe shall verify that each item in the array *WdgMSupervisedEntity* follows this rules:
1. *WdgMCheckpointRef* has a value of the form *&WdgMCheckPoint[i] with i < _WDGM_NR_OF_CHECKPOINTS*, and
2. _WdgMCheckpointLocInitialId has a value of 0.
The array can be found in *WdgM_PBcfg.c*.

SMI-27923
The user of MICROSAR Safe shall verify that each WdgM_StatusReportToRte member of struct WdgM_ConfigType and each WdgM_StatusReportToRte member of struct WdgM_SupervisedEntityType has a valid entry if WDGM_STATUS_REPORTING_MECHANISM is set to WDGM_USE_MODE_SWITCH_PORTS.
The functions must be implemented by the Rte and must have the following signature: Std_ReturnType (*WdgM_StatusReportToRte) (WdgMMode).
Both structs can be found in WdgM_PBcfg.c.

SMI-508
The user of MICROSAR Safe shall verify that for each core with ID *WDGM_NR_OF_WATCHDOGS_CORE<ID>* matches the actual number of configured WD devices.
The define can be found in *WdgM_PBcfg.h*.

SMI-509
The user of MICROSAR Safe shall verify that for each core with ID *WDGM_NR_OF_TRIGGER_MODES_CORE<ID>* matches the actual number of configured Watchdog Manager Trigger Modes.
The define can be found in *WdgM_PBcfg.h*.

SMI-510

The user of MICROSAR Safe shall verify that *WDGM_NR_OF_ALLOWED_CALLERS* matches the number of modules that call function *WdgM_SetMode()*.
The define can be found in *WdgM_PBcfg.h*.

SMI-511
The user of MICROSAR Safe shall verify that in *WdgMConfig_Mode<m>_core<ID>* (for each core with ID and every mode m), the field *WdgMCallersRef* points to *WdgMAllowedCallers* and *WdgMAllowedCallers* is an array of type *WdgM_CallersType* with a length of *WDGM_NR_OF_ALLOWED_CALLERS*.
The variable can be found in *WdgM_PBCfg.h*.

SMI-550
The user of MICROSAR Safe shall verify the types used by WdgM.

If the configuration parameter *WDGM_USE_RTE* is set to *STD_ON*, the types from *Rte_Type.h* are used:

Type	**Allowed Value**
WdgM_SupervisedEntityIdType	*uint16*
WdgM_CheckpointIdType	*uint16*
WdgM_ModeType	*uint8*
WdgM_LocalStatusType	*uint8*
WdgM_GlobalStatusType	*uint8*

The WdgM includes *WdgM_Rte_Includes.h* if and only if *WDGM_USE_RTE* is set to *STD_ON*.

If the configuration parameter *WDGM_USE_RTE* is set to *STD_OFF*, the types from WdgM are used:

Type	**Allowed Value**
WdgM_SupervisedEntityIdType	*uint16*
WdgM_CheckpointIdType	*uint16*
WdgM_ModeType	*uint8*
WdgM_LocalStatusType	*uint8*
WdgM_GlobalStatusType	*uint8*

SMI-551
The user of MICROSAR Safe shall verify the definitions used by WdgM.

If the configuration parameter *WDGM_USE_RTE* is set to *STD_ON*, the definitions from *Rte_WdgM_Type.h* are used.
If the configuration parameter *WDGM_USE_RTE* is set to *STD_OFF*, the definitions from WdgM are used.

Definition	**Value**
WDGM_LOCAL_STATUS_OK	0

WDGM_LOCAL_STATUS_FAILED	1
WDGM_LOCAL_STATUS_EXPIRED	2
WDGM_LOCAL_STATUS_DEACTIVATED	4
WDGM_GLOBAL_STATUS_OK	0
WDGM_GLOBAL_STATUS_FAILED	1
WDGM_GLOBAL_STATUS_EXPIRED	2
WDGM_GLOBAL_STATUS_STOPPED	3
WDGM_GLOBAL_STATUS_DEACTIVATED	4

The WdgM includes *Rte_WdgM_Type.h* if and only if *WDGM_USE_RTE* is set to *STD_ON*.

32.4 Safety features required from other components

SMI-372
This component requires setting a trigger condition and setting the triggering mode as safety features from WdgIf.
This requirement is fulfilled if the WdgIf by Vector is used.

SMI-3414
The user of MICROSAR Safe shall call the service to set the mode as expected by the Wdg stack.
If the watchdog is not properly set up, it may not provide the expected protection.

32.5 Dependencies to hardware

This component does not use a direct hardware interface.

33 Safety Manual XCP

33.1 Safety features

This component does not provide safety features.

SMI-178
This component is only partly developed according to ASIL development process. This part includes the disabling of the Xcp.

The main part of this component is developed according to a QM development process. Thus, this component shall only be enabled in an operating mode that do not impose risk for the health of persons.

33.2 Configuration constraints

SMI-3412
The user of MICROSAR Safe shall configure the following:

- Set `/MICROSAR/Xcp/XcpGeneral/XcpControl` to TRUE.

The user of MICROSAR Safe shall verify that the corresponding configuration switch is set in the Xcp protocol layer and all used transport layers:

File	Define	STD_ON/STD_OFF
Xcp_Cfg.h	XCP_CONTROL	STD_ON
CanXcp_Cfg.h	CANXCP_ENABLE_CONTROL	STD_ON
FrXcp_Cfg.h	FRXCP_ENABLE_CONTROL	STD_ON
TcpIpXcp_Cfg.h	TCPIPXCP_ENABLE_CONTROL	STD_ON

SMI-179
The user of MICROSAR Safe shall use the macros *XCP_ACTIVATE()* and *XCP_DEACTIVATE()* to activate and deactivate XCP protocol layer and transport layer components.
Activation and deactivation shall only be performed by a software component that is developed according to the highest ASIL that is allocated to the ECU.
XCP shall only be activated in an operating mode that does not impose risk for the health of persons.
Note: XCP is active by default.

SMI-183
The user of MICROSAR Safe shall make the following memory sections **read-only** for the XCP protocol layer and transport layer components as well as all other software with an ASIL lower than the highest ASIL allocated to the ECU:

- *XCP_START_SEC_VAR_INIT_UNSPECIFIED_SAFE*

- *CANXCP_START_SEC_VAR_INIT_UNSPECIFIED_SAFE*
- *FRXCP_START_SEC_VAR_INIT_UNSPECIFIED_SAFE*
- *TCPIPXCP_START_SEC_VAR_INIT_UNSPECIFIED_SAFE*

33.3 Additional verification measures

SMI-184
The user of MICROSAR Safe shall verify during integration testing that XCP is disabled during normal operation.

33.4 Safety features required from other components

SMI-177
This component requires an operating system with enabled memory partitioning.

33.5 Dependencies to hardware

This component does not use a direct hardware interface.

34 Glossary and Abbreviations

34.1 Glossary

Term	Definition
User of MICROSAR Safe	Integrator and user of components from MICROSAR Safe provided by Vector.
MICROSAR Safe	MICROSAR Safe comprises MICROSAR SafeBSW and MICROSAR SafeRTE as Safety Element out of Context. MICROSAR SafeBSW is a set of components, that are developed according to ISO 26262 [1], and are provided by Vector in the context of this delivery. The list of MICROSAR Safe components in this delivery can be taken from the documentation of the delivery.
Critical section	A section of code that needs to be protected from concurrent access. A critical section may be protected by using the AUTOSAR exclusive area concept.
Configuration data	Data that is used to adapt the MICROSAR Safe component to the specific use-case of the user of MICROSAR Safe. Configuration data typically comprises among others: feature selection, routing tables, channel tables, task priorities, memory block descriptions.
Generated code	Source code that is generated as a result of the configuration in DaVinci Configurator Pro
Partition	A set of memory regions that is accessible by tasks and ISRs. Synonym to OSApplication.

34.2 Abbreviations

Abbreviation	Description
ASIL	Automotive Safety Integrity Level
BSWMD	Basic Software Module Description
CPU	Central Processing Unit
CREQ	Component Requirement
EEPROM	Eletronically Ereasable and Programmable Read-only Memory
ECC	Error Correcting Code
ECU	Electronic Control Unit
EXT	Driver for an external hardware unit
ISO	International Standardization Organization
MCAL	Microcontroller Abstraction
MIP	Module Implementation Prefix
MSSV	MICROSAR Safe Silence Verifier
OS	Operating System
PDU	Protocol Data Unit
QM	Quality Management

RAM	Random Access Memory
SMI	Safety Manual Item
TCL	Tool Confidence Level

35 Contact

Visit our website for more information on

- News
- Products
- Demo software
- Support
- Training data
- Addresses

www.vector.com