MICROSAR SafeE2E Protection Wrapper

Safety Manual

Version 1.0.0

Authors	Jonas Wolf
Status	Released

Document Information

History

Author	Date	Version	Remarks
Jonas Wolf	2016-01-18	1.0.0	Initial creation

Caution
We have configured the programs in accordance with your specifications in the questionnaire. Whereas the programs do support other configurations than the one specified in your questionnaire, Vector´s release of the programs delivered to your company is expressly restricted to the configuration you have specified in the questionnaire.

Contents

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 your 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.

1.1.3 Definitions

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.

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

1. *Shall* means that the definition is an absolute requirement of the specification.
2. *Shall not* means that the definition is an absolute prohibition of the specification.
3. *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.
4. *Can* means that a definition is truly optional.

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 Documentation	1.2.1
[3]	Vector	Product Information MICROSAR Safe	1.1
[4]	Vector	Technical Reference MICROSAR Safe Silence Verifier	1.4

1.1.5 Overview

This document is automatically generated. The content of this document depends on the components and microcontroller of your delivery.

The structure of this document comprises:

- a general section that covers all assumptions and constraints that are always applicable, and
- a microcontroller specific section that covers all aspects of the selected microcontroller (only if microcontroller specific components are part of the delivery), and
- a section for each component that covers its constraints and necessary verification steps.

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.

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.

1.2.1 Technical Safety Requirements

These 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 are assigned an ASIL D 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.

1.2.1.1 Initialization

TSR-1
The system shall initialize the CPU, MPU, watchdog, and operating system.

Rationale: Initialization of the hardware, e.g. clocks, memory protection, scheduling etc. is necessary to enable the other safety requirements.
MICROSAR Safe Feature: The ECU State Manager (EcuM) is responsible for performing the configured driver initialization. After initialization the EcuM starts the operating system. The EcuM distributes the post-built loadable configuration information within the ECU. The documentation of the operating system describes which parts of the CPU it initializes. The startup code and main function is in the responsibility of the user of MICROSAR Safe.

1.2.1.2 Self-test

TSR-2
The system shall perform self-tests based on the requirements of the system.

Rationale: It may be necessary to periodically test individual components of the system to detect latent faults.
MICROSAR Safe Feature :The operating system provides a function to self-test the effectiveness of the MPU settings.
The ECU State Manager services callouts to user code that can check RAM consistency after wakeup.
Other hardware self-tests are usually performed by MCAL components not developed by Vector.
End-to-end protection of communication provides its own fault detection mechanisms.

1.2.1.3 Reset of ECU

TSR-3
The system shall reset itself in case of a detected fault.

Rationale: Resetting the CPU of the ECU is in most cases an appropriate measure to achieve a safe state.
It is assumed that the reset 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. EMC.
MICROSAR Safe Feature: MICROSAR Safe does not reset the ECU on its own (there may be exceptions for the operating system).
Functionality from ECU State Manager (setting the shutdown target) can be used to reset the ECU instead, if this is the intended fault reaction.

1.2.1.4 Non-volatile memory

The system must be designed 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 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 has to assure that the read data is identical to the data stored in non-volatile memory.
The absence of data still has to be handled by the application.
The availability may be increased by e.g. redundant storage.

1.2.1.4.1 Saving data

TSR-4
The system shall save information in non-volatile memory.

Rationale: see text in section 2.4.

MICROSAR Safe Feature: The intended block is written with the information provided by the application.

1.2.1.4.2 Loading data

TSR-5
The system shall retrieve the last stored information from non-volatile memory.

Rationale: see text in section 2.4.

MICROSAR Safe Feature: The intended block is read and the information is provided to the application.
The last or any previous completely stored block in non-volatile memory is returned by memory stack.
If CRC or ECC protections do not match block data an error is returned.
If data is stored redundantly, the redundant information is returned.
If no completely stored block is found, an error is returned.

1.2.1.5 Scheduling

1.2.1.5.1 Deterministic, hard real-time scheduling

TSR-6
The system shall execute the specified functions within their respective hard timing limits.

Rationale: Hard real-time scheduling may be used for scheduling safety mechanisms implemented in software.
This requirement is even more important for fail-operational systems, where one function may have to work, while another blocks the processor.
MICROSAR Safe Feature: The immediate priority ceiling protocol specified by AUTOSAR is capable of performing this task. The schedule tables specified by AUTOSAR may also be used on top of the scheduling algorithm.
The operating system of MICROSAR Safe implements the scheduling algorithm according to ISO 26262.

1.2.1.6 Partitioning

1.2.1.6.1 Memory partitioning

TSR-7
The system shall protect software applications from unspecified memory access.

Rationale: 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 available MPU in hardware for the effectiveness of the mechanism.
MICROSAR Safe Feature: Memory partitioning and context switching using AUTOSAR Operating Feature System SC3 mechanisms is implemented according to ISO 26262.
Adequate configuration of the memory partitions is in the responsibility of the user of MICROSAR Safe.

1.2.1.6.2 Time partitioning

1.1.1.1.1.1 Timing protection

TSR-8
The system shall detect timing faults in the software.

Rationale: Relying on a watchdog for timing protection is sometimes not sufficiently robust or efficient.
MICROSAR Safe Feature: The operating system of MICROSAR Safe implements the timing protection functionality of SC4 according to ISO 26262.

1.1.1.1.1.2 Killing of applications

TSR-9
The system shall terminate software applications.

Rationale: In combination with the timing protection this allows the software to continue operation in case of a software fault.
MICROSAR Safe Feature: The operating system of MICROSAR Safe implements the termination of applications according to ISO 26262.

1.2.1.7 Communication protection

1.2.1.7.1 Inter ECU communication

1.1.1.1.1.3 End-to-end protection

TSR-10
The system shall protect communication between its elements.

Rationale: Communication has to be protected against corruption, unintended replay and masquerading. The loss of a message must be detected.

This can be achieved using the end-to-end (E2E) protection mechanism defined by AUTOSAR.
MICROSAR Safe Feature: MICROSAR Safe implements the E2E functionality according to
ISO 26262. This also includes the CRC library functionality.

1.1.1.1.1.4 Protection by cryptographic algorithms

TSR-11
The system shall protect communication between its elements using cryptographic hash algorithms to detect accidental corruption of the communication.

Rationale: Cyclic redundancy codes (CRC) provide a specified hamming distance given a polynomial and data block size.
Cryptographic hash functions provide a probabilistic statement on data corruption detection depending on the hash function, data block size and hash value size.
MICROSAR Safe Feature: The Cryptographic Service Manager of MICROSAR Safe is implemented according to ISO 26262.
The Cryptographic Service Manager services the main function and functions to calculate a cryptographic hash function according to AUTOSAR specification.

1.2.1.7.2 Intra ECU communication

1.1.1.1.1.5 Intra OS application communication

TSR-16
The microcontroller software shall communicate within its applications.

Rationale: Software components need to communicate.
Protection of the memory against random hardware faults is expected by the system (e.g. via ECC RAM and lock-step mode).
MICROSAR Safe Feature: The RTE provides services to allow communication of software components within OS applications (intra-partition communication).

1.1.1.1.1.6 Inter OS application communication

TSR-12
The microcontroller software shall communicate between its applications.

Rationale: Multi-core systems may need to exchange safety-related information between applications.
Protection of the memory against random hardware faults is expected by the system (e.g. via ECC RAM and lock-step mode).
MICROSAR Safe Feature: The operating system of MICROSAR Safe implements the inter-OS application (IOS) functionality according to ISO 26262.
The RTE provides services to allow communication between OS applications (inter-partition communication).

1.2.1.8 Watchdog services

1.2.1.8.1 Program flow monitoring

TSR-13
The system shall provide a mechanism to detect faults in program flow.

Rationale. Program flow can be corrupted by random hardware faults or software faults.
MICROSAR Safe Feature: MICROSAR Safe watchdog stack implements program flow monitoring functionality according to ISO 26262.

1.2.1.8.2 Alive monitoring

TSR-14
The system shall provide a mechanism to detect stuck software.

Rationale: Alive monitoring is used to reset the software or controller in case it is unresponsive.
MICROSAR Safe Feature: MICROSAR Safe watchdog stack implements alive monitoring functionality according to ISO 26262.

1.2.1.8.3 Deadline monitoring

TSR-15
The system shall provide a mechanism to detect deadline violations.

Rationale: Deadline monitoring using the watchdog stack can be used to implement timing monitoring.
MICROSAR Safe Feature: MICROSAR Safe watchdog stack implements deadline monitoring functionality according to ISO 26262.

1.2.2 Environment

1.2.2.1 Safety Concept

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 to use 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.

This assumption is added to this Safety Manual, because it is used in Vector's 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.
Vector's MICROSAR SafeWatchdog can be used to fulfill this assumption.
If the functional safety concept also requires a logic monitoring, Vector's MICROSAR SafeWatchdog can be used to implement it.
See also SMI-14.

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

The communication components of MICROSAR Safe do not assume sending or receiving as a safety requirement. Vector always assumes that an end-to-end protection or equivalent mechanism is implemented on application level.
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.

1.2.2.2 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. It is added to this Safety Manual, because Vector assumes initialized components 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.
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-18
The user of MICROSAR Safe shall enable plausibility checks for the MICROSAR Safe components.

This setting is necessary to increase fault detection and prevent fault propagation. This setting can be found at /MICROSAR/EcuC/EcucGeneral/EcuCSafeBswChecks in the DaVinci Configurator.
This setting is enforced by an MSSV plug-in.
This setting does not enable error reporting to the DET component.

1.2.2.3 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.

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).
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.

SMI-99
The user of MICROSAR Safe shall verify that the memory mapping is consistent with the partitioning concept.
The 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.

1.2.2.4 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.

1.2.3 Process

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 MICORSAR Safe as intended by the user's safety concept.

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.
The possible implementations of exclusive areas are described in the Technical References.

SMI-5
The user of MICROSAR Safe shall verify all code that is changed during integration of MICROSAR Safe.

Code that is typically changed 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.
Support by Vector can be requested on a per-project basis.

SMI-30
The user of MICROSAR Safe shall only modify source code of MICRSAR 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 Safe.RTE.

SMI-19
The user of MICROSAR Safe shall execute the MICROSAR Safe Silence Verifier (MSSV).

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. 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.
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.

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).
Vector has evaluated the tools exclusively used by Vector during the development of MICROSAR Safe.

2 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).

2.1.1 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

2.1.2 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.

2.1.3 Additional verification measures

2.1.3.1 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.

2.1.3.2 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.

2.1.3.3 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()*

2.1.4 Dependencies to other components

2.1.4.1 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.

2.1.4.2 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.

2.1.5 Dependencies to hardware

This component does not use a direct hardware interface.

3 Contact

Visit our website for more information on

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

www.vector.com