Document Title	General Requirements on Basic Software Modules
Document Category	Requirement specification (RS)
Document UID	43
Document Status	published
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	R26-11

Document Change History

Date	Release	Changed by	Description
2026-11-26	R26-11	AUTOSAR Release Management	• Added requirement for handling of corrupted NV data ([SRS_BSW_00497]) • Added requirement for double initialization errors ([SRS_BSW_00498]) • Updated MISRA reference to MISRA C 2025 ([SRS_BSW_00007]) • Removed requirements for extended production errors
2025-11-27	R25-11	AUTOSAR Release Management	• New (Metadata-handling related) requirement ([SRS_BSW_00496]) • Removed (Parameter-Scope related) requirement ([SRS_BSW_00394]) • Status changed to "Obsolete" for requirement ([SRS_BSW_00492])
2024-11-27	R24-11	AUTOSAR Release Management	• Removed Transient Faults • Clarified meaning of machine readable module description ([SRS_BSW_00334]) • Allowed callbacks with void return ([SRS_BSW_00359])
2023-11-23	R23-11	AUTOSAR Release Management	• Removed (ARTI related) requirement ([SRS_BSW_00495])

△

2022-11-24	R22-11	AUTOSAR Release Management	• Clarifications related to security events and development error table ([SRS_BSW_00385], [SRS_BSW_00386], [SRS_BSW_00490]) • Older requirements set to valid (were draft before) • New (ARTI related) requirement ([SRS_BSW_00495])
2021-11-25	R21-11	AUTOSAR Release Management	• Allowance of function pointers ([SRS_BSW_00371]) • Exclusive use of AUTOSAR data types ([SRS_BSW_00304]) • Introduction of C99 standard in AUTOSAR ([SRS_BSW_00477])
2020-11-30	R20-11	AUTOSAR Release Management	• New naming convention ([SRS_BSW_00494]) • Introduced IDSM concept ([SRS_BSW_00488] - [SRS_BSW_00493]) • Clarification about no return after development error ([SRS_BSW_00369]) • Updated enum examples with values ([SRS_BSW_00441], [SRS_BSW_00377])
2019-11-28	R19-11	AUTOSAR Release Management	• No content changes • Changed Document Status from Final to published
2018-10-31	4.4.0	AUTOSAR Release Management	• Added requirement for classification of security events ([SRS_BSW_00488]) • Added requirement for errors for module initialization ([SRS_BSW_00487]) • Header File Cleanup • Obsolete references removed • Editorial Changes
2017-12-08	4.3.1	AUTOSAR Release Management	• Life cycle change for header files • Related standards and norms are updated • Editorial changes

▽

△

2016-11-30	4.3.0	AUTOSAR Release Management	• Interfaces for C90 has been added • Support for MISRA 2012 updated • Obsolete references removed • Editorial Changes
2015-07-31	4.2.2	AUTOSAR Release Management	• Introduce new requirement [SRS_BSW_00403] • Introduce new requirement [SRS_BSW_00351] • Modified requirement [SRS_BSW_00406] and [SRS_BSW_00450] • Debugging support marked as obsolete
2014-10-31	4.2.1	AUTOSAR Release Management	• Alignment of post-build configuration to SWS_BSWGeneral • Rephrasing of definition of runtime errors • Incorporation of concept SupportForPB-LAndPBSECUConfiguration • Editorial changes
2014-03-31	4.1.3	AUTOSAR Release Management	• Erased/modified requirements about standard header files providing a more abstract view • Improved definition of run-time errors • Editorial changes
2013-10-31	4.1.2	AUTOSAR Release Management	• Revised the management of interfaces and the corresponding types into a dedicated header file for one module • Deleted a redundant requirement • Editorial changes
2013-03-15	4.1.1	AUTOSAR Administration	• Interface for BSW Modules to DEM and Debouncing for DEM • Declaration and implementation requirements for the interrupt routines in the BSW modules • Function prototype and improvement callback functions of AUTOSAR Services • Improvement of safety and integrity

▽

△

2011-12-22 to 2005-05-31	4.0.3 to 1.0	AUTOSAR Administration	• Change descriptions to older releases can be found in AUTOSAR R21-11 release of this document.

Disclaimer

This work (specification and/or software implementation) and the material contained in it, as released by AUTOSAR, is for the purpose of information only. AUTOSAR and the companies that have contributed to it shall not be liable for any use of the work.

The material contained in this work is protected by copyright and other types of intellectual property rights. The commercial exploitation of the material contained in this work requires a license to such intellectual property rights.

This work may be utilized or reproduced without any modification, in any form or by any means, for informational purposes only. For any other purpose, no part of the work may be utilized or reproduced, in any form or by any means, without permission in writing from the publisher.

The work has been developed for automotive applications only. It has neither been developed, nor tested for non-automotive applications.

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

Table of Contents

1 Scope of Document 8
1.1 Constraints . . . 8
2 Conventions to be used 9
2.1 Document Conventions . . . 10
2.2 Requirements Guidelines . . . 10
2.2.1 Requirements structure . . . 10
2.2.2 Mapping to AUTOSAR releases . . . 10
3 Acronyms and abbreviations 12
4 Requirements Specification 13
4.1 Functional Overview . . . 13
4.2 Functional Requirements . . . 13
4.2.1 Configuration . . . 13
4.2.2 Wake-Up . . . 22
4.2.3 Initialization . . . 23
4.2.4 Normal Operation . . . 25
4.2.5 Shutdown Operation . . . 30
4.2.6 Fault Operation and Error Detection . . . 31
4.3 Non-Functional Requirements (Qualities) . . . 40
4.3.1 Software Architecture Requirements . . . 40
4.3.2 Software Integration Requirements . . . 42
4.3.3 Software Module Design Requirements . . . 46
4.3.4 Software Documentation Requirements . . . 72
5 References 79
5.1 Related standards and norms . . . 79
5.1.1 ISO 17356 . . . 79
A Change history of AUTOSAR traceable items 80
A.1 Traceable item history of this document according to AUTOSAR Release R26-11 . . . 80
A.1.1 Added Requirements in R26-11 . . . 80
A.1.2 Changed Requirements in R26-11 . . . 80
A.1.3 Deleted Requirements in R26-11 . . . 81
A.2 Traceable item history of this document according to AUTOSAR Release R25-11 . . . 81
A.2.1 Added Requirements in R25-11 . . . 81
A.2.2 Changed Requirements in R25-11 . . . 81
A.2.3 Deleted Requirements in R25-11 . . . 81
A.3 Traceable item history of this document according to AUTOSAR Release R24-11 . . . 82
A.3.1 Added Requirements in R24-11 . . . 82

A.3.2 Changed Requirements in R24-11 82
A.3.3 Deleted Requirements in R24-11 82
A.4 Traceable item history of this document according to AUTOSAR Release R23-11 82
A.4.1 Added Requirements in R23-11 82
A.4.2 Changed Requirements in R23-11 82
A.4.3 Deleted Requirements in R23-11 83
A.5 Traceable item history of this document according to AUTOSAR Release R22-11 83
A.5.1 Added Requirements in R22-11 83
A.5.2 Changed Requirements in R22-11 89
A.5.3 Deleted Requirements in R22-11 89
B Tracegroups 90

1 Scope of Document

The goal of this document is to define a common set of basic requirements that apply to all SW modules of the AUTOSAR Basic Software.

These requirements shall be adopted and refined for the specification of Basic SW modules .

The functional requirements defined in this document shall be referenced in each Software Specification (SWS) document of the AUTOSAR Basic Software.

1.1 Constraints

First scope for specification of requirements on Basic Software Modules are systems which are not safety relevant. For this reason safety requirements are assigned to medium priority.

2 Conventions to be used

Each requirement has its unique identifier starting with the prefix "BSW" (for "Basic Software"). For any review annotations, remarks or questions, please refer to this unique ID rather than chapter or page numbers!

- The representation of requirements in AUTOSAR documents follows the table specified in TPS_STDT_00078, see [1].
- In requirements, the following specific semantics shall be used (based on the Internet Engineering Task Force IETF).

The key words "MUST", "MUST NOT", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "MAY", and "OPTIONAL" in this document are to be interpreted as:

- SHALL: This word means that the definition is an absolute requirement of the specification.
- SHALL NOT: This phrase means that the definition is an absolute prohibition of the specification.
- MUST: This word means that the definition is an absolute requirement of the specification due to legal issues.
- MUST NOT: This phrase means that the definition is an absolute prohibition of the specification due to legal constraints.
- SHOULD: This word, or the adjective "RECOMMENDED", mean that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.
- SHOULD NOT: This phrase, or the phrase "NOT RECOMMENDED" mean that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.
- MAY: This word, or the adjective "OPTIONAL", means that an item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because the vendor feels that it enhances the product while another vendor may omit the same item. An implementation, which does not include a particular option, MUST be prepared to interoperate with another implementation, which does include the option, though perhaps with reduced functionality. In the same vein an implementation, which does include a particular option, MUST be prepared to interoperate with another implementation, which does not include the option (except, of course, for the feature the option provides.)

All requirements tables comply with the template TPS_STDT_00077.

2.1 Document Conventions

The representation of requirements in AUTOSAR documents follows the table specified in [TPS_STDT_00078], see [1, Standardization Template].

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

2.2 Requirements Guidelines

2.2.1 Requirements structure

Each module specific chapter contains a short functional description of the Basic Software Module. Requirements of the same kind within each chapter are grouped under the following headlines (where applicable):

Functional Requirements:

- Configuration (which elements of the module need to be configurable)
- Initialization
- Normal Operation
- Shutdown Operation
- Fault Operation
- . . .

Non-Functional Requirements:

- Timing Requirements
- Resource Usage
- Usability
- Output for other WGs (e.g. Description Templates, Tooling,...)
- . . .

2.2.2 Mapping to AUTOSAR releases

For each requirement defined in the document "General Requirements on Basic Software Modules", there shall be a reference to the AUTOSAR release(s) for which the requirement is valid. This is achieved by the row "AUTOSAR release" in the requirement description table.

This Requirements Specification contains general requirements that are valid for all SW modules that are part of the AUTOSAR Basic Software.

The obligatory part of the requirements is stated in the description of each requirement.

3 Acronyms and abbreviations

All acronyms and abbreviations relevant to this specification are included in the AUTOSAR Glossary [2].

4 Requirements Specification

This chapter describes all requirements driving the work to define the *General Requirements on Basic Software Modules*.

4.1 Functional Overview

The requirements on Basic Software cover the following domains:

- Body
- Powertrain
- Chassis
- Safety (assumption: covered, because hardware and system infrastructure are similar to the domains above)

The ECU application experience is taken from the following concrete applications:

- Sunroof and power window ECU
- Diesel engine ECU
- ESP ECU
- BMW, DC and VW standard software packages ('Standard Core', 'Standard Software Platform', 'Standard Software Core') including ISO 17356-3 OS [3], communication modules, bootloader, basic diagnostic functions for the domains listed above
- Infotainment control ECU

4.2 Functional Requirements

4.2.1 Configuration

[SRS_BSW_00344] BSW Modules shall support link-time configuration ⌈

Description:	Link-time configuration phase shall be supported. Link-time parameters are optional.
Rationale:	Allow configurable functionality of modules that are deployed as object code. Usually those modules are drivers.
Use Case:	–
Dependencies:	[SRS_BSW_00342] Usage of source code and object code

▽

△

Supporting Material:	–

⌋

[SRS_BSW_00404] BSW Modules shall support post-build configuration ⌈

Description:	Post-build configuration phase shall be supported. Post-build parameters are optional
Rationale:	Change ECU configuration after ECU production without an update of the whole application.

Use Case:	Type declaration of the config type
	`typedef struct ComM_ConfigType_Tag {`
	`...`
	`} ComM_ConfigType;`
	(in ComM.h)
	as a forward declaration use:
	`typedef struct ComM_ConfigType_Tag ComM_ConfigType;`
	`extern void ComM(ComM_ConfigType * ComMConfigPtr);`
	(in ComM.h)

Dependencies:	[SRS_BSW_00342] Usage of source code and object code
Supporting Material:	–

⌋

[SRS_BSW_00405] BSW Modules shall support multiple configuration sets ⌈

Description:	Modules of the AUTOSAR Basic Software shall be able to operate with more than one configuration set, selectable at start-up time.
Rationale:	Application of the same software to different cars.
Use Case:	–
Dependencies:	[SRS_BSW_00342] Usage of source code and object code
Supporting Material:	–

⌋

[SRS_BSW_00345] BSW Modules shall support pre-compile configuration ⌈

Rationale:	Static configuration is decoupled from implementation. Separation of configuration dependent data at compile time furthermore enhances flexibility, readability and reduces version management as no source code is affected.
Supporting Material:	–

⌋

[SRS_BSW_00159] All modules of the AUTOSAR Basic Software shall support a tool based configuration ⌈

Description:	All modules of the AUTOSAR Basic Software shall support a tool based configuration.
Rationale:	Integration into AUTOSAR methodology
Use Case:	The NvM can be automatically configured depending on the NV parameters and their corresponding attributes of the software components.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00167] All AUTOSAR Basic Software Modules shall provide configuration rules and constraints to enable plausibility checks ⌈

Description:	All AUTOSAR Basic Software Modules shall provide configuration rules and constraints to enable plausibility checks of configuration during ECU configuration time where possible.
Rationale:	Runtime efficiency: Checks can be made by a configuration tool or the preprocessor instead during runtime. Safety: Detect wrong or missing configurations as early as possible
Use Case:	–
Dependencies:	[SRS_BSW_00334] Provision of XML file
Supporting Material:	–

⌋

[SRS_BSW_00171] Optional functionality of a Basic-SW component that is not required in the ECU shall be configurable at pre-compile-time ⌈

Description:	Optional functionality of a Basic-SW component that is not required in the ECU shall be configurable at pre-compile-time (on/off).
Rationale:	Optional functionalities of Basic SW components which are disabled by static configuration shall not consume resources (RAM, ROM, runtime). Implementation example: in C language, preprocessing directives can be used. Ensure optimal resource consumption. There are many requirements marked with high importance but not all are used in each ECU thus resource overhead must be avoided.

▽

△

Use Case:	1. The development error detection is a statically configurable optional function that can be enabled and disabled. 2. The EEPROM write cycle reduction is a statically configurable optional function that can be enabled and disabled.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00170] The AUTOSAR SW Components shall provide information about their dependency from faults, signal qualities, driver demands ⌈

Description:	AUTOSAR SW-Components may depend on the system fault state or configuration demand of OEM or driver. These reconfiguration dependencies must be provided during ECU configuration time. This information must be used for cross checks and functional evaluation at ECU configuration time and for correct shut down/activation behavior at runtime.
Rationale:	Resolve the interdependencies between AUTOSAR SW-Components.
Use Case:	A fault of the steering angle sensor will lead to reduced function of the related AUTOSAR SW-Components. Example: • faults (CAN bus off, sensor defective, calibration data checksum error) • signal quality (lambda sensor not yet in operating temperature range) • driver demands (disable ESP) • ...
Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00380] Configuration parameters being stored in memory shall be placed into separate c-files ⌈

Description:	Configuration parameters being stored in memory shall be placed into separate c-files (effected parameters are those from link-time configuration as well as those from post-build time configuration).
Rationale:	Enable the use of different object files.
Use Case:	–
Dependencies:	[SRS_BSW_00346] Basic set of module files
Supporting Material:	[4, Layered Software Architecture]

⌋

[SRS_BSW_00419] If a pre-compile time configuration parameter is implemented as `const` it should be placed into a separate c-file ⌈

Description:	If a pre-compile time configuration parameter is implemented as `const` it should be placed into a separate c-file.
Rationale:	Enabling of object code integration. Separation of configuration from implementation.
Use Case:	–
Dependencies:	–
Supporting Material:	[4, Layered Software Architecture]

⌋

[SRS_BSW_00383] The Basic Software Module specifications shall specify which other configuration files from other modules they use at least in the description ⌈

Description:	The Basic Software Module specifications shall specify which other configuration files from other modules they use at least in the description.
Rationale:	Resolve compatibility issues
Use Case:	–
Dependencies:	[SRS_BSW_00384] List dependencies to other modules
Supporting Material:	–

⌋

[SRS_BSW_00384] The Basic Software Module specifications shall specify at least in the description which other modules they require ⌈

Description:	The Basic Software Module specifications shall specify at least in the description which other modules (in which versions) they require.
Rationale:	Resolve compatibility issues
Use Case:	–
Dependencies:	[SRS_BSW_00383] List dependencies of configuration files
Supporting Material:	–

⌋

[SRS_BSW_00388] Containers shall be used to group configuration parameters that are defined for the same object ⌈

Description:	Containers are used to group configuration parameters that are defined for the same object. Containers are to be defined whenever 1. Several configuration parameters logically belong together. 2. Configuration must be repeated with different parameter values for several entities of same type (e.g. the NvM has some parameters that are defined once for the whole module, which are collected in one container, and a set of parameters that are defined once per memory block, which are collected in another container. This second container is included in the first container and will be instantiated once for each memory block) 3. Containers may contain parameters of different configuration classes. This will not map to the software implementation!
Rationale:	Cluster the configuration parameters in order to ease the readability of code.
Use Case:	Header configuration file with sections for each container
Dependencies:	[SRS_BSW_00389] Containers shall have names

⌋

[SRS_BSW_00389] Containers shall have names ⌈

Description:	Containers shall have names - these names will map to section headers in the configuration header-files or configuration c-files containing the parameters
Rationale:	Enable referencing to the .XML document.
Use Case:	–
Dependencies:	–
Supporting Material:	[2, Glossary]

⌋

[SRS_BSW_00390] Parameter content shall be unique within the module ⌈

Description:	The same intention, logical contents or semantic shall be placed in one parameter only (There must not be several parameters with the same intention, logical contents or semantic)
Rationale:	Avoid multitude identical definitions. Ease the maintenance
Use Case:	–
Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00392] Parameters shall have a type ⌈

Description:	Each Parameter shall have a type. Types shall be based on primitive or, complex types defined within AUTOSAR specifications. I.e. they may be combined to structures, arrays etc. Parameters based on a "define" are not required to have an explicit cast to their type, they shall have an appropriate C suffix ("U" if of unsigned integer type, "L" if of integer long type and "F" if of single precision floating type).
Rationale:	–
Use Case:	–
Dependencies:	–
Supporting Material:	MISRA C Rule (see [5])

⌋

[SRS_BSW_00393] Parameters shall have a range ⌈

Description:	Each parameter shall have a list of valid values or the minimum as well as maximum values shall be specified.
Rationale:	–
Use Case:	E.g. the range is used to enable the consistency check by a tool.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00395] The Basic Software Module specifications shall list all configuration parameter dependencies ⌈

Description:	The Basic Software Module specifications must specify, via configuration constraint items, all dependencies to this or other modules configuration parameters. A dependency is for example: the value of another parameter influences or invalidates the setting of this parameter. A dependency shall be documented only once, i.e. if a dependency between two Basic Software Modules exists, then the configuration constraint item shall be described only in the Basic Software Module specification containing the influenced configuration parameter.
Rationale:	–
Use Case:	Specified parameter "Bit timing register" requires other parameters e.g., "input clock frequency" which is defined in another module.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00396] The Basic Software Module specifications shall specify the supported configuration classes for changing values and multiplicities for each parameter/container ⌈

Description:	There are three main configuration classes for changing values (applicable only to parameters) and multiplicities (applicable both to parameters and containers). The Basic Software Module specifications shall specify the classes to be supported per parameter/container. The classes are: - pre- compile time configuration - link time configuration - post build time configuration
Rationale:	Enable optimizing towards different goals of configuration.
Use Case:	–
Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00403] The Basic Software Module specifications shall specify for each parameter/container whether it supports different values or multiplicity in different configuration sets ⌈

Description:	For each container, the module shall be able to specify whether the multiplicity may be different in different configuration sets. For each parameter, the module shall be able to specify whether the multiplicity and/or the value may be different in different configuration sets.
Rationale:	Enable to specify restrictions that are necessary to optimize the implementation.
Use Case:	–
Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00397] The configuration parameters in pre-compile time are fixed before compilation starts ⌈

Description:	The configuration parameters in pre-compile time are fixed before compilation starts. The configuration of the SW element is done at source code level.
Rationale:	Ease generation of efficient code.
Use Case:	–
Dependencies:	–
Supporting Material:	[SRS_BSW_00345] Pre-compile-time configuration

⌋

[SRS_BSW_00398] The link-time configuration is achieved on object code basis in the stage after compiling and before linking ⌈

Description:	The link-time configuration is achieved on object code basis in the stage after compiling and before linking (locating).
Rationale:	Concept of configuration to support modules delivered as object code.
Use Case:	–
Dependencies:	–
Supporting Material:	[SRS_BSW_00344] Reference to link-time configuration

⌋

[SRS_BSW_00399] Parameter-sets shall be located in a separate segment and shall be loaded after the code ⌈

Description:	Parameter-sets are located in a separate segment and can be loaded after the code. (see definition of post-build time configuration in the AUTOSAR glossary). This means as well the memory layout of ext. conf. parameters must be known. This set of parameters may be optimized in a way (configuration is always located at the same address) that the pointer indirection is avoided.
Rationale:	–
Use Case:	Loadable CAN configuration or communication matrix.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00400] Parameter shall be selected from multiple sets of parameters after code has been loaded and started ⌈

Description:	Parameter will be selected from multiple sets of parameters after code has been loaded and started. During module startup (initialization) one of several configurations is selected. This configuration is typically a data structure that contains the relevant parameter values.
Rationale:	–
Use Case:	Reuse of ECUs.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00438] Configuration data shall be defined in a structure ⌈

Description:	In case of post-build configuration, or when one of multiple configuration sets shall be selectable at initialization time, the configuration parameters of a BSW module shall be reachable from a single base structure. The pointer to this structure shall be passed to the Init function of the BSW module.
Rationale:	1. Allow selection of one configuration set in case more than one set is available. 2. Allow moving of configuration in reprogrammable memory in case post-build configuration is applied.
Use Case:	Initialization concept for ComM or CanIf.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00402] Each module shall provide version information ⌈

Description:	The provided information shall be included in each module. This information shall include: Vendor and module identification numbers, AUTOSAR release version and software module version.
Rationale:	The published information contains data defined by the implementer of the SW module that doesn't change when the module is adapted (i.e. configured) to the actual HW/SW environment it is used in. It thus contains version and manufacturer information to ease the integration of different BSW modules.
Use Case:	–
Dependencies:	[SRS_BSW_00318]
Supporting Material:	–

⌋

4.2.2 Wake-Up

[SRS_BSW_00375] Basic Software Modules shall report wake-up reasons ⌈

Description:	All Basic Software Modules that implement wake-up interrupts shall report the wake-up reason to the ECU State Manager. Within this notification the ECU State Manager shall store the passed wake-up ID for later evaluation.
Rationale:	Allow ECU State Manager to decide which start-up sequence is chosen based on the wake-up reason.

▽

△

Use Case:	A body ECU can wake-up from 3 different wake-up sources. Depending on the wake-up reason, the ECU • blinks the door lock indication LEDs • performs a full start-up • evaluates the received key ID and decides to start-up and unlock or goto sleep again
Dependencies:	–
Supporting Material:	–

⌋

4.2.3 Initialization

[SRS_BSW_00101] The Basic Software Module shall be able to initialize variables and hardware in a separate initialization function ⌈

Description:	If a Basic Software Module needs to initialize variables and hardware resources, this should be done in a separate initialization function. This function shall be named `<Module name>_Init()`. This function shall only be called by the BswM or EcuM.
Rationale:	Interface to EcuM
Use Case:	–
Dependencies:	[SRS_BSW_00358] , [SRS_BSW_00414] , [SRS_BSW_00406]
Supporting Material:	–

⌋

[SRS_BSW_00416] The sequence of modules to be initialized shall be configurable ⌈

Description:	The sequence of modules to be initialized shall be configurable.
Rationale:	To enable the handling of dependencies of Basic SW-modules with the respect to environment, implementation and proprietary functionality the start-up sequence needs to be adaptable.
Use Case:	Start-up sequence is a proprietary functionality. Det dependency shall allow error detection during development.
Dependencies:	[SRS_BSW_00406]
Supporting Material:	–

⌋

[SRS_BSW_00406] API handling in uninitialized state ⌈

Description:	If a BSW Module requires initialization and the module is not initialized, then the module shall provide means to prevent undefined behavior of API calls. Additional information: 'Preventing undefined behavior' could mean e.g. reporting an error or deferring the error mitigation to the caller of the API or even resetting the SW to get to a defined state.
Rationale:	–
Use Case:	During optimization of init phase for fast startup, wrong init order has been configured and needs correction.
Dependencies:	[SRS_BSW_00369], [SRS_BSW_00450]
Supporting Material:	–

⌋

[SRS_BSW_00467] The init / deinit services shall only be called by BswM or EcuM ⌈

Description:	The init / deinit services shall only be called by BswM or EcuM
Rationale:	The module does not need to protect itself against untimely calls.
Dependencies:	[SRS_BSW_00101]
Supporting Material:	–

⌋

[SRS_BSW_00437] Memory mapping shall provide the possibility to define RAM segments which are not to be initialized during startup ⌈

Description:	Memory mapping shall provide the possibility to define RAM segments which are not to be initialized during startup (NoInit-Area). This shall be achieved by using/modifying linker and C startup routines.
Rationale:	There should be an area in the RAM, which will not be affected by a reset (clearing all memory). This area is used as storage for persistent data which are needed during normal operation (and that will not be stored in EEPROM).
Use Case:	Reset information is stored in RAM and has to be evaluated after reset.
Dependencies:	–
Supporting Material:	–

⌋

4.2.4 Normal Operation

[SRS_BSW_00168] SW components shall be tested by a function defined in a common API in the Basis-SW ⌈

Description:	If a SW component above or below RTE has the requirement to be tested by external devices e.g. in the garage, the required function shall be accessed via a common API from diagnostics services in Basic-SW (function, data interface).
Rationale:	Ensure less difference in handling and kind of API
Use Case:	Tester in the garage requires calibration of a certain SW-component e.g. steering angle sensor monitoring in the ESP. The interface must remain to be ready for moving this SW-component. This interface can also be used by XCP.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00423] BSW modules with AUTOSAR interfaces shall be describable with the means of the SW-C Template ⌈

Description:	BSW modules with AUTOSAR interfaces shall be describable with the means of the SW-C Template. The BSW description template shall therefore inherit the concepts of the SW-C Template for those BSW modules.
Rationale:	AUTOSAR Services are located in the BSW, but have to interact with AUTOSAR SW-Cs (above the RTE) via ports. Therefore the RTE generator shall be able to read the input and shall be able to generate proper RTE.
Use Case:	(1) SW-Cs use the service(s) related to the `NvM_Read` C-API of the NvM (2) SW-Cs use services of the EcuM in order to request or release the run mode
Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00424] BSW module main processing functions shall not be allowed to enter a wait state ⌈

Description:	BSW module main processing functions are not allowed to enter a wait state because the function must be able to be allocated to a basic task. (see extended and basic task according to AUTOSAR OS classification).
Rationale:	Typically, basic tasks are more efficient then extended tasks. Enables schedule ability analysis and predictability.
Use Case:	Enabling schedule ability analysis of the ECU.
Dependencies:	–

▽

△

Supporting Material:	–

⌋

[SRS_BSW_00425] The BSW module description template shall provide means to model the defined trigger conditions of schedulable objects ⌈

Description:	The BSW module description template shall provide means to model the following trigger conditions of schedulable objects: • Cyclic timings (fixed and selectable during runtime) • Sporadic events
Rationale:	The model of the timing behavior of a BSW module can serve for the purpose of (1) documentation (2) integration → supports the design of the schedule module.
Use Case:	–
Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00426] BSW Modules shall ensure data consistency of data which is shared between BSW modules ⌈

Description:	BSW Modules shall ensure data consistency of data which is shared between BSW modules. There are two possible scenarios. Scenario 1: the data is defined and managed within one BSW Module. In this case, Exclusive Areas shall be defined and documented in the BSW module description template of the managing module and used in the implementation. The exclusive areas shall be defined with a name and the accessing main functions, API services, callback functions and ISR functions. Scenario 2: the data is not managed by a BSW Module. This is only possible in case of special hardware resources like registers. In this case, the accessing modules need to disable and enable interrupts to ensure data consistency
Rationale:	To allow priority determination for preventing simultaneous access to shared resources.
Use Case:	Stop interrupt handler from corrupting a data buffer in Com due to simultaneous access via the RTE.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00427] ISR functions shall be defined and documented in the BSW module description template ⌈

Description:	ISR functions shall be defined and documented in the BSW module description template. The ISR functions shall be defined with a name and the category according to the AUTOSAR OS. In case of the intention to support memory protection a BSW module implementation shall at least support interrupt category 2.
Rationale:	Determination of locking scheme for a particular exclusive area.
Use Case:	Stop interrupt handler from corrupting a data buffer in Com due to simultaneous access via the RTE.
Dependencies:	–

⌋

[SRS_BSW_00428] A BSW module shall state if its main processing function(s) has to be executed in a specific order or sequence ⌈

Description:	A BSW module shall state if its main processing function(s) has to be executed in a specific order or sequence with respect to other BSW main processing function(s).
Rationale:	Improved integration of BSW modules.
Use Case:	Improved efficiency in the COM stack by ensuring receive and transmit call sequence.
Dependencies:	–

⌋

[SRS_BSW_00429] Access to OS is restricted ⌈

Description:	BSW modules shall only be allowed to use certain OS services. The services and their access shall be defined in SWS_BSW_General.
Rationale:	Simplification of the OS integration of BSW modules.
Use Case:	Integration of different BSW modules in one ECU.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00432] Modules should have separate main processing functions for read/receive and write/transmit data path ⌈

Description:	Modules which propagate data up (read, receive) or down (write, transmit) through the different layers of the BSW should have separate main processing functions for the read/receive and write/transmit data path.

▽

△

Rationale:	Enables efficient scheduling of the main processing functions in a more specific order to reduce execution time and latency.

Use Case:	`TASK(BSW_Scheduler_Communications) {`
	`...`
	`CanIf_MainFunction_Receive();`
	`Com_MainFunction_Receive();`
	`Com_MainFunction_Transmit();`
	`CanIf_MainFunction_Transmit();`
	`...`
	`}`

Dependencies:	[SRS_BSW_00373] Main processing function naming convention
Supporting Material:	–

⌋

[SRS_BSW_00433] Main processing functions are only allowed to be called from task bodies provided by the BSW Scheduler ⌈

Description:	Main processing functions are only allowed to be called from task bodies provided by the BSW Scheduler.
Rationale:	Indirect and in-transparent timing dependencies between BSW modules shall be prohibited.
Use Case:	–
Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00450] A Main function of a un-initialized module shall return immediately ⌈

Description:	If a Main function of an un-initialized module is called, then it shall return immediately without performing any functionality and without raising any errors.
Rationale:	Main Function processing of an un-initialized Module may result in undesired and non defined behaviour.
Use Case:	–
Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00461] Modules called by generic modules shall satisfy all interfaces requested by the generic module ⌈

Description:	If a generic module (e.g. PduR) requests an interface from an surrounding module, the surrounding module shall offer the interface, unless a configuration parameter exists which suppresses calling the interface. In case the respective module does not support the functionality of the interface, the module shall supply an 'empty function'.
Rationale:	Keep generic modules independent of specification of surrounding Modules.
Use Case:	Generic NM interface, ComM etc. need no adaptation to specific modules and CDDs
Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00451] Hardware registers shall be protected if concurrent access to these registers occur ⌈

Description:	In all cases where concurrent access to hardware registers may occur, the caller has to protect manipulation of such registers by disabling interrupts and using read-modify-write functions, unless there is specific hardware support (e.g. atomic instructions) which makes such precautions unnecessary.
Rationale:	The respective implementation restriction in the SWS General guarantees system consistency with no influence on system functionality. It only applies to BSW modules with direct access to hardware registers
Use Case:	CompletionOfCDD concept
Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00478] Timing limits of main functions ⌈

Description:	Basic Software Modules which require a periodic main function shall allow to configure the period time between $]0..\infty[$ seconds.
Rationale:	It should be avoided to standardize different upper limits for main functions. Therefore the upper limit should be open (∞). Also the lower number should exclude 0, since this value does not make sense. An implementation may restrict the upper limit to a reasonable time, but the specifications should not be limited. The lower limit is typically given by the used implementation and hardware.
Use Case:	Avoid fragmentation of different main functions caused by different upper limits.
Dependencies:	–

▽

△

Supporting Material:	–

⌋

[SRS_BSW_00496] BSW modules shall be able to handle PDU meta data ⌈

Description:	To be able to pass information that is not part of the PDU payload between layers, BSW modules shall be able to process PDU related meta data.
Rationale:	Often it is too expensive to integrate all relevant information into the PDU payload.
Use Case:	Handling of address information on J1939 buses, passing of transaction handles in SOME/IP.
Dependencies:	–
Supporting Material:	–

⌋

4.2.5 Shutdown Operation

[SRS_BSW_00336] Basic SW module shall be able to shutdown ⌈

Description:	If a Basic SW module needs to shutdown functionality (e.g. release hardware resources), this shall be done in a separate API function.
Rationale:	Interface to EcuM
Use Case:	–
Dependencies:	–
Supporting Material:	–

⌋

4.2.6 Fault Operation and Error Detection

[SRS_BSW_00337] Classification of development errors ⌈

Description:	All AUTOSAR Basic Software Modules shall report development relevant errors if development error detection is enabled: • Errors caused by software bugs • Errors caused by incorrect integration by the user • Errors caused by invalid configuration • Errors caused by bugs in the integration tools Development errors are handled like assertions: After calling the configured Det_ReportError hooks, the normal control flow of execution shall not be continued. Det shall stop execution of the entire process. This can be done for example with an endless loop or a halt statement or by creating something like an exception stack trace. If there is only one hook function configured, this might also do the exception handling and stop execution.
Rationale:	Extended error detection for debugging and especially integration.
Use Case:	The EEPROM driver provides internal checking of API parameters which is only activated for the first software integration test ('development build') and disabled afterwards ('deployment build').
Dependencies:	[SRS_BSW_00350] Development error detection keyword

⌋

[SRS_BSW_00369] All AUTOSAR Basic Software Modules shall not return specific development error codes via the API ⌈

Description:	All AUTOSAR Basic Software Modules shall not return specific development error codes via the API. In case of a detected development error, the error shall only be reported to the Det.
Rationale:	The production version of a module shall have a limited number of return values.
Use Case:	–
Dependencies:	[SRS_BSW_00337] ,[SRS_BSW_00327] ,[SRS_BSW_00357]
Supporting Material:	–

⌋

[SRS_BSW_00339] Reporting of production relevant error status ⌈

Description:	AUTOSAR Basic Software Modules shall report all production errors to the Dem (Diagnostic Event Manager).
Rationale:	• Central configuration and handling of error events instead of • spreading the handling all over the Basic Software. • Common reporting to the lamps • Common reporting to the garage • Centralized fail-safe reactions through FiM
Use Case:	Error events like (e.g `CANSM_E_BUS_OFF`) are reported to the Dem.
Dependencies:	[SRS_Diag_04063] Single Event ID for each monitoring path
Supporting Material:	–

⌋

[SRS_BSW_00422] Pre-de-bouncing of error status information is done within the Dem ⌈

Description:	Pre-de-bouncing of error status information reported via `Dem_SetEventStatus` is done within the Dem. Pre-de-bouncing is handled inside the Diagnostic Event Manager using AUTOSAR predefined generic signal de-bouncing libraries. The Diagnostic Event Manager shall define the interface to the libraries. By defining the interface it is possible for the user to implement further extensions for more complex pre-de-bouncing algorithms.
Rationale:	Central configuration and handling of error events instead of spreading the handling all over the Basic Software.
Use Case:	–
Dependencies:	[SRS_BSW_00339] Reporting of production relevant error status
Supporting Material:	–

⌋

Note related to [SRS_BSW_00422]: Figure 4.1 shows only one of several possible use cases (error detected and notified). The timer function shall be provided (in this example) in the pre-de-bouncing library of the Diagnostic Event Manager.

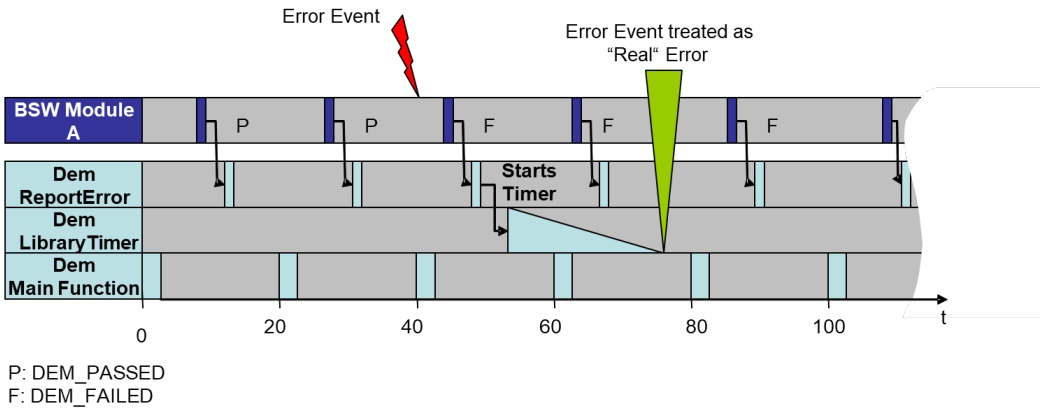


Figure 4.1: Pre-de-bouncing in Diagnostic Event Manager

[SRS_BSW_00417] Software which is not part of the SW-C shall report error events only after the Dem is fully operational. ⌈

Description:	Software which is not part of the SW-C shall report error events only after the Dem is fully operational.
Rationale:	It is only possible to store errors in error memory after the Dem is fully operational. To simplify error handling within Dem (and to gain efficiency) this requirement is needed.
Use Case:	Reporting of non plausible sensor values.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00323] All AUTOSAR Basic Software Modules shall check passed API parameters for validity ⌈

Description:	All AUTOSAR Basic Software Modules shall check passed API parameters for validity. The (minimum) conditions if a parameter needs to be treated as invalid shall be described for each parameter (e.g. check of reserved values). This checking shall be statically configurable (ON/OFF) per module with one single preprocessor switch.
Rationale:	Ease of debugging for development, efficient code for deployment.
Use Case:	The EEPROM driver provides internal checking of API parameters which is only activated for the first software integration test (’development build’) and disabled afterwards (’deployment build’).
Dependencies:	[SRS_BSW_00350],[SRS_BSW_00327]
Supporting Material:	–

⌋

[SRS_BSW_00004] All Basic SW Modules shall perform a pre-processor check of the versions of all imported include files ⌈

Description:	All Basic SW Modules shall perform a pre-processor check of the versions of all imported include files (Inter Module Checks).
Rationale:	Compatibility enforcement, error avoidance, ease of integration
Use Case:	The integration of incompatible imported files shall be avoided. The version numbers of all modules shall be listed in the Basic Software Description Template. During configuration a tool shall check whether the version numbers of all integrated modules belong to the same AUTOSAR major and minor release (same baseline). If not an error shall be reported. For the update of Basic Software Modules, version conflicts shall be detected. Example: • For included files from other modules, the AUTOSAR- MAJOR and MINOR Release Version shall be verified. I.e. Can.c includes Dem.h: Only MAJOR and MINOR Release versions shall be verified.
Dependencies:	[SRS_BSW_00003] ,[SRS_BSW_00318] ,[SRS_BSW_00402]
Supporting Material:	The term AUTOSAR baseline is defined in release management process.

⌋

[SRS_BSW_00409] All production code error ID symbols are defined by the Dem module and shall be retrieved by the other BSW modules from Dem configuration ⌈

Description:	All production code error ID symbols are defined by the Dem module and shall be retrieved by the other BSW modules from Dem configuration.
Rationale:	The error codes shall be defined in a central file, to simplify the include structure of the Dem.
Use Case:	–
Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00385] List possible error notifications ⌈

Description:	The BSW shall document all production errors, development errors and runtime errors which are supported by the BSW module.
Rationale:	Documentation, overview of errors
Use Case:	–
Supporting Material:	–

⌋

[SRS_BSW_00386] The BSW shall specify the configuration and conditions for detecting an error ⌈

Description:	The BSW shall specify the configuration for detecting an error. This configuration shall describe criteria and limits how the error is detected and possibly reset. This is applicable for production errors as well as for development errors. The BSW shall specify the conditions when the error is detected, this applies to all error types.
Rationale:	–
Use Case:	a) configuration of debounce counters (counting up/down), configuration of limits of these debounce counters etc., b) specify the library function which is to be used to debounce. c) specify whether the Diagnostic modules may request to delete errors. If so, specify how and when errors may be reset
Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00452] Classification of runtime errors ⌈

Description:	AUTOSAR Basic Software Modules may report runtime errors. Runtime errors are systematic faults that do not necessarily affect the overall system behavior (e.g. wrong PDU-Ids, wrong post-build configurations). Runtime errors are not implementation errors; they will not cause assertions and therefore not cause the abortion of the 'normal' control flow of execution (as Det will do). Runtime errors shall only be reported as an event in case of the occurrence (have set conditions only). In contrast to production errors, there is no reset conditions reported to an error handler. An error handler of runtime errors is executed synchronously and may only store the corresponding events to a memory, may call Dem and may execute any reasonable action.
Rationale:	Catch sporadic error events caused by seldom occurring systematic faults.
Use Case:	• `CANNM_E_NET_START_IND`: Reception of NM PDUs in Bus-Sleep Mode
Dependencies:	–

⌋

[SRS_BSW_00458] Classification of production errors ⌈

Description:	All AUTOSAR Basic Software Modules shall report a production error if this error is caused by any hardware problem, e.g., aging, deterioration, total hardware failure, bad production quality, incorrect assembly, etc. • and the same root cause is not detected as a production error by any other BSW module (usually, but not necessarily closer to the hardware) • and if at least one of the following criteria is met: – The error leads to an increase of emissions and must be detected to fulfill applicable regulations. – The error limits the capability of any other OBD relevant diagnostic monitors. – The error requires limp-home reactions, e.g. to prevent further damage to the hardware; or customer perceivable properties. – The garage shall be pointed to the failed component for repair actions. Production errors shall be defined in a granularity of standardized diagnostics trouble codes (e.g., SAE J2012), if possible. Note: Production errors are regular operation of the software, but not of the system. It is not any kind of exception handling. Software bugs or software misbehavior are no production errors.
Rationale:	Report errors that are useful in the field.
Use Case:	Flash is no longer writable due to aging, emission relevant adaptation maps can no longer be stored. The control unit must be replaced.
Dependencies:	If not specified by AUTOSAR, the real classification of a particular error beeing a production error may be selectable by configuration. Dependent of this classification the particular error may cause different reactions within the Dem.
Supporting Material:	–

⌋

[SRS_BSW_00488] Classification of security events ⌈

Description:	AUTOSAR Basic Software Modules and SWCs shall report security events to the module IdsM. A security event is the identified occurrence of an onboard system, service or network state indicating a possible breach of information security policy or failure of controls, or a previously unknown situation that can be security relevant. Additional information: • Security event: The identified occurrence of an onboard system, service or network state indicating a possible breach of information security policy or failure of controls, or a previously unknown situation that can be security relevant. Note: security events are not any kind of exception handling. Software bugs or software misbehavior are no security events.
Rationale:	Security events may be used to support off-board analysis of security incidents as part of an incident reponse process.
Use Case:	–

▽

△

Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00489] Reporting of security events ⌈

Description:	AUTOSAR Basic Software Modules shall report the configured security events to the IdsM (Intrusion Detection System Manager).
Rationale:	Central configuration and handling of security events instead of spreading the handling all over the Basic Software.
Use Case:	–
Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00490] List possible security events ⌈

Description:	The BSW shall document security events which are supported by the BSW module in a section "Security events". Each security event shall be documented by a "Security Event Short Name", and a "Security Event Description"
Rationale:	Documentation, overview of security events
Use Case:	–
Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00491] Specification of trigger conditions and context data ⌈

Description:	The BSW module shall specify the trigger conditions and context data for each reported security event. Additional information: • Trigger conditions: specify which situations lead to the reporting of a security event. • Context data: specifies which data is reported by the BSW module if the security event is triggered.
Rationale:	–
Use Case:	–
Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00493] Definition of security event ID symbols ⌈

Description:	All security event ID symbols are defined by the IdsM module and shall be retrieved by the other BSW modules from IdsM configuration.
Rationale:	The security ID symbols shall be defined in a central file, to simplify the include structure of the IdsM.
Use Case:	–
Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00469] Fault detection and healing of production errors ⌈

Description:	The detection of production errors shall distinguish between fault detection, failure free detection, and undecided state. Only detected faults and explicitly failure free detected states shall be reported.
Rationale:	• Avoid incorrect healing in case a failure still persists: Do not heal the OBD pending/confirmed state unless the vehicle is failure free. • Allow the system to heal if the repair is executed without using a garage tool to clear the error. • Heal only if the system is known to work, not in the absense of detected failures, i.e., ensure the correct computation of the OBD readiness information.
Use Case:	The driver re-connects a disconnected sensor, and the system is again working properly, and the production error is healed.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00470] Execution frequency of production error detection ⌈

Description:	State information are detected either by the change of the state or when checked (event-based or cyclic). Checks shall be executed as often as possible, at least once per related monitoring cycle (e.g. OBD driving cycle for emission relevant systems), or as often as required by applicable regulations, to the extend feasible.
Rationale:	• Timely detection of failures • Readiness / self-healing in case failures are absent • Ensure correct behavior of event handling during the enableconditions are not fulfilled (if enable-conditions are handled in Dem).
Use Case:	If a monitor is required to be continuous according to the regulations (CCR1968-2) the execution cycle shall be at least 2 times per second.
Dependencies:	–

▽

△

Supporting Material:	[6, CCR 1968-2]

⌋

[SRS_BSW_00471] Do not cause dead-locks on detection of production errors - the ability to heal from previously detected production errors ⌈

Description:	Production errors shall be able to heal, if a problem no longer persists.
Rationale:	Detected production errors may cause fail-safe / limp-home modes, usually through the FiM. During such operation, the detection algorithm may be disabled, preventing the error from healing. Therefore, care must be taken to avoid this situation or provide a means of healing, e.g., by starting without fail-safe / limp-home modes in the next operating cycle.
Use Case:	A component is detected as faulty and the error is reported to the Dem. As a consequence, the component is disabled and no further fault or fault free detection is possible. At the next operation cycle, the component is re-tested, and passes the tests, PASS is reported to the Dem.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00472] Avoid detection of two production errors with the same root cause. ⌈

Description:	Some production errors detect the same root cause as failure. To avoid duplicate error reports to the garage, detection of one error shall be disabled in case of the the other error, by a appropriate configuration of the FiM. Hence, the production error shall only be enabled when a permission is granted.
Rationale:	The garage will analyze all DTCs (resulting from production errors), possibly causing unnecessary repair operations if there was only one root cause.
Use Case:	This situation shall be avoided: The garage reads out two production error trouble codes, one pointing to a disconnected wiring harness, and the other to a broken control unit. The control unit is detected as broken due to the disconnected wiring harness. The garage replaces both the control unit and the wiring harness, causing unnecessary repair cost.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00497] Handling of corrupted NV data. ⌈

Description:	If an AUTOSAR Basic Software Module requires storage of its data in non-volatile memory, it shall define the behavior when the NV data is corrupted.
Rationale:	Ensure defined error behavior.
Use Case:	–
Dependencies:	–
Supporting Material:	Further information can be found in [7, SWS BSWGeneral]

⌋

4.3 Non-Functional Requirements (Qualities)

4.3.1 Software Architecture Requirements

[SRS_BSW_00161] The AUTOSAR Basic Software shall provide a microcontroller abstraction layer which provides a standardized interface to higher software layers ⌈

Description:	The AUTOSAR Basic Software shall provide a microcontroller abstraction layer which provides a standardized interface to higher software layers.
Rationale:	Portability and reusability. Encapsulate implementation details of a specific microcontroller from higher software layers.
Use Case:	Exchange microcontroller ST10 with STAR12 without affecting higher software layers interfacing with the microcontroller abstraction layer.
Dependencies:	–
Supporting Material:	[4]

⌋

[SRS_BSW_00162] The AUTOSAR Basic Software shall provide a hardware abstraction layer ⌈

Description:	The AUTOSAR Basic Software shall provide a hardware abstraction layer which provides a stable interface to higher software layers which is independent from the ECU hardware layout.
Rationale:	Keep the impact of changes in the ECU hardware layout as small as possible. Portability and reusability of modules of higher software layers. Flexibility for changes in the ECU hardware layout.

▽

△

Use Case:	• Change the hardware layout of the ECU (e.g. PortA.5 → PortD.7) without affecting software layers interfacing with the hardware abstraction layer. • Use the NvM with an internal and/or external EEPROM. • Provide uniform access to analog signals using the on-chip ADC or an external ADC ASIC.
Dependencies:	–
Supporting Material:	[4]

⌋

[SRS_BSW_00005] Modules of the μC Abstraction Layer (MCAL) may not have hard coded horizontal interfaces ⌈

Description:	Modules of the μC Abstraction Layer (MCAL) may not have hard coded horizontal interfaces. Necessary interactions (e.g. GPT triggered ADC conversion) shall be implemented by using statically configurable notifications (callbacks).
Rationale:	Avoidance of strong coupling, ease of integration, better structure
Use Case:	–
Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00415] Interfaces which are provided exclusively for one module shall be separated into a dedicated header file ⌈

Description:	Interfaces and the corresponding types which are provided exclusively for one module should be separated into a dedicated header file. This should prevent the inclusion of the <ModuleName>.h file. The format of the file name shall be: <ModuleName>_<User>.h Comment: Common definitions for different interfaces (e.g. types) shall be defined in a common header file (e.g. <Module Name>.h).
Rationale:	Encapsulate an interface between modules in an include file
Use Case:	Example: CanIf_Pdur.h, CanIf_Nm.h
Dependencies:	[SRS_BSW_00346] Basic set of module files.
Supporting Material:	< Module name > shall be derived from "List of Basic Software Modules", [7] (2. . . 8 characters). <User> shall be the user module from the same list.

⌋

4.3.2 Software Integration Requirements

[SRS_BSW_00164] The Implementation of interrupt service routines shall be done by the Operating System, complex drivers or modules ⌈

Description:	Only the Operating System, complex drivers and modules of the microcontroller abstraction layer are allowed to implement interrupt service routines. If a transition from an interrupt service routine to an operating system task is needed, it shall take place at the lowest level possible of the Basic Software. In the case of CAT2 ISRs this shall be at the latest in the RTE. In the case of CAT1 ISRs this shall be at the latest in the Interface layer. This means: no interrupts on application level.
Rationale:	Portability and reusability. The implementation of interrupt service routines is highly microcontroller dependent.
Use Case:	Exchange microcontroller ST10 with STAR12 without affecting higher software layers.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00325] The runtime of interrupt service routines and functions that are running in interrupt context shall be kept short ⌈

Description:	The runtime of interrupt service routines and functions that are running in interrupt context should be kept short. Where an interrupt service routine is likely to take a long time, an operating system task should be used instead.
Rationale:	Real time behavior, avoid blocking of the whole system.
Use Case:	An ISR calls a callback which is calling other callbacks.
Dependencies:	[SRS_BSW_00333] Documentation of callback function context
Supporting Material:	–

⌋

[SRS_BSW_00342] It shall be possible to create an AUTOSAR ECU out of modules provided as source code and modules provided as object code, even mixed ⌈

Description:	It shall be possible to create an AUTOSAR ECU out of modules provided as source code and modules provided as object code, even mixed.
Rationale:	Allow both: • IP protection and guaranteed test coverage : object code • High efficiency and configurability at ECU configuration time (by integrator) : source code
Use Case:	Some simple drivers could be provided as object code. More complex and configurable modules could be provided as source code or even generated code.
Dependencies:	[SRS_BSW_00344] Configuration at Runtime
Supporting Material:	–

⌋

[SRS_BSW_00343] The unit of time for specification and configuration of Basic SW modules shall be preferably in physical time unit ⌈

Description:	The unit of time for specification and configuration of Basic SW modules shall be preferably in physical time unit, not ticks. Nevertheless for some module "tick" parameters are accepted
Rationale:	The duration of a "tick" varies from system to system.
Use Case:	The software specification defines the unit (e.g. μs, s) and software configuration uses these units. OS Modules require time parameter values in ticks.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00160] Configuration files of AUTOSAR Basic SW module shall be readable for human beings ⌈

Description:	Files holding configuration data for AUTOSAR Basic SW modules shall have a format that is readable and understandable by human beings.
Rationale:	Plausibility checking, comparison of different versions of configuration data.
Use Case:	XML is readable.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00453] BSW Modules shall be harmonized ⌈

Description:	If an SWS of a BSW module is allowed to be linked to more than one implementation of another BSW module into an AUTOSAR binary image, then all involved SWS's shall ensure that all externally visible C identifiers (i.e. types, variables, macros, functions, etc) are defined such that no conflicts can arise for surrounding BSW modules using these multiple implementations at compile time and that no ambiguity exists at link time.
Rationale:	If the rule is not followed, systems with multiple implementations of one BSW Module will mostly get an error at compile time or link time.
Use Case:	In CAN Driver there are 2 type definitions i) Can_IdType ii) Can_PduType which are used in CanIf. Can_IdType can be uint16 or uint32 type. If there are 2 CAN drivers implemented in one Autosar system by two different vendors and both implementations defines Can_IdType differently, then it will lead to compilation / linking failure in the system. Hence it should be made sure that there are no ambiguities.
Dependencies:	[SRS_BSW_00456]
Supporting Material:	–

⌋

[SRS_BSW_00456] A Header file shall be defined in order to harmonize BSW Modules ⌈

Description:	If more than one implementation of a BSW Module is linked into an Autosar system which results in conflict of externally visible C Identifiers (i.e. types, variables, macros etc), a common header file may define all the conflicting identifiers. The header file shall be named as <Module Abbreviation>_GeneralTypes.h Module Abbreviation is defined in Basic Software Module List. It refers to BSW Module which has more than one implementation.
Rationale:	BSW systems with multiple implementations of one BSW Module will mostly get an error at compile time or link time, if they are not harmonized.
Use Case:	–
Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00457] Callback functions of Application software components shall be invoked by the Basis SW ⌈

Description:	An AUTOSAR Basic Software module shall only invoke the callback functions of Application Software Components and/or Sensor/Actuator SW-Components through the Client Server communication of the RTE. CDDs are not affected by this requirement.
Rationale:	RTE shall not be bypassed if AUTOSAR Basic Software Modules are calling callbacks provided by Application SW-Cs and/or Sensor/Actuator SW-Cs, because only these components are restricted to having only AUTOSAR interfaces. This is to support memory partitioning.
Use Case:	–
Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00479] Interfaces for handling request from external devices ⌈

Description:	Drivers for external devices shall use and offer the same interfaces as internal drivers when calling or being called by the interface module.
Rationale:	In general, the driver for external devices shall follow the same SWS specification. For external drivers, when calling Det, use the same module ID as the internal drivers.
Use Case:	System which uses an internal and an independent external HW Wdg module.
Dependencies:	–
Supporting Material:	[SRS_BSW_00005]

⌋

[SRS_BSW_00483] BSW Modules shall handle buffer alignments internally ⌈

Description:	BSW modules which require certain alignment of buffers shall not impose any additional requirements on the users. I.e. Buffers passed as arguments shall be treated as specified by their base types; alignment results from base type and platform specifics.
Rationale:	Avoid conflicting alignment requirements within software stack. It shall be possible to allocate RAM buffers without the need to consider alignment requirements throughout the software stack.
Use Case:	Interoperability of components; avoid "hidden" restrictions in API usage (imposing stricter alignments limits the value range for pointer parameters). Especially drivers shall hide HW/peripheral's alignment requirements from upper layers; they shall not map a HW's/peripheral's alignment requirements to data buffers, which would result in propagating them to upper layers.
Dependencies:	–

▽

△

Supporting Material:	–

⌋

4.3.3 Software Module Design Requirements

[SRS_BSW_00007] All Basic SW Modules written in C language shall conform to the MISRA C 2025 Standard. ⌈

Description:	MISRA C (see [5]) describes programming rules for the C programming language and a process to implement and follow these rules. Only in technically reasonable, exceptional cases MISRA violations are permissible. Such violations against MISRA rules shall be clearly identified and documented within comments in the C source code (including rationale why MISRA rule is violated). Examples of MISRA rules violations shall look like: /* MR12 RULE XX VIOLATION: Reason */ /* MR12 DIR XX VIOLATION: Reason */
Rationale:	Portability, maintainability, error avoidance, safety
Use Case:	Software for safety relevant systems
Dependencies:	–
Supporting Material:	[5]

⌋

[SRS_BSW_00300] Unique Name of Basic Software Modules ⌈

Description:	AUTOSAR shall identify all AUTOSAR Basic Software Modules by a unique name. Related file names shall contain the module name. Additional Information: Convention for module related files: • <Module name>*.* • Module name: 2..8 letters, derived from [7]
Rationale:	The module name serves as an identifier and classification mechanism in order to group module related files.
Use Case:	Example: Can.c, Can.h
Dependencies:	–
Supporting Material:	[7]

⌋

[SRS_BSW_00413] An index-based accessing of the instances of BSW modules shall be done ⌈

Description:	If instances of BSW modules are characterized by: • same vendor **and** • same functionality **and** • same hardware device they shall be accessed index based.
Rationale:	–
Use Case:	–
Dependencies:	[SRS_BSW_00347] Naming separation of drivers
Supporting Material:	–

⌋

[SRS_BSW_00347] A Naming seperation of different instances of BSW drivers shall be in place ⌈

Description:	Driver modules shall be named according to the following rules (only for implementation, not for the software specification): • First the module name has to be listed: <Module Abbreviation> • After that the vendor Id defined in the AUTOSAR vendor list has to be given <Vendor Id> • At last a vendor specific name (the vendor API infix) follows <Vendor API infix> • Only for API names, last name shall be <API Service name> • All parts shall be separated by underscores "_". • This naming extension applies to the following externally visible elements of the module: – File names – API names – Published parameters – Memory allocation keyword • For API names, <Vendor specific name> should be followed by "_" and then <API Service Name>.
Rationale:	Avoidance of name clashes
Use Case:	Examples: • CAN driver: `Can_21_Ext_CanDriver.c` • Published parameters: `CAN_21_EXT_SW_MAJOR_VERSION` • API: `Can_21_Ext_Init`
Dependencies:	–

▽

△

Supporting Material:	[7] List of Basic Software Modules (Module Abbreviations)

⌋

[SRS_BSW_00441] Naming convention for type, macro and function ⌈

Description:	All AUTOSAR Basic Software Modules shall label enumeration literals and #defines according to the following scheme: • Composition: <Module Abbreviation>_<Specific name> • <Module Abbreviation> shall be written in UPPERCASE • <Specific name> shall be written in UPPERCASE • <Module Abbreviation> and <Specific name> shall be separated by underscore • If <Specific name> consists of several words, they shall be separated by underscore The `#define` `E_OK` and `E_NOT_OK` are exceptions to this.
Rationale:	Enhance readability and unique classification of enumeration literals and #defines identifiers.

Use Case:	Example #define:
	`#define EEP_PARAM_CONFIG`
	`#define EEP_SIZE`
	
	Example enumeration literals:
	`typedef enum`
	`{`
	`EEP_DRA_CONFIG = 0,`
	`EEP_ARE = 1,`
	`EEP_EV = 2`
	`} Eep_NotificationType;`

Dependencies:	[SRS_BSW_00331] [SRS_BSW_00327] [SRS_BSW_00335]
Supporting Material:	–

⌋

[SRS_BSW_00305] Data types naming convention ⌈

Description:	All AUTOSAR Basic Software Modules shall label data types according to the following scheme: • Composition of type: <Module name>_<Type name>Type • Only one underscore between module name and type name • < Type name > shall be written in UpperCamelCase. Note: Basic AUTOSAR types ([SRS_BSW_00304]) need not support the scheme defined here.
Rationale:	Enhance readability and unique classification of data type identifiers.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00307] Global variables naming convention ⌈

Description:	• All AUTOSAR Basic Software Modules shall label global variables according to the following scheme: • Composition of name: <Module name>_<Variable name> • Only one underscore between module name and variable name • Spelling of name: First letter of each word upper case, consecutive letters lower case
Rationale:	Enhance readability and unique classification of global variables.
Use Case:	–
Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00310] API naming convention ⌈

Description:	All AUTOSAR Basic Software Modules shall implement an API based on the following naming rules: • Composition of API: <Module name>_ServiceName() `<Mip>_<Sn>` • Where <Mip> is the Module implementation prefix and <Sn> is the API Service name • Module name: 2..8 letters, derived from [7] • Only one underscore between module name and service name • Spelling of API: First letter of each word upper case, consecutive letters lower case

▽

△

Rationale:	Avoidance of name clashes, uniform AUTOSAR API; The API shows to which module it belongs
Use Case:	• `Can_TransmitFrame()` • `Nm_RequestBusCommunication()` • `Adc_Init()` • `Eep_Write()` • `Nvm_GetState()`
Dependencies:	–
Supporting Material:	[7]

⌋

[SRS_BSW_00494] ServiceInterface argument with a pointer datatype ⌈

Description:	A ServiceInterface argument with a pointer datatype (DATA_REFERENCE) shall have a Ptr postfix in its argument type and -name. No other argument type shall use this postfix in its argument type or -name. The argument direction of a pointer datatype is always IN.
Rationale:	A pointer datatype in ServiceInterface has limitations in use, so it should be easily identifiable. The argument direction IN is because the RTE just transfers the pointer and not the content behind the pointer. An argument direction other than IN would theoretically allocate a buffer for this pointer in the RTE, wherefore the RTE is rejecting such configurations (according to SWS_Rte_07662).
Use Case:	–
Dependencies:	–
Supporting Material:	–

⌋

Note: [SRS_BSW_00494] does not apply on regular C-API.

[SRS_BSW_00373] The main processing function of each AUTOSAR Basic Software Module shall be named according the defined convention ⌈

Description:	The main processing function of each AUTOSAR Basic Software Module shall be named according to the following rule: <Module name>_MainFunction_<module specific extension> () Module specific extension shall be used to distinguish between multiple main processing functions of one module (e.g. Cluster index, Rx /Tx . . .). If only one main processing function exists in one module no module specific extension is required. It is responsibility of the modules to either define one main processing function and handle all the processing internally or define multiple main processing functions with appropriate module specific extensions. This depends on Module requirements. Main processing functions shall have no parameters and no return value. Main processing functions shall not be re-entrant.
Rationale:	Many modules have one or more functions that have to be called cyclically (e.g. within an OS Task) and that do the main work of the module. These shall have unique names.

Use Case:	Possible main processing function of EEPROM driver:
	`void Eep_MainFunction(void)`
	Possible main processing functions of FlexRay driver:
	`void Fr_MainFunction_TxClst1(void)`
	`void Fr_MainFunction_TxClst2(void)`
	`void Fr_MainFunction_RxClst1(void)`
	`void Fr_MainFunction_RxClst2(void)`
	Please Note: The Use case is no recommendation for the particular Module, it just illustrates Main processing function possibilities.

Supporting Material:	<Module name> shall be derived from "List of Basic Software Modules", [7] (2. . . 8 characters)

⌋

[SRS_BSW_00327] Error values naming convention ⌈

Description:	All AUTOSAR Basic Software Modules shall apply the following naming rules for all error values: • Error values shall have only CAPITAL LETTERS • Naming convention: <MODULENAME>_E_<ERRORNAME> • If <ERRORNAME> consists of several words, they shall be separated by underscores

▽

△

Rationale:	Avoidance of name clashes, uniform AUTOSAR error values; The error shows to which module it belongs.
Use Case:	The EEPROM driver has the following error values: • `EEP_E_BUSY` • `EEP_E_PARAM_ADDRESS` • `EEP_E_PARAM_LENGTH` • `EEP_E_WRITE_FAILED`
Dependencies:	[SRS_BSW_00331] [SRS_BSW_00369]
Supporting Material:	< MODULENAME > shall be derived from "List of Basic Software Modules", [7] (2. . . 8 characters)

⌋

[SRS_BSW_00335] Status values naming convention ⌈

Description:	All AUTOSAR Basic Software Modules shall apply the following naming rules for status values that are visible outside of the module: • Status values shall have only CAPITAL LETTERS • If <STATUSNAME> consists of several words, they shall be separated by underscores
Rationale:	Avoidance of name clashes, uniform AUTOSAR status values; The status value shows to which module it belongs.
Use Case:	The Eeprom driver has the following status values: • `EEP_UNINIT` • `EEP_IDLE` • `EEP_BUSY`
Dependencies:	[SRS_BSW_00331] Separation of error and status values
Supporting Material:	< MODULENAME > shall be derived from "List of Basic Software Modules", [7] (2. . . 8 characters)

⌋

[SRS_BSW_00350] All AUTOSAR Basic Software Modules shall allow the enabling/disabling of detection and reporting of development errors. ⌈

Description:	All AUTOSAR Basic Software Modules shall allow the enabling/disabling of detection and reporting of development errors. It shall be configurable and the default value of the configuration shall be that those error type is disabled.
Rationale:	Provide module wide debug instrumentation facilities. Each defined keyword has to be properly documented.

▽

△

Use Case:	Example:
	In Eep.h:
	`#define EEP_DEV_ERROR_DETECT STD_ON /* detection module wide enabled */`
	In source Eep.c:
	`#include "Eep.h"`
	`...`
	`#if ( EEP_DEV_ERROR_DETECT == STD_ON )`
	`..`
	`.. development errors to be detected`
	`..`
	`#endif /* EEP_DEV_ERROR_DETECT */`

Dependencies:	[SRS_BSW_00337],
Supporting Material:	< MODULENAME > shall be derived from "List of Basic Software Modules", [7] (2. . . 8 characters)

⌋

[SRS_BSW_00408] All AUTOSAR Basic Software Modules configuration parameters shall be named according to a specific naming rule ⌈

Description:	All AUTOSAR Basic Software Modules configuration parameters shall be named according to the following naming rules: • Naming convention: <Module Abbreviation><ParameterName> < Module Abbreviation > is the prefix derived from [7]. < ParameterName > may consist of several words which may or may not be separated by underscore. The configuration parameter name can either be in UpperCamelCase or Uppercase
Rationale:	Avoidance of name clashes, uniform AUTOSAR configuration naming.
Use Case:	Example: `CanIfTxConfirmation`
Dependencies:	–
Supporting Material:	< Module Abbreviation > shall be derived from "List of Basic Software Modules", [7] (2. . . 8 characters)

⌋

[SRS_BSW_00410] Compiler switches shall have defined values ⌈

Description:	Compiler switches shall be compared with defined values. Simple checks if a compiler switch is defined shall not be used. In general the symbols which switch functionality on or off are defined in Std_Types.h
Rationale:	C-Language allows asking for defined symbols. This shall be avoided.

Use Case:	Example:
	Do :
	`#if ( EEP_DEV_ERROR_DETECT == STD_ON )`
	`..`
	Don't:
	`#ifdef EEP_DEV_ERROR_DETECT`
	`..`

Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00411] AUTOSAR Basic Software Modules shall apply a naming rule for enabling/disabling the existence of API(s) ⌈

Description:	AUTOSAR Basic Software Modules shall apply the following naming rule to the configuration parameters for enabling/disabling API(s): <Module Abbreviation><FunctionName>Api <Module Abbreviation> is the prefix derived from [7]. <FunctionName> is the name of the API without the module implementation prefix and without any underscore. The default value of the configuration parameter shall be `FALSE` which indicates that this API is not available.
Rationale:	Exclude API(s) (features) which are not required in a system.
Use Case:	Example: `CanTpReadParameterApi`
Dependencies:	[SRS_BSW_00408]
Supporting Material:	–

⌋

[SRS_BSW_00463] Naming convention of callout prototypes ⌈

Description:	Each callout function shall be mapped to its own memory section and memory class. These memory classes will then be mapped to the actually implemented memory classes at integration time.
	The following naming convention shall be used:
	— Start section definition: —
	`#define MSN_START_SEC_CBN_CODE`
	— Stop section definition: —
	`#define MSN_STOP_SEC_CBN_CODE`
	— Function prototype definition: —
	`void MSN_Cbn (void);`
	Where:
	MSN: Module Short Name as officially defined in AUTOSAR (see supporting material).
	CBN: Call Back Name, which shall have the same spelling of the Callback name including module reference but using only capital letters.
	Cbn: Callback name using the conventional Camel Case notation for API names.

Rationale:	The memory segment used for a callout is not known to the module developer. The integrator needs the freedom to map callouts independently from the module's design.

Use Case:	In order to ensure uniqueness, it is recommended to use the function's name to derive the name of the memory section and the name of the memory class.
	For example:
	`#define COM_START_SEC_COM_SOMECALLOUT_CODE`
	`#include "Com_MemMap.h"`
	`void Com_SomeCallout(void);`
	`#define COM_STOP_SEC_COM_SOMECALLOUT_CODE`
	`#include "Com_MemMap.h"`

Dependencies:	–
Supporting Material:	[7]

⌋

[SRS_BSW_00464] File names shall be considered case sensitive regardless of the filesystem in which they are used ⌈

Description:	File names shall be considered case sensitive regardless of the filesystem in which they are used.
Rationale:	Some file systems do not distinguish between file names spelled with the same letters but with different cases. Allowing such variability in the definitions can cause ambiguities.
Use Case:	If different implementers implement modules using same names with different cases, the compile and link process shall have unpredictable results depending on the file system on which they are executed, leading eventually to errors (source or object file not found). Example of wrong implementation: The file name "ModuleAbc.h" is defined in a SWS; "moduleabc.h" and "ModuleAbc.h" are implemented by two different implementers and then included in modules developed by different implementers. If the file "moduleabc.h" is included with the directive `#include <ModuleAbc.h>` on a case sensitive file system, the file won't be found.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00465] It shall not be allowed to name any two files so that they only differ by the cases of their letters ⌈

Description:	It shall not be allowed to name any two files so that they only differ by the cases of their letters.
Rationale:	Problems deriving potentially ambiguous name definitons must be avoided already in the specification phase
Use Case:	In a SWS the include files: RTE.h rte.h are defined and they are specified to contain different information. At compile time a compiler running in a file system which does not distinguish between cases shall include one or the other in a non predictable order.
Dependencies:	[SRS_BSW_00464]
Supporting Material:	–

⌋

[SRS_BSW_00480] Null pointer errors shall follow a naming rule ⌈

Description:	NULL pointer error naming convention. The name for the development errors for NULL pointer violations is `<MIP>_E_PARAM_POINTER`.
Rationale:	Harmonization of standard
Use Case:	–
Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00487] Errors for module initialization shall follow a naming rule ⌈

Description:	The name for the development errors for uninitialized modules is `<MIP>_E_UNINIT`.
Rationale:	Harmonization of standard
Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00498] Errors for module double initialization shall follow a naming rule ⌈

Description:	The name pattern for the development errors for calling the initialization function of a module when the module is currently already initialized (or in an initialized state) is `<MIP>_E_ALREADY_INITIALIZED`.
Rationale:	Harmonization of standard
Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00481] Invalid configuration set selection errors shall follow a naming rule ⌈

Description:	Invalid configuration set selection error naming convention The name for the invalid configuration set selection errors `<MIP>_E_INIT_FAILED`.
Rationale:	Harmonization of standard
Use Case:	–
Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00346] All AUTOSAR Basic Software Modules shall provide at least a basic set of module files ⌈

Description:	All AUTOSAR Basic Software Modules shall provide a standardized set of unique header files which separates source code from configuration. The exact structure shall be defined in [7, SWS BSW General] including the naming convention using the module name.
Rationale:	Source code and configuration are strictly separated. User defined configurations will not imply a change of the original source code. Other BSW Modules which need to access configuration data can do this without need for source code change.
Use Case:	Separate post built configuration data from precompile configuration data, source code from configuration data in general etc..
Dependencies:	[SRS_BSW_00345], [SRS_BSW_00347], [SRS_BSW_00314], [SRS_BSW_00419]
Supporting Material:	< Module name > shall be derived from "List of Basic Software Modules", [7] (2. . . 8 characters)

⌋

[SRS_BSW_00314] All internal driver modules shall separate the interrupt frame definition from the service routine ⌈

Description:	All internal driver modules shall separate the interrupt frame definition from the service routine in the following way: • <Module name>_Irq.c: implementation of interrupt frame • <Module name>.c: implementation of service routine called from interrupt frame
Rationale:	Flexibility using different compilers and/or different OS integrations
Use Case:	The interrupt could be realized as ISR frame of the operating system or implemented directly without changing the driver code. The service routine can be called directly during module test without the need of causing an interrupt.
Dependencies:	–
Supporting Material:	< Module name > shall be derived from "List of Basic Software Modules", [7] (2. . . 8 characters)

⌋

[SRS_BSW_00447] Standardizing Include file structure of BSW Modules Implementing Autosar Service ⌈

Description:	• A Basic Software Module implementing an Autosar Service shall include its Application Types Header file in the Module Header File. • Data Types used in Standard Interface and Standard AUTOSAR Interface shall only be defined in RTE Types Header file only. • A Basic Software Module implementing an Autosar Service shall include Rte_<ModuleShortName>.h as AUTOSAR Service Application Header File, providing the interface for interaction with the RTE. • A Basic Software Module implementing an Autosar Service shall include its AUTOSAR Service Application Header File in module files, which are using RTE interfaces. The Application Header file shall not be included in module files, which are in included directly or indirectly by other modules. Data Type NvM_RequestResultType used in BSW C-API `NvM_GetErrorStatus` and in the AUTOSAR Interface "NvMService" operation `GetErrorStatus (OUT NvM_RequestResultType RequestResultPtr);` is same. The proper types shall be generated in Rte_Type.h. Rte_Type.h shall be included in BSW module header file via Rte_"Service"_Type.h Rte_Type.h shall be included in SW-C module header file via Rte_"Swc"_Type.h
Rationale:	Standardizing Include Header file structure will allow common data types to be defined in RTE Types header files. This will avoid double and inconsistent definition of data types in both BSW and Software Component. This will also avoid type casts if SW-Cs are communicating with Autosar Services.
Use Case:	All BSW Services which are called by Application SW-C and share data types. E.g. Asynchronous NvRAM Block request result returned by the operation `GetErrorStatus` and API service `NvM_GetErrorStatus`.
Dependencies:	–
Supporting Material:	Please see the Figure "Relationships between RTE Header Files" and related information in Chapter "RTE Modules" of [8, SWS Rte].

⌋

[SRS_BSW_00348] All AUTOSAR standard types and constants shall be placed and organized in a standard type header file ⌈

Description:	All AUTOSAR standard types and constants shall be placed and organized in a standard type header file. Standard type header file naming convention: Std_Types.h This standard type header file shall • include the Platform specific type header (Platform_Types.h) • define the type Std_ReturnType • define values for `E_OK` and `E_NOT_OK` • define values for `STD_ON`, `STD_OFF`, `STD_HIGH`, `STD_LOW`, `STD_ACTIVE`, `STD_IDLE`
Rationale:	Provide uniform framework wide access to standard types to be used by all modules.
Use Case:	Each module that uses AUTOSAR integer data types and/or the standard return type shall include the file Std_Types.h.
Dependencies:	[SRS_BSW_00357], [SRS_BSW_00353]

Supporting Material:	Important note for implementation of this header file:
	Because `E_OK` is already defined within [3, OSEK OS], `E_OK` has to be checked for being already defined:
	`/* for ISO 17356-3 compliance this typedef has been added */`
	`#ifndef STATUSTYPEDEFINED`
	`#define STATUSTYPEDEFINED`
	`typedef unsigned char StatusType;`
	`#define E_OK 0`
	`#endif`

⌋

[SRS_BSW_00353] All type definitions of target and compiler specific scope shall be placed and organized in a single type header ⌈

Description:	All type definitions of target and compiler specific scope shall be placed and organized in a single type header. Name of platform types header file: Platform_Types.h
Rationale:	Separate compiler and μC-specific types from standard types.
Use Case:	Changing the microcontroller and/or compiler shall only affect a limited number of files.
Dependencies:	[SRS_BSW_00308], [SRS_BSW_00348]

▽

△

Supporting Material:	–

⌋

[SRS_BSW_00301] All AUTOSAR Basic Software Modules shall only import the necessary information ⌈

Description:	All AUTOSAR Basic Software Modules shall only import the necessary information (i.e. header files) that is required to fulfill the modules functional requirements.
Rationale:	Promote defensive module layout. Modules shall not import functionality that could be misused. Shorten compile times.
Use Case:	–
Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00302] All AUTOSAR Basic Software Modules shall only export information needed by other modules ⌈

Description:	All AUTOSAR Basic Software Modules shall export only that kind of information in their correspondent header-files explicitly needed by other modules.
Rationale:	Prevent other modules accessing functionality and data that is 'none of their business'.
Use Case:	The NvM shall not know all processor registers because someone has included the processor register file in another header file used by the NvM.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00328] All AUTOSAR Basic Software Modules shall avoid the duplication of code ⌈

Description:	All AUTOSAR Basic Software Modules should avoid the duplication of code.
Rationale:	Avoid bugs during maintenance
Use Case:	A module contains 4 code segments which are equal. During maintenance of the module 3 of them have been updated, 1 has been forgotten → bug.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00312] Shared code shall be reentrant ⌈

Description:	All AUTOSAR Basic Software Modules implementing shared code shall ensure reentrancy if code is exposed to preemptive or parallel environments. For multi-core systems, reentrancy shall be ensured for unrestricted concurrent execution of that service on several cores (concurrency safety).
Rationale:	Shared code eases functional composition, reusability, code size reduction and maintainability. As a drawback, shared code shall be implemented reentrant if it is used in preemptive environments or on multiple partitions in parallel. Please note that an implementation that is reentrant on single core systems might not be concurrency safe when used in a Multi-Core environment.
Use Case:	A subroutine or function is reentrant if a single copy of the routine can be called from several task contexts simultaneously without conflict. Use the following reentrancy techniques: • Avoid use of static and/or global variables • Guard static and/or global variables using blocking mechanisms • Use dynamic stack variables
Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00006] The source code of software modules above the μC Abstraction Layer (MCAL) shall not be processor and compiler dependent. ⌈

Description:	Those software modules have to be developed once and shall be compilable for all processor platforms without any changes. Any necessary processor or compiler specific instructions (e.g. memory locators, pragmas, use of atomic bit manipulations etc.) have to be exported to macros and include files.
Rationale:	Minimize number of variants and development effort
Use Case:	NvM, Network Management, . . .
Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00439] Enable BSW modules to handle interrupts ⌈

Description:	Autosar shall allow BSW modules to define and handle Interrupts.
Rationale:	–
Use Case:	In the case where the entire driver is delivered as source this isn't a problem. In the case where the MCAL BSW module is delivered as object code, the interrupt handler could be written as a pair of small stubs (a cat1 stub and a cat2 stub) that are delivered as source, compiled as necessary, and simply call the main handler.

▽

△

Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00448] Module SWS shall not contain requirements from other modules ⌈

Description:	It shall not be allowed for a module SWS to add requirements from other modules: • If a requirement is missing, then raise an Rfc, possibly resulting in a valid requirement within the module. • For this valid requirement give reference of the document where original requirement resides.
Rationale:	Increase consistency between SWS documents, ease change management of documents.
Use Case:	CAN Driver SWS using requirements from MCU Driver SRS. In this case there shall be a valid CAN requirement in SRS which refers to the particular requirement in MCU Driver SRS
Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00449] BSW Service APIs used by Autosar Application Software shall return a Std_ReturnType ⌈

Description:	Every BSW Service API called by application software via RTE shall return a Std_ReturnType, return value. Refer to the Port Interface Section of the respective module, to confirm if the APIs are accessed by the RTE.
Rationale:	RTE call of BSW service always expect a return value of Std_ReturnType
Use Case:	RTE always expects return type of Std_ReturnType for the BSW Service API Call, any other return type or void shall cause incompatibility between the RTE and BSW.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00357] For success/failure of an API call a standard return type shall be defined ⌈

Description:	For success/failure of an API call, a return type is defined in Std_Types.h which indicates the success or failure of the call.
Rationale:	Enforces usage of already defined types instead of attempting to override existing ones. If different success states can occur and they are of interest for the caller then different return values need to be defined.
Dependencies:	[SRS_BSW_00348], [SRS_BSW_00377],[SRS_BSW_00359]
Supporting Material:	–

⌋

[SRS_BSW_00377] A Basic Software Module can return a module specific types ⌈

Description:	A Basic Software Module can return a module specifictypes.

Rationale:	Example for possibility 1:
	`uint8 Can_Write(...)`
	return values: `E_OK` (0), `CAN_BUSY` (1), `E_OK` is taken from Std_Types.h, `CAN_BUSY` is #defines in can.h.
	Note: no strong type checking possible because return type is uint8 and values are only #defines. `E_OK` can be used.
	Example for possibility 2:
	`Can_ReturnType Can_Write(...)`
	Return values: `CAN_OK`, `CAN_BUSY`, Can_ReturnType is an enumeration type in can.h:
	`typedef enum`
	`{`
	`CAN_OK = 0,`
	`CAN_BUSY = 2,`
	`} Can_ReturnType;`
	Note: strong type checking possible because only the values of the enumeration may be assigned to variables of type Can_ReturnType. `E_OK` cannot be used here!

Dependencies:	[SRS_BSW_00357]
Supporting Material:	–

⌋

[SRS_BSW_00304] All AUTOSAR Basic Software Modules shall use only AUTOSAR data types instead of native C data types ⌈

Description:	All AUTOSAR Basic Software Modules shall not use the native C data types.
Rationale:	MISRA C compliance, portability, reusability. The usage of native C-data types (e.g. char, int, short, long, float, double, ...) is forbidden as size and/or sign and/or accuracy are not unambiguously defined and therefore are platform specific.
Use Case:	Portability between different architectures.
Dependencies:	[SRS_BSW_00353]
Supporting Material:	[SRS_BSW_00007] MISRA C, [5]

⌋

[SRS_BSW_00378] AUTOSAR shall provide a boolean type ⌈

Description:	For simple logical values and for API return values (if applicable) AUTOSAR shall provide a boolean type. The only allowed operations shall be: assignment, return, test for quality.
Rationale:	Repeating requests of several WGs to define a boolean data type.

Use Case:	API return value. Example:
	In file Eep.h:
	`#include "Std_Types.h" /* this automatically includes Platform_Types.h */`
	`boolean Eep_Busy(void) {...}`
	In calling module:
	`if (Eep_Busy() == FALSE) {...}`

Dependencies:	–
Supporting Material:	Compiler vendors that provide a boolean data type that cannot be disabled have to change their compiler (i.e. make it ANSI C compliant).

⌋

[SRS_BSW_00306] AUTOSAR Basic Software Modules shall be compiler and platform independent ⌈

Description:	All AUTOSAR Basic Software Modules shall not use compiler or platform specific keywords directly.
Rationale:	Direct use of not standardized keywords like "_near", "_far", "_pascal" in the frameworks source code will create compiler and platform dependencies that must strictly be avoided. If no precautions were made, portability and reusability of influenced code is deteriorated and effective release management is costly and hard to maintain.
Dependencies:	–

▽

△

Supporting Material:	–

⌋

[SRS_BSW_00308] AUTOSAR Basic Software Modules shall not define global data in their header files, but in the C file ⌈

Description:	AUTOSAR Basic Software Modules shall not define global data in their header files. If global variables have to be used, the definition shall take place in the C file.
Rationale:	Avoid multiple definition and uncontrolled spreading of global data, limit visibility of global variables.
Use Case:	–
Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00309] All AUTOSAR Basic Software Modules shall indicate all global data with read-only purposes by explicitly assigning the const keyword ⌈

Description:	All AUTOSAR Basic Software Modules shall indicate all global data with read-only purposes by explicitly assigning the const keyword.
Rationale:	In principle, all global data shall be avoided due to extra blocking efforts when used in preemptive runtime environments. Unforeseen effects are to occur if no precautions were made. If data is intended to serve as constant data, global exposure is permitted only if data is explicitly declared read-only using the const qualifier.
Use Case:	`const uint8 MaxPayload = 0x18;`
Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00484] Input parameters of scalar and enum types shall be passed as a value. ⌈

Description:	All input parameters of scalar or enum type shall be passed as a value..
Use Case:	For example a function named <Mip>_SomeFunction with a return type of Std_ReturnType and a single parameter named SomeParameter of type uint8 is defined with the following signature: Std_ReturnType <Mip>_SomeFunction(uint8 SomeParameter);

▽

△

Supporting Material:	–

⌋

[SRS_BSW_00485] Input parameters of structure type shall be passed as a reference to a constant structure ⌈

Description:	All input parameters of structure type shall be passed as a reference constant structure
Rationale:	Passing input parameters of structure type by value would result in additional run-time overhead due to efforts for copying the whole structure.
Use Case:	For example a function named <Mip>_SomeFunction with a return type of Std_ReturnType and a single parameter named SomeParameter of type Some Structure (where SomeStructure is a struct) is defined with the following signature: Std_ReturnType <Mip>_SomeFunction(const SomeStructure * Some Parameter);
Supporting Material:	–

⌋

[SRS_BSW_00486] Input parameters of array type shall be passed as a reference to the constant array base type ⌈

Description:	All input parameters of array type shall be passed as a reference to the constant array base type
Rationale:	This effectively matches the behavior specified in the ISO-C:99 namely that a "declaration of a parameter as 'array of type' shall be adjusted to 'qualified pointer to type'".
Use Case:	For example a function named <Mip>_SomeFunction with a return type of Std_ReturnType and a single parameter named SomeParameter of type array of uint8 is defined with the following signature: Std_ReturnType <Mip>_SomeFunction(const uint8 * SomeParameter);
Supporting Material:	–

⌋

[SRS_BSW_00358] The return type of init() functions implemented by AUTOSAR Basic Software Modules shall be void ⌈

Description:	The return type of init() functions implemented by AUTOSAR Basic Software Modules shall be void.
Rationale:	Errors in initialization data shall be detected during configuration time (e.g. by configuration tool).

▽

△

Use Case:	–
Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00414] Init functions shall have a pointer to a configuration structure as single parameter ⌈

Description:	For post-build time configuration, or when multiple configuration sets are available, the pointer to the base configuration structure (see [SRS_BSW_00438]) shall be passed to the init function of the BSW module. For pre-compile and link time configuration, when only one configuration set is available, a NULL_PTR shall be passed for this parameter. <Mip>_ConfigType It shall be used for init function argument
Rationale:	–
Use Case:	Example: `void Eep_Init (const Eep_ConfigType *ConfigPtr)`
Dependencies:	[SRS_BSW_00101], [SRS_BSW_00358], [SRS_BSW_00400]
Supporting Material:	–

⌋

[SRS_BSW_00359] Callback Function Return Types for AUTOSAR BSW ⌈

Description:	All AUTOSAR Basic Software Modules callback functions shall use void return type unless an error is returned. Additional Information: The caller of the callback function shall consider the case that the environment (RTE) can return infrastructure errors (refer [SWS_Rte_02593]) e.g. in case the partition is currently not available. In case the callback is used as notification only, the caller can assume that always E_OK is returned. Older releases of the RTE required C/S operations to return a Std_ReturnType, but this has changed, so void is also possible. For backward compatibility reason existing interface may deviate from the requirement.
Rationale:	Callbacks could be used for notifications.
Use Case:	–
Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00360] AUTOSAR Basic Software Modules callback functions are allowed to have parameters ⌈

Description:	AUTOSAR Basic Software Modules callback functions are allowed to have parameters.
Rationale:	Enhance flexibility and scope of callback functionality.
Use Case:	If callback functions do serve as simple triggers, no parameter is necessary to be passed. If additional data is to be passed to the caller within the callback scope, it shall be possible to forward the contents of that data using a parameter.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00440] The callback function invocation by the BSW module shall follow the signature provided by RTE to invoke servers via `Rte_Call` API ⌈

Description:	The callback function invocation by the BSW module, which is routed via RTE shall follow the signature provided by RTE to invoke servers via `Rte_Call` API.
Rationale:	The callback function has to be to be compatible to `Rte_Call` API of the RTE to enable a type safe configuration and implementation of AUTOSAR Services and IO Hardware Abstraction. Instance pointers are in Basic Software not allowed.
Use Case:	–
Dependencies:	[SRS_BSW_00359]
Supporting Material:	–

⌋

[SRS_BSW_00330] It shall be allowed to use macros instead of functions where source code is used and runtime is critical ⌈

Description:	It shall be allowed to use macros instead of functions where source code is used and runtime is critical. It shall be allowed to use inline functions for the same purpose. Inline functions have the advantage (compared to macros) that the compiler can do type checking of function parameters and return values.
Rationale:	Improve runtime behavior.
Use Case:	–
Dependencies:	Macros as well as inline functions are only possible when source code is delivered.
Supporting Material:	Attention has to be paid within reentrant systems. MISRA C (see [5])

⌋

[SRS_BSW_00331] All Basic Software Modules shall strictly separate error and status information ⌈

Description:	All Basic Software Modules shall strictly separate error and status information.
Rationale:	Common API specification of AUTOSAR Basic Software Modules.
Dependencies:	–
Supporting Material:	[SRS_BSW_00327] Error values naming convention [SRS_BSW_00335] Status values naming convention

⌋

[SRS_BSW_00462] All Standardized Autosar Interfaces shall have unique requirement Id / number ⌈

Description:	All Standardized Autosar Interfaces shall have unique requirement Id / number. The purpose of the standardized AUTOSAR Interface definition is to provide a standard which has to be considered by Software Components defining Service ports. Therefore the Port of the Software Component has to be at least compatible to the definition in the related SWS document.
Rationale:	The standardized Autosar Interfaces definitions are not binding without a requirement Id.
Use Case:	A SWC deviating from the Operation names will hinder the integration process. This is because the Ports of the Service and the Ports of the Service User (SWC) are NOT compatible.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00454] An alternative interface without a parameter of category DATA_REFERENCE shall be available. ⌈

Description:	In case an AUTOSAR interface supports a parameter of category DATA_REFERENCE, an alternative interface without such a parameter shall be available.
Rationale:	A DATA_REFERENCE will show up as a pointer to data at the interface level. AUTOSAR BSW can not do a full safety check on the pointer because the size of the data is not known. Therefore, if safety is an issue, the alternative interface needs to be available and to be used. In general, to avoid such problems, AUTOSAR Interfaces should not use a DATA_REFERENCE.
Use Case:	ECUs with safety requirements where an application with lower privileges passes a DATA_REFERENCE to the BSW with higher privileges.
Dependencies:	–

▽

△

Supporting Material:	–

⌋

[SRS_BSW_00477] The functional interfaces of AUTOSAR BSW modules shall be specified in C99 ⌈

Description:	The specification of functional interfaces of AUTOSAR BSW modules shall be specified in C99 according to ISO/IEC 9899:1999. This implies that languages, which can interface to C99 can be used for application programming. The number of significant characters allowed for external identifier can exceed the specified minimum (31 characters).
Rationale:	A useful reduction of programming languages to current programming languages reduces the impacts on AUTOSAR definitions and specifications due to logical and/or technical differences of different programming languages.
Use Case:	AUTOSAR implementation in C, C++.
Dependencies:	–
Supporting Material:	[9, ISO C99]

⌋

[SRS_BSW_00459] It shall be possible to concurrently execute a service offered by a BSW module in different partitions ⌈

Description:	If a service supports concurrent execution in different partitions , the implementation of the service shall ensure that concurrent handling of calls is performed in a multi-core safe manner, i.e. several calls from different partitions to the same service at the same time do not interfere with each other. This can be implemented, for example, by using exclusive areas and re-entrant code.
Rationale:	Performance, error avoidance.
Use Case:	BSW running on multi core systems
Dependencies:	[SRS_BSW_00426]
Supporting Material:	–

⌋

[SRS_BSW_00460] Reentrancy Levels ⌈

Description:	If BSW is executed in multiple partitions, all functions in a BSW module entity shall conform to the reentrancy level enforced by the API description of the implemented Bsw module entry, or to a stricter level. If the description of a module entity contains the optional reentrancy level attribute, this level must be compliant to the reentrancy requirements of the implemented entry, and the implementation must conform to the reentrancy level enforced by the description of the module entity. If a module can be invoked locally in multiple partitions, reentrancy also implies safe execution in parallel on multiple cores.
Rationale:	Performance, error avoidance.
Use Case:	BSW running on multi core systems
Dependencies:	[SRS_BSW_00426]
Supporting Material:	–

⌋

4.3.4 Software Documentation Requirements

[SRS_BSW_00009] All Basic SW Modules shall be documented according to a common standard. ⌈

Description:	The module documentation shall contain at least the following items: • Cover sheet with title, version number, date, author, document status, document name • Change history with version number, date, author, change description, document status • Table of contents (navigable) • Functional overview • Source file list and description • Module requirements • Used resources (interrupts, μC peripherals etc.) • Integration description (OS, interface to other modules etc.) • Configuration description with parameter, description, unit, validrange, default value, relation to other parameters The module documentation shall also contain examples for • the correct usage of the API • the configuration of the module
Rationale:	User acceptance, maintainability, usability
Use Case:	Standard Core

▽

△

Dependencies:	[SRS_BSW_00010], [SRS_BSW_00333]
Supporting Material:	–

⌋

[SRS_BSW_00401] Documentation of multiple instances of configuration parameters shall be available ⌈

Description:	"Multiplicity" defines how many times an entity (in this case configuration parameter) is instanciated. The multiplicity of each configuration parameter has to be documented. It shall be documented what determines the number of entries (e.g. "one per frame").
Rationale:	Overall (throughout the complete Basic Software) harmonization of configuration parameter naming.
Use Case:	Id of a PDU is multiple time present dependent on the number of PDUs to be sent/received.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00172] The scheduling strategy that is built inside the Basic Software Modules shall be compatible with the strategy used in the system ⌈

Description:	The scheduling strategy that is built inside the Basic Software Modules shall be compatible with the strategy used in the system. To achieve this, the following items shall be traced by BSW specific SWS: • polling / event driven • cooperative / pre-emptive • for each cyclic function: invocation rate (either fixed value or allowed range) • execution order (dependencies to other modules) • synchronous / asynchronous processing • minimum and maximum function runtime (WCET) • maximum interrupt rate
Rationale:	Today scheduling mechanisms differ between ECUs. A Basic Software Module provides several entry points to be accessed by the other Basic Software Modules/surrounding system. E.g. a function can react directly on event or by a scheduled polling. The differences may result in difference in real-time requirements, system load, latency etc.!

▽

△

Use Case:	On the one hand it is possible to avoid any direct function call between BSW modules by using only scheduling and data interface - more deterministic. On the other hand it is possible that beside the scheduling additional functional interfaces exists to control BSW modules - less deterministic. The integrating SW-system and its SW-architecture might restrict direct function calls between SW-components. Thus not any SW-component will fit in this SW-system.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00010] The memory consumption of all Basic SW Modules shall be documented for a defined configuration for all supported platforms. ⌈

Description:	For software integration the following data shall be available for each supported platform: - RAM/ROM consumption
Rationale:	Due to stability of documentation, this information is provided in a separate document for each supported platform. If a further platform is added, the module documentation remains unvalid
Use Case:	Microcontroller selection, software integration, configuration of operating system
Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00333] For each callback function it shall be specified if it is called from interrupt context or not ⌈

Description:	For each callback function it shall be specified if it is called from interrupt context or not.
Rationale:	User awareness. The code inside a callback function called from an ISR has to be kept short.
Use Case:	Some notification function is called from an ISR of the CAN driver. The user filling this callback function has to know that the function is running in interrupt context!
Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00374] All Basic Software Modules shall provide a readable module vendor identification ⌈

Description:	All Basic Software Modules shall provide a readable module vendor identification in their published parameters.
	Naming convention:
	`<MODULENAME>_VENDOR_ID`
	The vendor ID shall be represented in uint16 (16 bit).
	The format of the vendor identification shall be only:
	`#define <MODULENAME>_VENDOR_ID 0x0000u`
	without any cast to allow a verification in pre-processor.

Rationale:	Allow identification of module vendor
Use Case:	`EEP_VENDOR_ID`
Dependencies:	–
Supporting Material:	• < MODULENAME > shall be derived from "List of Basic Software Modules", [7] (2. . . 8 characters) • For AUTOSAR vendor ID list see [10].

⌋

[SRS_BSW_00379] All software modules shall provide a module identifier in the header file and in the module XML description file. ⌈

Description:	All software modules shall provide a module ID both in the header file and in the module XML description file. The value shall be taken from the Basic Software Module List. Naming convention: <MODULENAME>_MODULE_ID The module ID shall be represented in uint16 (16 bit).
Rationale:	Required for error reporting to Default Error Tracer (Det).
Use Case:	In file Eep.h: `#define EEP_MODULE_ID 90`
Dependencies:	[SRS_BSW_00334] Provision of XML file
Supporting Material:	• < MODULENAME > shall be derived from "List of Basic Software Modules", [7] (2. . . 8 characters) • [7], column 'Module ID', defines the module IDs.

⌋

[SRS_BSW_00003] All software modules shall provide version and identification information ⌈

Description:	All software modules shall provide a readable software version number in all import header files. Version number macros can be used for checking (Inter Module Checks) and reading out the software version of a software module during compile time and runtime. It is preferred to derive this information from the version management system automatically.
Rationale:	Compatibility checking, configuration supervision
Use Case:	–
Dependencies:	[SRS_BSW_00004], [SRS_BSW_00318]
Supporting Material:	–

⌋

[SRS_BSW_00318] Each AUTOSAR Basic Software Module file shall provide version numbers in the header file ⌈

Description:	Each AUTOSAR Basic Software Module file shall provide version numbers in the header file as defined below: Naming convention: • <MODULENAME>_SW_MAJOR_VERSION • <MODULENAME>_SW_MINOR_VERSION • <MODULENAME>_SW_PATCH_VERSION • <MODULENAME>_AR_RELEASE_MAJOR_VERSION • <MODULENAME>_AR_RELEASE_MINOR_VERSION • <MODULENAME>_AR_RELEASE_REVISION_VERSION AR: Major/Minor/Revision Release Version number of AUTOSAR specification which the appropriate implementation is based on. SW: Major/minor/patch version number of the vendor specific implementation of the module. The numbering shall be vendor specific Each number shall be represent able as uint8 (8 bit).
Rationale:	Allow version identification and version checking in between software modules.

▽

△

Use Case:	Example: Adc vendor module version 1.14.9; implemented according to the AUTOSAR Release 4.0, Revision 1
	`#define ADC_SW_MAJOR_VERSION 1`
	`#define ADC_SW_MINOR_VERSION 14`
	`#define ADC_SW_PATCH_VERSION 9`
	`#define ADC_AR_RELEASE_MAJOR_VERSION 4`
	`#define ADC_AR_RELEASE_MINOR_VERSION 0`
	`#define ADC_AR_RELEASE_REVISION_VERSION 1`

Dependencies:	[SRS_BSW_00321], [SRS_BSW_00374], [SRS_BSW_00402]
Supporting Material:	< MODULENAME > shall be derived from "List of Basic Software Modules", [7] (2. . . 8 characters)

⌋

[SRS_BSW_00321] The version numbers of AUTOSAR Basic Software Modules shall be enumerated according specific rules ⌈

Description:	The version numbers of AUTOSAR Basic Software Modules shall be enumerated according to the following rules: • Increasing a more significant digit of a version number resets all less significant digits • The PATCH_VERSION is incremented if the module is still upwards and downwards compatible (e.g. bug fixed) • The MINOR_VERSION is incremented if the module is still downwards compatible (e.g. validfunctionality added) • The MAJOR_VERSION is incremented if the module is not compatible any more (e.g. existing API valid)
Rationale:	Provide unambiguous version identification for each module, provide version cross check as well as basic version retrieval facilities. Compatibility is always visible!
Use Case:	Example: ADC module with version 1.14.2: • Versions 1.14.2 and 1.14.9 are exchangeable. 1.14.2 may contain bugs • Version 1.14.2 can be used instead of 1.12.0, but not vice versa • Version 1.14.2 cannot be used instead of 1.15.4 or 2.0.0
Dependencies:	[SRS_BSW_00318]
Supporting Material:	–

⌋

[SRS_BSW_00341] Module documentation shall contains all needed informations ⌈

Description:	All needed informations by user of a module shall be stated in the documentation of the module.
Rationale:	Opportunity to identify uniquely the specific microprocessor, including known bugs in the silicon so that its compatibility with the software can be established.
Use Case:	Different mask revisions of e.g. TriCore
Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00334] Machine readable module description ⌈

Description:	All Basic Software Modules shall provide an machine readable description of the meta data which is required for the SW integration process.
Rationale:	Ensure system consistency and correctness, support automatic processing (e.g. interface checks)
Use Case:	–
Dependencies:	–
Supporting Material:	–

⌋

[SRS_BSW_00351] Encapsulation of compiler specific methods to map objects ⌈

Description:	AUTOSAR shall define header files which encapsulate compiler and platform specific differences in memory mapping such that BSW modules and SWC can be implemented of compiler and platform.
Rationale:	AUTOSAR focuses on embedded systems with only restricted memory resources. Therefore a precise mapping of objects (data,code) is needed.
Use Case:	Storage of different objects in memory with fast access times.
Dependencies:	–
Supporting Material:	[RS_BRF_00057]

⌋

5 References

[1] Standardization Template
AUTOSAR_FO_TPS_StandardizationTemplate

[2] Glossary
AUTOSAR_FO_TR_Glossary

[3] ISO 17356-3: Road vehicles – Open interface for embedded automotive applications – Part 3: OSEK/VDX Operating System (OS)

[4] Layered Software Architecture
AUTOSAR_CP_EXP_LayeredSoftwareArchitecture

[5] MISRA C:2025 Guidelines for the use of the C language in critical systems, ISBN-13: 978-1911700197

[6] Motor Vehicle Pollution Control Devices
https://www.iso.org

[7] General Specification of Basic Software Modules
AUTOSAR_CP_SWS_BSWGeneral

[8] Specification of RTE Software
AUTOSAR_CP_SWS_RTE

[9] ISO/IEC 9899:1999
https://www.iso.org

[10] AUTOSAR Vendor ID List
https://www.autosar.org/vendor-id

5.1 Related standards and norms

5.1.1 ISO 17356

[3, OSEK OS]

A Change history of AUTOSAR traceable items

A.1 Traceable item history of this document according to AUTOSAR Release R26-11

A.1.1 Added Requirements in R26-11

Number	Heading
[SRS_BSW_00497]	Handling of corrupted NV data.
[SRS_BSW_00498]	Errors for module double initialization shall follow a naming rule

Table A.1: Added Requirements in R26-11

A.1.2 Changed Requirements in R26-11

Number	Heading
[SRS_BSW_00007]	All Basic SW Modules written in C language shall conform to the MISRA C 2025 Standard.
[SRS_BSW_00304]	All AUTOSAR Basic Software Modules shall use only AUTOSAR data types instead of native C data types
[SRS_BSW_00330]	It shall be allowed to use macros instead of functions where source code is used and runtime is critical
[SRS_BSW_00339]	Reporting of production relevant error status
[SRS_BSW_00353]	All type definitions of target and compiler specific scope shall be placed and organized in a single type header
[SRS_BSW_00385]	List possible error notifications
[SRS_BSW_00386]	The BSW shall specify the configuration and conditions for detecting an error
[SRS_BSW_00392]	Parameters shall have a type
[SRS_BSW_00402]	Each module shall provide version information
[SRS_BSW_00406]	API handling in uninitialized state
[SRS_BSW_00411]	AUTOSAR Basic Software Modules shall apply a naming rule for enabling/ disabling the existence of API(s)
[SRS_BSW_00458]	Classification of production errors
[SRS_BSW_00469]	Fault detection and healing of production errors

Table A.2: Changed Requirements in R26-11

A.1.3 Deleted Requirements in R26-11

Number	Heading
[SRS_BSW_00407]	Each BSW module shall provide a function to read out the version information of a dedicated module implementation
[SRS_BSW_00466]	Classification of extended production errors
[SRS_BSW_00482]	Get version information function shall follow a naming rule
[SRS_BSW_00492]	Reporting of security events during startup

Table A.3: Deleted Requirements in R26-11

A.2 Traceable item history of this document according to AUTOSAR Release R25-11

A.2.1 Added Requirements in R25-11

Number	Heading
[SRS_BSW_00496]	BSW modules shall be able to handle PDU meta data

Table A.4: Added Requirements in R25-11

A.2.2 Changed Requirements in R25-11

Number	Heading
[SRS_BSW_00348]	All AUTOSAR standard types and constants shall be placed and organized in a standard type header file
[SRS_BSW_00490]	List possible security events
[SRS_BSW_00492]	Reporting of security events during startup
[SRS_BSW_00493]	Definition of security event ID symbols

Table A.5: Changed Requirements in R25-11

A.2.3 Deleted Requirements in R25-11

Number	Heading
[SRS_BSW_00394]	The Basic Software Module specifications shall specify the scope of the configuration parameters

Table A.6: Deleted Requirements in R25-11

A.3 Traceable item history of this document according to AUTOSAR Release R24-11

A.3.1 Added Requirements in R24-11

none

A.3.2 Changed Requirements in R24-11

Number	Heading
[SRS_BSW_00300]	Unique Name of Basic Software Modules
[SRS_BSW_00334]	Machine readable module description
[SRS_BSW_00347]	A Naming seperation of different instances of BSW drivers shall be in place
[SRS_BSW_00359]	Callback Function Return Types for AUTOSAR BSW
[SRS_BSW_00385]	List possible error notifications
[SRS_BSW_00406]	API handling in uninitialized state
[SRS_BSW_00488]	Classification of security events

Table A.7: Changed Requirements in R24-11

A.3.3 Deleted Requirements in R24-11

Number	Heading
[SRS_BSW_00473]	Classification of transient faults

Table A.8: Deleted Requirements in R24-11

A.4 Traceable item history of this document according to AUTOSAR Release R23-11

A.4.1 Added Requirements in R23-11

none

A.4.2 Changed Requirements in R23-11

none

A.4.3 Deleted Requirements in R23-11

Number	Heading
[SRS_BSW_00495]	If tracing is enabled, all AUTOSAR Basic Software Modules should allow tracing its entry and exit points.

Table A.9: Deleted Requirements in R23-11

A.5 Traceable item history of this document according to AUTOSAR Release R22-11

A.5.1 Added Requirements in R22-11

Number	Heading
[SRS_BSW_00003]	All software modules shall provide version and identification information
[SRS_BSW_00004]	All Basic SW Modules shall perform a pre-processor check of the versions of all imported include files
[SRS_BSW_00005]	Modules of the μC Abstraction Layer (MCAL) may not have hard coded horizontal interfaces
[SRS_BSW_00006]	The source code of software modules above the μC Abstraction Layer (MCAL) shall not be processor and compiler dependent.
[SRS_BSW_00007]	All Basic SW Modules written in C language shall conform to the MISRA C 2012 Standard.
[SRS_BSW_00009]	All Basic SW Modules shall be documented according to a common standard.
[SRS_BSW_00010]	The memory consumption of all Basic SW Modules shall be documented for a defined configuration for all supported platforms.
[SRS_BSW_00101]	The Basic Software Module shall be able to initialize variables and hardware in a separate initialization function
[SRS_BSW_00159]	All modules of the AUTOSAR Basic Software shall support a tool based configuration
[SRS_BSW_00160]	Configuration files of AUTOSAR Basic SW module shall be readable for human beings
[SRS_BSW_00161]	The AUTOSAR Basic Software shall provide a microcontroller abstraction layer which provides a standardized interface to higher software layers
[SRS_BSW_00162]	The AUTOSAR Basic Software shall provide a hardware abstraction layer
[SRS_BSW_00164]	The Implementation of interrupt service routines shall be done by the Operating System, complex drivers or modules
[SRS_BSW_00167]	All AUTOSAR Basic Software Modules shall provide configuration rules and constraints to enable plausibility checks
[SRS_BSW_00168]	SW components shall be tested by a function defined in a common API in the Basis-SW
[SRS_BSW_00170]	The AUTOSAR SW Components shall provide information about their dependency from faults, signal qualities, driver demands

▽

△

Number	Heading
[SRS_BSW_00171]	Optional functionality of a Basic-SW component that is not required in the ECU shall be configurable at pre-compile-time
[SRS_BSW_00172]	The scheduling strategy that is built inside the Basic Software Modules shall be compatible with the strategy used in the system
[SRS_BSW_00300]	All AUTOSAR Basic Software Modules shall be identified by an unambiguous name
[SRS_BSW_00301]	All AUTOSAR Basic Software Modules shall only import the necessary information
[SRS_BSW_00302]	All AUTOSAR Basic Software Modules shall only export information needed by other modules
[SRS_BSW_00304]	All AUTOSAR Basic Software Modules shall use only AUTOSAR data types instead of native C data types
[SRS_BSW_00305]	Data types naming convention
[SRS_BSW_00306]	AUTOSAR Basic Software Modules shall be compiler and platform independent
[SRS_BSW_00307]	Global variables naming convention
[SRS_BSW_00308]	AUTOSAR Basic Software Modules shall not define global data in their header files, but in the C file
[SRS_BSW_00309]	All AUTOSAR Basic Software Modules shall indicate all global data with read-only purposes by explicitly assigning the const keyword
[SRS_BSW_00310]	API naming convention
[SRS_BSW_00312]	Shared code shall be reentrant
[SRS_BSW_00314]	All internal driver modules shall separate the interrupt frame definition from the service routine
[SRS_BSW_00318]	Each AUTOSAR Basic Software Module file shall provide version numbers in the header file
[SRS_BSW_00321]	The version numbers of AUTOSAR Basic Software Modules shall be enumerated according specific rules
[SRS_BSW_00323]	All AUTOSAR Basic Software Modules shall check passed API parameters for validity
[SRS_BSW_00325]	The runtime of interrupt service routines and functions that are running in interrupt context shall be kept short
[SRS_BSW_00327]	Error values naming convention
[SRS_BSW_00328]	All AUTOSAR Basic Software Modules shall avoid the duplication of code
[SRS_BSW_00330]	It shall be allowed to use macros instead of functions where source code is used and runtime is critical
[SRS_BSW_00331]	All Basic Software Modules shall strictly separate error and status information
[SRS_BSW_00333]	For each callback function it shall be specified if it is called from interrupt context or not
[SRS_BSW_00334]	All Basic Software Modules shall provide an XML file that contains the meta data
[SRS_BSW_00335]	Status values naming convention

▽

△

Number	Heading
[SRS_BSW_00336]	Basic SW module shall be able to shutdown
[SRS_BSW_00337]	Classification of development errors
[SRS_BSW_00339]	Reporting of production relevant error status
[SRS_BSW_00341]	Module documentation shall contains all needed informations
[SRS_BSW_00342]	It shall be possible to create an AUTOSAR ECU out of modules provided as source code and modules provided as object code, even mixed
[SRS_BSW_00343]	The unit of time for specification and configuration of Basic SW modules shall be preferably in physical time unit
[SRS_BSW_00344]	BSW Modules shall support link-time configuration
[SRS_BSW_00345]	BSW Modules shall support pre-compile configuration
[SRS_BSW_00346]	All AUTOSAR Basic Software Modules shall provide at least a basic set of module files
[SRS_BSW_00347]	A Naming seperation of different instances of BSW drivers shall be in place
[SRS_BSW_00348]	All AUTOSAR standard types and constants shall be placed and organized in a standard type header file
[SRS_BSW_00350]	All AUTOSAR Basic Software Modules shall allow the enabling/disabling of detection and reporting of development errors.
[SRS_BSW_00351]	Encapsulation of compiler specific methods to map objects
[SRS_BSW_00353]	All integer type definitions of target and compiler specific scope shall be placed and organized in a single type header
[SRS_BSW_00357]	For success/failure of an API call a standard return type shall be defined
[SRS_BSW_00358]	The return type of init() functions implemented by AUTOSAR Basic Software Modules shall be void
[SRS_BSW_00359]	All AUTOSAR Basic Software Modules callback functions shall avoid return types other than void if possible
[SRS_BSW_00360]	AUTOSAR Basic Software Modules callback functions are allowed to have parameters
[SRS_BSW_00369]	All AUTOSAR Basic Software Modules shall not return specific development error codes via the API
[SRS_BSW_00373]	The main processing function of each AUTOSAR Basic Software Module shall be named according the defined convention
[SRS_BSW_00374]	All Basic Software Modules shall provide a readable module vendor identification
[SRS_BSW_00375]	Basic Software Modules shall report wake-up reasons
[SRS_BSW_00377]	A Basic Software Module can return a module specific types
[SRS_BSW_00378]	AUTOSAR shall provide a boolean type
[SRS_BSW_00379]	All software modules shall provide a module identifier in the header file and in the module XML description file.
[SRS_BSW_00380]	Configuration parameters being stored in memory shall be placed into separate c-files
[SRS_BSW_00383]	The Basic Software Module specifications shall specify which other configuration files from other modules they use at least in the description

▽

△

Number	Heading
[SRS_BSW_00384]	The Basic Software Module specifications shall specify at least in the description which other modules they require
[SRS_BSW_00385]	List possible error notifications
[SRS_BSW_00386]	The BSW shall specify the configuration and conditions for detecting an error
[SRS_BSW_00388]	Containers shall be used to group configuration parameters that are defined for the same object
[SRS_BSW_00389]	Containers shall have names
[SRS_BSW_00390]	Parameter content shall be unique within the module
[SRS_BSW_00392]	Parameters shall have a type
[SRS_BSW_00393]	Parameters shall have a range
[SRS_BSW_00394]	The Basic Software Module specifications shall specify the scope of the configuration parameters
[SRS_BSW_00395]	The Basic Software Module specifications shall list all configuration parameter dependencies
[SRS_BSW_00396]	The Basic Software Module specifications shall specify the supported configuration classes for changing values and multiplicities for each parameter/container
[SRS_BSW_00397]	The configuration parameters in pre-compile time are fixed before compilation starts
[SRS_BSW_00398]	The link-time configuration is achieved on object code basis in the stage after compiling and before linking
[SRS_BSW_00399]	Parameter-sets shall be located in a separate segment and shall be loaded after the code
[SRS_BSW_00400]	Parameter shall be selected from multiple sets of parameters after code has been loaded and started
[SRS_BSW_00401]	Documentation of multiple instances of configuration parameters shall be available
[SRS_BSW_00402]	Each module shall provide version information
[SRS_BSW_00403]	The Basic Software Module specifications shall specify for each parameter/container whether it supports different values or multiplicity in different configuration sets
[SRS_BSW_00404]	BSW Modules shall support post-build configuration
[SRS_BSW_00405]	BSW Modules shall support multiple configuration sets
[SRS_BSW_00406]	A static status variable denoting if a BSW module is initialized shall be initialized with value 0 before any APIs of the BSW module is called
[SRS_BSW_00407]	Each BSW module shall provide a function to read out the version information of a dedicated module implementation
[SRS_BSW_00408]	All AUTOSAR Basic Software Modules configuration parameters shall be named according to a specific naming rule
[SRS_BSW_00409]	All production code error ID symbols are defined by the Dem module and shall be retrieved by the other BSW modules from Dem configuration
[SRS_BSW_00410]	Compiler switches shall have defined values

▽

△

Number	Heading
[SRS_BSW_00411]	All AUTOSAR Basic Software Modules shall apply a naming rule for enabling/disabling the existence of the API
[SRS_BSW_00413]	An index-based accessing of the instances of BSW modules shall be done
[SRS_BSW_00414]	Init functions shall have a pointer to a configuration structure as single parameter
[SRS_BSW_00415]	Interfaces which are provided exclusively for one module shall be separated into a dedicated header file
[SRS_BSW_00416]	The sequence of modules to be initialized shall be configurable
[SRS_BSW_00417]	Software which is not part of the SW-C shall report error events only after the Dem is fully operational.
[SRS_BSW_00419]	If a pre-compile time configuration parameter is implemented as `const` it should be placed into a separate c-file
[SRS_BSW_00422]	Pre-de-bouncing of error status information is done within the Dem
[SRS_BSW_00423]	BSW modules with AUTOSAR interfaces shall be describable with the means of the SW-C Template
[SRS_BSW_00424]	BSW module main processing functions shall not be allowed to enter a wait state
[SRS_BSW_00425]	The BSW module description template shall provide means to model the defined trigger conditions of schedulable objects
[SRS_BSW_00426]	BSW Modules shall ensure data consistency of data which is shared between BSW modules
[SRS_BSW_00427]	ISR functions shall be defined and documented in the BSW module description template
[SRS_BSW_00428]	A BSW module shall state if its main processing function(s) has to be executed in a specific order or sequence
[SRS_BSW_00429]	Access to OS is restricted
[SRS_BSW_00432]	Modules should have separate main processing functions for read/receive and write/transmit data path
[SRS_BSW_00433]	Main processing functions are only allowed to be called from task bodies provided by the BSW Scheduler
[SRS_BSW_00437]	Memory mapping shall provide the possibility to define RAM segments which are not to be initialized during startup
[SRS_BSW_00438]	Configuration data shall be defined in a structure
[SRS_BSW_00439]	Enable BSW modules to handle interrupts
[SRS_BSW_00440]	The callback function invocation by the BSW module shall follow the signature provided by RTE to invoke servers via `Rte_Call` API
[SRS_BSW_00441]	Naming convention for type, macro and function
[SRS_BSW_00447]	Standardizing Include file structure of BSW Modules Implementing Autosar Service
[SRS_BSW_00448]	Module SWS shall not contain requirements from other modules
[SRS_BSW_00449]	BSW Service APIs used by Autosar Application Software shall return a Std_ReturnType
[SRS_BSW_00450]	A Main function of a un-initialized module shall return immediately

▽

△

Number	Heading
[SRS_BSW_00451]	Hardware registers shall be protected if concurrent access to these registers occur
[SRS_BSW_00452]	Classification of runtime errors
[SRS_BSW_00453]	BSW Modules shall be harmonized
[SRS_BSW_00454]	An alternative interface without a parameter of category DATA_ REFERENCE shall be available.
[SRS_BSW_00456]	A Header file shall be defined in order to harmonize BSW Modules
[SRS_BSW_00457]	Callback functions of Application software components shall be invoked by the Basis SW
[SRS_BSW_00458]	Classification of production errors
[SRS_BSW_00459]	It shall be possible to concurrently execute a service offered by a BSW module in different partitions
[SRS_BSW_00460]	Reentrancy Levels
[SRS_BSW_00461]	Modules called by generic modules shall satisfy all interfaces requested by the generic module
[SRS_BSW_00462]	All Standardized Autosar Interfaces shall have unique requirement Id / number
[SRS_BSW_00463]	Naming convention of callout prototypes
[SRS_BSW_00464]	File names shall be considered case sensitive regardless of the filesystem in which they are used
[SRS_BSW_00465]	It shall not be allowed to name any two files so that they only differ by the cases of their letters
[SRS_BSW_00466]	Classification of extended production errors
[SRS_BSW_00467]	The init / deinit services shall only be called by BswM or EcuM
[SRS_BSW_00469]	Fault detection and healing of production errors and extended production errors
[SRS_BSW_00470]	Execution frequency of production error detection
[SRS_BSW_00471]	Do not cause dead-locks on detection of production errors - the ability to heal from previously detected production errors
[SRS_BSW_00472]	Avoid detection of two production errors with the same root cause.
[SRS_BSW_00473]	Classification of transient faults
[SRS_BSW_00477]	The functional interfaces of AUTOSAR BSW modules shall be specified in C99
[SRS_BSW_00478]	Timing limits of main functions
[SRS_BSW_00479]	Interfaces for handling request from external devices
[SRS_BSW_00480]	Null pointer errors shall follow a naming rule
[SRS_BSW_00481]	Invalid configuration set selection errors shall follow a naming rule
[SRS_BSW_00482]	Get version information function shall follow a naming rule
[SRS_BSW_00483]	BSW Modules shall handle buffer alignments internally
[SRS_BSW_00484]	Input parameters of scalar and enum types shall be passed as a value.
[SRS_BSW_00485]	Input parameters of structure type shall be passed as a reference to a constant structure

▽

△

Number	Heading
[SRS_BSW_00486]	Input parameters of array type shall be passed as a reference to the constant array base type
[SRS_BSW_00487]	Errors for module initialization shall follow a naming rule
[SRS_BSW_00488]	Classification of security events
[SRS_BSW_00489]	Reporting of security events
[SRS_BSW_00490]	List possible security events
[SRS_BSW_00491]	Specification of trigger conditions and context data
[SRS_BSW_00492]	Reporting of security events during startup
[SRS_BSW_00493]	Definition of security event ID symbols
[SRS_BSW_00494]	ServiceInterface argument with a pointer datatype
[SRS_BSW_00495]	If tracing is enabled, all AUTOSAR Basic Software Modules should allow tracing its entry and exit points.

Table A.10: Added Requirements in R22-11

A.5.2 Changed Requirements in R22-11

none

A.5.3 Deleted Requirements in R22-11

none

B Tracegroups

Defined Trace Groups	
Identifier	**Included Requirements**
TG_BSW	[SRS_BSW_00004] [SRS_BSW_00101] [SRS_BSW_00159] [SRS_BSW_00167] [SRS_BSW_00168] [SRS_BSW_00170] [SRS_BSW_00171] [SRS_BSW_00323] [SRS_BSW_00336] [SRS_BSW_00337] [SRS_BSW_00339] [SRS_BSW_00344] [SRS_BSW_00345] [SRS_BSW_00369] [SRS_BSW_00375] [SRS_BSW_00380] [SRS_BSW_00383] [SRS_BSW_00384] [SRS_BSW_00385] [SRS_BSW_00386] [SRS_BSW_00388] [SRS_BSW_00389] [SRS_BSW_00390] [SRS_BSW_00392] [SRS_BSW_00393] [SRS_BSW_00395] [SRS_BSW_00396] [SRS_BSW_00397] [SRS_BSW_00398] [SRS_BSW_00399] [SRS_BSW_00400] [SRS_BSW_00402] [SRS_BSW_00403] [SRS_BSW_00404] [SRS_BSW_00405] [SRS_BSW_00406] [SRS_BSW_00409] [SRS_BSW_00416] [SRS_BSW_00417] [SRS_BSW_00419] [SRS_BSW_00422] [SRS_BSW_00423] [SRS_BSW_00424] [SRS_BSW_00425] [SRS_BSW_00426] [SRS_BSW_00427] [SRS_BSW_00428] [SRS_BSW_00429] [SRS_BSW_00432] [SRS_BSW_00433] [SRS_BSW_00437] [SRS_BSW_00438] [SRS_BSW_00450] [SRS_BSW_00451] [SRS_BSW_00452] [SRS_BSW_00458] [SRS_BSW_00461] [SRS_BSW_00467] [SRS_BSW_00469] [SRS_BSW_00470] [SRS_BSW_00471] [SRS_BSW_00472] [SRS_BSW_00478] [SRS_BSW_00488] [SRS_BSW_00489] [SRS_BSW_00490] [SRS_BSW_00491] [SRS_BSW_00493] [SRS_BSW_00496] [SRS_BSW_00497]
TG_BSW_NonFunctional	[SRS_BSW_00003] [SRS_BSW_00005] [SRS_BSW_00006] [SRS_BSW_00007] [SRS_BSW_00009] [SRS_BSW_00010] [SRS_BSW_00160] [SRS_BSW_00161] [SRS_BSW_00162] [SRS_BSW_00164] [SRS_BSW_00172] [SRS_BSW_00300] [SRS_BSW_00301] [SRS_BSW_00302] [SRS_BSW_00304] [SRS_BSW_00305] [SRS_BSW_00306] [SRS_BSW_00307] [SRS_BSW_00308] [SRS_BSW_00309] [SRS_BSW_00310] [SRS_BSW_00312] [SRS_BSW_00314] [SRS_BSW_00318] [SRS_BSW_00321] [SRS_BSW_00325] [SRS_BSW_00327] [SRS_BSW_00328] [SRS_BSW_00330] [SRS_BSW_00331] [SRS_BSW_00333] [SRS_BSW_00334] [SRS_BSW_00335] [SRS_BSW_00341] [SRS_BSW_00342] [SRS_BSW_00343] [SRS_BSW_00346] [SRS_BSW_00347] [SRS_BSW_00348] [SRS_BSW_00350] [SRS_BSW_00351] [SRS_BSW_00353] [SRS_BSW_00357] [SRS_BSW_00358] [SRS_BSW_00359] [SRS_BSW_00360] [SRS_BSW_00373] [SRS_BSW_00374] [SRS_BSW_00377] [SRS_BSW_00378] [SRS_BSW_00379] [SRS_BSW_00401] [SRS_BSW_00408] [SRS_BSW_00410] [SRS_BSW_00411] [SRS_BSW_00413] [SRS_BSW_00414] [SRS_BSW_00415] [SRS_BSW_00439] [SRS_BSW_00440] [SRS_BSW_00441] [SRS_BSW_00447] [SRS_BSW_00448] [SRS_BSW_00449] [SRS_BSW_00453] [SRS_BSW_00454] [SRS_BSW_00456] [SRS_BSW_00457] [SRS_BSW_00459] [SRS_BSW_00460] [SRS_BSW_00462] [SRS_BSW_00463] [SRS_BSW_00464] [SRS_BSW_00465] [SRS_BSW_00477] [SRS_BSW_00479] [SRS_BSW_00480] [SRS_BSW_00481] [SRS_BSW_00483] [SRS_BSW_00484] [SRS_BSW_00485] [SRS_BSW_00486] [SRS_BSW_00487] [SRS_BSW_00494] [SRS_BSW_00498]

Table B.1: Trace Groups of this document