Document Title	General Requirements on Basic Software Modules
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	043
Document Classification	Auxiliary
Document Version	4.2.0
Document Status	Final
Part of Release	4.1
Revision	3

Document Change History			
Date	**Version**	**Changed by**	**Change Description**
31.03.2014	4.2.0	AUTOSAR Release Management	• Erased/modified requirements about standard header files providing a more abstract view • Improved definition of run-time errors • Editorial changes
31.10.2013	4.1.0	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
27.02.2013	4.0.0	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

Document Change History			
Date	**Version**	**Changed by**	**Change Description**

18.11.2011	3.2.0	AUTOSAR Administration	• Improvement of safety and integrity:
			  ○ Limitation on callers for Init and definite functions
			  ○ Re-entrant handling
			  ○ New implementation requirements for the interrupt routines in the BSW modules
			• Adaptation to the Include structure of the BSW modules. (e.g. RTE headers handling)
			• The format of VENDOR_ID adapted to ease the verification

22.10.2010	3.1.0	AUTOSAR Administration	• Changed Requirement [SRS_BSW_00416] (sequence of initialisation): added check of uninitialized module calls. • Changed Requirement [SRS_BSW_00004] (version check): reworded to specify pass criteria of checks. • Changed Requirement [SRS_BSW_00346] (Basic set of module files): added Link-time and Post-Build configuration header files. • Changed Requirement [SRS_BSW_00408] (Configuration parameter naming convention): requirement relaxed. • Changed Requirement [SRS_BSW_00440] (Function Prototype for Callback functions of AUTOSAR): modified callback call mechanism through RTE. • Changed Requirement [SRS_BSW_00414] (Parameter if init function): added check on coherence of configuration type (pre-compile, link time, post-build) and pointer passed to API. • Added Requirement [SRS_BSW_00462] (Requirement Id for Standardized Autosar Interface): AUTOSAR Standard Interfaces description has now a Requirement ID and is binding.

Document Change History

Date	Version	Changed by	Change Description
02.12.2009	3.0.0	AUTOSAR Administration	• Added New Requirements: [BSW00443], [BSW00444], [BSW00445], [BSW00446], [SRS_BSW_00442], [SRS_BSW_00448], [SRS_BSW_00447], [SRS_BSW_00450], [SRS_BSW_00453], [SRS_BSW_00455], [SRS_BSW_00456], [SRS_BSW_00457, [SRS_BSW_00449] • Removed Requirements : ○ [BSW00434] The Schedule Module provides an API for exclusive areas. ○ [BSW00431] The BSW Scheduler module implements task bodies. These requirements are available in SRS RTE SRS_Rte_00222, RTE00225 respectively. • Changed requirements: [SRS_BSW_00416], [SRS_BSW_00407], SRS_BSW_00379], [SRS_BSW_00435], [SRS_BSW_00305], [SRS_BSW_00429], [SRS_BSW_00318], [SRS_BSW_00004], [SRS_BSW_00402], [SRS_BSW_00373], [SRS_BSW_00406], [SRS_BSW_00414], [SRS_BSW_00347], [SRS_BSW_00343], [SRS_BSW_00003], [SRS_BSW_00347] • Legal disclaimer revised
23.06.2008	2.2.1	AUTOSAR Administration	• Legal disclaimer revised
10.12.2007	2.2.0	AUTOSAR Administration	• [SRS_BSW_00439] Declaration of interrupt handlers and ISRs • [SRS_BSW_00440] Function prototype for callback functions of AUTOSAR Services • [SRS_BSW_00441] Enumeration literals and define naming convention • Changes done for Interrupt Handling, Configuration Parameter Naming Convention and AUTOSAR Services • Document meta information extended • Small layout adaptations made

Document Change History			
Date	**Version**	**Changed by**	**Change Description**
26.01.2007	2.1.0	AUTOSAR Administration	• Interface for BSW Modules to DEM and Debouncing for DEM • Changes in Configuration Requirements • Module Headerfile Structure • Naming separation of different instances of BSW drivers • Legal disclaimer revised • “Advice for users” revised • “Revision Information” added
23.05.2006	2.0.0	AUTOSAR Administration	Second release
23.06.2005	1.0.0	AUTOSAR Administration	Initial release

Disclaimer

This specification 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 specification.

The material contained in this specification is protected by copyright and other types of Intellectual Property Rights. The commercial exploitation of the material contained in this specification requires a license to such Intellectual Property Rights.

This specification 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 specification may be utilized or reproduced, in any form or by any means, without permission in writing from the publisher.

The AUTOSAR specifications have been developed for automotive applications only. They have neither been developed, nor tested for non-automotive applications.

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

Advice for users

AUTOSAR specifications may contain exemplary items (exemplary reference models, "use cases", and/or references to exemplary technical solutions, devices, processes or software).

Any such exemplary items are contained in the specifications for illustration purposes only, and they themselves are not part of the AUTOSAR Standard. Neither their presence in such specifications, nor any later documentation of AUTOSAR conformance of products actually implementing such exemplary items, imply that intellectual property rights covering such exemplary items are licensed under the same rules as applicable to the AUTOSAR Standard.

Table of Contents

1 Scope of this document

The goal of AUTOSAR WP Architecture and 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 by the work packages responsible 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 How to read this document

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!

2.1 Conventions used

- The representation of requirements in AUTOSAR documents follows the table specified in [TPS_STDT_00078].

- 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.

2.2 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 WPs (e.g. Description Templates, Tooling,...)
- ...

2.3 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 Acronym and abbrevations

Acronym:	*Description:*
Interrupt frame	An interrupt frame is the code which is generated by the compiler or the assembler code for prefix and postfix of interrupt routines. This code is Microcontroller specific
ISR	Interrupt Service Routine. Also used as a macro to declare in C a cat2 interrupt service routine.

Abbreviation:	*Description:*
Cat2	Category 2. Cat2 ISRs are supported by the OS and can make OS calls.
Cat1	Category 1. Cat1 interrupts are not supported by the OS and are only allowed to make a very small selection of OS calls to enable and disable all interrupts.

4 General Requirements on Basic Software

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 OSEK OS, communication modules, bootloader, basic diagnostic functions for the domains listed above
- Infotainment control ECU

4.1 Functional Requirements

4.1.1 Configuration

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

[

Type:	Valid
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:	--

]()

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

[

Type:	Valid
Description:	Post-build configuration phase shall be supported. Post-build parameters are optional
Rationale:	As long as there is only one set of configuration data (i.e. we have no

	multiple configuration sets) the references can be resolved as constant pointers. The indirections shall be kept as simple as possible

Use Case:	type declaration of the Config Type
	`typedef struct ComM_ConfigType_Tag {`
	`...`
	`} ComM_ConfigType;` (in ComM_Cfg.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:	--

](

4.1.1.3 [SRS_BSW_00405] BSW Modules shall support multiple configuration sets

[

Type:	Valid
Description:	Modules of the AUTOSAR Basic Software that operate on more than one post build time configurable data entity shall use a reference (pointer) to an external configuration instance.
Rationale:	Application of the same software to different cars.
Use Case:	--
Dependencies:	[SRS_BSW_00342] Usage of source code and object code
Supporting Material:	--

](

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

[

Type:	Valid
Description:	
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.
Use Case:	
Dependencies:	[SRS_BSW_00158] Separation of configuration from implementation
Supporting Material:	--

](

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

[

Type:	Valid
Description:	All modules of the AUTOSAR Basic Software shall support a tool based configuration.

Rationale:	Integration into AUTOSAR methodology
Use Case:	The NVRAM manager can be automatically configured depending on the NV parameters and their corresponding attributes of the software components.
Dependencies:	--
Supporting Material:	--

]()

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

[

Type:	Valid
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:	--

]()

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

[

Type:	Valid
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:	--

]()

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

[

Type:	Valid
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:	--

](RS_BRF_01480)

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

[

Type:	Valid
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_00381] Separate configuration header file for pre--compile time parameters [SRS_BSW_00346] Basic set of module files
Supporting Material:	Layered Software Architecture ([DOC_LAYERED_ARCH])

]()

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

[

Type:	Valid
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:	[SRS_BSW_00380] Separate C--Files for configuration parameters
Supporting Material:	Layered Software Architecture ([DOC_LAYERED_ARCH])

](

4.1.1.11 [SRS_BSW_00381] The pre--compile time parameters shall be placed into a separate configuration header file

[

Type:	Valid
Description:	The pre--compile time parameters shall be placed into a separate configuration header file.
Rationale:	Keep the configuration data separate.
Use Case:	--
Dependencies:	[SRS_BSW_00345] Pre--compile--time configuration
Supporting Material:	--

](

4.1.1.12 [SRS_BSW_00412] References to c--configuration parameters shall be placed into a separate h--file

[

Type:	Valid
Description:	References to c--configuration parameters (link time and post--build time) shall be placed into a separate h--file. The h--file shall be the same as pre--compile time parameters.
Rationale:	Put the references to c--configuration parameters in the same header file as pre--compile time parameters to enable access to the configuration data.
Use Case:	--
Dependencies:	[SRS_BSW_00345], [SRS_BSW_00346]
Supporting Material:	--

](RS_BRF_01056)

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

[

Type:	Valid
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:	--

](

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

[

Type:	Valid

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:	--

](RS_BRF_01064)

4.1.1.15 [SRS_BSW_00387] The Basic Software Module specifications shall specify how the callback function is to be implemented

[

Type:	Valid
Description:	The Basic Software Module specifications shall specify how the callback function is to be implemented. (Pre--compile macro, pointer at link time, array of pointers at post--build time and pointer at post--build time)
Rationale:	
Use Case:	If a pre--compile time callback function (macro) shall be valid to a post build time multiple configuration--set callback function (pointer to a function). The implementation will change significantly.
Dependencies:	--
Supporting Material:	See Glossary ([GLOSSARY]) and ECU Configuration (WP ECU Configuration) ([ECU_CONF_SRS])

](RS_BRF_01064)

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

[

Type:	Valid
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 NVRAM manager 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
Supporting Material:	

]()

4.1.1.17 [SRS_BSW_00389] Containers shall have names

[

Type:	Valid

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:	See Glossary ([GLOSSARY])

](RS_BRF_01024)

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

[

Type:	Valid
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:	--

](RS_BRF_01024)

4.1.1.19 [SRS_BSW_00392] Parameters shall have a type

[

Type:	Valid
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:	--

](RS_BRF_01024)

4.1.1.20 [SRS_BSW_00393] Parameters shall have a range

[

Type:	Valid
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:	--

](RS_BRF_01384)

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

[

Type:	Valid
Description:	A parameter may only be applicable for the module it is defined in. In this case, the parameter is marked as "local". Alternatively, the parameter may be shared with other modules (i.e. exported).
Rationale:	Increase the uniformity of the use of this attribute and let as single entity (BSW UML model) be the source for import information.
Use Case:	Importing and exporting could be achieved in different ways: external reference, redefinition in the other module.
Dependencies:	--
Supporting Material:	

]()

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

[

Type:	Valid
Description:	The Basic Software Module specifications must list configuration parameters of this or other modules this parameter relies on. A dependency is for example: the value of another parameter influences or invalidates the setting of this 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:	--

](RS_BRF_01136)

4.1.1.23 [SRS_BSW_00396] The Basic Software Module specifications shall specify one classe (of the three) to be supported

[

Type:	Valid
Description:	There are three main configuration classes. The Basic Software Module specifications must specify the classes to be supported (per parameter). The classes are: -- pre-- compile time configuration -- link time configuration -- post build time configuration (could be either loadable or multiple)
Rationale:	Enable optimizing towards different goals of configuration.
Use Case:	--
Dependencies:	--
Supporting Material:	--

](RS_BRF_02200)

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

[

Type:	Valid
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

]()

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

[

Type:	Valid
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

]()

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

[

Type:	Valid
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:	--

]()

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

[

Type:	Valid
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 (see definition of post--build time configuration in the AUTOSAR glossary).
Rationale:	--
Use Case:	Reuse of ECUs.
Dependencies:	--
Supporting Material:	--

]()

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

[

Type:	Valid
Description:	Configuration data shall be defined in a structure. This structure shall be pointed to by configuration pointers. Only EcuM contains pointers to the data structures containing the post-build. If there is at least one module with the configuration class “post build selectable” then the EcuM shall determine which pointer to the configuration parameters is required to be passed to the init functions. If there are no modules in the configuration class “post build selectable” but one or more modules are in the “post build” class then a fixed pointer shall be passed to the init functions by EcuM.
Rationale:	Allow configurable functionality of modules that are deployed as object code. Usually those modules are drivers.
Use Case:	Initialization concept for ComM or CanIf.
Dependencies:	--
Supporting Material:	--

]()

4.1.1.29 [SRS_BSW_00402] Each module shall provide version information

[

Type:	Valid
Description:	The provided informationshall 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_00407], [SRS_BSW_00318]
Supporting Material:	--

](RS_BRF_01032)

4.1.2 Wake--Up

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

⌈

Type:	Valid
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:	--

⌋(RS_BRF_01104,RS_BRF_01440)

4.1.3 Initialization

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

⌈

Type:	Valid
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 ECU state manager
Use Case:	--
Dependencies:	[SRS_BSW_00358] ,[SRS_BSW_00414] , [SRS_BSW_00406]
Supporting Material:	--

⌋()

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

⌈

Type:	Valid
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. Each SWS shall specify that all calls of a non initialized module which are in un-initialized state must raise a DET error. This would lead to the detection of such issues during development
Use Case:	Start-up sequence is a proprietary functionality. DET dependency shall allow error detection during development.
Dependencies:	[SRS_BSW_00406]
Supporting Material:	--

](RS_BRF_01208)

4.1.3.3 [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

[

Type:	Valid
Description:	If the Development Error Tracer (DET) Error is enabled, module APIs shall check if the module is initialized i.e. the static initialization status variable of the module is not zero. If the Module is not initialized and Development Error Tracer (DET) is enabled, then a) The Module's API shall report error to DET. b) The Module's API function shall return an error status when it has a return type or return without further processing when it has no return type. Module Initialization and initialization check shall not be performed for i) Init Functions , Reason :- The Initialization of the static variable is done in the Init Functions, hence no checks required ii) Version Check API , Reason :- It is possible to call Version Check API, without Initializing the module. iii) Libraries , Reason .- They are generally stateless and may not have initialization dependencies. **Please Note** :- For optimization reasons, if Development Error Detection is switched off, the static variable and the check are optional.
Rationale:	When development error detection is enabled, functions should report 'Module not initialized' to the Development Error Tracer (DET) if the module is not initialized. Without initializing the static status variable in Module initialization, 'Module not initialized' check cannot be performed.
Use Case:	The call "Can_Write()" to the Can driver causes a call Det_ReportError (ModuleId, ApiId, ErrorId); in case the Can driver is not initialized. In this case the return value of the "Can_Write()" function will be "E_NOT_OK".
Dependencies:	[SRS_BSW_00407], [SRS_BSW_00369],
Supporting Material:	--

](RS_BRF_01136)

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

[

Type:	Valid
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.
Use Case:	
Dependencies:	[SRS_BSW_00101]
Supporting Material:	--

](

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

[

Type:	valid
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:	--

](RS_BRF_00057)

4.1.4 Normal Operation

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

[

Type:	Valid
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:	--

](RS_BRF_02144)

4.1.4.2 [SRS_BSW_00407] Each BSW module shall provide a function to read out the version information of a dedicated module implementation

[

Type:	Valid
Description:	Each BSW module shall provide a function to read out the version information of a dedicated module implementation. This API shall be pre--compile time configurable (see SRS_BSW_00411). It shall be possible to call this function at any time (e.g. before the init function is called).
Rationale:	If problems are detected within an ECU during lifetime this enables the garage to check the version of the modules. The AUTOSAR specification version number is checked during compile time (see requirement SRS_BSW_00004) and therefore not required in this API.
Use Case:	With this API the garage can read out version information which is implemented in a dedicated (erroneous) ECU to enable the decision whether a software update might be sufficient, or not.
Dependencies:	[SRS_BSW_00318] ,[SRS_BSW_00374] ,[SRS_BSW_00411], [SRS_BSW_00406]
Supporting Material:	--

](RS_BRF_01352)

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

[

Type:	Valid
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:	--

]()

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

[

Type:	Valid
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:	--

]()

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

[

Type:	Valid
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:	--

](RS_BRF_01464)

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

[

Type:	Valid
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:	--

]()

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

[

Type:	Valid
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:	--

]()

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

[

Type:	Valid
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:	--

]()

4.1.4.9 [SRS_BSW_00429] BSW modules shall be only allowed to use OS objects and/or related OS services

[

Type: Valid

Description: BSW modules are only allowed to use OS objects and/or related OS services according to the following table:

Objects / Service	**RTE / BSW Scheduler / CDD / DBG**	**EcuM**	**MCAL**	**StbM**	**Other BSW Modules**
OS Objects					
OS Object “Task”	✓				
OS Object “ISR”	✓		✓		
OS Object “Alarm”	✓				
OS Object “Counters”	✓				
OS Object “Schedule tables”	✓				
OS Object “Resource”	✓				
OS Object “Message”					
OS Services					
Activate Task	✓				
Terminate Task	✓				
Chain Task	✓				
Schedule	✓				
GetTaskID	✓				
GetTaskState	✓				
DisableAllInterrupts	✓	✓			
EnableAllInterrupts	✓	✓			
SuspendAllInterrupts	✓		✓		
ResumeAllInterrupts	✓		✓		
SuspendOSInterrupts	✓		✓		
ResumeOSInterrupts	✓		✓		

GetResource	✓				
ReleaseResource	✓				
SetEvent	✓				
ClearEvent	✓				
GetEvent	✓				
WaitEvent	✓				
GetAlarmBase	✓				
GetAlarm	✓				
SetRelAlarm	✓				
SetAbsAlarm	✓				
CancelAlarm	✓				
GetActiveApplicationMode	✓	✓			
StartOS		✓			
ShutdownOS		✓			
GetApllicationID	✓				
StartScheduleTable	✓	✓			
StopScheduleTable	✓	✓			
NextScheduleTable	✓	✓			
SyncScheduleTable	✓	✓		✓	
GetScheduleTableStatus	✓	✓		✓	
SetScheduleTableAsync	✓	✓			
IncrementCounter	✓				
GetCounterValue	✓	✓	✓	✓	✓
GetElapsedCounterValue	✓	✓	✓	✓	✓
TerminateApplication	✓				

CDD : Complex Driver
StbM : Synchronized Time-base Manager

Rationale:	Simplification of the OS integration of BSW modules.
Use Case:	Integration of different BSW modules in one ECU.
Dependencies:	--
Supporting Material:	--

]()

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

[

Type:	Valid
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:	--

](RS_BRF_01352)

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

[

Type:	Valid
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:	--

]()

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

[

Type:	Valid
Description:	If a Main function of a un-initialized module is called from the BSW Scheduler, 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:	--

]()

4.1.4.13 [SRS_BSW_00442] The AUTOSAR architecture shall support standardized debugging and tracing features

[

Type:	Valid
Description:	The AUTOSAR architecture shall support standardized debugging and tracing features for basic software, RTE and software components. The debugging feature shall be optional. When the debugging is supported, the following condition should be taken care of 1. Each variable that shall be accessible for debugging, shall be defined as global Variable. 2. All type definitions of variables which shall be debugged, shall be accessible by the standard module header file "Modulename".h. 3. Debug variables shall be declared in a separate debug header file. 4. The declaration of debug variables in the header file shall be such, that it is possible to calculate the size of the variables by C-"size of" operation. 5. Each variable which is available for debugging shall be described in

	respective Basic Software Module Description. 6. If the individual module SWS specifies any module information (e.g. state machine related data, counters), this information shall be available for debugging.
Rationale:	To fit the Debugging Concept in the Autosar Architecture
Use Case:	--
Dependencies:	--
Supporting Material:	--

]()

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

[

Type:	Valid
Description:	If a generic module (e.g. PDU Router) 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, COM Manager etc. need no adaptation to specific modules and CDDs
Dependencies:	--
Supporting Material:	--

](RS_BRF_01016)

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

[

Type:	Valid
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:	--

]()

4.1.5 Shutdown Operation

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

[

Type:	Valid
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 ECU state manager
Use Case:	--
Dependencies:	--
Supporting Material:	--

](RS_BRF_01096)

4.1.6 Fault Operation and Error Detection

4.1.6.1 [SRS_BSW_00337] Classification of development errors

[

Type:	**Changed (by the TF Production 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: the normal control flow of execution will be aborted by halting or reseting of the entire ECU. Monitors of development errors shall not stay in the production code (deployment build). For switching the configuration the Standard Types STD_ON and STD_OFF shall be used.
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
Supporting Material:	

](RS_BRF_02168)

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

[

Type:	Valid
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. If the API-- function which detected the error has a return type it shall return a value which indicates an error.

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:	--

](RS_BRF_02168)

4.1.6.3 [SRS_BSW_00339] Reporting of production relevant error status

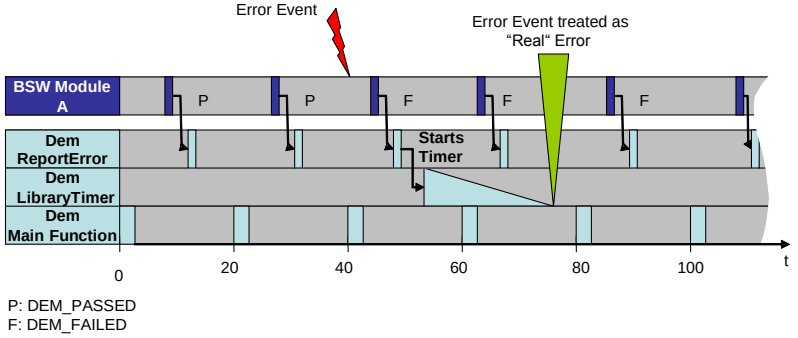
[

Type:	**Changed (by the TF Production Errors)**
Description:	AUTOSAR Basic Software Modules shall report all production errors and extended 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_BUSOFF) are reported to the DEM.
Dependencies:	[RS_BSWMD_00069] Configuration for production errors and extended production errors [SRS_Diag_04063] Single Event ID for each monitoring path
Supporting Material:	--

](RS_BRF_02184,RS_BRF_02168)

4.1.6.4 [SRS_BSW_00422] Pre-de-bouncing of error status information is done within the DEM

[

Type:	Valid
Description:	Pre--de--bouncing of error status information reported via `Dem_ReportErrorStatus` 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:	This is 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.
Dependencies:	[SRS_BSW_00339] Reporting of production relevant error status
Supporting Material:	--

⌋()

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

⌈

Type:	Valid
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:	--

⌋(RS_BRF_02184,RS_BRF_02168)

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

⌈

Type:	Valid
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:	--

⌋(RS_BRF_01384)

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

⌈

Type:	Valid
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 [ARReleaseManagement].

](

4.1.6.8 [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

[

Type:	Valid
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:	--

](

4.1.6.9 [SRS_BSW_00385] List possible error notifications

[

Type:	**Changed (by the TF Production Errors)**
Description:	The BSW shall document all production errors, extended production errors, development errors and runtime errors which are supported by the BSW module.
Rationale:	Documentation, overview of errors
Use Case:	--
Dependencies:	
Supporting Material:	--

](RS_BRF_02184,RS_BRF_02168)

4.1.6.10 [SRS_BSW_00386] The BSW shall specify the configuration for detecting an error

[

Type:	Valid
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 code errors as well as for development errors.
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:	--

](RS_BRF_02184,RS_BRF_02168,RS_BRF_02176)

4.1.6.11 [SRS_BSW_00452] Classification of runtime errors

[

Type:	New
Description:	AUTOSAR Basic Software Modules may report runtime errors. Runtime errors are not implementation errors; they will not cause assertions and therefore even not cause the abortion of the 'normal' control flow of execution (as DET will do). Runtime errors are not production errors; they will not cause diagnostic trouble codes (at least not automatically) and therefore not be handled by DEM. But runtime errors are transient hardware fault (e.g. register flip due to particle effects) or systematic faults, that does not influence the overall software (e.g. wrong PDU-Ids, wrong post-build configurations)).Monitors of runtime errors may stay in the code for the production code (deployment build). Runtime errors shall only be reported as an event in case of the occurence (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 may only store the corresponding events to a memory, may call DEM and may return handling instructions or execute any reasonable action. An error handle of runtime errors exists in both development and production code.
Rationale:	Catch sporadic error events caused by transient hardware faults or seldom occurring systematic faults.
Use Case:	• CAN controller goes offline due to bit-flip in its control register. • Ethernet controller changed state of operation sporadically. • Improper error handling has been defined in a software state machine (e.g. because of not and considered race conditions).
Dependencies:	--
Supporting Material:	

](RS_BRF_02184,RS_BRF_02168)

4.1.6.12 [SRS_BSW_00458] Classification of production errors

[

Type:	New
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 or an extended production error may be selectable by configuration. Dependent of this classification the particular error may cause different reactions within the Dem.
Supporting Material:	--

](RS_BRF_02184,RS_BRF_02168,RS_BRF_02176)

4.1.6.13 [SRS_BSW_00466] Classification of extended production errors

[

Type:	New
Description:	AUTOSAR Basic Software Modules may report extended production errors (to the module Dem) if this error is caused • by any hardware problem of the ECU itself, e.g., a memory transactions failed, • by a misbehavior of the embedding environment, e.g., the loss of messages due to any problem of the communication channel AND • this error does not comply to any criteria of the production error definition, notably ○ OBD relevance ○ direct limp-home reactions ○ direct repair actions in the garage • the error cause is already covered by any other production error Extended production errors shall define set and reset conditions. Note: Extended 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 'Extended production errors'. Note: Extended production errors may not be entered in the primary event memory of the module Dem.
Rationale:	Extended production errors may be used • to deduce 'real' production errors by tying several values influencing the state of the ECU together • to gain more detailed information of the real cause of a production error
Use Case:	--
Dependencies:	If not specified by AUTOSAR, the real classification of a particular error being a production error or an extended production error may be selectable by configuration. Dependent of this classification the particular error may cause different reactions within the Dem.

Supporting Material:	--

](RS_BRF_02184,RS_BRF_02168,RS_BRF_02176)

4.1.6.14 [SRS_BSW_00469] Fault detection and healing of production errors and extended production errors

[

Type:	New
Description:	The detection of production errors and extended 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:	--

](RS_BRF_02184,RS_BRF_02168,RS_BRF_02176)

4.1.6.15 [SRS_BSW_00470] Execution frequency of production error detection

[

Type:	New
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:	CCR1968-2

](RS_BRF_02096,RS_BRF_02224)

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

[

Type:	New
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:	--

]()

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

[

Type:	New
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:	--

](RS_BRF_00129,RS_BRF_02248)

4.2 Non--functional Requirements

4.2.1 Software Architecture Requirements

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

[

Type:	Valid
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:	[DOC_LAYERED_ARCH]

](RS_BRF_01008,RS_BRF_01016)

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

[

Type:	Valid
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 NVRAM manager 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:	[DOC_LAYERED_ARCH]

](RS_BRF_01016,RS_BRF_01856,RS_BRF_01864,RS_BRF_01872,RS_BRF_01880,RS_BRF_01888,RS_BRF_01896,RS_BRF_01904,RS_BRF_01912,RS_BRF_01920,RS_BRF_01928,RS_BRF_01936)

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

[

Type:	Valid
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:	--

]()

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

[

Type:	Valid
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 WP Architecture “List of Basic Software Modules”, [DOC_MOD_LIST] (2…8 characters). <User> shall be the user module from the same list.

]()

4.2.2 Software Integration Requirements

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

[

Type:	Valid
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:	--

](RS_BRF_02056)

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

[

Type:	Valid
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:	--

]()

4.2.2.3 [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

[

Type:	Valid
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:	--

]()

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

[

Type:	Valid
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:	--

]()

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

[

Type:	Valid
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:	--

]()

4.2.2.6 [SRS_BSW_00453] BSW Modules shall be harmonized

[

Type:	Valid
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:	--

](RS_BRF_01016)

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

[

Type:	Valid
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:	--

](RS_BRF_01016)

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

[

Type:	Valid
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:	--

](RS_BRF_01064)

4.2.3 Software Module Design Requirements

4.2.3.1 Software quality

4.2.3.1.1 [SRS_BSW_00007] All Basic SW Modules written in C language shall conform to the MISRA C 2004 Standard.

[

Type:	Valid
Description:	MISRA C 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).
Rationale:	Portability, maintainability, error avoidance, safety
Use Case:	Software for safety relevant systems
Dependencies:	--
Supporting Material:	--

](RS_BRF_01056)

4.2.3.2 Naming conventions

4.2.3.2.1 [SRS_BSW_00300] All AUTOSAR Basic Software Modules shall be identified by an unambiguous name

[

Type:	Valid
Description:	All AUTOSAR Basic Software Modules shall be identified by an unambiguous name. The module name is always part of related files. Convention for module related files: - <Module name>_*.* - Spelling of module name: First letter of each word upper case, consecutive letters lower case - Module name: 2..8 letters, derived from WP Architecture SW Module List - Wildcard replacement according to module related file set (either basic and recommended)
Rationale:	The module name serves as an identifier and classification mechanism in order to group module related files.
Use Case:	Example: Eep.c, Eep.h, Eep_Cfg.h
Dependencies:	--
Supporting Material:	WP Architecture SW Module List (Module Abbreviations)

](RS_BRF_01024)

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

[

Type:	Valid
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:	--

](

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

[

Type:	Valid
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: o File names o API names o Published parameters o Memory allocation keyword ▪ For API names, <Vendor specific name> should be followed by "_" and then <API Service Name>. ▪ For the creation of file names, no trailing underscore shall be added. ▪ For Published parameters and Memory allocation keyword names, <Vendor Specific name> shall have a trailing underscores.
Rationale:	Avoidance of name clashes
Use Case:	Examples: • EEPROM (LD): `Eep_21_LDExtEepDriver.c` • Published parameters: `EEP_21_LDEXT_SW_MAJOR_VERSION` • API: Eep_21_LDExt_Init()
Dependencies:	--
Supporting Material:	[DOC_MOD_LIST] List of Basic Software Modules (Module Abbreviations)

](

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

[

Type:	Valid
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 # defines 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,`
	`EEP_ARE,`
	`EEP_EV`
	`} Eep_NotificationType;`

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

](RS_BRF_01024)

4.2.3.2.5 [SRS_BSW_00305] Data types naming convention

[

Type:	Valid
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.
Use Case:	•
Dependencies:	--
Supporting Material:	BMW Standard Core Programming Guidelines

](RS_BRF_01024)

4.2.3.2.6 [SRS_BSW_00307] Global variables naming convention

[

Type:	Valid
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:	--

](RS_BRF_01024)

4.2.3.2.7 [SRS_BSW_00310] API naming convention

[

Type:	Valid
Description:	All AUTOSAR Basic Software Modules shall implement an API based on the following naming rules: - Composition of API: <Module name>_ServiceName() - Module name: 2..8 letters, derived from WP Architecture SW Module List - 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:	WP Architecture SW Module List (Module Abbreviations)

](RS_BRF_01024)

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

[

Type:	Valid
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 not recommendation for the particular Module, it just illustrates Main processing function possibilities.

Dependencies:	
Supporting Material:	<Module name> shall be derived from WP Architecture “List of Basic Software Modules”, [DOC_MOD_LIST] (2…8 charactersWP Architecture SW Module List (Module Abbreviations))

]()

4.2.3.2.9 [SRS_BSW_00327] Error values naming convention

[

Type:	Valid
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 WP Architecture “List of Basic Software Modules”, [DOC_MOD_LIST] (2…8 characters)

](RS_BRF_01024)

4.2.3.2.10 [SRS_BSW_00335] Status values naming convention

[

Type:	Valid
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 WP Architecture “List of Basic Software Modules”, [DOC_MOD_LIST] (2…8 characters)

](RS_BRF_01024)

4.2.3.2.11 [SRS_BSW_00350] All AUTOSAR Basic Software Modules shall apply a specific naming rule for enabling/disabling the detection and reporting of development errors

[

Type:	Valid
Description:	All AUTOSAR Basic Software Modules shall apply the following naming rule for enabling/disabling the detection and reporting of development errors: <MODULENAME>_DEV_ERROR_DETECT
Rationale:	Provide module wide debug instrumentation facilities. Each defined keyword has to be properly documented.

Use Case:	Example:
	
	In Eep_Cfg.h:
	`#define EEP_DEV_ERROR_DETECT STD_ON /* detection module wide enabled */`
	…
	
	In source Eep.c:
	`#include "Eep_Cfg.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 WP Architecture “List of Basic Software Modules”, [DOC_MOD_LIST] (2…8 characters)

](RS_BRF_01024)

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

[

Type:	Valid
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 AUTOSAR_WP Architecture_BasicSoftwareModules.xls.

	< 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` PDUR_E_INIT_FAILED
Dependencies:	--
Supporting Material:	< Module Abbreviation > shall be derived from WP1.1.2 “List of Basic Software Modules”, [DOC_MOD_LIST] (2…8 characters)

]()

4.2.3.2.13 [SRS_BSW_00410] Compiler switches shall have defined values

[

Type:	Valid
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:	--

](RS_BRF_01616)

4.2.3.2.14 [SRS_BSW_00411] All AUTOSAR Basic Software Modules shall apply a naming rule for enabling/disabling the existence of the API

[

Type:	Valid
Description:	All AUTOSAR Basic Software Modules shall apply the following naming rule for enabling/disabling the existence of the API.
Rationale:	Enable/Disable the reading out of version information
Use Case:	Example: In Eep_Cfg.h: `#define EEP_VERSION_INFO_API STD_ON /*API enabled */` …
Dependencies:	[SRS_BSW_00407]
Supporting Material:	< MODULENAME > shall be derived from WP Architecture “List of Basic Software Modules”, [DOC_MOD_LIST] (2…8 characters)

](RS_BRF_01024)

4.2.3.2.15 [SRS_BSW_00463] Naming convention of callout prototypes

[

Type:	Valid
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: --- FUNC(void, MSN_CBN_CODE) 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”
	FUNC(void, COM_SOMECALLOUT_CODE) Com_SomeCallout(void);
	#define COM_STOP_SEC_COM_SOMECALLOUT_CODE
	#include “Com_MemMap.h”

Dependencies:	--
Supporting Material:	“List of Basic Software Modules”, UID [150]

](RS_BRF_01024)

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

[

Type:	Valid

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_cfg.h" is defined in a SWS; "moduleabc_cfg.h" and "ModuleAbc_Cfg.h" are implemented by two different implementers and then included in modules developed by different implementers. If the file “moduleabc_cfg” is included with the directive #include <ModuleAbc_Cfg.h” on a case sensitive file system, the file won´t be found.
Dependencies:	--
Supporting Material:	--

]()

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

[

Type:	Valid
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_cfg.h rte_cfg.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:	--

]()

4.2.3.2.18 [SRS_BSW_00468] Specification documents shall define naming conventions if applicable

[

Type:	Valid
Description:	Specification documents shall define naming conventions if applicable
Rationale:	Naming conventions must be defined in specifications documents for a more standardized and clear documentation.
Use Case:	

Dependencies:	
Supporting Material:	

](RS_BRF_01024,RS_BRF_01192)

4.2.3.3 Module file structure

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

[

Type:	Valid
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 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_00158],[SRS_BSW_00345], [SRS_BSW_00347], [SRS_BSW_00412],[SRS_BSW_00314], [SRS_BSW_00419]
Supporting Material:	< Module name > shall be derived from WP Architecture “List of Basic Software Modules”, [DOC_MOD_LIST] (2…8 characters)

](RS_BRF_02080)

4.2.3.3.2 [SRS_BSW_00158] All modules of the AUTOSAR Basic Software shall strictly separate configuration from implementation

[

Type:	Valid
Description:	All modules of the AUTOSAR Basic Software shall strictly separate configuration from implementation.
Rationale:	Easy and clear configuration.
Use Case:	The file Adc_Cfg.h contains the pre--compile time configurable parameters to set the properties of the module Adc. Post build configuration parameters are stored in the file Adc_PBcfg.c
Dependencies:	[SRS_BSW_00345],[SRS_BSW_00346]
Supporting Material:	--

]()

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

[

Type:	Valid
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 WP Architecture “List of Basic Software Modules”, [DOC_MOD_LIST] (2…8 characters)

](RS_BRF_01144)

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

[

Type:	Valid
Description:	I. A Basic Software Module implementing an Autosar Service shall include its Application Types Header file in the Module Header File. II. Data Types used in Standard Interface and Standard AUTOSAR Interface shall only be defined in RTE Types Header file only. III. 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. IV. 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 RTE_SWS

]()

4.2.3.4 Standard header files

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

[

Type:	Valid
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) • include the compiler specific language extension header (Compiler.h) • define the type Std_ReturnType • define values for E_OK and E_NOT_OK
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 OSEK OS, E_OK has to be checked for being already defined:
	`/* for OSEK compliance this typedef has been added */`
	`#ifndef STATUSTYPEDEFINED`
	`#define STATUSTYPEDEFINED`
	`typedef unsigned char StatusType;`
	`#define E_OK 0`
	`#endif`

]()

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

[

Type:	Valid
Description:	All integer 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 integer 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:	--

](RS_BRF_02080)

4.2.3.4.3 [SRS_BSW_00361] All mappings of not standardized keywords of compiler specific scope shall be placed and organized in a compiler specific type and keyword header

[

Type:	Valid
Description:	All mappings of not standardized keywords of compiler specific scope shall be placed and organized in a compiler specific type and keyword header. Name of compiler specific type/keyword header file: Compiler.h
Rationale:	Provision of a compiler specific header containing proprietary pre--processor directives as well as wrapper macros for all specialized language extensions.
Use Case:	Different compilers can require extended keywords to be placed in different places It is not possible to accommodate the different implementations with inline macros, so a function--like macro style is adopted instead. This macro wraps the return type of the function and therefore permits additions to made, such as __far__, either before or after the return type.
Dependencies:	[SRS_BSW_00306], [SRS_BSW_00348]
Supporting Material:	--

](RS_BRF_02080)

4.2.3.5 Module Design

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

[

Type:	Valid
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:	--

]()

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

[

Type:	Valid
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 NVRAM Manager shall not know all processor registers because someone has included the processor register file in another header file used by the NVRAM manager.
Dependencies:	--
Supporting Material:	--

](RS_BRF_02024)

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

[

Type:	Valid
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:	--

](RS_BRF_02072,RS_BRF_02112,RS_BRF_02032)

4.2.3.5.4 [SRS_BSW_00312] Shared code shall be reentrant

[

Type:	Valid
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:	AUTOSAR_SRS_BSWGeneral.doc, section 7.1.7

]()

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

[

Type:	Valid
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:	NVRAM Manager, Network Management, …
Dependencies:	--
Supporting Material:	--

](RS_BRF_01000)

4.2.3.5.6 [SRS_BSW_00439] Enable BSW modules to handle interrupts

[

Type:	Valid
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:	--

]()

4.2.3.5.7 [SRS_BSW_00448] Module SWS shall not contain requirements from Other Modules

[

Type:	Valid
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 validrequirement 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 validCAN requirement in SRS which refers to the particular requirement in MCU Driver SRS
Dependencies:	--
Supporting Material:	--

]()

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

[

Type:	Valid
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:	--

]()

4.2.3.6 Types and keywords

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

[

Type:	Valid
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.
Use Case:	
Dependencies:	[SRS_BSW_00348], [SRS_BSW_00377],[SRS_BSW_00359]
Supporting Material:	--

](RS_BRF_01024)

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

[

Type:	Valid
Description:	1. A Basic Software Module can return a module specifictypes.

Rationale:	Example for possibility 1:
	`uint8 Can_Write(…)`
	return values: `E_OK` (0), `CAN_E_BUSY` (1), `CAN_E_FAILED` (2)
	`E_OK` is taken from `Std_Types.h`, `CAN_E_BUSY` and `CAN_E_FAILED` are `#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_E_BUSY, CAN_E_FAILED`
	
	`Can_ReturnType` is an enumeration type in can.h:
	`typedef enum`
	`{`
	`  CAN_OK = 0,`
	`  CAN_E_BUSY,`
	`  CAN_E_FAILED`
	`} 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!

Use Case:	
Dependencies:	[SRS_BSW_00357]
Supporting Material:	--

]()

4.2.3.6.3 [SRS_BSW_00304] All AUTOSAR Basic Software Modules shall use the following data types instead of native C data types

[

Type:	Valid
Description:	All AUTOSAR Basic Software Modules shall not use the native C data types.
Rationale:	MISRA--C compliance. The usage of native C--data types (`char, int, short, long`) is forbidden as size and sign are not unambiguously defined and therefore are platform specific. Portability, reusability
Use Case:	The '_least' data types can be chosen if optimal performance is required (e.g. for loop counters). `uint8_least … uint32_least` could all be 32 bit on a 32 bit platform.
Dependencies:	[SRS_BSW_00353]
Supporting Material:	[SRS_BSW_00007] HIS MISRA C

](

4.2.3.6.4 [SRS_BSW_00378] AUTOSAR shall provide a boolean type

[

Type:	Valid
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 WPs 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).

]()

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

[

Type:	Valid
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.
Use Case:	
Dependencies:	[SRS_BSW_00361]
Supporting Material:	--

]()

4.2.3.7 Global data

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

[

Type:	Valid
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:	--

](RS_BRF_01056)

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

[

Type:	Valid
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 modifier keyword.
Use Case:	`const uint8 MaxPayload = 0x18;`
Dependencies:	--
Supporting Material:	--

]()

4.2.3.8 Interface and API

4.2.3.8.1 [SRS_BSW_00371] The passing of function pointers as API parameter is forbidden for all AUTOSAR Basic Software Modules

[

Type:	Valid
Description:	The passing of function pointers as API parameter is forbidden for all AUTOSAR Basic Software Modules.
Rationale:	• HIS MISRA C • Protected Operating System compatibility • Callbacks shall be defined statically at compile time, not during runtime
Use Case:	No, forbidden!!!
Dependencies:	[SRS_BSW_00007]
Supporting Material:	--

](RS_BRF_01056)

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

[

Type:	Valid
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:	--

](RS_BRF_01056)

4.2.3.8.3 [SRS_BSW_00414] The init function may have parameters

[

Type:	Valid
Description:	The init function may have parameters. If post build time configuration is required, the pointer to the configuration shall be passed. If post build time configuration is required (with and without instances) the naming convention for the configuration pointer type shall be: <Module name>_ConfigType. If a module provides different variants where only some are supporting post build time, multiple (selectable) configuration parameter sets, all variants shall have a pointer as parameter. In this case the pre-compile variant and the link-time variant shall get a NULL as parameter, what shall be tested in case of enabled Development Error Tracer (DET). If instances of the module have to be addressed, the index and the according pointer to the configuration shall be passed. If a lower module includes a configuration pointer then the module, that calls the init function for the lower module, shall also have a configuration pointer. This implies that every module that is not a leaf module needs a pointer. In the case of leaf modules, if the module has a post build variant then the init function shall have a pointer.
Rationale:	--

Use Case:	Example:
	`void NvM_Init (void)`
	Or in case of multiple (selectable) configurable configuration parameter sets:
	`void Eep_Init (const Eep_ConfigType *ConfigPtr)`
	Or in case of an instance index:
	`void Fr_Init (uint8 Fr_CtrlIdx, const Fr_ConfigType *ConfigPtr)`

Dependencies:	[SRS_BSW_00101], [SRS_BSW_00358], [SRS_BSW_00400]
Supporting Material:	--

](RS_BRF_01056)

4.2.3.8.4 [SRS_BSW_00359] All AUTOSAR Basic Software Modules callback functions shall avoid return types other than void if possible

[

Type:	Valid

Description:	All AUTOSAR Basic Software Modules callback functions shall avoid return types other than void if possible. Callback functions routed to Software Components (SWCs) via the RTE shall be typed by Std_ReturnType, not void. 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 servers' partition is currently not available. In case the callback is used as notification only, the caller can assume that always E_OK is returned.
Rationale:	Callbacks could be used for notifications.
Use Case:	--
Dependencies:	--
Supporting Material:	--

](RS_BRF_01056,RS_BRF_01064)

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

[

Type:	Valid
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:	--

](RS_BRF_01056,RS_BRF_01064)

4.2.3.8.6 [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

[

Type:	Valid
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:	--

](RS_BRF_01056,RS_BRF_01064)

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

[

Type:	Valid
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:	MISRA--C Attention has to be paid within reentrant systems.

]()

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

[

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

]()

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

[

Type:	Valid
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:	--

](RS_BRF_01056,RS_BRF_01024)

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

[

Type:	Valid
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:	--

](RS_BRF_01056)

4.2.3.9 Concurrency

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

[

Type:	Valid
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,[SRS_BSW_00412]
Supporting Material:	--

](RS_BRF_01160,RS_BRF_02040)

4.2.3.9.2 [SRS_BSW_00460] Reentrancy Levels

[

Type:	Valid
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:	--

](RS_BRF_01160,RS_BRF_02040)

4.2.4 Software Documentation Requirements

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

[

Type:	Valid
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:	--

](RS_BRF_01192)

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

[

Type:	Valid
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:	--

]()

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

[

Type:	Valid
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:	--

](RS_BRF_01320)

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

[

Type:	Valid
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:	--

]()

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

[

Type:	Valid
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:	--

](RS_BRF_01064)

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

[

Type:	Valid
Description:	All Basic Software Modules shall provide a readable module vendor identification (according to HIS) 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 WP Architecture “List of Basic Software Modules”, [DOC_MOD_LIST] (2…8 characters) • HIS Software Supplier Identifications [STD_HIS_SUPPLIER_IDS]

](RS_BRF_01032)

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

[

Type:	Valid
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 Development 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 WP Architecture “List of Basic Software Modules”, [DOC_MOD_LIST] (2…8 characters) • Basic Software Module List, Column ‘Module ID‘, defines the module IDs.

](RS_BRF_01056,RS_BRF_01032)

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

[

Type:	Valid
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:	--

](RS_BRF_01032)

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

[

Type:	Valid
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 WP Architecture “List of Basic Software Modules”, [DOC_MOD_LIST] (2…8 characters)

](RS_BRF_01032)

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

[

Type:	Valid
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:	--

](RS_BRF_01032)

4.2.4.11 [SRS_BSW_00341] Module documentation shall contains all needed informations

[

Type:	Valid
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:	--

](RS_BRF_01032)

4.2.4.12 [SRS_BSW_00334] All Basic Software Modules shall provide an XML file that contains the meta data

[

Type:	Valid
Description:	All Basic Software Modules shall provide an XML file that contains the meta data which is required for the SW integration process.
Rationale:	• Being able to have several drivers of the same type (e.g. 2 different external flash drivers) on the same ECU without name clash • Ensure system consistency and correctness

Use Case:	`<function_provided>`
	`  <name>Eep_Write</name>`
	`  <prototype>Eep_ST16RF42_Write</prototype>`
	`</function_provided>`
	
	ST16RF42 is the type of the external EEPROM

Dependencies:	--
Supporting Material:	[ECU_CONF_SWS]

](RS_BRF_01032)

5 References

5.1 Deliverables of AUTOSAR

[DOC_LAYERED_ARCH] Layered Software Architecture
AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf

[DOC_MOD_LIST] List of Basic Software Modules
AUTOSAR_TR_BSWModuleList.pdf

[ECU_CONF_SRS] Requirements on ECU Configuration
AUTOSAR_RS_ECUConfiguration.pdf

[ECU_CONF_SWS] Specification of ECU Configuration
AUTOSAR_TPS_ECUConfiguration.pdf

[GLOSSARY] Glossary,
AUTOSAR_TR_Glossary.pdf

[DOC_STDTYPE_SWS] Specification of Standard Types,
AUTOSAR_SWS_StandardTypes.pdf

[DOC_MEMMAP_SWS] Specification of Memory Mapping,
AUTOSAR_SWS_MemoryMapping.pdf

[DOC_BSWSCHED_SWS] Specification of BSW Scheduler,
AUTOSAR_SWS_BSW_Scheduler.pdf

[ARReleaseManagement] Definition of Release Management Process,
AUTOSAR_PD_ReleaseManagementProcess.pdf

[TPS_STDT_0078] Software Standardization Template
AUTOSAR_TPS_StandardizationTemplate.pdf

5.2 Related standards and norms

5.2.1 OSEK

[STD_OSEK_OS] OSEK/VDX Operating System Specification
http://www.osek--vdx.org

5.2.2 HIS

[STD_HIS_SUPPLIER_IDS] HIS Software Supplier Identifications
http://www.automotive--his.de/his--ergebnisse.htm

[STD_HIS_MISRA_SUBSET] HIS Common Subset of MISRA C
http://www.automotive--his.de/his--ergebnisse.htm