Document Title	General Requirements on SPAL
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	9
Document Status	published
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	R19-11

Document Change History

Date	Release	Changed by	Change Description
2019-11-28	R19-11	AUTOSAR Release Management	• No content changes • Changed Document Status from Final to published
2018-10-31	4.4.0	AUTOSAR Release Management	• Editorial changes
2017-12-08	4.3.1	AUTOSAR Release Management	• SRS requirements refer to BMW specifications • Removed references to HIS • minor corrections / clarifications / editorial changes; for details please refer to the ChangeDocumentation
2016-11-30	4.3.0	AUTOSAR Release Management	• Editorial changes
2014-10-31	4.2.1	AUTOSAR Release Management	• Editorial changes
2013-03-15	4.1.1	AUTOSAR Administration	• Link Requirement with BSW Feature Document • Updating format of requirements according to TPS_StandardizationTemplate
2010-09-30	3.1.5	AUTOSAR Administration	• Changes in SRS_SPAL_12461: removed “All other registers shall be initialized by the start-up code” from description
2010-02-02	3.1.4	AUTOSAR Administration	• Legal disclaimer revised

Document Change History

Date	Release	Changed by	Change Description
2008-08-13	3.1.1	AUTOSAR Administration	• Legal disclaimer revised
2007-12-21	3.0.1	AUTOSAR Administration	• Document meta information extended • Small layout adaptations made
2007-01-24	2.1.15	AUTOSAR Administration	• Updated the use case in SRS_SPAL_12092 • Deleted the supporting material in SRS_SPAL_12077 since it was referencing a rejected requirement BSW12161 • • Legal disclaimer revised • “Advice for users” revised • “Revision Information” added
2006-05-16	2.0	AUTOSAR Administration	• Split the document. Each SPAL driver now has its own requirements document. • Changed the wake-up requirements. • Added 3 requirements • Changed 9 requirements • Rejected 5 requirements
2005-05-31	1.0	AUTOSAR Administration	• Initial release

Disclaimer

Table of Contents

1 Scope of Document

This document specifies general requirements on Basic Software Modules of the following software layers:

- Microcontroller Abstraction Layer
- ECU Abstraction Layer

Those modules are of the following type:

- Drivers for µC-internal and external peripherals
- Handlers
- Interfaces

The selection of modules is derived from the WP Architecture BSW Module List and Layered Architecture. The following modules are in scope:

- Memory drivers and interfaces (internal/external EEPROM, Flash, Flash EEPROM Emulation)
- I/O drivers (PORT, ADC, DIO, PWM, ICU, OCU)
- I/O Hardware Abstraction
- ECU onboard communication drivers and handlers (SPI)
- System drivers (internal/external Watchdog, MCU, GPT, RAM test)

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 Requirements Guidelines

Existing specifications shall be referenced (in form of a single requirement). Differences to these specifications are specified as additional requirements.

2.1 Conventions Used

- The representation of requirements in AUTOSAR documents follows the table specified in [5].

- In requirements, the following specific semantics are used (taken from Request for Comment RFC 2119 from the Internet Engineering Task Force IETF)

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119. Note that the requirement level of the document in which they are used modifies the force of these words.

- MUST: This word, or the terms "REQUIRED" or "SHALL", mean that the definition is an absolute requirement of the specification.
- MUST NOT: This phrase, or the phrase "SHALL NOT", means that the definition is an absolute prohibition of the specification.
- 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.)

2.2 Requirements quality

All Requirements shall have the following properties:

- Redundancy
  Requirements shall not be repeated within one requirement or in other requirements
- Clearness
  All requirements shall allow one possibility of interpretation only. Only technical terms of the glossary may be used.
- Atomicity
  Each Requirement shall only contain one requirement. A Requirement is atomic if it cannot be split up in further requirements.
- Testability
  Requirements shall be testable by analysis, review or test.
- Traceability
  The source and status of a requirement shall be visible at all times.

2.3 Requirements identification

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!

2.4 Requirements structure

Each chapter shall be structured in the following way:

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,...)
- ...

3 Acronyms and abbreviations

Acronyms and abbreviations that have a local scope are not contained in the AUTOSAR glossary. These must appear in a local glossary.

Acronym:	***Description:***
CS	Chip Select
DIO	Digital Input Output
ECU	Electric Control Unit
ICU	Interrupt Capture Unit
MAL	Old name of Microcontroller Abstraction Layer (replaced by MCAL because ‘MAL’ is a French term meaning ‘bad’)
MCAL	Microcontroller Abstraction Layer
MCU	Microcontroller Unit
MMU	Memory Management Unit
Master	A device controlling other devices (slaves, see below)
Slave	A device being completely controlled by a master device
NMI	Non Maskable Interrupt
OS	Operating System
OCU	Output Compare Unit
PLL	Phase Locked Loop
PWM	Pulse Width Modulation
RX	Reception (in the context of bus communication)
SPAL	Standard Peripheral Abstraction Layer (The name of this working group)
SFR	Special Function Register
RTE	Runtime Environment
WP	Work Package

Acronym:	***Description:***
STD	Standard
REQ	Requirement
UNINIT	Uninitialized (= not initialized)

As this is a document from professionals for professionals, all other terms are expected to be known.

4 Conceptual Issues

4.1 General Rules

1. Don’t do anything within our callbacks that exceeds 50 µs runtime this will affect the system performance too much.
2. Each driver specification is designed so that the driver itself will take care of atomicity and data integrity for data inside of the driver.
3. Application buffers shall be passed as pointers from the user to the driver.

4.2 List of drivers not affected by the clock frequency

The clock frequency is a parameter that has a very large influence to most of the drivers included in WP “Specification / Standardization of BSW”. Below is a list of the software modules that do not have a direct dependency on the clock frequency:

- PORT
- DIO
- RAM test

Conclusion: Most of the drivers have a strong dependency of the clock frequency therefore it is very important to carefully consider the influence across the whole system when configuring each software component.

4.3 MCAL relevant ECU Power Modes

The drivers included in WP “Specification / Standardization of BSW” shall support the ECU power modes defined in the Specification of ECU State Manager.
Different clock modes have to be supported. All drivers shall support re-initialization with different configuration settings. Please refer to the document “ECU State Manager”.

4.4 Wake-up Scenarios

Due to different timing requirements (e.g. ECUs that wake up cyclically and only check some inputs and go to sleep again as fast as possible) different initialization procedures are necessary. Examples:

- Initialization after Wake-up
- Initialization after Power On Reset

Conclusion: It is not possible to have a standard wake-up sequence. This sequence depends on the microcontroller hardware and the system requirements. Current specified concept allows for a standardized way to handle wake-up signaling and offers the possibility to customize the actual wake-up sequence.

4.5 Scheduling and integration of drivers

Today, 90% of the functions of known ECUs are scheduled cooperatively. Reasons are:

- Technical: Lower overhead (task switch time and task stack consumption) in comparison to pre-emptive systems
- Technical: Better possibility to create a deterministic behavior
- Technical: It is easier to reach stable 95% system load with a cooperative system than with a full pre-emptive
- Historical: Many ECUs are using a cooperative scheduling concept

For this reason, all drivers shall allow to be used within a cooperatively scheduled system. They shall not implement blocking code and expect that they are pre-empted by the operating system. Implementation hint: use state machines instead of linear code.

5 Requirements Tracing

Requirement	Description	Satisfied by
RS_BRF_01064	AUTOSAR BSW shall provide callback functions in order to access upper layer modules	SRS_SPAL_00157, SRS_SPAL_12056
RS_BRF_01096	AUTOSAR shall support start-up and shutdown of ECUs	SRS_SPAL_12057, SRS_SPAL_12068, SRS_SPAL_12125, SRS_SPAL_12163, SRS_SPAL_12461, SRS_SPAL_12463
RS_BRF_01104	AUTOSAR shall support sleep and wake-up of ECUs and buses	SRS_SPAL_12067, SRS_SPAL_12069, SRS_SPAL_12267
RS_BRF_01152	AUTOSAR shall support limited dynamic reconfiguration	SRS_SPAL_12265
RS_BRF_01440	AUTOSAR services shall support system diagnostic functionality	SRS_SPAL_12064
RS_BRF_01496	AUTOSAR shall standardize how events which move an ECU out of the SLEEP mode are handled	SRS_SPAL_12267
RS_BRF_02168	AUTOSAR diagnostics shall provide a central classification and handling of abnormal operative conditions	SRS_SPAL_00157, SRS_SPAL_12064
RS_BRF_02232	AUTOSAR shall support development with run-time assertion checks	SRS_SPAL_00157, SRS_SPAL_12448

6 Requirement Specification

6.1 Functional Requirements

6.1.1 General Requirements

This chapter contains general requirements which apply to all modules of the Microcontroller and ECU Abstraction Layers, but not necessarily to Basic Software Modules of other layers.

6.1.1.1 Configuration

6.1.1.1.1 [SRS_SPAL_12263] The implementation of all driver modules shall allow the configuration of specific module parameter types at link time

⌈

Type:	Valid
Description:	The implementation of all driver modules shall allow the configuration of the following module parameter types at link time: • values written to hardware registers • values used within the driver module (e.g. timings) • callback functions Those parameters shall be placed in a module external initialization data structure.
Rationale:	Delivery of driver modules as object code
Use Case:	Internal development models of e.g. SVDO and Hella
Dependencies:	[SRS_SPAL_12264] Specification of configuration items
Supporting Material:	Sophisticated software design techniques are necessary to achieve similar scalability and resource efficiency like source code.

⌋()

6.1.1.1.2 [SRS_SPAL_12056] All driver modules shall allow the static configuration of notification mechanism

⌈

Type:	Valid
Description:	All driver modules shall allow the static configuration of notification mechanisms. Pointers to callback functions shall not be passed via the API.
Rationale:	Flexibility and scalability
Use Case:	Give the possibility to run a driver within a protected operating system. Callbacks passed by the API and “pointing anywhere” cannot be used within a protected OS. MISRA recommends avoiding dynamic pointers to functions.
Dependencies:	--
Supporting Material:	--

⌋(RS_BRF_01064)

6.1.1.1.3 [SRS_SPAL_12267] Wakeup sources shall be initialized by MCAL drivers and/or the MCU driver

⌈

Type:	Valid
Description:	Wakeup sources shall be initialized by MCAL drivers and/or the MCU driver. Possible wake-up sources are e.g. reset, watchdog, NMI, interrupt etc.
Rationale:	Allow the configuration of MCU to wake-up.
Use Case:	The GPT interrupt is enabled by the GPT driver and should wake-up the MCU from Idle/Sleep/Stop mode.
Dependencies:	--
Supporting Material:	--

⌋(RS_BRF_01104,RS_BRF_01496)

6.1.1.2 Initialization

6.1.1.2.1 [SRS_SPAL_12057] All driver modules shall implement an interface for initialization

⌈

Type:	Valid
Description:	All driver modules shall implement an interface for initialization. This service shall initialize all module global variables and those SFRs that are used by this module.
Rationale:	Basic functionality.
Use Case:	--
Dependencies:	[SRS_SPAL_12125] Initialization of hardware resources
Supporting Material:	--

⌋(RS_BRF_01096)

6.1.1.2.2 [SRS_SPAL_12125] All driver modules shall only initialize the configured resources

⌈

Type:	Valid
Description:	All driver modules shall only initialize the configured resources. Resources that are not configured in the configuration file shall not be touched.
Rationale:	Allow integration with complex drivers without resource conflicts.
Use Case:	Timer channels 0..3 are used by the GPT driver, timer channels 4..6 are used by complex drivers
Dependencies:	[SRS_SPAL_12057] Driver module initialization
Supporting Material:	--

⌋(RS_BRF_01096)

6.1.1.2.3 [SRS_SPAL_12163] All driver modules shall implement an interface for de-initialization

⌈

Type:	Valid
Description:	All driver modules shall implement an interface for de-initialization. This service shall reset all module global variables and all SFRs that are used by this module to their default reset value. Values of registers which are not writeable are excluded.
Rationale:	Shut down the module. Create the same conditions like before initialization. Empty queues.
Use Case:	--
Dependencies:	--

Supporting Material:	

⌋(RS_BRF_01096)

6.1.1.2.4 [SRS_SPAL_12461] Specific rules regarding initialization of controller registers shall apply to all driver implementations

⌈

Type:	Valid
Description:	The following rules regarding initialization of controller registers shall apply to all driver implementations: 1. If the hardware allows for only one usage of the register, the driver module implementing that functionality is responsible for initializing the register 2. If the register can affect several hardware modules and if it is an I/O register it shall be initialized by the PORT driver 3. If the register can affect several hardware modules and if it is not an I/O register it shall be initialized by the MCU driver 4. One-time writable registers that require initialization directly after reset shall be initialized by the startup code
Rationale:	Unambiguous initialization of controller registers, no changes in driver implementation needed for different configurations.
Use Case:	1) All registers concerning the flash module shall be initialized by the flash driver 2) I/O Registers that can be used either for CAN, ADC or DIO shall be initialized by the PORT driver 3) Registers that affect the clock settings of different hardware modules shall be initialized by the MCU driver 4) Registers affecting the mapping of the register set, RAM or EEPROM shall be initialized in the startup code
Dependencies:	--
Supporting Material:	I/O register: Everything that can affect the functionality of a port pin.

⌋(RS_BRF_01096)

6.1.1.2.5 [SRS_SPAL_12462] The register initialization settings shall be published

⌈

Type:	Valid
Description:	The implementers of the respective driver modules have to publish all register initialization settings in the driver modules documentation.
Rationale:	The configurator (human or tool responsible for configuring the software) needs to get the register settings of the register that are not initialized directly by the driver
Use Case:	--
Dependencies:	SRS_SPAL_12461
Supporting Material:	--

⌋()

6.1.1.2.6 [SRS_SPAL_12463] The register initialization settings shall be combined and forwarded

⌈

Type:	Valid
Description:	The configurator shall combine all initialization settings from different drivers and check them for consistency (dependency and conflict). If this check is successful it shall forward those combined settings to the

	module that is responsible for initializing the hardware. If there are any inconsistencies, the configurator has to raise an error and the system build process has to be restarted.
Rationale:	Make sure all controller registers are used in a consistent way and all driver requirements on register initialization settings are fulfilled.
Use Case:	--
Dependencies:	[SRS_SPAL_12461], [SRS_SPAL_12462]
Supporting Material:	--

⌋(RS_BRF_01096)

6.1.1.2.7 [SRS_SPAL_12068] The modules of the MCAL shall be initialized in a defined sequence

⌈

Type:	Valid
Description:	The modules of the MCAL shall be initialized in the following sequence: 1. disable global interrupts 2. initialize overall registers (MCAL system module) 3. initialize all drivers 4. global interrupts may be enabled
Rationale:	Defined initialization sequence without side effects.
Use Case:	Power On Reset
Dependencies:	--
Supporting Material:	--

⌋(RS_BRF_01096)

6.1.1.2.8 [SRS_SPAL_12069] All drivers of the SPAL that wake up from a wake-up interrupt shall report the wake-up reason

⌈

Type:	Valid
Description:	All drivers of the SPAL that wake up from a wake-up interrupt shall report the wake-up reason to the ECU State Manager via the IO hardware abstraction. Notifications come from SPAL-drivers and shall be handled within the IO hardware abstraction module before the wake up reason is sent to the ECU state manager. Implementation hint: Usually this notification is done from the ISR of the wake-up.
Rationale:	The ECU State Manager needs the wake-up reason. It allows guaranteeing low consumption. For the ICU for instance, it avoids the report of non valid wake-up reasons (spikes).
Use Case:	The ISR of the associated wake-up interrupt calls the wake-up report function of the ECU State Manager if wake-up occurs.
Dependencies:	--
Supporting Material:	--

⌋(RS_BRF_01104)

6.1.1.3 Normal Operation

6.1.1.3.1 [SRS_SPAL_00157] All drivers and handlers of the AUTOSAR Basic Software shall implement notification mechanisms of drivers and handlers

⌈

Type:	Valid
Description:	All drivers and handlers of the AUTOSAR Basic Software shall implement the following notification mechanisms (configurable per module) for use within the Basic Software: • Polling (by reading a status information) • Callback functions • Error reporting function of the Default Error Tracer • Event reporting function of the Diagnostic Event Manager
Rationale:	Flexible integration Avoidance of strong coupling and dependencies
Use Case:	The completion of an EEPROM write command can be signaled via a callback function or by setting status information (which is accessible via the module interface). A fault occurred during EEPROM writing (cell defective) can be signaled to the Diagnostic Event Manager.
Dependencies:	Review annotation #35 of Mr. Schumpelt/Bosch
Supporting Material:	--

⌋(RS_BRF_01064, RS_BRF_02232, RS_BRF_02168)

6.1.1.3.2 [SRS_SPAL_12169] All driver modules that provide different operation modes shall provide a service for mode selection

⌈

Type:	Valid
Description:	All driver modules that provide different operation modes shall provide a service for mode selection. This service allows switching from one operation mode to another operation mode without the need of de-initialization and new initialization.
Rationale:	Allow operation mode changes where a full de-initialization and a new initialization would cause not desired artifacts.
Use Case:	Switch EEPROM driver from normal mode to burst mode
Dependencies:	[SRS_SPAL_12064] Change of operation mode during running operation
Supporting Material:	--

⌋()

6.1.1.3.3 [SRS_SPAL_12063] All driver modules shall only support raw value mode

⌈

Type:	Valid
Description:	All driver modules shall only support raw value mode. In this mode values passed via the API services are used directly without further scaling.
Rationale:	Scaling and adaptation to physical values is task of the ECU Abstraction Layer. Raw value mode provides the highest performance.
Use Case:	The I/O Hardware Abstraction converts a raw ADC value to a scaled value (e.g. voltage) and the other way round.
Dependencies:	--
Supporting Material:	--

⌋()

6.1.1.3.4 [SRS_SPAL_12075] All drivers with random streaming capabilities shall use application buffers

⌈

Type:	Valid
Description:	All drivers with random streaming capabilities (memory drivers) shall use application buffers. The caller shall not change the data during job processing of the driver.
Rationale:	Minimal RAM consumption, runtime efficiency
Use Case:	The EEPROM write service gets a pointer to the source data to be written. During EEPROM write operation the driver reads data from the application buffer. The EEPROM driver does not provide an own data buffer.
Dependencies:	--
Supporting Material:	--

⌋()

6.1.1.3.5 [SRS_SPAL_12129] The ISRs shall be responsible for resetting the interrupt flags and calling the according notification function

⌈

Type:	Valid
Description:	The ISRs shall be responsible for resetting the interrupt flags and calling the according notification function.
Rationale:	The notification functions can be user defined and therefore not allowed to have direct access to hardware.
Use Case:	--
Dependencies:	--
Supporting Material:	--

⌋()

6.1.1.4 Fault Operation

6.1.1.4.1 [SRS_SPAL_12064] All driver modules shall raise an error if the change of the operation mode leads to degradation of running operations

⌈

Type:	Valid
Description:	All driver modules shall raise an error if the change of the operation mode leads to degradation of running operations. The running operation shall be maintained. Further comment: This error condition shall never happen in correct system designs.
Rationale:	--
Use Case:	The SPI EEPROM operation mode is valid during a running SPI communication sequence.
Dependencies:	[SRS_SPAL_12169] Control of operation mode
Supporting Material:	--

⌋(RS_BRF_02168,RS_BRF_01440)

6.1.1.4.2 [SRS_SPAL_12448] All driver modules shall have a specific behavior after a development error detection

⌈

Type:	Valid
Description:	In case of a development error detection, all driver modules shall • report the error to the Default Error Tracer (DET) • skip the desired functionality (leave service without any action) • in case of standard return value return E_NOT_OK • in case of arbitrary return values (e.g. Dio_ReadPort) return 0
Rationale:	Uniform behavior of all SPAL modules. Avoid processing of wrong API parameters and thus avoid damage to hardware or dangerous system behavior.
Use Case:	The development error detection is enabled for a Driver. The driver service is called with a faulty input parameter value. The service shall NOT process the command (which might result in a serious malfunction).
Dependencies:	[SRS_SPAL_00157] Notification mechanisms of drivers and handlers
Supporting Material:	--

⌋(RS_BRF_02232)

6.1.1.5 Shutdown Operation

6.1.1.5.1 [SRS_SPAL_12067] All driver modules shall set their wake-up conditions depending on the selected operation mode

⌈

Type:	valid
Description:	All driver modules shall set their wake-up conditions depending on the selected operation mode.
Rationale:	Allow enabling of module specific wake-up interrupts.
Use Case:	Example: The ECU state manager switches the ECU power mode to 'ECU_POWERMODE_SLEEP'. The modules 'GPT' and 'ICU' enable specific wake-up interrupts according to their configuration related to 'ECU_POWERMODE_SLEEP'.
Dependencies:	[SRS_SPAL_12169] Control of operation mode
Supporting Material:	--

⌋(RS_BRF_01104)

6.2 Non-Functional Requirements

6.2.1 Timing requirements

6.2.1.1 [SRS_SPAL_12077] All drivers shall provide a non blocking implementation

⌈

Type:	Valid
Description:	All drivers shall provide a non blocking implementation. Note: 'blocking implementation' in this requirement means 'insensible, uncooperative usage of processor time' like long term loops.
Rationale:	Avoid undetermined waiting times. Allow all drivers to be used within a

	cooperatively scheduled system.
Use Case:	The waiting loop for the 'ADC Conversion Ready Flag' shall have an additional timeout condition.
Dependencies:	--
Supporting Material:	--

⌋()

6.2.1.2 [SRS_SPAL_12078] The drivers shall be coded in a way that is most efficient in terms of memory and runtime resources

⌈

Type:	Valid
Description:	The drivers shall be coded in a way that is most efficient in terms of memory and runtime resources.
Rationale:	Avoid waste of resources.
Use Case:	Usage of the driver in embedded automotive systems.
Dependencies:	--
Supporting Material:	--

⌋()

6.2.2 Software design requirements

6.2.2.1 [SRS_SPAL_12092] The driver's API shall be accessed by its handler or manager

⌈

Type:	Valid
Description:	If a driver is controlled by a handler or a manager, it is not allowed to bypass the handler/manager and access the driver's API directly. If a driver does not have a handler/manager above, it may be accessed directly.
Rationale:	Consistent access. Handlers and Managers shall not be bypassed.
Use Case:	The EEPROM driver is controlled exclusively by the NVRAM Manager via the EEPROM Abstraction module and the Memory Abstraction Interface. No other form of access to the EEPROM driver's API shall be allowed.
Dependencies:	--
Supporting Material:	--

⌋()

6.2.2.2 [SRS_SPAL_12265] Configuration data shall be kept constant

⌈

Type:	Valid
Description:	The contents of the init structure passed to the module via the init function shall be kept constant and available during runtime. Comment: Usually, this init data structure is located in ROM.
Rationale:	The module could access this structure at any time.
Use Case:	--
Dependencies:	--
Supporting Material:	--

⌋(RS_BRF_01152)

6.2.3 Process requirements

6.2.3.1 [SRS_SPAL_12264] Specification of configuration items shall be provided

⌈

Type:	Valid
Description:	The SWS (software specification) shall specify for each configuration element • whether it is configurable before or after compile time • where this configuration item is located (init data structure, configuration header file *_Cfg.h)
Rationale:	Enable correct implementation of configuration parameters that allow for object code delivery
Use Case:	--
Dependencies:	[SRS_SPAL_12263] Configuration after compile time
Supporting Material:	--

⌋()

7 References

7.1 Deliverables of AUTOSAR

[1] Glossary
AUTOSAR_TR_Glossary.pdf

[2] Layered Software Architecture
AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf

[3] General Requirements on Basic Software Modules
AUTOSAR_SRS_BSWGeneral.pdf

[4] Specification of ECU State Manager
AUTOSAR_SWS_ECUStateManager.pdf

[5] Software Standardization Template
AUTOSAR_TPS_StandardizationTemplate.pdf

7.2 Related standards and norms

None.