Document Title	Specification of EEPROM Driver
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	21

Document Status	published
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	R23-11

Document Change History

Date	Release	Changed by	Description
2023-11-23	R23-11	AUTOSAR Release Management	• Editorial Changes • Assigned new ID [SWS_Eep_00247] to a duplicate ID under EEP_E_ERASE_FAILED
2022-11-24	R22-11	AUTOSAR Release Management	• Proper implementation of TPS_STDT_00042
2021-11-25	R21-11	AUTOSAR Release Management	• Removed [SWS_Eep_00047] • EepJobCallCycle renamed to EepMainFunctionPeriod and moved from EepInitConfiguration to EepGeneral
2020-11-30	R20-11	AUTOSAR Release Management	• Editorial Changes
2019-11-28	R19-11	AUTOSAR Release Management	• MCAL Multicore Disctribution concept is changed from draft to Final • Changed Document Status from Final to published
2018-10-31	4.4.0	AUTOSAR Release Management	• MCAL Multicore Distribution
2017-12-08	4.3.1	AUTOSAR Release Management	• Changed EEP_E_TIMEOUT and EEP_E_BUSY from Development error to Runtime error • Changed description of [ECUC_Eep_00189]

△

2016-11-30	4.3.0	AUTOSAR Release Management	• Obsolete chapter “7.11 Support for Debugging” and sub chapter “10.2.1 Variants” are removed • Byte-wise read/write/erase access adaptation • Alignment of DataBuffers passed to functions
2015-07-31	4.2.2	AUTOSAR Release Management	• DET renaming and adaptation • Chapter 7 adaptation for error classification
2014-10-31	4.2.1	AUTOSAR Release Management	• Added pass/fail criteria and additional attributes for extended production errors • Removed redundant SWS IDs with respect to NULL_PTR check for `Eep_Init()`
2014-03-31	4.1.3	AUTOSAR Release Management	• Corrected formatting of requirements [SWS_Eep_00102], [SWS_Eep_00068] and [SWS_Eep_00137]
2013-10-31	4.1.2	AUTOSAR Release Management	• Removed the ’Timing’ row from the `Eep_MainFunction` API table • Editorial changes • Removed chapter(s) on change documentation
2013-03-15	4.1.1	AUTOSAR Administration	• MemMap.h changed to Eep_MemMap.h • Added Extended Production Errors
2011-12-22	4.0.3	AUTOSAR Administration	• Min max values of FloatParamDef parameters added for EEP178 & EEP185 • Replaced Module short name by module abbreviation
2010-09-30	3.1.5	AUTOSAR Administration	• Added DET errors `EEP_E_PARAM_POINTER`, `EEP_E_TIMEOUT` • Version check section (section 7.10) modified

▽

△

2010-02-02	3.1.4	AUTOSAR Administration	• Made hidden text visible in [SWS_Eep_00003], [SWS_Eep_00030], [SWS_Eep_00128] • Clarified optional callback notifications • Reworked external SPI EEPROM configuration example • Support VARIANT-POST-BUILD instead of VARIANT-LINK-TIME • Clarified synchronous behavior of `Eep_Cancel()` • Added support for debugging • Added DEM error codes for HW failure, removed SPI error • Changed job result to `MEMIF_BLOCK_INCONSISTENT` for differing data compare job • Replaced `Gpt_Init()` with `Eep_Init()` • Made `Dem_ReportErrorStatus()` a mandatory interface • Legal disclaimer revised
2008-08-13	3.1.1	AUTOSAR Administration	• Legal disclaimer revised
2007-12-21	3.0.1	AUTOSAR Administration	• Minor rewording of requirement ([SWS_Eep_00005]). • Introduction of new requirements ([SWS_Eep_00161] and [SWS_Eep_00162]) for NULL_PTR check. • Updates to [SWS_Eep_00028] and Figure 4 to correct spelling of `MEMIF_JOB_CANCELED` • Document meta information extended • Small layout adaptations made

▽

△

2007-01-24	2.1.15	AUTOSAR Administration	• Constant name correction • Limitation of erase cycles • Link-time configuration versus config pointer check • Job result for compare jobs is not specified • Legal disclaimer revised • Release Notes added • “Advice for users” revised • “Revision Information” added
2006-05-16	2.0	AUTOSAR Administration	• Document structure adapted to common Release 2.0 SWS Template. • adaptation to the new memory abstraction architecture • cancel function now asynchronous • deletion of two specifications elements that could lead to a misinterpretation of the described “write-cycle-reduction” functionality
2005-05-31	1.0	AUTOSAR Administration	• Initial Release

Disclaimer

Contents

1 Introduction and functional overview

This specification describes the functionality and API for an EEPROM driver. This specification is applicable to drivers for both internal and external EEPROMs.

The EEPROM driver provides services for reading, writing, erasing to/from an EEPROM. It also provides a service for comparing a data block in the EEPROM with a data block in the memory (e.g. RAM).

The behaviour of those services is asynchronous.

A driver for an internal EEPROM accesses the microcontroller hardware directly and is located in the Microcontroller Abstraction Layer. A driver for an external EEPROM uses handlers (SPI in most cases) or drivers to access the external EEPROM device. It is located in the ECU Abstraction Layer.

The functional requirements and the functional scope are the same for both types of drivers. Hence the API is semantically identical.

2 Acronyms and Abbreviations

Acronyms and abbreviations which have a local scope and therefore are not contained in the AUTOSAR glossary [1] must appear in a local glossary.

Acronym:	Description:
Data block	A data block may contain 1..n bytes and is used within the API of the EEPROM driver. Data blocks are passed with • Address offset in EEPROM • Pointer to memory location • Length to the EEPROM driver.
Data unit	The smallest data entity in EEPROM. The entities may differ for read/write/erase operation. Example 1: Motorola STAR12 Read: 1 byte Write: 2 bytes Erase: 4 bytes Example 2: external SPI EEPROM device Read/Write/Erase: 1 byte
Normal mode Burst mode	Some external SPI EEPROM devices provide the possibility of different access modes: • Normal mode: The data exchange with the EEPROM device via SPI is performed byte wise. This allows for cooperative SPI usage together with other SPI devices like I/O ASICs, external watchdogs etc. • Burst mode: The data exchange with the EEPROM device via SPI is performed block wise. The block size depends on the EEPROM properties, an example is 64 bytes. Because large blocks are transferred, the SPI is blocked by the EEPROM access in burst mode. This mode is used during ECU start-up and shut-down phases where fast reading/writing of data is required.
EEPROM cell	Smallest physical unit of an EEPROM device that holds the data. Usually 1 byte.

Table 2.1: Acronyms used in the scope of this Document

Abbreviation:	Description:
EEPROM	Electrically Erasable and Programmable Read Only Memory
NVRAM	Non Volatile Random Access Memory
NvM	Module name of NVRAM Manager [2]
EcuM	Module name of ECU State Manager [3]
DEM	Module name of Diagnostic Event Manager [4]
DET	Module name of Default Error Tracer [5]

Table 2.2: Abbreviations used in the scope of this Document

3 Related documentation

3.1 Input documents

[1] Glossary
AUTOSAR_FO_TR_Glossary

[2] Specification of NVRAM Manager
AUTOSAR_CP_SWS_NVRAMManager

[3] Specification of ECU State Manager
AUTOSAR_CP_SWS_ECUStateManager

[4] Specification of Diagnostic Event Manager
AUTOSAR_CP_SWS_DiagnosticEventManager

[5] Specification of Default Error Tracer
AUTOSAR_CP_SWS_DefaultErrorTracer

[6] General Specification of Basic Software Modules
AUTOSAR_CP_SWS_BSWGeneral

[7] Layered Software Architecture
AUTOSAR_CP_EXP_LayeredSoftwareArchitecture

[8] Specification of MCU Driver
AUTOSAR_CP_SWS_MCUDriver

[9] Specification of SPI Handler/Driver
AUTOSAR_CP_SWS_SPIHandlerDriver

[10] Specification of Memory Abstraction Interface
AUTOSAR_CP_SWS_MemoryAbstractionInterface

[11] Requirements on EEPROM Driver
AUTOSAR_CP_SRS_EEPROMDriver

[12] General Requirements on Basic Software Modules
AUTOSAR_CP_SRS_BSWGeneral

[13] General Requirements on SPAL
AUTOSAR_CP_SRS_SPALGeneral

3.2 Related specification

AUTOSAR provides a General Specification on Basic Software modules [6, SWS BSW General], which is also valid for EEPROM Driver.

Thus, the specification SWS BSW General shall be considered as additional and required specification for EEPROM Driver.

4 Constraints and assumptions

4.1 Limitations

The EEPROM driver does not provide mechanisms for providing data integrity (e.g. checksums, redundant storage, etc.).

The setting of the EEPROM write protection is not provided.

4.2 Applicability to car domains

No restrictions.

4.3 Applicability to safety related environments

This module can be used within safety relevant systems if the upper layer software provides following mechanisms for safety related data:

- Checksum protection
- Checking integrity before using data
- Redundant storage
- Verification of data after it has been written to EEPROM. For this, the compare function of the EEPROM driver can be used.

5 Dependencies to other modules

There are two classes of EEPROM drivers:

1. EEPROM drivers for onchip EEPROM.
   These are part of the Microcontroller Abstraction Layer (see [7]).
2. EEPROM drivers for external EEPROM devices.
   These are part of the ECU Abstraction Layer (see [7]).

[SWS_Eep_00082] ⌈The source code of external EEPROM drivers shall be independent of the microcontroller platform.⌋*()*

The internal EEPROM may depend on the system clock, prescaler(s) and PLL. Thus, changes of the system clock (e.g. PLL on – PLL off) may also affect the clock settings of the EEPROM hardware. Module EEPROM Driver do not take care of setting the registers which configure the clock, prescaler(s) and PLL in its init function. This has to be done by the MCU module [8].

A driver for an external EEPROM depends on the API and capabilities of the used onboard communication handler (e.g. SPI Handler/Driver [9]).

EEPROM driver is part of Memory Abstraction Architecture and for this reason some types depend on Memory Interface (MemIf) module [10].

5.1 File structure

[SWS_Eep_00228] ⌈If the module implementation uses custom interrupt processing, the interrupt service routines shall be placed in `Eep_Irq.c`.⌋*()*

6 Requirements Tracing

Uptracing to requirements in [11], [12], [13].

Requirement	Description	Satisfied by
[SRS_BSW_00101]	The Basic Software Module shall be able to initialize variables and hardware in a separate initialization function	[SWS_Eep_00004]
[SRS_BSW_00323]	All AUTOSAR Basic Software Modules shall check passed API parameters for validity	[SWS_Eep_00016] [SWS_Eep_00017] [SWS_Eep_00018]
[SRS_BSW_00327]	Error values naming convention	[SWS_Eep_00247] [SWS_Eep_00248] [SWS_Eep_00249] [SWS_Eep_00250] [SWS_Eep_00252] [SWS_Eep_00253] [SWS_Eep_00254] [SWS_Eep_00255]
[SRS_BSW_00331]	All Basic Software Modules shall strictly separate error and status information	[SWS_Eep_00247] [SWS_Eep_00248] [SWS_Eep_00249] [SWS_Eep_00250] [SWS_Eep_00252] [SWS_Eep_00253] [SWS_Eep_00254] [SWS_Eep_00255]
[SRS_BSW_00335]	Status values naming convention	[SWS_Eep_00138]
[SRS_BSW_00337]	Classification of development errors	[SWS_Eep_00000] [SWS_Eep_00200] [SWS_Eep_00201] [SWS_Eep_00202] [SWS_Eep_00203] [SWS_Eep_00247] [SWS_Eep_00248] [SWS_Eep_00249] [SWS_Eep_00250] [SWS_Eep_00252] [SWS_Eep_00253] [SWS_Eep_00254] [SWS_Eep_00255]
[SRS_BSW_00357]	For success/failure of an API call a standard return type shall be defined	[SWS_Eep_00138]
[SRS_BSW_00369]	All AUTOSAR Basic Software Modules shall not return specific development error codes via the API	[SWS_Eep_00033]
[SRS_BSW_00377]	A Basic Software Module can return a module specific types	[SWS_Eep_00138]
[SRS_BSW_00385]	List possible error notifications	[SWS_Eep_00000] [SWS_Eep_00247] [SWS_Eep_00248] [SWS_Eep_00249] [SWS_Eep_00250] [SWS_Eep_00252] [SWS_Eep_00253] [SWS_Eep_00254] [SWS_Eep_00255]
[SRS_BSW_00390]	Parameter content shall be unique within the module	[SWS_Eep_00094] [SWS_Eep_00095]
[SRS_BSW_00398]	The link-time configuration is achieved on object code basis in the stage after compiling and before linking	[SWS_Eep_00094]
[SRS_BSW_00402]	Each module shall provide version information	[SWS_Eep_00095]
[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	[SWS_Eep_00006] [SWS_Eep_00033]
[SRS_Eep_00087]	The EEPROM driver shall provide an asynchronous read function	[SWS_Eep_00009] [SWS_Eep_00013] [SWS_Eep_00256]
[SRS_Eep_00088]	The EEPROM driver shall provide an asynchronous write function	[SWS_Eep_00014] [SWS_Eep_00015] [SWS_Eep_00063] [SWS_Eep_00090] [SWS_Eep_00256]

△

Requirement	Description	Satisfied by
[SRS_Eep_00089]	The EEPROM driver shall provide an asynchronous erase function	[SWS_Eep_00019] [SWS_Eep_00020] [SWS_Eep_00070] [SWS_Eep_00072]
[SRS_Eep_00090]	The EEPROM driver shall provide a synchronous cancel function	[SWS_Eep_00021] [SWS_Eep_00027] [SWS_Eep_00028] [SWS_Eep_00215] [SWS_Eep_00216]
[SRS_Eep_00091]	The EEPROM driver shall provide a synchronous function which returns the job processing status	[SWS_Eep_00029]
[SRS_Eep_00092]	The EEPROM driver shall only write data if at least one data value of the affected erasable block is different from the data value to be written	[SWS_Eep_00060] [SWS_Eep_00064]
[SRS_Eep_00094]	The EEPROM driver shall handle the EEPROM memory segmentation	[SWS_Eep_00063] [SWS_Eep_00070] [SWS_Eep_00072] [SWS_Eep_00090]
[SRS_Eep_00095]	The EEPROM driver shall handle only one job at the same time	[SWS_Eep_00033] [SWS_Eep_00036]
[SRS_Eep_12047]	The EEPROM driver shall provide a function that has to be called for job processing	[SWS_Eep_00030] [SWS_Eep_00032]
[SRS_Eep_12050]	The job processing function of the EEPROM driver shall process only as much data as the EEPROM hardware can handle	[SWS_Eep_00051] [SWS_Eep_00054] [SWS_Eep_00057] [SWS_Eep_00069]
[SRS_Eep_12051]	The same requirements shall apply for an external and internal EEPROM driver	[SWS_Eep_00088]
[SRS_Eep_12072]	In fast mode, one cycle of the job processing function of the EEPROM driver shall limit the block size that is read from EEPROM to the configured maximum block size	[SWS_Eep_00054] [SWS_Eep_00055] [SWS_Eep_00073]
[SRS_Eep_12091]	The EEPROM driver shall provide an asynchronous compare function	[SWS_Eep_00025] [SWS_Eep_00026] [SWS_Eep_00256]
[SRS_Eep_12124]	The EEPROM driver for an external SPI EEPROM device shall access the SPI depending on the current EEPROM mode	[SWS_Eep_00052] [SWS_Eep_00053] [SWS_Eep_00055] [SWS_Eep_00073]
[SRS_Eep_12156]	The EEPROM driver shall provide a synchronous selection function	[SWS_Eep_00042] [SWS_Eep_00130] [SWS_Eep_00132]
[SRS_Eep_12157]	In normal mode, one cycle of the job processing function of the EEPROM driver shall limit the block size that is read from EEPROM to the configured default block size	[SWS_Eep_00051] [SWS_Eep_00052] [SWS_Eep_00053]
[SRS_SPAL_00157]	All drivers and handlers of the AUTOSAR Basic Software shall implement notification mechanisms of drivers and handlers	[SWS_Eep_00024] [SWS_Eep_00029] [SWS_Eep_00045] [SWS_Eep_00046]
[SRS_SPAL_12056]	All driver modules shall allow the static configuration of notification mechanism	[SWS_Eep_00049]
[SRS_SPAL_12057]	All driver modules shall implement an interface for initialization	[SWS_Eep_00004]
[SRS_SPAL_12064]	All driver modules shall raise an error if the change of the operation mode leads to degradation of running operations	[SWS_Eep_00033]

▽

△

Requirement	Description	Satisfied by
[SRS_SPAL_12075]	All drivers with random streaming capabilities shall use application buffers	[SWS_Eep_00037]
[SRS_SPAL_12448]	All driver modules shall have a specific behavior after a development error detection	[SWS_Eep_00016] [SWS_Eep_00017] [SWS_Eep_00018] [SWS_Eep_00033]

Table 6.1: RequirementsTracing

7 Functional specification

7.1 General behavior

[SWS_Eep_00088] ⌈The Eep SWS shall be valid both for internal and external EEPROMs.

The Eep SWS defines asynchronous services for EEPROM operations (read/write/erase/compare).⌋*(SRS_Eep_12051)*

[SWS_Eep_00036] ⌈The Eep module shall not buffer jobs. The Eep module shall accept only one job at a time. During job processing, the Eep module shall accept no other job.⌋*(SRS_Eep_00095)*

Note: when running in production mode it is assumed that the Eep user will never issue jobs concurrently; therefore error handling for this requirement is restricted to development, see [SWS_Eep_00033].

[SWS_Eep_00037] ⌈The Eep module shall not buffer data to be read or written. The Eep module shall use application data buffers that are referenced by a pointer passed via the API.⌋*(SRS_SPAL_12075)*

[SWS_Eep_00256] ⌈Eep driver shall handle data buffer alignment internally. Instead of imposing any requirements on a RAM buffers' alignments (as they are uint8*) it shall handle passed pointers as being just byte-aligned.⌋*(SRS_Eep_00087, SRS_Eep_00088, SRS_Eep_12091)*

7.2 Error Classification

This section describes how the Eep module has to manage the several error classes that may occur during the life cycle of this basic software.

The general requirements document of AUTOSAR [12] specifies that all basic software modules must distinguish (according to the product life cycle) 3 error types:

- Development errors: These errors should be detected and fixed during development phase. In most cases, these errors are software errors.
- Runtime errors: These errors are software exceptions that may occur in the production (i.e. series) code, due to software real time
- Production errors: These errors are hardware errors and software exceptions that cannot be avoided and are expected to occur in the production (i.e. series) code.

7.2.1 Development Errors

[SWS_Eep_00000] Definiton of development errors in module Eep ⌈

Type of error	*Related error code*	*Error value*
Invalid configuration set selection	EEP_E_INIT_FAILED	0x10
Invalid configuration set selection	EEP_E_PARAM_ADDRESS	0x11
Invalid configuration set selection	EEP_E_PARAM_DATA	0x12
Invalid configuration set selection	EEP_E_PARAM_LENGTH	0x13
API service called without module initialization	EEP_E_UNINIT	0x20
API service called with a NULL pointer	EEP_E_PARAM_POINTER	0x23

⌋*(SRS_BSW_00337, SRS_BSW_00385)*

7.2.2 Runtime Errors

[SWS_Eep_00251] Definiton of runtime errors in module Eep ⌈

Type of error	*Related error code*	*Error value*
API service called while driver still busy	EEP_E_BUSY	0x21
Timeout exceeded	EEP_E_TIMEOUT	0x22

⌋*()*

7.2.3 Transient Faults

There are no transient faults.

7.2.4 Production Errors

There are no production errors.

7.2.5 Extended Production Errors

7.2.5.1 EEP_E_ERASE_FAILED

[SWS_Eep_00242] ⌈

Error Name:	EEP_E_ERASE_FAILED	
Short Description:	EEPROM erase failed (HW)	
Long Description:	The Eeprom module reports this error when EEPROM erase job fails due to a hardware error.	
Detection Criteria:	Fail	EEPROM erase job failed (see [SWS_Eep_00255]).
	Pass	EEPROM erase job finished successfully (see [SWS_Eep_00247]).
Secondary Parameters:	N/A	
Time Required:	N/A	
Monitor Frequency:	Implementation specific	

⌋*()*

[SWS_Eep_00255] ⌈The production error code `EEP_E_ERASE_FAILED` shall be reported with FAILED when the Eeprom erase function failed.⌋*(SRS_BSW_00337, SRS_BSW_00385, SRS_BSW_00327, SRS_BSW_00331)*

[SWS_Eep_00247] ⌈The production error code `EEP_E_ERASE_FAILED` shall be reported with PASSED when the Eeprom erase function was executed successfully.⌋ *(SRS_BSW_00337, SRS_BSW_00385, SRS_BSW_00327, SRS_BSW_00331)*

7.2.5.2 EEP_E_WRITE_FAILED

[SWS_Eep_00243] ⌈

Error Name:	EEP_E_WRITE_FAILED	
Short Description:	EEPROM write failed (HW)	
Long Description:	The Eeprom module reports this error when EEPROM write job fails due to a hardware error.	
Detection Criteria:	Fail	EEPROM write job failed (see [SWS_Eep_00249]).
	Pass	EEPROM write job finished successfully (see [SWS_Eep_00248]).
Secondary Parameters:	N/A	

▽

△

Time Required:	N/A
Monitor Frequency:	Implementation specific

⌋()

[SWS_Eep_00249] ⌈The production error code EEP_E_WRITE_FAILED shall be reported with FAILED when the Eeprom write function failed⌋*(SRS_BSW_00337, SRS_BSW_00385, SRS_BSW_00327, SRS_BSW_00331)*

[SWS_Eep_00248] ⌈The production error code EEP_E_WRITE_FAILED shall be reported with PASSED when the Eeprom write function was executed successfully.⌋ *(SRS_BSW_00337, SRS_BSW_00385, SRS_BSW_00327, SRS_BSW_00331)*

7.2.5.3 EEP_E_READ_FAILED

[SWS_Eep_00244] ⌈

Error Name:	EEP_E_READ_FAILED	
Short Description:	EEPROM read failed (HW)	
Long Description:	The Eeprom module reports this error when EEPROM read job fails due to a hardware error.	
Detection Criteria:	Fail	EEPROM read job failed (see [SWS_Eep_00250]).
	Pass	EEPROM read job finished successfully (see [SWS_Eep_00252]).
Secondary Parameters:	N/A	
Time Required:	N/A	
Monitor Frequency:	Implementation specific	

⌋()

[SWS_Eep_00250] ⌈The production error code EEP_E_READ_FAILED shall be reported with FAILED when the Eeprom read function failed.⌋*(SRS_BSW_00337, SRS_BSW_00385, SRS_BSW_00327, SRS_BSW_00331)*

[SWS_Eep_00252] ⌈The production error code EEP_E_READ_FAILED shall be reported with PASSED when the Eeprom read function was executed successfully.⌋ *(SRS_BSW_00337, SRS_BSW_00385, SRS_BSW_00327, SRS_BSW_00331)*

7.2.5.4 EEP_E_COMPARE_FAILED

[SWS_Eep_00245] ⌈

Error Name:	EEP_E_COMPARE_FAILED	
Short Description:	EEPROM compare failed (HW)	
Long Description:	The Eeprom module reports this error when EEPROM compare job fails due to a hardware error.	
Detection Criteria:	Fail	EEPROM compare job failed (see [SWS_Eep_00253]).
	Pass	EEPROM compare job finished successfully (see [SWS_Eep_00254]).
Secondary Parameters:	N/A	
Time Required:	N/A	
Monitor Frequency:	Implementation specific	

⌋()

[SWS_Eep_00253] ⌈The production error code `EEP_E_COMPARE_FAILED` shall be reported with FAILED when the Eeprom compare function failed.⌋*(SRS_BSW_00337, SRS_BSW_00385, SRS_BSW_00327, SRS_BSW_00331)*

[SWS_Eep_00254] ⌈The production error code `EEP_E_COMPARE_FAILED` shall be reported with PASSED when the Eeprom compare function was executed successfully.⌋ *(SRS_BSW_00337, SRS_BSW_00385, SRS_BSW_00327, SRS_BSW_00331)*

7.3 Error detection

For details refer to the chapter 7.3 “Error Detection” in [6].

7.3.1 API parameter checking

[SWS_Eep_00016] ⌈If development error detection for the module Eep is enabled: the functions `Eep_Read()`, `Eep_Write()`, `Eep_Compare()` and `Eep_Erase()` shall check that `DataBufferPtr` is not NULL. If `DataBufferPtr` is NULL, they shall raise development error `EEP_E_PARAM_DATA`, otherwise (if no development error detection is enabled) it shall return with `E_NOT_OK`.⌋*(SRS_BSW_00323, SRS_SPAL_12448)*

[SWS_Eep_00017] ⌈If development error detection for the module Eep is enabled: the functions `Eep_Read()`, `Eep_Write()`, `Eep_Compare()` and `Eep_Erase()` shall check that `EepromAddress` is valid. If `EepromAddress` is not within the valid EEPROM address range they shall raise development error `EEP_E_PARAM_ADDRESS`, otherwise (if no development error detection is enabled) it shall return with `E_NOT_OK`.⌋ *(SRS_BSW_00323, SRS_SPAL_12448)*

[SWS_Eep_00018] ⌈If development error detection for the module Eep is enabled: the functions `Eep_Read()`, `Eep_Write()`, `Eep_Compare()` and `Eep_Erase()` shall check that the parameter `Length` is within the specified minimum and maximum values:

- Min.: 1
- Max.: `EepSize` - `EepromAddress`

If the parameter `Length` is not within the specified minimum and maximum values, they shall raise development error `EEP_E_PARAM_LENGTH`, otherwise (if no development error detection is enabled) it shall return with `E_NOT_OK`.⌋*(SRS_BSW_00323, SRS_SPAL_12448)*

7.3.2 EEPROM state checking

[SWS_Eep_00033] ⌈The functions `Eep_SetMode()`, `Eep_Read()`, `Eep_Write()`, `Eep_Compare()` and `Eep_Erase()` shall check the EEPROM state for being `MEMIF_IDLE`. If the EEPROM state is not `MEMIF_IDLE` , the called function shall

- raise the development error `EEP_E_UNINIT` if the module has not been initialized yet and if development error detection for the module Eep is enabled
- raise the runtime error `EEP_E_BUSY` according to the EEPROM state
- reject the service with `E_NOT_OK` (except `Eep_SetMode()` because this service has no return value).

⌋*(SRS_BSW_00406, SRS_BSW_00369, SRS_SPAL_12064, SRS_SPAL_12448, SRS_Eep_00095)*

7.3.3 EEPROM job encounters Hardware Failure

[SWS_Eep_00200] ⌈The production error code `EEP_E_ERASE_FAILED` shall be reported when the EEPROM erase function failed.⌋*(SRS_BSW_00337)*

[SWS_Eep_00201] ⌈The production error code `EEP_E_WRITE_FAILED` shall be reported when the EEPROM write function failed.⌋*(SRS_BSW_00337)*

[SWS_Eep_00202] ⌈The production error code `EEP_E_READ_FAILED` shall be reported when the EEPROM read function failed.⌋*(SRS_BSW_00337)*

[SWS_Eep_00203] ⌈The production error code `EEP_E_COMPARE_FAILED` shall be reported when the EEPROM compare function failed.⌋*(SRS_BSW_00337)*

7.3.4 Timeout Supervision

[SWS_Eep_00234] ⌈The runtime error code EEP_E_TIMEOUT shall be reported when the timeout supervision of a read, write, erase or compare job failed.⌋*()*

7.4 Error notification

For details refer to the chapter 7.2 “Error classification” in [6].

7.5 Processing of jobs - general requirements

[SWS_Eep_00128] ⌈The Eep module shall allow to be configured for interrupt or polling controlled job processing (if this is supported by the EEPROM hardware) through the configuration parameter EepUseInterrupts (see [ECUC_Eep_00163]).⌋*()*

[SWS_Eep_00129] ⌈If interrupt controlled job processing is supported and enabled, the external interrupt service routine located in Eep_Irq.c shall call an additional job processing function.⌋*()*

Hint:

The function Eep_MainFunction is still required for processing of jobs without hardware interrupt support (e.g. for read and compare jobs) and for timeout supervision.

[SWS_Eep_00246] ⌈If the underlying EEPROM technology requires a certain alignment of the read address or length information and if the address and/or length parameter for a read or compare Job are not correctly aligned, the function Eep_MainFunction shall internally compensate for this missing alignment, that is the function Eep_MainFunction shall provide byte-wise read access to the flash memory, regardless of any alignment restrictions imposed by the Hardware.⌋*()*

Additional general requirements only applicable for SPI EEPROM drivers:

[SWS_Eep_00056] ⌈For an Eep module driving an external EEPROM through SPI: If the SPI access fails, the Eep module shall behave as specified in [SWS_Eep_00068].⌋*()*

[SWS_Eep_00052] ⌈For an Eep module driving an external EEPROM through SPI: In normal EEPROM mode, the Eep module shall access the external EEPROM by usage of SPI channels that are configured for normal access to the SPI EEPROM.⌋*(SRS_Eep_12157, SRS_Eep_12124)*

[SWS_Eep_00053] ⌈For an Eep module driving an external EEPROM through SPI: The Eep’s configuration shall be such that the value of the configuration parameter

EepNormalReadBlockSize fits to the number of bytes that are readable in normal SPI mode.⌋(*SRS_Eep_12157*, *SRS_Eep_12124*)

[SWS_Eep_00055] ⌈For an Eep module driving an external EEPROM through SPI: In fast EEPROM mode, the Eep module shall access the external EEPROM by usage of SPI channels that are configured for burst access to the SPI EEPROM.⌋(*SRS_Eep_12072*, *SRS_Eep_12124*)

[SWS_Eep_00073] ⌈For an Eep module driving an external EEPROM through SPI: The Eep's configuration shall be such that the value of the configuration parameter EepFastReadBlockSize fits to the number of bytes that are readable in burst SPI mode.⌋(*SRS_Eep_12072*, *SRS_Eep_12124*)

7.6 Processing of read jobs

[SWS_Eep_00130] ⌈The Eep module shall provide two different read modes:

- normal mode
- fast mode

⌋(*SRS_Eep_12156*)

[SWS_Eep_00132] ⌈For an Eep module driving an external EEPROM: in case the external EEPROM does not support the burst mode, the Eep module shall accept a selection of fast read mode, but shall behave the same as in normal mode (don't care of mode parameter).⌋(*SRS_Eep_12156*)

[SWS_Eep_00051] ⌈In normal EEPROM mode, the Eep module shall read within one job processing cycle a number of bytes specified by the parameter EepNormalReadBlockSize.⌋(*SRS_Eep_12157*, *SRS_Eep_12050*)

Example:

- EepNormalReadBlockSize = 4
- Number of bytes to read: 21
- Required number of job processing cycles: 6
- Resulting read pattern: 4-4-4-4-4-1

[SWS_Eep_00054] ⌈In fast EEPROM mode, the Eep module shall read within one job processing cycle a number of bytes specified by the parameter EepFastReadBlockSize.⌋(*SRS_Eep_12072*, *SRS_Eep_12050*)

Example:

- EepFastReadBlockSize = 32
- Number of bytes to read: 110

- Required number of job processing cycles: 4
- Resulting read pattern: 32-32-32-14

[SWS_Eep_00058] ⌈When a read job is finished successfully, the Eep module shall set the EEPROM state to `MEMIF_IDLE` and shall set the job result to `MEMIF_JOB_OK`. If configured, the Eep module shall call the notification defined in the configuration parameter `EepJobEndNotification`.⌋*()*

[SWS_Eep_00068] ⌈When an error is detected during read job processing, the Eep module shall abort the job, shall set the EEPROM state to `MEMIF_IDLE` and shall set the job result to `MEMIF_JOB_FAILED`. If configured, the Eep module shall call the notification defined in the configuration parameter `EepJobErrorNotification`.⌋*()*

7.7 Processing of write jobs

[SWS_Eep_00057] ⌈The Eep module shall only write (and erase) as many bytes to the EEPROM as supported by the EEPROM hardware within one job processing cycle.

For internal EEPROMs, usually 1 data word can be written per time. Some external EEPROMs provide a RAM buffer (e.g. page buffer) that allows writing many bytes in one step.⌋*(SRS_Eep_12050)*

[SWS_Eep_00133] ⌈The Eep module shall provide two different write modes:

- normal mode
- fast mode

⌋*()*

[SWS_Eep_00134] ⌈For the case of an Eep module driving an external EEPROM: if the external EEPROMs does not provide burst mode, the Eep module shall accept a selection of fast mode, but shall behave the same as in normal mode (don't care of mode parameter).⌋*()*

[SWS_Eep_00097] ⌈In normal EEPROM mode, the Eep module shall write (and erase) within one job processing cycle a number of bytes specified by the parameter `EepNormalWriteBlockSize`.⌋*()*

Example:

- `EepNormalWriteBlockSize` = 1
- Number of bytes to write: 4
- Required number of job processing cycles: 4
- Resulting write pattern: 1-1-1-1

[SWS_Eep_00098] ⌈In fast EEPROM mode, the Eep module shall write (and erase) within one job processing cycle a number of bytes specified by the parameter EepFastWriteBlockSize.⌋*()*

Example:

- EepFastWriteBlockSize = 16
- Number of bytes to write: 55
- Required number of job processing cycles: 4
- Resulting write pattern: 16-16-16-7

[SWS_Eep_00060] ⌈If the value to be written to an EEPROM cell is already contained in the EEPROM cell, the Eep module should[1] skip the programming of that cell if it is configured to do so through the configuration parameter EepWriteCycleReduction.⌋*(SRS_Eep_00092)*

[SWS_Eep_00059] ⌈The Eep module shall erase an EEPROM cell before writing to it if this is not done automatically by the EEPROM hardware.⌋*()*

[SWS_Eep_00063] ⌈The Eep module shall preserve data of affected EEPROM cells by performing read – modify – write operations, if the number of bytes to be written are smaller than the erasable and/or writeable data units.⌋*(SRS_Eep_00088, SRS_Eep_00094)*

[SWS_Eep_00090] ⌈The Eep module shall preserve data of affected EEPROM cells by performing read – modify – write operations, if the given parameters (EepromAddress and Length) do not align with the erasable/writeable data units.⌋*(SRS_Eep_00088, SRS_Eep_00094)*

[SWS_Eep_00064] ⌈The Eep module shall keep the number of read – modify – write operations during writing a data block as small as possible.⌋*(SRS_Eep_00092)*

[SWS_Eep_00219] ⌈When a write job is finished successfully, the Eep module shall set the EEPROM state to MEMIF_IDLE and shall set the job result to MEMIF_JOB_OK. If configured, the Eep module shall call the notification defined in the configuration parameter EepJobEndNotification.⌋*()*

[SWS_Eep_00222] ⌈When an error is detected during write job processing, the Eep module shall abort the job, shall set the EEPROM state to MEMIF_IDLE and shall set the job result to MEMIF_JOB_FAILED. If configured, the Eep module shall call the notification defined in the configuration parameter EepJobErrorNotification.⌋*()*

Note: The verification of data written to EEPROM is not done within the write job processing function. If this is required for a data block, the compare function has to be called after the write job has been finished. This optimizes write speed, because data verification (read back and comparing data after writing) is only done where required.

[1] This feature is not mandatory but it depends on the EEPROM hardware manufacturer specification.

7.8 Processing of erase jobs

[SWS_Eep_00069] ⌈The Eep module shall erase only as many bytes to the EEPROM as supported by the EEPROM hardware within one job processing cycle.⌋*(SRS_Eep_12050)*

[SWS_Eep_00070] ⌈The Eep module shall use block erase commands if supported by the EEPROM hardware and if the given parameters (`EepromAddress` and `Length`) are aligned to erasable blocks.⌋*(SRS_Eep_00089, SRS_Eep_00094)*

[SWS_Eep_00072] ⌈The Eep module shall preserve the contents of affected EEPROM cells by using read – modify – write operations, if the given erase parameters (`EepromAddress` and `Length`) do not align with the erasable data units.⌋*(SRS_Eep_00089, SRS_Eep_00094)*

[SWS_Eep_00220] ⌈When an erase job is finished successfully, the Eep module shall set the EEPROM state to `MEMIF_IDLE` and shall set the job result to `MEMIF_JOB_OK`. If configured, the Eep module shall call the notification defined in the configuration parameter `EepJobEndNotification`.⌋*()*

[SWS_Eep_00223] ⌈When an error is detected during erase job processing, the Eep module shall abort the job, shall set the EEPROM state to `MEMIF_IDLE` and shall set the job result to `MEMIF_JOB_FAILED`. If configured, the Eep module shall call the notification defined in the configuration parameter `EepJobErrorNotification`.⌋*()*

7.9 Processing of compare jobs

For processing of compare jobs, the following EEPROM mode related requirements are applicable: [SWS_Eep_00130], [SWS_Eep_00132], [SWS_Eep_00051], [SWS_Eep_00054].

[SWS_Eep_00221] ⌈When a compare job is finished successfully, the Eep module shall set the EEPROM state to `MEMIF_IDLE` and shall set the job result to `MEMIF_JOB_OK`. If configured, the Eep module shall call the notification defined in the configuration parameter `EepJobEndNotification`.⌋*()*

[SWS_Eep_00224] ⌈When an error is detected during compare job processing, the Eep module shall abort the job, shall set the EEPROM state to `MEMIF_IDLE` and shall set the job result to `MEMIF_JOB_FAILED`. If configured, the Eep module shall call the notification defined in the configuration parameter `EepJobErrorNotification`.⌋*()*

[SWS_Eep_00075] ⌈When it is detected during compare job processing that the compared data areas are not equal, the EEPROM driver shall abort the job, set the EEPROM state to `MEMIF_IDLE` and the job result to `MEMIF_BLOCK_INCONSISTENT`. If configured, the callback function `Eep_JobErrorNotification` shall be called.⌋*()*

Requirements only applicable for SPI EEPROM drivers:

For processing of compare jobs, the following read job requirements are applicable: [SWS_Eep_00052], [SWS_Eep_00053], [SWS_Eep_00055], [SWS_Eep_00073].

7.10 Version check

For details refer to the chapter 5.1.8 “Version Check” in [6].

8 API specification

8.1 Imported types

In this chapter all types included from the following modules are listed:

[SWS_Eep_00138] Definition of imported datatypes of module Eep ⌈

Module	*Header File*	*Imported Type*
Dem	Rte_Dem_Type.h	Dem_EventIdType
	Rte_Dem_Type.h	Dem_EventStatusType
MemIf	MemIf.h	MemIf_JobResultType
	MemIf.h	MemIf_StatusType
Std	Std_Types.h	Std_ReturnType
	Std_Types.h	Std_VersionInfoType

⌋(*SRS_BSW_00335*, *SRS_BSW_00357*, *SRS_BSW_00377*)

8.2 Type definitions

8.2.1 Eep_ConfigType

[SWS_Eep_00225] Definition of datatype Eep_ConfigType ⌈

Name	Eep_ConfigType	
Kind	Structure	
Elements	Implementation Specific	
	Type	–
	Comment	The contents of the initialization data structure are EEPROM specific.
Description	This is the type of the external data structure containing the initialization data for the EEPROM driver.	
Available via	Eep.h	

⌋()

8.2.2 Eep_AddressType

[SWS_Eep_00226] Definition of datatype Eep_AddressType ⌈

Name	Eep_AddressType
Kind	Type
Derived from	uint

▽

△

Range	8 / 16 / 32 bits	–	Size depends on target platform and EEPROM device.
Description	Used as address offset from the configured EEPROM base address to access a certain EEPROM memory area.		
Available via	Eep.h		

⌋()

[SWS_Eep_00113] ⌈The type `Eep_AddressType` shall have 0 as lower limit for each EEPROM device.⌋()

[SWS_Eep_00217] ⌈The EEPROM module shall add a device specific base address to the address type `Eep_AddressType` if necessary.⌋()

8.2.3 Eep_LengthType

[SWS_Eep_00227] Definition of datatype Eep_LengthType ⌈

Name	Eep_LengthType		
Kind	Type		
Derived from	uint		
Range	Same as Eep_AddressType	–	Is the same type as Eep_Address Type because of arithmetic operations. Size depends on target platform and EEPROM device.
Description	Specifies the number of bytes to read/write/erase/compare.		
Available via	Eep.h		

⌋()

8.3 Function definitions

8.3.1 Eep_Init

[SWS_Eep_00143] Definition of API function Eep_Init ⌈

Service Name	Eep_Init	
Syntax	`void Eep_Init ( const Eep_ConfigType* ConfigPtr )`	
Service ID [hex]	0x00	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	ConfigPtr	Pointer to configuration set.
Parameters (inout)	None	

▽

△

Parameters (out)	None
Return value	None
Description	Service for EEPROM initialization.
Available via	Eep.h

⌋()

[SWS_Eep_00004] ⌈The function `Eep_Init` shall initialize all EEPROM relevant registers with the values of the structure referenced by the parameter `ConfigPtr`.⌋ *(SRS_BSW_00101, SRS_SPAL_12057)*

[SWS_Eep_00006] ⌈After having finished the module initialization, the function `Eep_Init` shall set the EEPROM state to `MEMIF_IDLE` and shall set the job result to `MEMIF_JOB_OK`.⌋*(SRS_BSW_00406)*

[SWS_Eep_00044] ⌈The function `Eep_Init` shall set the EEPROM mode to the configured default mode.⌋*()*

[SWS_Eep_00115] ⌈The Eep's user shall not call the function `Eep_Init` during a running operation.⌋*()*

8.3.2 Eep_SetMode

[SWS_Eep_00144] Definition of API function Eep_SetMode ⌈

Service Name	Eep_SetMode	
Syntax	`void Eep_SetMode ( MemIf_ModeType Mode )`	
Service ID [hex]	0x01	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	Mode	MEMIF_MODE_SLOW: Slow read access / normal SPI access. MEMIF_MODE_FAST: Fast read access / SPI burst access.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Sets the mode.	
Available via	Eep.h	

⌋()

[SWS_Eep_00042] ⌈The function `Eep_SetMode` shall set the EEPROM operation mode to the given `Mode` parameter.

The function `Eep_SetMode` checks the EEPROM state according to requirement [SWS_Eep_00033].⌋*(SRS_Eep_12156)*

[SWS_Eep_00116] ⌈The Eep's user shall not call the function `Eep_SetMode` during a running operation.⌋*()*

8.3.3 Eep_Read

[SWS_Eep_00145] Definition of API function Eep_Read ⌈

Service Name	Eep_Read	

Syntax	`Std_ReturnType Eep_Read (`
	`  Eep_AddressType EepromAddress,`
	`  uint8* DataBufferPtr,`
	`  Eep_LengthType Length`
	`)`	

Service ID [hex]	0x02	
Sync/Async	Asynchronous	
Reentrancy	Non Reentrant	
Parameters (in)	EepromAddress	Address offset in EEPROM (will be added to the EEPROM base address). Min.: 0 Max.: EEP_SIZE - 1
	Length	Number of bytes to read Min.: 1 Max.: EEP_SIZE - Eeprom Address
Parameters (inout)	None	
Parameters (out)	DataBufferPtr	Pointer to destination data buffer in RAM
Return value	Std_ReturnType	E_OK: read command has been accepted E_NOT_OK: read command has not been accepted
Description	Reads from EEPROM.	
Available via	Eep.h	

⌋()

[SWS_Eep_00009] ⌈The function Eep_Read shall copy the given parameters, initiate a read job, set the EEPROM status to MEMIF_BUSY, set the job result to MEMIF_JOB_PENDING and return.⌋(*SRS_Eep_00087*)

[SWS_Eep_00013] ⌈The Eep module shall execute the read job asynchronously within the Eep module's job processing function. During job processing the Eep module shall read a data block of size Length from EepromAddress + EEPROM base address to *DataBufferPtr.

The function Eep_Read checks the API parameters according to requirements [SWS_Eep_00016], [SWS_Eep_00017], [SWS_Eep_00018].

The function Eep_Read checks the EEPROM state according to requirement [SWS_Eep_00033].⌋(*SRS_Eep_00087*)

[SWS_Eep_00117] ⌈The Eep's user shall only call Eep_Read after the Eep module has been been initialized.⌋()

[SWS_Eep_00118] ⌈The Eep's user shall not call the function Eep_Read during a running Eep module job (read/write/erase/compare).⌋()

8.3.4 Eep_Write

[SWS_Eep_00146] Definition of API function Eep_Write ⌈

Service Name	Eep_Write	

Syntax	`Std_ReturnType Eep_Write (`
	`  Eep_AddressType EepromAddress,`
	`  const uint8* DataBufferPtr,`
	`  Eep_LengthType Length`
	`)`	

Service ID [hex]	0x03	
Sync/Async	Asynchronous	
Reentrancy	Non Reentrant	
Parameters (in)	EepromAddress	Address offset in EEPROM (will be added to the EEPROM base address). Min.: 0 Max.: EEP_SIZE - 1 This target address will be added to the EEPROM base address.
	DataBufferPtr	Pointer to source data
	Length	Number of bytes to write Min.: 1 Max.: EEP_SIZE - Eeprom Address
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: write command has been accepted `E_NOT_OK`: write command has not been accepted
Description	Writes to EEPROM.	
Available via	Eep.h	

⌋*()*

[SWS_Eep_00014] ⌈The function `Eep_Write` shall copy the given parameters, initiate a write job, set the EEPROM status to `MEMIF_BUSY`, set the job result to `MEMIF_JOB_PENDING` and return.⌋*(SRS_Eep_00088)*

[SWS_Eep_00015] ⌈The Eep module shall execute the write job asynchronously within the Eep module's job processing function. During job processing the Eep module shall write a data block of size `Length` from *`DataBufferPtr` to `EepromAddress` + EEPROM base address.

The function `Eep_Write` checks the API parameters according to requirements [SWS_Eep_00016], [SWS_Eep_00017], [SWS_Eep_00018].

The function `Eep_Write` checks the EEPROM state according to requirement [SWS_Eep_00033].⌋*(SRS_Eep_00088)*

[SWS_Eep_00119] ⌈The Eep module's user shall only call the function `Eep_Write` after the Eep module has been initialized.⌋*()*

[SWS_Eep_00120] ⌈The Eep module's user shall not call the function `Eep_Write` during a running Eep module job (read/write/erase/compare).⌋*()*

8.3.5 Eep_Erase

[SWS_Eep_00147] Definition of API function Eep_Erase ⌈

Service Name	Eep_Erase	

Syntax	`Std_ReturnType Eep_Erase (`
	`  Eep_AddressType EepromAddress,`
	`  Eep_LengthType Length`
	`)`	

Service ID [hex]	0x04	
Sync/Async	Asynchronous	
Reentrancy	Non Reentrant	
Parameters (in)	EepromAddress	Start address in EEPROM Min.: 0 Max.: EEP_SIZE - 1 This address will be added to the EEPROM base address.
	Length	Number of bytes to erase Min.: 1 Max.: EEP_SIZE - Eeprom Address
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: erase command has been accepted `E_NOT_OK`: erase command has not been accepted
Description	Service for erasing EEPROM sections.	
Available via	Eep.h	

⌋*()*

[SWS_Eep_00019] ⌈The function `Eep_Erase` shall copy the given parameters, initiate an erase job, set the EEPROM status to `MEMIF_BUSY`, set the job result to `MEMIF_JOB_PENDING` and return.⌋*(SRS_Eep_00089)*

[SWS_Eep_00020] ⌈The Eep module shall execute the erase job asynchronously within the Eep module's job processing function. The Eep module shall erase an EEPROM block starting from `EepromAddress` + EEPROM base address of size `Length`.

The function `Eep_Erase` checks the API parameters according to requirements [SWS_Eep_00016], [SWS_Eep_00017], [SWS_Eep_00018].

The function `Eep_Erase` checks the EEPROM state according to requirement [SWS_Eep_00033].⌋*(SRS_Eep_00089)*

[SWS_Eep_00121] ⌈The Eep module's user shall only call the function `Eep_Erase` after the Eep module has been initialized.⌋*()*

[SWS_Eep_00122] ⌈The Eep module's user shall not call the function `Eep_Erase` during a running Eep job (read/write/erase/compare).⌋*()*

8.3.6 Eep_Compare

[SWS_Eep_00148] Definition of API function Eep_Compare ⌈

Service Name	Eep_Compare	

Syntax	`Std_ReturnType Eep_Compare (`
	`  Eep_AddressType EepromAddress,`
	`  const uint8* DataBufferPtr,`
	`  Eep_LengthType Length`
	`)`	

Service ID [hex]	0x05	
Sync/Async	Asynchronous	
Reentrancy	Non Reentrant	
Parameters (in)	EepromAddress	Address offset in EEPROM (will be added to the EEPROM base address). Min.: 0 Max.: EEP_SIZE - 1 This target address will be added to the EEPROM base address.
	DataBufferPtr	Pointer to data buffer (compare data)
	Length	Number of bytes to compare Min.: 1 Max.: EEP_SIZE - Eeprom Address
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: compare command has been accepted `E_NOT_OK`: compare command has not been accepted
Description	Compares a data block in EEPROM with an EEPROM block in the memory.	
Available via	Eep.h	

⌋*()*

[SWS_Eep_00025] ⌈The function `Eep_Compare` shall copy the given parameters, initiate a compare job, set the EEPROM status to `MEMIF_BUSY`, set the job result to `MEMIF_JOB_PENDING` and return.⌋*(SRS_Eep_12091)*

[SWS_Eep_00026] ⌈The Eep module shall execute the compare job asynchronously within the Eep module's job processing function. During job processing the Eep module shall compare the EEPROM data block at `EepromAddress` + EEPROM base address of size `Length` with the data block at *`DataBufferPtr` of the same length.

The service `Eep_Compare` checks the API parameters according to requirements [SWS_Eep_00016], [SWS_Eep_00017], [SWS_Eep_00018].

The service `Eep_Compare` checks the EEPROM state according to requirement [SWS_Eep_00033].⌋*(SRS_Eep_12091)*

[SWS_Eep_00123] ⌈The Eep module's user shall only call the function `Eep_Compare` after the Eep module has been initialized.⌋*()*

[SWS_Eep_00124] ⌈The Eep module's user shall not call the function `Eep_Compare` during a running Eep job (read/write/erase/compare).⌋*()*

8.3.7 Eep_Cancel

[SWS_Eep_00149] Definition of API function Eep_Cancel ⌈

Service Name	Eep_Cancel
Syntax	`void Eep_Cancel ( void )`
Service ID [hex]	0x06
Sync/Async	Synchronous
Reentrancy	Non Reentrant
Parameters (in)	None
Parameters (inout)	None
Parameters (out)	None
Return value	None
Description	Cancels a running job.
Available via	Eep.h

⌋*()*

[SWS_Eep_00215] ⌈The function `Eep_Cancel` shall cancel an ongoing EEPROM read, write, erase or compare job.⌋*(SRS_Eep_00090)*

[SWS_Eep_00021] ⌈The function `Eep_Cancel` shall abort a running job synchronously so that directly after returning from this function a new job can be requested by the upper layer.⌋*(SRS_Eep_00090)*

Note: The function `Eep_Cancel` is synchronous in its behavior but at the same time asynchronous w.r.t. the underlying hardware. The job of the `Eep_Cancel` function (i.e. make the module ready for a new job request) is finished when it returns to the caller (hence it is synchronous), but on the other hand e.g. an erase job might still be ongoing in the hardware device (hence it is asynchronous w.r.t. the hardware).

[SWS_Eep_00027] ⌈The function `Eep_Cancel` shall set the EEP module state to `MEMIF_IDLE`.⌋*(SRS_Eep_00090)*

[SWS_Eep_00216] ⌈If configured, `Eep_Cancel` shall call the error notification function defined in `EepJobErrorNotification` in order to inform the caller about the cancelation of a job.⌋*(SRS_Eep_00090)*

[SWS_Eep_00028] ⌈The function `Eep_Cancel` shall set the job result to `MEMIF_JOB_CANCELED` if the job result currently has the value `MEMIF_JOB_PENDING`. Otherwise it shall leave the job result unchanged.⌋*(SRS_Eep_00090)*

[SWS_Eep_00136] ⌈The Eep module's user shall not call the `Eep_Cancel()` function during a running `Eep_MainFunction()` function.

[SWS_Eep_00136] can be achieved by one of the following scheduling configurations:

- Possibility 1: the job functions of the NVRAM manager and the EEPROM driver are synchronized (e.g. called sequentially within one task)

- Possibility 2: the task that calls the `Eep_MainFunction` function cannot be preempted by another task.

⌋()

Note: The states and data of the affected EEPROM cells will be undefined when canceling an ongoing write or erase job with the function `Eep_Cancel`.

Only the NVRAM Manager is authorized to use the function `Eep_Cancel`.

Canceling any job on-going with the service `Eep_Cancel` in an external EEPROM device might set this one in a blocking state.

8.3.8 Eep_GetStatus

[SWS_Eep_00150] Definition of API function Eep_GetStatus ⌈

Service Name	Eep_GetStatus	
Syntax	`MemIf_StatusType Eep_GetStatus ( void )`	
Service ID [hex]	0x07	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	None	
Return value	MemIf_StatusType	See document [3]
Description	Returns the EEPROM status.	
Available via	Eep.h	

⌋()

[SWS_Eep_00029] ⌈The function `Eep_GetStatus` shall return the EEPROM status synchronously.⌋*(SRS_SPAL_00157, SRS_Eep_00091)*

8.3.9 Eep_GetJobResult

[SWS_Eep_00151] Definition of API function Eep_GetJobResult ⌈

Service Name	Eep_GetJobResult
Syntax	`MemIf_JobResultType Eep_GetJobResult ( void )`
Service ID [hex]	0x08
Sync/Async	Synchronous
Reentrancy	Reentrant

▽

△

Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	None	
Return value	MemIf_JobResultType	See document [3]
Description	This service returns the result of the last job.	
Available via	Eep.h	

⌋()

[SWS_Eep_00024] ⌈The function `Eep_GetJobResult` shall synchronously return the result of the last job that has been accepted by the Eep module.⌋(*SRS_SPAL_00157*)

The services read/write/compare/erase share the same job status. Only the result of the last accepted job can be queried. Every new job that has been accepted by the EEPROM driver overwrites the job result with `MEMIF_JOB_PENDING`.

8.3.10 Eep_GetVersionInfo

[SWS_Eep_00152] Definition of API function Eep_GetVersionInfo ⌈

Service Name	Eep_GetVersionInfo	
Syntax	`void Eep_GetVersionInfo ( Std_VersionInfoType* versioninfo )`	
Service ID [hex]	0x0a	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	versioninfo	Pointer to where to store the version information of this module.
Return value	None	
Description	Service to get the version information of this module.	
Available via	Eep.h	

⌋()

[SWS_Eep_00239] ⌈If development error detection for the module Eep is enabled, and if the function `Eep_GetVersionInfo` is called with a NULL Pointer, the function `Eep_GetVersionInfo` shall raise the development error `EEP_E_PARAM_POINTER`, otherwise (if no development error detection is enabled) it shall return without any action.⌋()

8.4 Callback notifications

This chapter lists all functions provided by the Eep module to lower layer modules.

The EEPROM Driver is specified for either an internal microcontroller peripheral or an SPI external device. In the first case, the module belongs to the lowest layer of AUTOSAR Software Architecture hence this module specification has not identified any callback functions. In the second case, the module belongs to the ECU abstraction layer of AUTOSAR Software Architecture hence this module should provide callback notifications according to the SPI Handler/Driver specification requirements but those can not be specified here because they depend on module detailed design. That means, they depend on number of SPI Jobs and SPI Sequences that will be used.

[SWS_Eep_00137] ⌈In case the Eep module support an SPI external device, the Eep module shall provide additional callback notifications according to the SPI Handler/-Driver specification requirements.⌋*()*

8.5 Scheduled functions

This chapter lists all functions provided by the Eep module and called directly by the Basic Software Module Scheduler.

8.5.1 Eep_MainFunction

[SWS_Eep_00153] Definition of scheduled function Eep_MainFunction ⌈

Service Name	Eep_MainFunction

Syntax	`void Eep_MainFunction (`
	`  void`
	`)`

Service ID [hex]	0x09
Description	Service to perform the processing of the EEPROM jobs (read/write/erase/compare) .
Available via	SchM_Eep.h

⌋*()*

[SWS_Eep_00030] ⌈The function `Eep_MainFunction` shall perform the processing of the EEPROM read, write, erase and compare jobs.⌋*(SRS_Eep_12047)*

[SWS_Eep_00031] ⌈When a job has been initiated, the Eep's user shall call the function `Eep_MainFunction` cyclically until the job is finished.⌋*()*

Note: The function `Eep_MainFunction` may also be called cyclically if no job is currently pending.

[SWS_Eep_00084] ⌈The configuration parameter `EepMainFunctionPeriod` (see [ECUC_Eep_00170]) shall be used for internal timing of the EEPROM driver (deadline monitoring, write and erase timing etc.) if needed by the implementation and/or the underlying hardware.⌋*()*

[SWS_Eep_00032] ⌈The function `Eep_MainFunction` shall return without action if no job is pending.⌋*(SRS_Eep_12047)*

[SWS_Eep_00204] ⌈The function Eep_MainFunction shall set the job result to MEMIF_JOB_FAILED and report the error code EEP_E_ERASE_FAILED to the DEM if an EEPROM erase job fails due to a hardware error.⌋*()*

[SWS_Eep_00205] ⌈The function Eep_MainFunction shall set the job result to MEMIF_JOB_FAILED and report the error code EEP_E_WRITE_FAILED to the DEM if an EEPROM write job fails due to a hardware error.⌋*()*

[SWS_Eep_00206] ⌈The function Eep_MainFunction shall set the job result to MEMIF_JOB_FAILED and report the error code EEP_E_READ_FAILED to the DEM if an EEPROM read job fails due to a hardware error.⌋*()*

[SWS_Eep_00207] ⌈The function Eep_MainFunction shall set the job result to MEMIF_JOB_FAILED and report the error code EEP_E_COMPARE_FAILED to the DEM if an EEPROM compare job fails due to a hardware error.⌋*()*

[SWS_Eep_00235] ⌈The function Eep_MainFunction shall provide a timeout monitoring for the currently running job. That is it shall supervise the deadline of the read / compare / erase or write job.⌋*()*

[SWS_Eep_00236] ⌈The function Eep_MainFunction shall check whether the configured maximum erase time (see [ECUC_Eep_00178] EepEraseTime) has been exceeded. If this is the case, the function Eep_MainFunction shall raise the runtime error EEP_E_TIMEOUT.⌋*()*

[SWS_Eep_00237] ⌈The function Eep_MainFunction shall check whether the expected maximum write time (see note below) has been exceeded. If this is the case, the function Eep_MainFunction shall raise the runtime error EEP_E_TIMEOUT.⌋*()*

Note: The expected maximum write time depends on the current mode of the Eep module (see [SWS_Eep_00144]), the configured number of bytes to write in this mode (see [ECUC_Eep_00174] and [ECUC_Eep_00169] respectively), the size of a EEPROM write data unit (see [ECUC_Eep_00186]) and last the maximum time to write one data unit (see [ECUC_Eep_00185]). The number of bytes to write divided by the size of one EEPROM data unit yields the number of data units to write in one cycle. This multiplied with the maximum write time for one EEPROM data unit gives the expected maximum write time.

[SWS_Eep_00238] ⌈The function Eep_MainFunction shall check whether the expected maximum read / compare time (see note below) has been exceeded. If this is the case, the function Eep_MainFunction shall raise the runtime error EEP_E_TIMEOUT.⌋*()*

Note: There are currently no published parameters standardized for read / compare timings; these are difficult to standardize as they mostly depend on whether the EEPROM device is internal or external e.g. connected via SPI. Depending on the exact configuration being used, the implementation may use vendor-specific parameters similar as described for write jobs above. The configured number of bytes to read (and to compare) is coupled to the expected read / compare times which should be supervised by the Eep_MainFunction.

8.6 Expected interfaces

This chapter lists all functions the Eep module requires from other modules.

8.6.1 Mandatory interfaces

This chapter defines all interfaces which are required to fulfill the core functionality of the module.

[SWS_Eep_00154] Definition of mandatory interfaces in module Eep ⌈

API Function	*Header File*	*Description*
Dem_SetEventStatus	Dem.h	Called by SW-Cs or BSW modules to report monitor status information to the Dem. BSW modules calling Dem_SetEventStatus can safely ignore the return value. This API will be available only if ({Dem/Dem ConfigSet/DemEventParameter/DemEvent ReportingType} == STANDARD_REPORTING)
Det_ReportRuntimeError	Det.h	Service to report runtime errors. If a callout has been configured then this callout shall be called.

⌋()

8.6.2 Optional interfaces

This chapter defines all interfaces which are required to fulfill an optional functionality of EEPROM Driver module.

[SWS_Eep_00155] Definition of optional interfaces in module Eep ⌈

API Function	*Header File*	*Description*
Det_ReportError	Det.h	Service to report development errors.

⌋()

8.6.3 Configurable interfaces

In this chapter all interfaces are listed where the target function could be configured. The target function is usually a call-back function. The name of these interfaces is not fixed because they are configurable.

[SWS_Eep_00049] ⌈Notification callback functions are configurable through their corresponding configuration parameters. If no callback function is configured, there shall be no asynchronous notification.⌋*(SRS_SPAL_12056)*

Note: The EEP implementation needs to be able to cope with the use case that post build configuration does not specify a callback, in case no notification is required. This

may internally be realized by setting the callback function pointer in the initialization data structure to null.

8.6.3.1 End Job Notification

[SWS_Eep_00045] ⌈The Eep module shall call the callback function defined in the configuration parameter `EepJobEndNotification` when a job has been completed with a positive result:

- Read finished & OK
- Write finished & OK
- Erase finished & OK
- Compare finished & data blocks are equal

⌋*(SRS_SPAL_00157)*

[SWS_Eep_00157] Definition of configurable interface Eep_JobEndNotification ⌈

Service Name	Eep_JobEndNotification
Syntax	`void Eep_JobEndNotification ( void )`
Sync/Async	Synchronous
Reentrancy	Don't care
Parameters (in)	None
Parameters (inout)	None
Parameters (out)	None
Return value	None
Description	This callback function provided by the module user is called when a job has been completed with a positive result.
Available via	Eep.h

⌋*()*

[SWS_Eep_00126] ⌈The callback function defined in the configuration parameter `EepJobEndNotification` shall be callable on interrupt level.⌋*()*

8.6.3.2 Error Job Notification

[SWS_Eep_00046] ⌈The Eep module shall call the callback function defined in the configuration parameter `EepJobErrorNotification` when a job has been canceled or aborted with negative result:

- Read aborted
- Write aborted or failed
- Erase aborted or failed

- Compare aborted or data blocks are not equal.

⌋(*SRS_SPAL_00157*)

[SWS_Eep_00158] Definition of configurable interface Eep_JobErrorNotification ⌈

Service Name	Eep_JobErrorNotification
Syntax	`void Eep_JobErrorNotification ( void )`
Sync/Async	Synchronous
Reentrancy	Don't care
Parameters (in)	None
Parameters (inout)	None
Parameters (out)	None
Return value	None
Description	This callback function provided by the module user is called when a job has been canceled or finished with negative result.
Available via	Eep.h

⌋()

[SWS_Eep_00127] ⌈The callback function defined in the configuration parameter `EepJobErrorNotification` shall be callable on interrupt level.⌋()

9 Sequence diagrams

9.1 Initialization

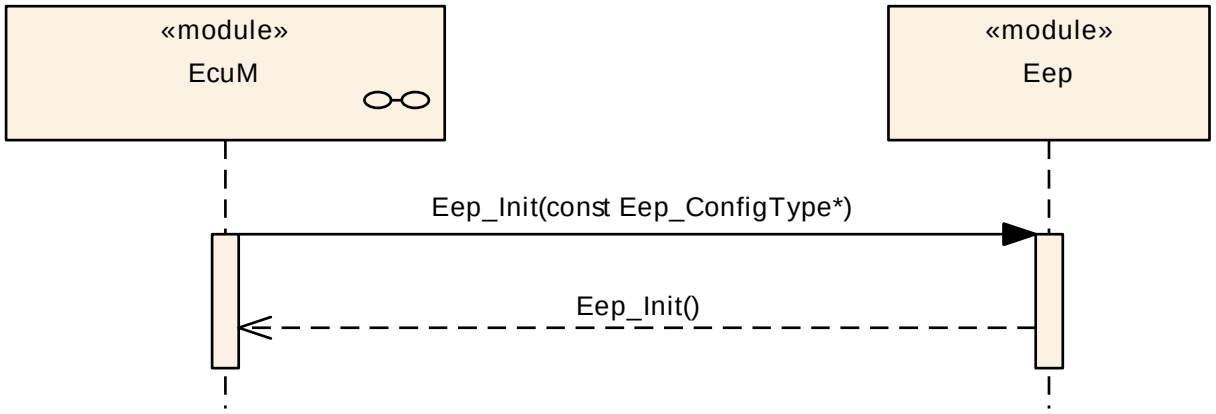

Figure 9.1: Initialization

9.2 Read/write/erase/compare

The following sequence diagram shows the write function as an example. The sequence for read, compare and erase is the same, only the processed block sizes may vary.

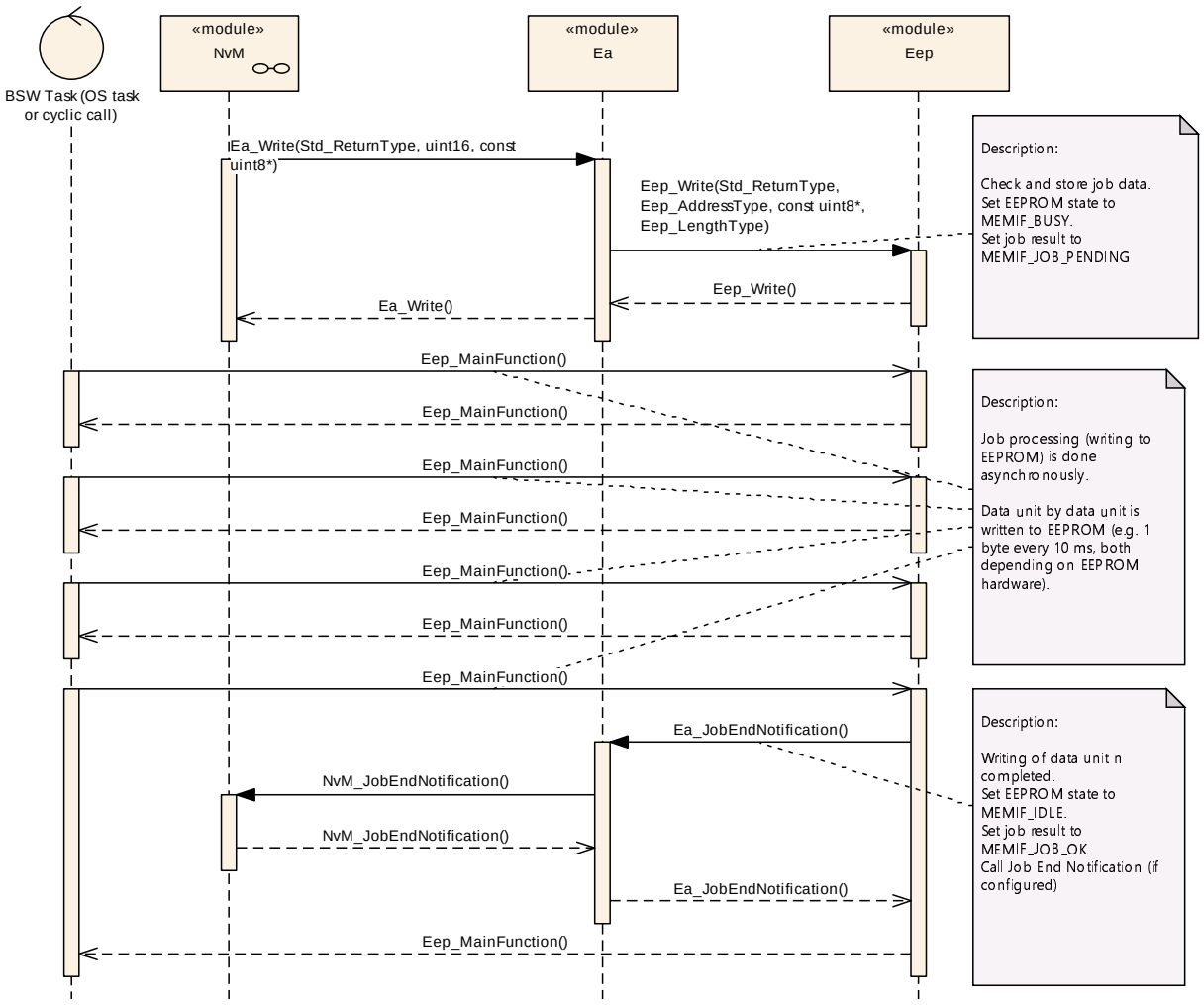

Figure 9.2: Write job

9.3 Cancelation of a running job

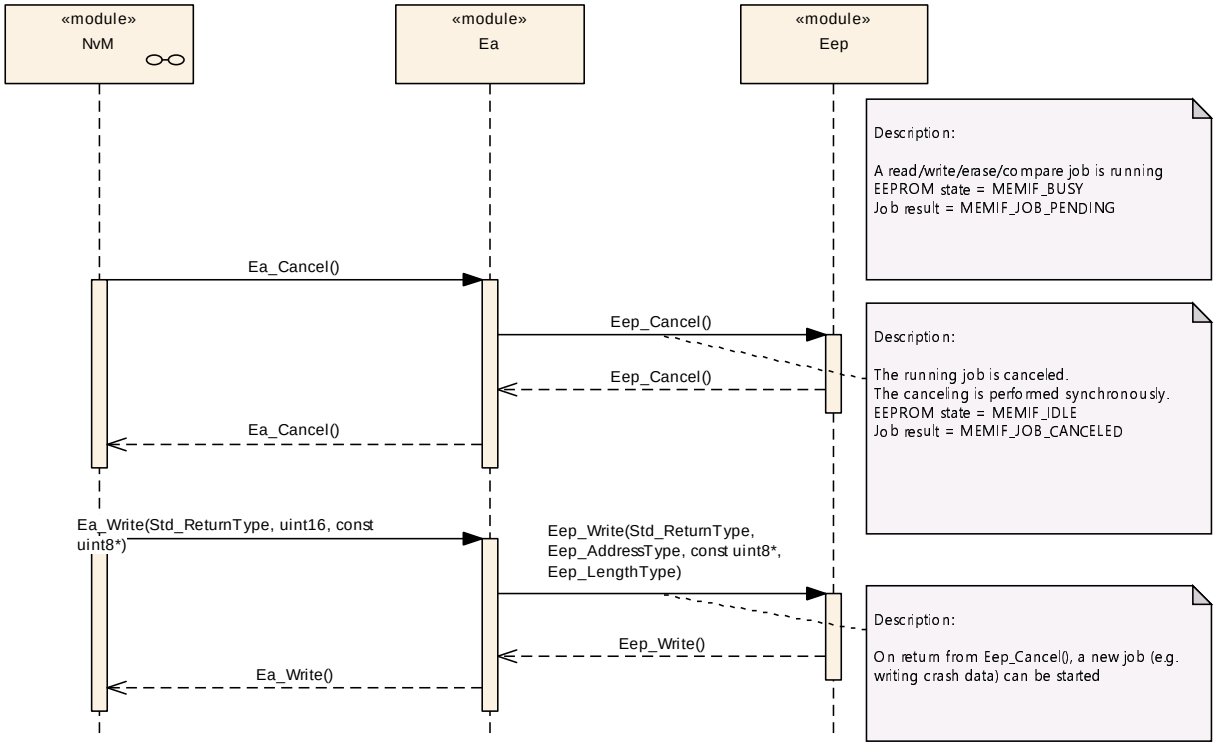

Figure 9.3: Cancelation of a running job

10 Configuration specification

10.1 How to read this chapter

For details refer to the chapter 10.1 “Introduction to configuration specification” in [6].

10.2 Containers and configuration parameters

The following chapters summarize all configuration parameters. The detailed meanings of the parameters describe Chapter 7 and Chapter 8. Further hardware / implementation specific parameters can be added if necessary.

10.2.1 Eep

SWS Item	[ECUC_Eep_00205]
Module Name	Eep
Description	Configuration of the Eep (internal or external EEPROM driver) module. Its multiplicity describes the number of EEPROM drivers present, so there will be one container for each EEPROM driver in the ECUC template. When no EEPROM driver is present then the multiplicity is 0.
Post-Build Variant Support	true
Supported Config Variants	VARIANT-POST-BUILD, VARIANT-PRE-COMPILE

Included Containers		
Container Name	**Multiplicity**	**Scope / Dependency**
EepGeneral	1	Container for general configuration parameters of the EEPROM driver. These parameters are always pre-compile.
EepInitConfiguration	1	Container for runtime configuration parameters of the EEPROM driver. Implementation Type: Eep_ConfigType.
EepPublishedInformation	1	Additional published parameters not covered by Common PublishedInformation container. Note that these parameters do not have any configuration class setting, since they are published information.

10.2.2 EepGeneral

SWS Item	[ECUC_Eep_00085]
Container Name	EepGeneral
Parent Container	Eep

▽

△

Description	Container for general configuration parameters of the EEPROM driver. These parameters are always pre-compile.
Configuration Parameters	

SWS Item	**[ECUC_Eep_00188]**		
Parameter Name	EepDevErrorDetect		
Parent Container	EepGeneral		
Description	Switches the development error detection and notification on or off. • true: detection and notification is enabled. • false: detection and notification is disabled.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

SWS Item	**[ECUC_Eep_00189]**		
Parameter Name	EepDriverIndex		
Parent Container	EepGeneral		
Description	Specifies the InstanceId of this module instance. If only one instance is present it shall have the Id 0.		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 254		
Default value	–		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: ECU		

SWS Item	**[ECUC_Eep_00170]**		
Parameter Name	EepMainFunctionPeriod		
Parent Container	EepGeneral		
Description	Call cycle time of the EEPROM driver's main function. Unit: [s]		
Multiplicity	0..1		
Type	EcucFloatParamDef		
Range	]0 .. INF[		
Default value	–		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	

▽

△

Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

SWS Item	**[ECUC_Eep_00163]**		
Parameter Name	EepUseInterrupts		
Parent Container	EepGeneral		
Description	Switches to activate or deactivate interrupt controlled job processing. true: Interrupt controlled job processing enabled. false: Interrupt controlled job processing disabled.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local dependency: Usually, this is only supported by some internal EEPROM peripherals.		

SWS Item	**[ECUC_Eep_00164]**		
Parameter Name	EepVersionInfoApi		
Parent Container	EepGeneral		
Description	Pre-processor switch to enable / disable the API to read out the modules version information. true: Version info API enabled. false: Version info API disabled.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

SWS Item	**[ECUC_Eep_00165]**		
Parameter Name	EepWriteCycleReduction		
Parent Container	EepGeneral		
Description	Switches to activate or deactivate write cycle reduction (EEPROM value is read and compared before being overwritten). true: Write cycle reduction enabled. false: Write cycle reduction disabled.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants

▽

△

	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

SWS Item	**[ECUC_Eep_00206]**		
Parameter Name	EepEcucPartitionRef		
Parent Container	EepGeneral		
Description	Maps the EEP driver to zero or one ECUC partition to make the driver API available in this partition.		
Multiplicity	0..1		
Type	Reference to EcucPartition		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: ECU		

No Included Containers

10.2.3 EepInitConfiguration

SWS Item	**[ECUC_Eep_00039]**
Container Name	EepInitConfiguration
Parent Container	Eep
Description	Container for runtime configuration parameters of the EEPROM driver. Implementation Type: Eep_ConfigType.
Configuration Parameters	

SWS Item	**[ECUC_Eep_00166]**		
Parameter Name	EepBaseAddress		
Parent Container	EepInitConfiguration		
Description	This parameter is the EEPROM device base address. Implementation Type: Eep_AddressType.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	–		
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	

▽

△

	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**[ECUC_Eep_00167]**		
Parameter Name	EepDefaultMode		
Parent Container	EepInitConfiguration		
Description	This parameter is the default EEPROM device mode after initialization. Implementation Type: MemIf_ModeType.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	MEMIF_MODE_FAST	The driver is working in fast mode (fast read access / SPI burst access).	
	MEMIF_MODE_SLOW	The driver is working in slow mode.	
Default value	MEMIF_MODE_SLOW		
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**[ECUC_Eep_00168]**		
Parameter Name	EepFastReadBlockSize		
Parent Container	EepInitConfiguration		
Description	Number of bytes read within one job processing cycle in fast mode. If the hardware does not support burst mode this parameter shall be set to the same value as Eep NormalReadBlockSize. Implementation Type: Eep_LengthType.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	–		
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**[ECUC_Eep_00169]**		
Parameter Name	EepFastWriteBlockSize		
Parent Container	EepInitConfiguration		
Description	Number of bytes written within one job processing cycle in fast mode. If the hardware does not support burst mode this parameter shall be set to the same value as Eep NormalWriteBlockSize. Implementation Type: Eep_LengthType.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		

▽

△

Default value	–		
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local dependency: This parameter is optional and only available if the hardware allows writing several bytes in one step (e.g. external EEPROMs with burst mode capability).		

SWS Item	**[ECUC_Eep_00171]**		
Parameter Name	EepJobEndNotification		
Parent Container	EepInitConfiguration		
Description	This parameter is a reference to a callback function for positive job result (see EEP045).		
Multiplicity	0..1		
Type	EcucFunctionNameDef		
Default value	–		
Regular Expression	–		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**[ECUC_Eep_00172]**		
Parameter Name	EepJobErrorNotification		
Parent Container	EepInitConfiguration		
Description	This parameter is a reference to a callback function for negative job result (see EEP046).		
Multiplicity	0..1		
Type	EcucFunctionNameDef		
Default value	–		
Regular Expression	–		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	[ECUC_Eep_00173]		
Parameter Name	EepNormalReadBlockSize		
Parent Container	EepInitConfiguration		
Description	Number of bytes read within one job processing cycle in normal mode. Implementation Type: Eep_LengthType.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	–		
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	[ECUC_Eep_00174]		
Parameter Name	EepNormalWriteBlockSize		
Parent Container	EepInitConfiguration		
Description	Number of bytes written within one job processing cycle in normal mode. Implementation Type: Eep_LengthType.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	–		
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local dependency: This parameter is optional and only available if the hardware allows configuration.		

SWS Item	[ECUC_Eep_00175]		
Parameter Name	EepSize		
Parent Container	EepInitConfiguration		
Description	This parameter is the used size of EEPROM device in bytes. Implementation Type: Eep_LengthType.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	–		
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

Included Containers		
Container Name	**Multiplicity**	**Scope / Dependency**
EepDemEventParameterRefs	0..1	Container for the references to DemEventParameter elements which shall be invoked using the API Dem_SetEventStatus in case the corresponding error occurs. The EventId is taken from the referenced DemEventParameter's DemEventId symbolic value. The standardized errors are provided in this container and can be extended by vendor-specific error references.
EepExternalDriver	0..1	This container is present for external EEPROM drivers only. Internal EEPROM drivers do not use the parameter listed in this container, hence its multiplicity is 0 for internal drivers.

10.2.4 EepDemEventParameterRefs

SWS Item	**[ECUC_Eep_00200]**
Container Name	EepDemEventParameterRefs
Parent Container	EepInitConfiguration
Description	Container for the references to DemEventParameter elements which shall be invoked using the API Dem_SetEventStatus in case the corresponding error occurs. The Event Id is taken from the referenced DemEventParameter's DemEventId symbolic value. The standardized errors are provided in this container and can be extended by vendor-specific error references.
Configuration Parameters	

SWS Item	**[ECUC_Eep_00204]**		
Parameter Name	EEP_E_COMPARE_FAILED		
Parent Container	EepDemEventParameterRefs		
Description	Reference to the DemEventParameter which shall be issued when the error "EEPROM compare failed (HW)" has occurred.		
Multiplicity	0..1		
Type	Symbolic name reference to DemEventParameter		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**[ECUC_Eep_00201]**
Parameter Name	EEP_E_ERASE_FAILED
Parent Container	EepDemEventParameterRefs
Description	Reference to the DemEventParameter which shall be issued when the error "EEPROM erase failed (HW)" has occurred.
Multiplicity	0..1
Type	Symbolic name reference to DemEventParameter

△

Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**[ECUC_Eep_00203]**		
Parameter Name	EEP_E_READ_FAILED		
Parent Container	EepDemEventParameterRefs		
Description	Reference to the DemEventParameter which shall be issued when the error "EEPROM read failed (HW)" has occurred.		
Multiplicity	0..1		
Type	Symbolic name reference to DemEventParameter		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**[ECUC_Eep_00202]**		
Parameter Name	EEP_E_WRITE_FAILED		
Parent Container	EepDemEventParameterRefs		
Description	Reference to the DemEventParameter which shall be issued when the error "EEPROM write failed (HW)" has occurred.		
Multiplicity	0..1		
Type	Symbolic name reference to DemEventParameter		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

No Included Containers

10.2.5 EepExternalDriver

SWS Item	[ECUC_Eep_00190]
Container Name	EepExternalDriver
Parent Container	EepInitConfiguration
Description	This container is present for external EEPROM drivers only. Internal EEPROM drivers do not use the parameter listed in this container, hence its multiplicity is 0 for internal drivers.
Configuration Parameters	

SWS Item	[ECUC_Eep_00176]		
Parameter Name	EepSpiReference		
Parent Container	EepExternalDriver		
Description	Reference to SPI sequence (required for external EEPROM drivers).		
Multiplicity	1..*		
Type	Symbolic name reference to SpiSequence		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

No Included Containers

10.2.6 SPI specific extension

[SWS_Eep_00094] ⌈In case of an external SPI EEPROM device, the following parameters shall also be located or referenced (according to the configuration methodology) in the external data structure of type `Eep_ConfigType` (see [ECUC_Eep_00039]). They shall be used as API parameters for accessing the SPI Handler/Driver API services. The symbolic names for those parameters are published in the module's description file (see [SWS_Eep_00095]).

- All required SPI channels
- All required SPI sequences
- All required SPI jobs

⌋(*SRS_BSW_00390*, *SRS_BSW_00398*)

10.3 Published parameters

10.3.1 Basic subset

For details refer to the chapter 10.3 “Published Information” in [6].

10.3.2 SPI specific extension

[SWS_Eep_00095] ⌈In case of an external SPI EEPROM device, the following parameters shall be published additionally in the module’s description file (see EEP038):

- All SPI channels that are required for EEPROM access (read, write, erase).
- Those channels shall be linked to construct SPI jobs that are linked with chip selected handling. This depends on the specific EEPROM device.
- Those jobs shall be assigned to SPI sequences to be scheduled for SPI transfer.

A complete list of required parameters is specified in the SPI Handler/Driver Software Specification.⌋*(SRS_BSW_00390, SRS_BSW_00402)*

10.3.3 EepPublishedInformation

SWS Item	[ECUC_Eep_00111]
Container Name	EepPublishedInformation
Parent Container	Eep
Description	Additional published parameters not covered by CommonPublishedInformation container. Note that these parameters do not have any configuration class setting, since they are published information.
Configuration Parameters	

SWS Item	[ECUC_Eep_00177]		
Parameter Name	EepAllowedWriteCycles		
Parent Container	EepPublishedInformation		
Description	Specified maximum number of write cycles under worst case conditions of specific EEPROM hardware (e.g. +90°C)		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	–		
Post-Build Variant Value	false		
Value Configuration Class	Published Information	X	All Variants
Scope / Dependency	scope: local		

SWS Item	[ECUC_Eep_00178]		
Parameter Name	EepEraseTime		
Parent Container	EepPublishedInformation		
Description	Maximum time for erasing one EEPROM data unit.		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	[0 .. INF]		
Default value	–		
Post-Build Variant Value	false		
Value Configuration Class	Published Information	X	All Variants
Scope / Dependency	scope: local		

SWS Item	[ECUC_Eep_00179]		
Parameter Name	EepEraseUnitSize		
Parent Container	EepPublishedInformation		
Description	Size of smallest erasable EEPROM data unit in bytes.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	–		
Post-Build Variant Value	false		
Value Configuration Class	Published Information	X	All Variants
Scope / Dependency	scope: local		

SWS Item	[ECUC_Eep_00180]		
Parameter Name	EepEraseValue		
Parent Container	EepPublishedInformation		
Description	Value of an erased EEPROM cell.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 255		
Default value	–		
Post-Build Variant Value	false		
Value Configuration Class	Published Information	X	All Variants
Scope / Dependency	scope: local		

SWS Item	[ECUC_Eep_00181]		
Parameter Name	EepMinimumAddressType		
Parent Container	EepPublishedInformation		
Description	Minimum expected size of Eep_AddressType.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	–		
Post-Build Variant Value	false		
Value Configuration Class	Published Information	X	All Variants
Scope / Dependency	scope: local		

SWS Item	[ECUC_Eep_00182]		
Parameter Name	EepMinimumLengthType		
Parent Container	EepPublishedInformation		
Description	Minimum expected size of Eep_LengthType.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	–		
Post-Build Variant Value	false		
Value Configuration Class	Published Information	X	All Variants
Scope / Dependency	scope: local		

SWS Item	[ECUC_Eep_00183]		
Parameter Name	EepReadUnitSize		
Parent Container	EepPublishedInformation		
Description	Size of smallest readable EEPROM data unit in bytes.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	–		
Post-Build Variant Value	false		
Value Configuration Class	Published Information	X	All Variants
Scope / Dependency	scope: local		

SWS Item	[ECUC_Eep_00187]		
Parameter Name	EepSpecifiedEraseCycles		
Parent Container	EepPublishedInformation		
Description	Number of erase cycles specified for the EEP device (usually given in the device data sheet).		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	–		
Post-Build Variant Value	false		
Value Configuration Class	Published Information	X	All Variants
Scope / Dependency	scope: local		

SWS Item	[ECUC_Eep_00184]		
Parameter Name	EepTotalSize		
Parent Container	EepPublishedInformation		
Description	Total size of EEPROM in bytes. Implementation Type: Eep_LengthType.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	–		

▽

△

Post-Build Variant Value	false		
Value Configuration Class	**Published Information**	X	All Variants
Scope / Dependency	scope: local		

SWS Item	**[ECUC_Eep_00185]**		
Parameter Name	EepWriteTime		
Parent Container	EepPublishedInformation		
Description	Maximum time for writing one EEPROM data unit.		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	[0 .. INF]		
Default value	–		
Post-Build Variant Value	false		
Value Configuration Class	**Published Information**	X	All Variants
Scope / Dependency	scope: local		

SWS Item	**[ECUC_Eep_00186]**		
Parameter Name	EepWriteUnitSize		
Parent Container	EepPublishedInformation		
Description	Size of smallest writeable EEPROM data unit in bytes.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	–		
Post-Build Variant Value	false		
Value Configuration Class	**Published Information**	X	All Variants
Scope / Dependency	scope: local		

No Included Containers

10.4 Configuration example – external SPI EEPROM device

The following chapter shall provide a better understanding of how and where configuration parameters are defined and used. For the following use case a detailed implementation and configuration example is given:

Use case

- Implement and configure a driver for operating an external EEPROM device accessed over SPI.
- Use the AUTOSAR SPI Handler/Driver, utilizing internal buffers (IB) for command communication and external buffers (EB) for data.
- Configure and perform an SPI read command.

The example assumes a certain fixed format and order of SPI commands to read from the external EEPROM device. The SPI API functions have been chosen for operating this exemplary device in order to demonstrate the basic principles of SPI bus interaction. When implementing a driver for a real-life device, the sequence of operation will most likely differ. The detailed selection of SPI API functions and parameters to be used and configured needs to be derived from studying the device's data sheet in combination with the SPI handler/driver specification [9].

Be aware that the use of the SPI API functions is exemplary; their exact signatures and configuration may change. The valid reference is always the current SPI SWS.

10.4.1 External SPI EEPROM device usage scenario

The following scenario is assumed in this example:

The external EEPROM device is an SPI slave device, the EEPROM driver to be implemented uses the SPI handler/driver module for the SPI master. The external device is addressed by a dedicated Chip Select line which will be asserted by the SPI master whenever a job operating on the device is being executed.

The external EEPROM uses serial op-code processing: After the device is selected with its Chip Select line going low, the first byte will be transmitted over the device's SI line. This byte contains an 8-bit Read-operation op-code (0x03), immediately followed by an 8-bit address byte. Upon completion, any data on the SI line will be ignored. The data (D7-D0) at the specified address is then shifted out onto the SO line. If only one byte is to be read, the CS line shall be driven high after the data comes out, otherwise the read sequence will be continued, with the address being automatically incremented and data shifted out on consecutive data.

Whenever the EEPROM driver's user wants to read data, the EEPROM driver forwards the read request to the SPI handler/driver via a number of selected SPI API calls. In order to follow the request/response behavior described above, the SPI needs to be configured exactly to fit the expected communication protocol. Therefore, an important development task consists in correctly configuring the SPI driver for communication with the external EEPROM device. Based on this configuration, the actual implementation of the EEPROM driver uses the SPI API functions in combination with the configured handle IDs for assigning jobs to the SPI handler/driver:

The EEPROM driver implementation may use a combination of external and internal SPI buffers for achieving the communication with the SPI handler:

Upon reception of an `Eep_Read()` request, the EEPROM driver writes the EEPROM source address in an SPI-channel internal buffer using `Spi_WriteIB()`. Next, it sets up an SPI external buffer specifying the requested number of bytes to be read using `Spi_SetupEB()`. It then calls `Spi_AsyncTransmit()` in order to initiate an SPI sequence EepReadSequence configured to match exactly the hardware access protocol outlined above.

Once the SPI read sequence has finished, the SPI handler/driver notifies the EEPROM driver by calling `Spi_SeqEndNotification`. The driver can now safely access the EEPROM data through the assigned external buffer and in turn finish the EEPROM read job.

10.4.2 Configuration of SPI parameters

In order to use the SPI handler/driver, the EEPROM driver implementer needs to create an SPI configuration, containing a complete set of SPI configuration containers such that the required functionality is configured.

Following a top-down view, an `SpiSequence` *EepReadSequence* configuration container handles one complete read sequence. *EepReadSequence* in turn uses an `SpiJob` *EepReadJob* for handling the details of a read job. This includes a reference to an `SpiExternalDevice` representing the EEPROM device with its specified Chip Select line as well as logic level characteristics like e.g. Baud Rate, Polarity or DataShiftEdge.

EepReadJob is further broken down into an ordered list of `SpiChannel`s which when executed in order will perform the required SPI bus communication with the external device:

1. *EepChCommand* is used for sending the ReadCommand byte, using a default data constant for the read op-code.
2. *EepChAddress* is used for sending the device read address utilizing an internal buffer.
3. *EepChReadData* is used for reading the requested EEPROM data into an externally (to SPI) provided buffer.

Roughly, the work flow of configuring the SPI module for an EEPROM read command contains the following steps:

1. In the EcuConfiguration for Spi, create a container *EepDriver* of type `SpiDriver` representing the external EEPROM driver. It will hold sub containers of type `SpiExternalDevice`, `SpiChannel`, `SpiJob` and `SpiSequence` to be created in the steps below.
2. Look up the external device's SPI characteristics in its data sheet and set up a container *EepDevice* of type `SpiExternalDevice` accordingly. Specify the Chip Select line to be used in *EepDevice*.
3. Look up the details of the SPI read command sequence in the device's data sheet.
4. Within *EepDriver*, define one `SpiChannel` each for transmitting the Read command opcode, the EEPROM source address and for receiving the data transmitted by the device in response to the request, e.g.
    - *EepChCommand*

- *EepChAddress*
- *EepChReadData*

5. Define SPI Channel attributes for each channel based on the communication sequence described in the device data sheet. In particular, configure buffers, i.e *EepChAddress* to use an internal buffer and *EepChReadData* to use an external buffer. For the fixed read-command opcode, `SpiDefaultData` can be used.
6. Define the `SpiJob` *EepReadJob* and set it up to work on *EepDevice*. Specify the ordered list of `SpiJob`s to be executed for performing the read job. In this example, the job consists of the channel list *EepChCommand*, *EepChAddress*, *EepChReadData*.
7. Define the `SpiSequence` *EepReadSequence* containing the list of `SpiJob`s required to perform the desired functionality. In this example, *EepReadSequence* contains only one job, *EepReadJob*. Fill in the callback function symbols to be provided by the EEPROM driver, e.g. *Eep_ReadSequenceEndNotification*.
8. Publish all defined attributes for SPI usage in the EEPROM driver as an XML description file according to SPI SWS.

10.4.3 Generation of SPI configuration data

As part of the SPI configuration described above, each `SpiSequence`, `SpiJob` and `SpiChannel` has been assigned a handle ID. Based on the XML file, an SPI include file will be generated which publishes this information.

```
#define Spi_EepReadSequence 10
#define Spi_EepReadJob 20
#define Spi_EepChCommand 31
#define Spi_EepChAddress 32
#define Spi_EepChReadData 33
```

10.4.4 SPI API usage

Upon receiving an `Eep_Read()` request, the EEPROM driver first needs to transfer the necessary information for executing the read command to the SPI handler/driver. It uses the `Spi_WriteIB()` function to set the device read address in the internal buffer allocated to the *EepChAddress* channel:

```
Spi_WriteIB(Spi_EepChAddress, &EepromAddress);
```

Next, the external buffer is set up for reading the EEPROM device data to:

```
Spi_SetupEB(Spi_EepChReadData, NULL, buf_data, length);
```

Finally, the Read sequence is initiated by calling `Spi_AsyncTransmit`:

```
Spi_AsyncTransmit(Spi_EepReadSequence);
```

After initiating the transfer, `Eep_Read()` returns.

The rest of the transfer is autonomously handled by the SPI handler/driver. Once the SPI sequence has finished, the SPI handler will notify the EEPROM driver using the callback `Spi_SeqEndNotification`. The EEPROM driver main function should ensure that either the sequence has finished successfully and in turn finish up the `Eep_Read()` request accordingly by signaling `EepJobEndNotification`; or upon reception of an error it should trigger an `EepJobErrorNotification` and report an `EEP_E_READ_FAILED` production error to the DEM.

A Not applicable requirements

[SWS_Eep_NA_00241] ⌈These requirements are not applicable to this specification.⌋*(SRS_BSW_00170, SRS_BSW_00399, SRS_BSW_00400, SRS_BSW_00375, SRS_BSW_00416, SRS_BSW_00168, SRS_BSW_00423, SRS_BSW_00424, SRS_BSW_00426, SRS_BSW_00427, SRS_BSW_00428, SRS_BSW_00429, SRS_BSW_00432, SRS_BSW_00433, SRS_BSW_00336, SRS_BSW_00422, SRS_BSW_00417, SRS_BSW_00161, SRS_BSW_00162, SRS_BSW_00005, SRS_BSW_00164, SRS_BSW_00325, SRS_BSW_00342, SRS_BSW_00343, SRS_BSW_00007, SRS_BSW_00413, SRS_BSW_00347, SRS_BSW_00307, SRS_BSW_00301, SRS_BSW_00302, SRS_BSW_00328, SRS_BSW_00312, SRS_BSW_00006, SRS_BSW_00378, SRS_BSW_00306, SRS_BSW_00308, SRS_BSW_00309, SRS_BSW_00330, SRS_BSW_00331, SRS_BSW_00009, SRS_BSW_00401, SRS_BSW_00172, SRS_BSW_00010, SRS_BSW_00341, SRS_BSW_00334, SRS_SPAL_12267, SRS_SPAL_12163, SRS_SPAL_12068, SRS_SPAL_12069, SRS_SPAL_12063, SRS_SPAL_12129, SRS_SPAL_12067, SRS_SPAL_12077, SRS_SPAL_12078, SRS_SPAL_12092, SRS_SPAL_12265)*

B Change history of AUTOSAR traceable items

B.1 Traceable item history of this document according to AUTOSAR Release R23-11

B.1.1 Added Specification Items in R23-11

Number	Heading
[SWS_Eep_00247]	

Table B.1: Added Specification Items in R23-11

B.1.2 Changed Specification Items in R23-11

Number	Heading
[SWS_Eep_00138]	Definition of imported datatypes of module Eep
[SWS_Eep_00145]	Definition of API function Eep_Read
[SWS_Eep_00146]	Definition of API function Eep_Write
[SWS_Eep_00147]	Definition of API function Eep_Erase
[SWS_Eep_00148]	Definition of API function Eep_Compare
[SWS_Eep_00150]	Definition of API function Eep_GetStatus
[SWS_Eep_00151]	Definition of API function Eep_GetJobResult
[SWS_Eep_00154]	Definition of mandatory interfaces in module Eep
[SWS_Eep_00155]	Definition of optional interfaces in module Eep
[SWS_Eep_00225]	Definition of datatype Eep_ConfigType
[SWS_Eep_00226]	Definition of datatype Eep_AddressType
[SWS_Eep_00227]	Definition of datatype Eep_LengthType
[SWS_Eep_00242]	
[SWS_Eep_00244]	

Table B.2: Changed Specification Items in R23-11

B.1.3 Deleted Specification Items in R23-11

none