Document Title	Specification of EEPROM Driver
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	021
Document Classification	Standard

Document Version	3.2.0
Document Status	Final
Part of Release	4.0
Revision	3

Document Change History

Date	Version	Changed by	Change Description
02.11.2011	3.2.0	AUTOSAR Administration	• Min max values of FloatParamDef parameters added for EEP178 & EEP185 • Replaced Module short name by module abbreviation
15.10.2010	3.1.0	AUTOSAR Administration	• Added DET errors EEP_E_PARAM_POINTER, EEP_E_TIMEOUT • Version check section (section 7.10) modified
30.11.2009	3.0.0	AUTOSAR Administration	• Made hidden text visible in EEP003, EEP030, EEP128 • 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
23.06.2008	2.2.1	AUTOSAR Administration	Legal disclaimer revised

12.12.2007	2.2.0	AUTOSAR Administration	• Minor rewording of requirement (EEP005). • Introduction of new requirements (EEP161 and EEP162) for NULL_PTR check. • Updates to EEP028 and Figure 4 to correct spelling of MEMIF_JOB_CANCELLED • Document meta information extended • Small layout adaptations made
31.01.2007	2.1.0	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
25.04.2006	2.0.0	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
30.06.2005	1.0.0	AUTOSAR Administration	Initial Release

Disclaimer

Advice for users

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

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

Table of Contents

1 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 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.

Abbreviation:	***Description:***
EEPROM	Electrically Erasable and Programmable Read Only Memory
NVRAM	Non Volatile Random Access Memory
NvM	Module name of NVRAM Manager
EcuM	Module name of ECU State Manager
DEM	Module name of Diagnostic Event Manager
DET	Module name of Development Error Tracer

3 Related documentation

3.1 Input documents

[1] Layered Software Architecture
AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf

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

[3] Specification of Memory Abstraction Interface
AUTOSAR_SWS_MemoryAbstractionInterface.pdf

[4] Specification of SPI Handler/Driver
AUTOSAR_SWS_SPIHandlerDriver.pdf

[5] Specification of ECU Configuration
AUTOSAR_TPS_ECUConfiguration.pdf

[6] Requirements on EEPROM Driver
AUTOSAR_SRS_EEPROMDriver.pdf

[7] Specification of Development Error Tracer
AUTOSAR_SWS_DevelopmentErrorTracer.pdf

[8] Specification of Diagnostics Event Manager
AUTOSAR_SWS_DiagnosticEventManager.pdf

[9] AUTOSAR Glossary
AUTOSAR_TR_Glossary.pdf

[10] Specification of MCU Driver
AUTOSAR_SWS_MCUDriver.pdf

[11] Basic Software Module Description Template
AUTOSAR_TPS_BSWModuleDescriptionTemplate.pdf

[12] List of Basic Software Modules
AUTOSAR_TR_BSWModuleList.pdf

3.2 Related standards and norms

[13] HIS Specification I/O Drivers, V2.1.3

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.
2. EEPROM drivers for external EEPROM devices.
   These are part of the ECU Abstraction Layer.

[EEP082] ⌈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 [10].

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

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

5.1 File structure

[EEP159] ⌈The module Eep shall provide a file `Eep_Lcfg.c` containing all link time configurable parameters.⌋ ()

[EEP160] ⌈The module Eep shall provide a file `Eep_PBcfg.c` containing all post build time configurable parameters.⌋ ()

[EEP228] ⌈If the module implementation uses custom interrupt processing, the interrupt service routines shall be placed in `Eep_Irq.c`⌋ ()

[EEP083] ⌈The module Eep shall adhere to the following include file structure:

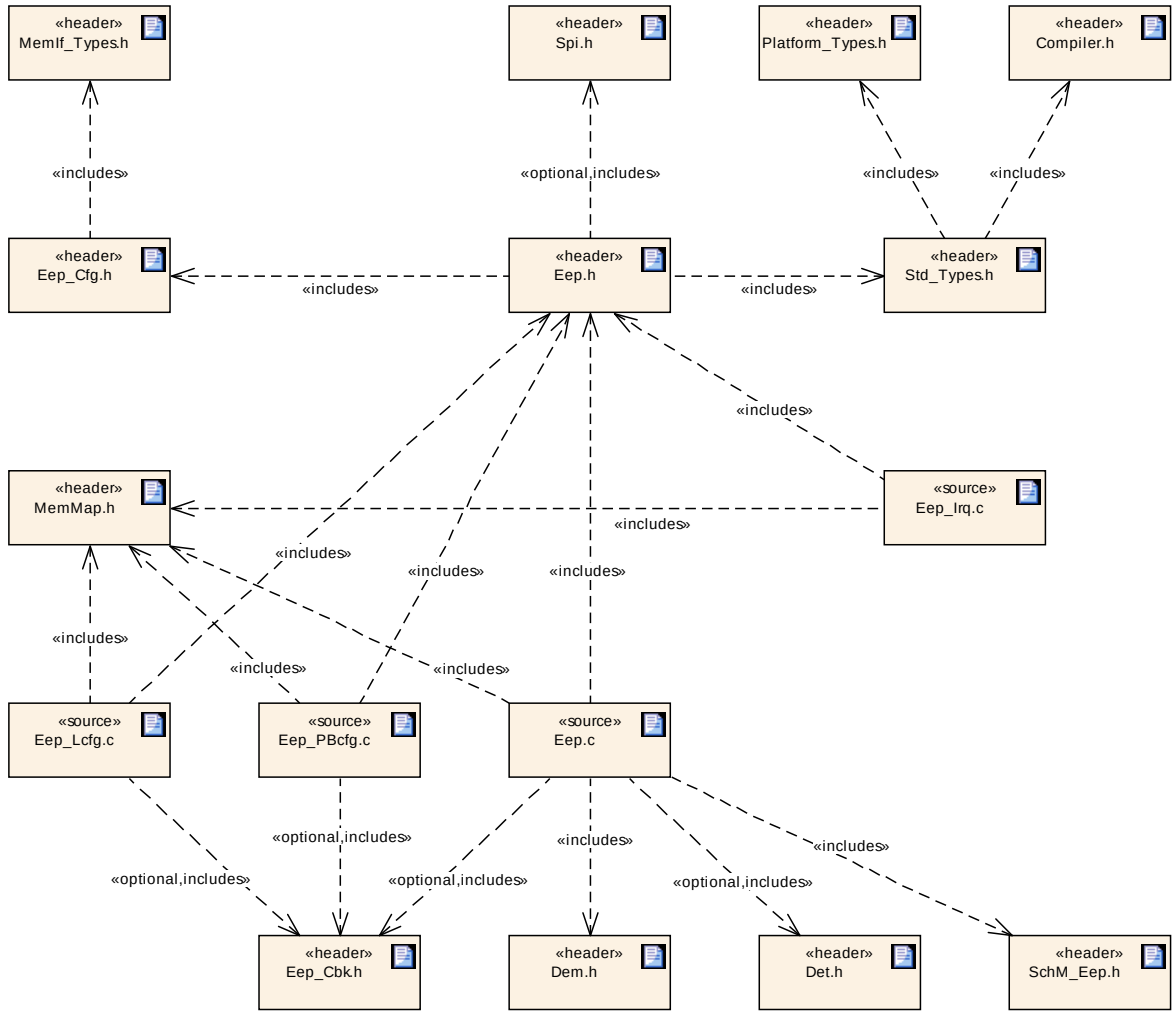

Figure 1

⌋(BSW00412, BSW00415)

[EEP102] ⌈The Eep module shall include `Eep.h`, `MemMap.h`, `Dem.h` and `SchM_Eep.h`. It shall optionally include `Det.h` and `Eep_Cbk.h`

By inclusion of Dem.h the APIs to report errors as well as the required Event Id symbols are included. This specification defines the name of the Event Id symbols which are provided by XML to the DEM configuration tool. The DEM configuration tool assigns ECU dependent values to the Event Id symbols and publishes the symbols in Dem_IntErrId.h. ⌋(BSW00384)

[EEP229] ⌈`Eep.h` shall include `Eep_Cfg.h` and `Std_Types.h`⌋()

[EEP230] ⌈In case of a driver for an external SPI EEPROM, `Eep.h` shall include `Spi.h`⌋()

[EEP231] ⌈If present, `Eep_Irq.c` shall include `Eep.h` and `MemMap.h`⌋()

[EEP232] ⌈If present, `Eep_Lcfg.c` shall include `Eep.h` and `MemMap.h`. It shall optionally include `Eep_Cbk.h`⌋ ()

[EEP233] ⌈If present, `Eep_PBcfg.c` shall include `Eep.h` and `MemMap.h`. It shall optionally include `Eep_Cbk.h`⌋ ()

6 Requirements traceability

Requirement	Satisfied by
-	EEP059
-	EEP235
-	EEP234
-	EEP119
-	EEP145
-	EEP147
-	EEP128
-	EEP153
-	EEP236
-	EEP146
-	EEP206
-	EEP151
-	EEP133
-	EEP143
-	EEP228
-	EEP106
-	EEP154
-	EEP220
-	EEP148
-	EEP219
-	EEP120
-	EEP122
-	EEP136
-	EEP044
-	EEP117
-	EEP031
-	EEP160
-	EEP225
-	EEP129
-	EEP152
-	EEP137
-	EEP207
-	EEP158
-	EEP162
-	EEP134
-	EEP226
-	EEP144
-	EEP205

-	EEP135
-	EEP233
-	EEP161
-	EEP229
-	EEP056
-	EEP115
-	EEP058
-	EEP150
-	EEP221
-	EEP124
-	EEP082
-	EEP217
-	EEP098
-	EEP222
-	EEP157
-	EEP116
-	EEP159
-	EEP204
-	EEP237
-	EEP126
-	EEP084
-	EEP149
-	EEP100
-	EEP231
-	EEP224
-	EEP121
-	EEP127
-	EEP155
-	EEP239
-	EEP227
-	EEP104
-	EEP068
-	EEP230
-	EEP238
-	EEP232
-	EEP075
-	EEP118
-	EEP097
-	EEP113
-	EEP223
-	EEP123
BSW00301	EEP241

BSW00302	EEP241
BSW00306	EEP241
BSW00307	EEP241
BSW00308	EEP241
BSW00309	EEP241
BSW00312	EEP241
BSW00323	EEP016, EEP017, EEP018, EEP005
BSW00324	EEP241
BSW00325	EEP241
BSW00326	EEP241
BSW00328	EEP241
BSW00330	EEP241
BSW00331	EEP241
BSW00334	EEP241
BSW00335	EEP138
BSW00336	EEP241
BSW00337	EEP003, EEP000, EEP201, EEP200, EEP203, EEP202
BSW00338	EEP218, EEP001
BSW00339	EEP002
BSW00341	EEP241
BSW00342	EEP241
BSW00343	EEP241
BSW00347	EEP241
BSW00350	EEP218, EEP001
BSW00355	EEP241
BSW00357	EEP138
BSW00369	EEP218, EEP001, EEP033
BSW00375	EEP241
BSW00377	EEP138
BSW00378	EEP241
BSW00384	EEP102
BSW00385	EEP003, EEP000
BSW00386	EEP218, EEP001
BSW00399	EEP241
BSW004	EEP091
BSW00400	EEP241
BSW00401	EEP241
BSW00406	EEP006, EEP033
BSW00407	EEP108, EEP107
BSW00409	EEP103
BSW00411	EEP108
BSW00412	EEP083

BSW00413	EEP241
BSW00415	EEP083
BSW00416	EEP241
BSW00417	EEP241
BSW00420	EEP241
BSW00421	EEP002
BSW00422	EEP241
BSW00423	EEP241
BSW00424	EEP241
BSW00426	EEP241
BSW00427	EEP241
BSW00428	EEP241
BSW00429	EEP241
BSW00431	EEP241
BSW00432	EEP241
BSW00433	EEP241
BSW00434	EEP241
BSW00442	EEP210, EEP212, EEP211, EEP214, EEP213, EEP209, EEP208
BSW005	EEP241
BSW006	EEP241
BSW007	EEP241
BSW009	EEP241
BSW010	EEP241
BSW087	EEP013, EEP009
BSW088	EEP063, EEP014, EEP015, EEP090
BSW089	EEP070, EEP072, EEP019, EEP020
BSW090	EEP216, EEP021, EEP215, EEP028, EEP027
BSW091	EEP029
BSW092	EEP060, EEP064
BSW094	EEP063, EEP070, EEP072, EEP090
BSW095	EEP033, EEP036
BSW101	EEP004
BSW12047	EEP032, EEP030
BSW12050	EEP057, EEP051, EEP054, EEP069
BSW12051	EEP088
BSW12056	EEP047, EEP049
BSW12057	EEP004
BSW12062	EEP004
BSW12063	EEP241
BSW12064	EEP033
BSW12067	EEP241
BSW12068	EEP241

BSW12069	EEP241
BSW12072	EEP055, EEP054, EEP073
BSW12075	EEP037
BSW12077	EEP241
BSW12078	EEP241
BSW12091	EEP026, EEP025
BSW12092	EEP241
BSW12124	EEP055, EEP052, EEP053, EEP073
BSW12129	EEP241
BSW12156	EEP132, EEP042, EEP130
BSW12157	EEP051, EEP052, EEP053
BSW12163	EEP241
BSW12265	EEP241
BSW12267	EEP241
BSW12448	EEP016, EEP017, EEP018, EEP218, EEP001, EEP005, EEP033
BSW157	EEP045, EEP047, EEP046, EEP029, EEP024
BSW161	EEP241
BSW162	EEP241
BSW164	EEP241
BSW168	EEP241
BSW170	EEP241
BSW172	EEP241

7 Functional specification

7.1 General behavior

[EEP088] ⌈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). ⌋(BSW12051)

[EEP036] ⌈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. ⌋(BSW095)

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 EEP033.

[EEP037] ⌈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. ⌋(BSW12075)

7.2 Error classification

[EEP104] ⌈Development error values are of type `uint8`. ⌋()

[EEP103] ⌈The Eep module shall take the values for production code Event Ids out of the file Dem_IntErrId.h. The Eep module shall include the file Dem.h (which in its turn includes Dem_IntErrId.h). ⌋(BSW00409)

[EEP000] ⌈The Eep module shall detect the following errors depending on its build options (development/production mode):

Type or error	*Relevance*	*Related error code*	*Value [hex]*

API service called with wrong parameter	Development	`EEP_E_PARAM_CONFIG`
		`EEP_E_PARAM_ADDRESS`
		`EEP_E_PARAM_DATA`
		`EEP_E_PARAM_LENGTH`
			`0x10`
			`0x11`
			`0x12`
			`0x13`

API service called with a NULL pointer	Development	`EEP_E_PARAM_POINTER`	`0x23`
API service called without module initialization	Development	`EEP_E_UNINIT`	`0x20`
API service called while driver still busy	Development	`EEP_E_BUSY`	`0x21`
Timeout exceeded	Development	`EEP_E_TIMEOUT`	`0x22`
EEPROM erase failed (HW)	Production	`EEP_E_ERASE_FAILED`	Assigned by DEM

EEPROM write failed (HW)	Production	EEP_E_WRITE_FAILED	Assigned by DEM
EEPROM read failed (HW)	Production	EEP_E_READ_FAILED	Assigned by DEM
EEPROM compare failed (HW)	Production	EEP_E_COMPARE_FAILED	Assigned by DEM

⌋(BSW00337, BSW00385)

[EEP003] ⌈The Eep module shall add additional errors that are detected because of specific implementation and/or specific hardware properties to the EEPROM device specific implementation specification. The Eep module shall define the classification and enumeration of these additional error so that they are compatible to the errors defined in EEP000. ⌋(BSW00337, BSW00385)

7.3 Error detection

[EEP001] ⌈The detection of development errors is configurable (*ON / OFF*) at pre-compile time. The switch `EepDevErrorDetect` (see chapter 10) shall activate or deactivate the detection of all development errors. ⌋(BSW00338, BSW00369, BSW00386, BSW00350, BSW12448)

[EEP218] ⌈If the `EepDevErrorDetect` switch is enabled, API parameter checking is enabled. The detailed description of the detected errors can be found in chapter 7.2, chapter 7.3.1 and chapter 8. ⌋(BSW00338, BSW00369, BSW00386, BSW00350, BSW12448)

[EEP106] ⌈The detection of production code errors cannot be switched off. ⌋()

7.3.1 API parameter checking

[EEP016] ⌈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` and return with `E_NOT_OK`. ⌋(BSW00323, BSW12448)

[EEP017] ⌈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` and return with `E_NOT_OK`. ⌋(BSW00323, BSW12448)

[EEP018] ⌈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` and return with `E_NOT_OK`.⌋(BSW00323, BSW12448)

7.3.2 EEPROM state checking

[EEP033] ⌈If development error detection for the module Eep is enabled: 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 development error `EEP_E_BUSY` or `EEP_E_UNINIT` according to the EEPROM state
- reject the service with `E_NOT_OK` (except `Eep_SetMode()`because this service has no return value) ⌋(BSW00406, BSW00369, BSW12064, BSW12448, BSW095)

7.3.3 EEPROM job encounters Hardware Failure

[EEP200] ⌈The production error code `EEP_E_ERASE_FAILED` shall be reported when the EEPROM erase function failed. ⌋(BSW00337)

[EEP201] ⌈The production error code `EEP_E_WRITE_FAILED` shall be reported when the EEPROM write function failed. ⌋(BSW00337)

[EEP202] ⌈The production error code `EEP_E_READ_FAILED` shall be reported when the EEPROM read function failed. ⌋(BSW00337)

[EEP203] ⌈The production error code `EEP_E_COMPARE_FAILED` shall be reported when the EEPROM compare function failed. ⌋(BSW00337)

7.3.4 Timeout Supervision

[EEP234] ⌈The development 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

[EEP002] ⌈The Eep module shall report production errors to the Diagnostic Event Manager. ⌋ (BSW00339, BSW00421)

[EEP100] ⌈The Eep module shall report detected development errors to the *Det_ReportError* service of the Development Error Tracer (DET) if the pre-processor switch `EepDevErrorDetect` is set (see EEP188_Conf). ⌋ ()

7.5 Processing of jobs – general requirements

[EEP128] ⌈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 EEP163_Conf). ⌋ ()

[EEP129] ⌈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.

Additional general requirements only applicable for SPI EEPROM drivers:

[EEP056] ⌈For an Eep module driving an external EEPROM through SPI: If the SPI access fails, the Eep module shall behave as specified in EEP068. ⌋ ()

[EEP052] ⌈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. ⌋ (BSW12157, BSW12124)

[EEP053] ⌈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. ⌋ (BSW12157, BSW12124)

[EEP055] ⌈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. ⌋(BSW12072, BSW12124)

[EEP073] ⌈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. ⌋(BSW12072, BSW12124)

7.6 Processing of read jobs

[EEP130] ⌈The Eep module shall provide two different read modes:

- normal mode
- fast mode⌋(BSW12156)

[EEP132] ⌈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). ⌋(BSW12156)

[EEP051] ⌈In normal EEPROM mode, the Eep module shall read within one job processing cycle a number of bytes specified by the parameter `EepNormalReadBlockSize`. ⌋(BSW12157, BSW12050)

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

[EEP054] ⌈In fast EEPROM mode, the Eep module shall read within one job processing cycle a number of bytes specified by the parameter `EepFastReadBlockSize`. ⌋(BSW12072, BSW12050)

Example:

- `EepFastReadBlockSize` = 32
- Number of bytes to read: 110
- Required number of job processing cycles: 4
- Resulting read pattern: 32-32-32-14

[EEP058] ⌈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`. ⌋()

[EEP068] ⌈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

[EEP057] ⌈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. ⌋ (BSW12050)

[EEP133] ⌈The Eep module shall provide two different write modes:

- normal mode
- fast mode⌋ ()

[EEP134] ⌈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). ⌋ ()

[EEP097] ⌈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

[EEP098] ⌈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

[EEP060] ⌈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. ⌋ (BSW092)

[EEP059] ⌈The Eep module shall erase an EEPROM cell before writing to it if this is not done automatically by the EEPROM hardware. ⌋ ()

[EEP063] ⌈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. ⌋ (BSW088, BSW094)

[EEP090] ⌈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. ⌋ (BSW088, BSW094)

[EEP064] ⌈The Eep module shall keep the number of read – modify – write operations during writing a data block as small as possible. ⌋ (BSW092)

[EEP219] ⌈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`. ⌋ ()

[EEP222] ⌈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

[EEP069] ⌈The Eep module shall erase only as many bytes to the EEPROM as supported by the EEPROM hardware within one job processing cycle.⌋ (BSW12050)

[EEP070] ⌈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.⌋ (BSW089, BSW094)

[EEP072] ⌈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.⌋ (BSW089, BSW094)

[EEP220] ⌈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`.⌋ ()

[EEP223] ⌈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: EEP130, EEP132, EEP051, EEP054.

[EEP221] ⌈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`.⌋ ()

[EEP224] ⌈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`.⌋ ()

[EEP075] ⌈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: EEP052, EEP053, EEP055, EEP073.

7.10 Version check

[EEP091] ⌈The Eep module shall perform Inter Module Checks to avoid integration of incompatible files.
The imported included files shall be checked by preprocessing directives.

The following version numbers shall be verified:
- <MODULENAME>_AR_RELEASE_MAJOR_VERSION
- <MODULENAME>_AR_RELEASE_MINOR_VERSION
Where <MODULENAME> is the module abbreviation of the other (external) modules which provide header files included by the Eep module.

If the values are not identical to the expected values, an error shall be reported. ⌋ (BSW004)

7.11 Support for Debugging

[EEP208] ⌈Each variable that shall be accessible by AUTOSAR Debugging shall be defined as global variable. ⌋(BSW00442)

[EEP209] ⌈All type definitions of variables which shall be debugged shall be accessible by the header file `Eep.h`. ⌋(BSW00442)

[EEP210] ⌈The declaration of variables in the header file shall be such that it is possible to calculate the size of the variables by C-“sizeof”. ⌋(BSW00442)

[EEP211] ⌈Variables available for debugging shall be described in the respective Basic Software Module Description. ⌋(BSW00442)

[EEP212] ⌈The EEPROM module state shall be available for debugging. ⌋ (BSW00442)

[EEP213] ⌈The job result shall be available for debugging. ⌋(BSW00442)

[EEP214] ⌈The EEPROM operation mode shall be available for debugging. ⌋ (BSW00442)

8 API specification

8.1 Imported types

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

[EEP138] ⌈

Module	*Imported Type*
Dem	Dem_EventIdType
	Dem_EventStatusType
MemIf	MemIf_JobResultType
	MemIf_ModeType
	MemIf_StatusType
Std_Types	Std_ReturnType
	Std_VersionInfoType

⌋ (BSW00335, BSW00357, BSW00377)

8.2 Type definitions

8.2.1 Eep_ConfigType

[EEP225] ⌈

Name:	`Eep_ConfigType`	
Type:	`Structure`	
Range:	`Implementation Specific`	The contents of the initialisation data structure are EEPROM specific.
Description:	This is the type of the external data structure containing the initialization data for the EEPROM driver.	

⌋ ()

8.2.2 Eep_AddressType

[EEP226] ⌈

Name:	`Eep_AddressType`		
Type:	`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.		

⌋ ()

[EEP113] ⌈The type Eep_AddressType shall have 0 as lower limit for each EEPROM device. ⌋ ()

[EEP217] ⌈The EEPROM module shall add a device specific base address to the address type Eep_AddressType if necessary. ⌋ ()

8.2.3 Eep_LengthType

[EEP227] ⌈

Name:	`Eep_LengthType`		
Type:	`uint`		
Range:	`Same as Eep_AddressType`	- -	Is the same type as Eep_AddressType because of arithmetic operations. Size depends on target platform and EEPROM device.
Description:	Specifies the number of bytes to read/write/erase/compare.		

⌋ ()

8.3 Function definitions

8.3.1 Eep_Init

[EEP143] ⌈

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.	

⌋ ()

[EEP004] ⌈The function Eep_Init shall initialize all EEPROM relevant registers with the values of the structure referenced by the parameter `ConfigPtr.` ⌋ (BSW101, BSW12057, BSW12062)

[EEP005] ⌈If development error detection for the module Eep is enabled; if the function Eep_Init is called with a NULL configPtr and if a variant containing postbuild multiple selectable configuration parameters is used (VariantPB), the function Eep_Init shall raise the development error `EEP_E_PARAM_CONFIG` and return without any action. ⌋ (BSW00323, BSW12448)

[EEP161] ⌈For variants with no postbuild multiple selectable configuration parameters (Variant PC), the EEP module's environment shall pass a `NULL` pointer to the function Eep_Init().⌋ ()

[EEP162] ⌈The initialization function of this module shall always have a pointer as a parameter, even though for Variant PC no configuration set shall be given. Instead a NULL pointer shall be passed to the initialization function. ⌋ ()

[EEP006] ⌈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.` ⌋ (BSW00406)

[EEP044] ⌈The function Eep_Init shall set the EEPROM mode to the configured default mode⌋ ()

[EEP115] ⌈The Eep's user shall not call the function Eep_Init during a running operation. ⌋ ()

8.3.2 Eep_SetMode

[EEP144] ⌈

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.	

⌋ ()

[EEP042] ⌈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 EEP033. ⌋ (BSW12156)

[EEP116] ⌈The Eep's user shall not call the function Eep_SetMode during a running operation. ⌋ ()

8.3.3 Eep_Read

[EEP145] ⌈

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 - EepromAddress
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.	

⌋ ()

[EEP009] ⌈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. ⌋ (BSW087)

[EEP013] ⌈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 EEP016, EEP017, EEP018.

The function Eep_Read checks the EEPROM state according to requirement EEP033. ⌋ (BSW087)

[EEP117] ⌈The Eep's user shall only call Eep_Read after the Eep module has been been initialized. ⌋ ()

[EEP118] ⌈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

[EEP146] ⌈

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 - EepromAddress
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.	

⌋()

[EEP014] ⌈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. ⌋(BSW088)

[EEP015] ⌈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 EEP016, EEP017, EEP018.

The function Eep_Write checks the EEPROM state according to requirement EEP033. ⌋(BSW088)

[EEP119] ⌈The Eep module's user shall only call the function Eep_Write after the Eep module has been initialized. ⌋()

[EEP120] ⌈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

[EEP147] ⌈

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 - EepromAddress
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.	

⌋()

[EEP019] ⌈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. ⌋(BSW089)

[EEP020] ⌈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 EEP016, EEP017, EEP018.

The function Eep_Erase checks the EEPROM state according to requirement EEP033. ⌋(BSW089)

[EEP121] ⌈The Eep module's user shall only call the function Eep_Erase after the Eep module has been initialized. ⌋()

[EEP122] ⌈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

[EEP148] ⌈

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 - EepromAddress
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.	

⌋()

[EEP025] ⌈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. ⌋(BSW12091)

[EEP026] ⌈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 EEP016, EEP017, EEP018.

The service Eep_Compare checks the EEPROM state according to requirement EEP033. ⌋(BSW12091)

[EEP123] ⌈The Eep module's user shall only call the function Eep_Compare after the Eep module has been initialized. ⌋()

[EEP124] ⌈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

[EEP149] ⌈

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.

⌋()

[EEP215] ⌈The function `Eep_Cancel` shall cancel an ongoing EEPROM read, write, erase or compare job. ⌋(BSW090)

[EEP021] ⌈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. ⌋(BSW090)

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).

[EEP027] ⌈The function `Eep_Cancel` shall set the EEP module state to `MEMIF_IDLE`. ⌋(BSW090)

[EEP216] ⌈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. ⌋(BSW090)

[EEP028] ⌈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. ⌋(BSW090)

[EEP136] ⌈The Eep module's user shall not call the `Eep_Cancel()` function during a running `Eep_MainFunction()` function.

EEP136 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

[EEP150] ⌈

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.	

⌋()

[EEP029] ⌈The function Eep_GetStatus shall return the EEPROM status synchronously. ⌋(BSW157, BSW091)

8.3.9 Eep_GetJobResult

[EEP151] ⌈

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.	

⌋()

[EEP024] ⌈The function Eep_GetJobResult shall synchronously return the result of the last job that has been accepted by the Eep module. ⌋(BSW157)

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

[EEP152] ⌈

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.	

⌋()

[EEP239] ⌈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` and return without any action. ⌋()

[EEP107] ⌈The function Eep_GetVersionInfo shall return the version information of this module. The version information includes:

- Module Id
- Vendor Id
- Vendor specific version numbers (BSW00407). ⌋(BSW00407)

[EEP135] ⌈If source code for caller and callee of the function Eep_GetVersionInfo is available, the Eep module should realize this function as a macro. The Eep module should define this macro in the module's header file. ⌋()

[EEP108] ⌈The function Eep_GetVersionInfo shall be pre compile time configurable `On/Off` by the configuration parameter: `EepVersionInfoApi.` ⌋ (BSW00407, BSW00411)

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.

[EEP137] ⌈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

[EEP153] ⌈

Service name:	Eep_MainFunction

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

Service ID[hex]:	0x09
Timing:	FIXED_CYCLIC
Description:	Service to perform the processing of the EEPROM jobs (read/write/erase/compare) .

⌋ ()

[EEP030] ⌈The function `Eep_MainFunction` shall perform the processing of the EEPROM read, write, erase and compare jobs. ⌋ (BSW12047)

[EEP031] ⌈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.

[EEP084] ⌈The configuration parameter EepJobCallCycle (see EEP170_Conf) 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.⌋ ()

[EEP032] ⌈The function Eep_MainFunction shall return without action if no job is pending.⌋ (BSW12047)

[EEP204] ⌈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.⌋ ()

[EEP205] ⌈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.⌋ ()

[EEP206] ⌈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.⌋ ()

[EEP207] ⌈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.⌋ ()

[EEP235] ⌈If development error detection for the module Eep is enabled, 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.⌋ ()

[EEP236] ⌈If development error detection for the module Eep is enabled, the function Eep_MainFunction shall check whether the configured maximum erase time (see EEP178_Conf EepEraseTime) has been exceeded. If this is the case, the function Eep_MainFunction shall raise the development error EEP_E_TIMEOUT.⌋ ()

[EEP237] ⌈If development error detection for the module Eep is enabled, 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 development error EEP_E_TIMEOUT.⌋ ()

Note: The expected maximum write time depends on the current mode of the Eep module (see EEP144), the configured number of bytes to write in this mode (see EEP174_Conf and EEP169_Conf respectively), the size of a EEPROM write data unit (see EEP186_Conf) and last the maximum time to write one data unit (see EEP185_Conf). 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.

[EEP238] ⌈If development error detection for the module Eep is enabled, 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 development 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.

[EEP154] ⌈

API function	*Description*
Dem_ReportErrorStatus	Queues the reported events from the BSW modules (API is only used by BSW modules). The interface has an asynchronous behavior, because the processing of the event is done within the Dem main function.

⌋()

8.6.2 Optional Interfaces

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

[EEP155] ⌈

API function	*Description*
Det_ReportError	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.

[EEP047] ⌈If a callback function is being configured at post build time, the initialization data structure Eep_ConfigType shall contain a corresponding function pointer. ⌋(BSW12056, BSW157)

[EEP049] ⌈Notification callback functions are configurable through their corresponding configuration parameters. If no callback function is configured, there shall be no asynchronous notification. ⌋(BSW12056)

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

[EEP045] ⌈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⌋(BSW157)

[EEP157] ⌈

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.

⌋()

[EEP126] ⌈The callback function defined in the configuration parameter EepJobEndNotification shall be callable on interrupt level. ⌋()

8.6.3.2 Error Job Notification

[EEP046] ⌈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. ⌋(BSW157)

[EEP158] ⌈

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.

⌋()

[EEP127] ⌈The callback function defined in the configuration parameter EepJobErrorNotification shall be callable on interrupt level. ⌋()

9 Sequence diagrams

9.1 Initialization

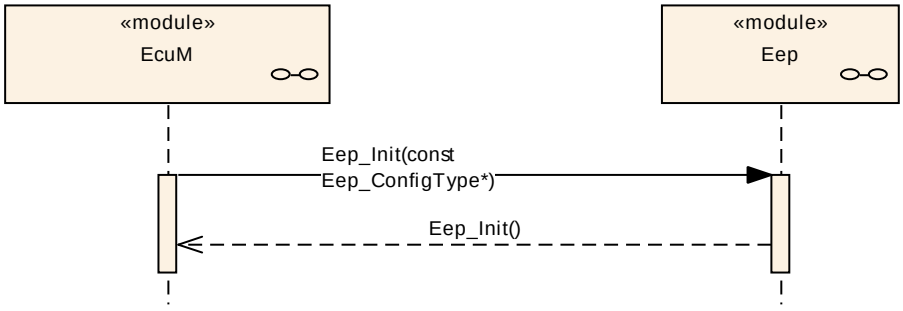

Figure 2

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.

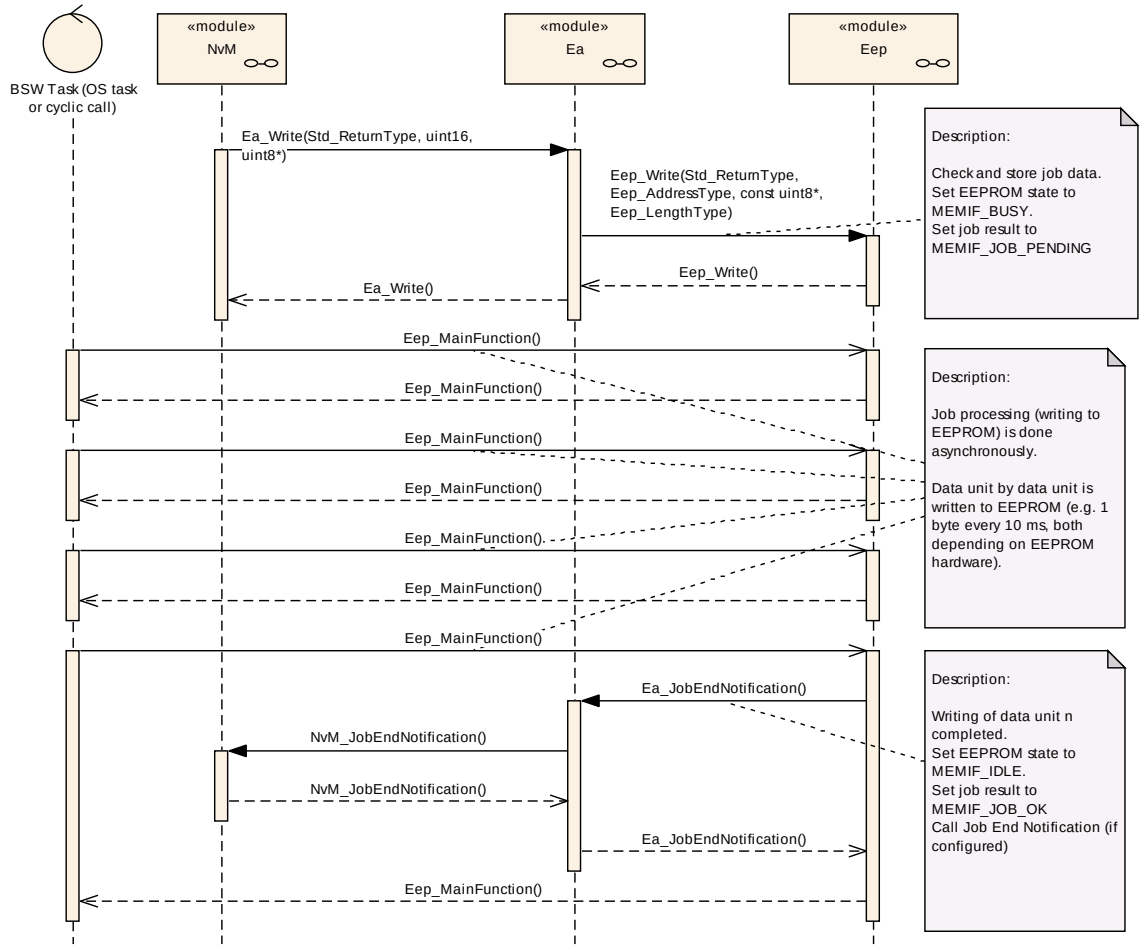
«module»
NvM
«module»
Ea
«module»
Eep
BSW Task (OS task or cyclic call)
Ea_Write(Std_ReturnType, uint16, uint8*)
Eep_Write(Std_ReturnType, Eep_AddressType, const uint8*, Eep_LengthType)
Eep_Write()
Ea_Write()
Eep_MainFunction()
Eep_MainFunction()
Eep_MainFunction()
Eep_MainFunction()
Eep_MainFunction()
Eep_MainFunction()
Eep_MainFunction()
Ea_JobEndNotification()
NvM_JobEndNotification()
NvM_JobEndNotification()
Ea_JobEndNotification()
Eep_MainFunction()
Description:
Check and store job data.
Set EEPROM state to MEMIF_BUSY.
Set job result to MEMIF_JOB_PENDING
Description:
Job processing (writing to EEPROM) is done asynchronously.
Data unit by data unit is written to EEPROM (e.g. 1 byte every 10 ms, both depending on EEPROM hardware).
Description:
Writing of data unit n completed.
Set EEPROM state to MEMIF_IDLE.
Set job result to MEMIF_JOB_OK
Call Job End Notification (if configured)

Figure 3

9.3 Cancelation of a running job

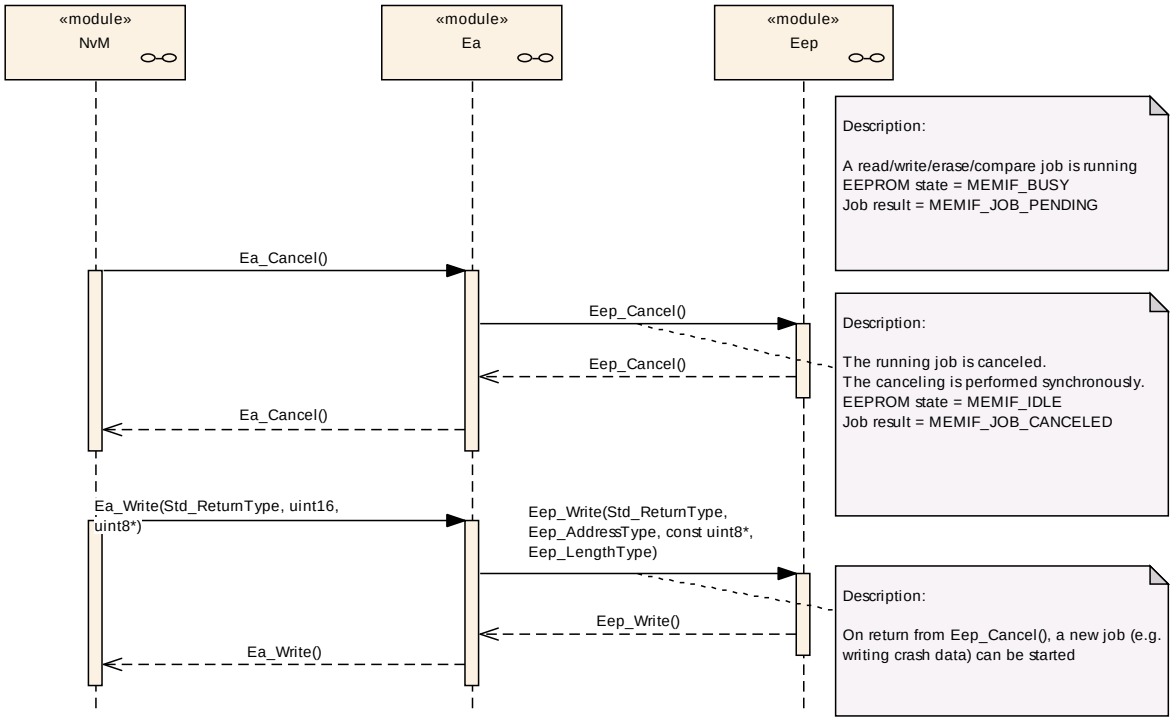

Figure 4

10 Configuration specification

10.1 How to read this chapter

In addition to this section, it is highly recommended to read the documents:
- AUTOSAR Layered Software Architecture [1]
- AUTOSAR ECU Configuration Specification [5]
  This document describes the AUTOSAR configuration methodology and the AUTOSAR configuration metamodel in detail.

The following is only a short survey of the topic and it will not replace the ECU Configuration Specification document.

10.1.1 Configuration and configuration parameters

Configuration parameters define the variability of the generic part(s) of an implementation of a module. This means that only generic or configurable module implementation can be adapted to the environment (software/hardware) in use during system and/or ECU configuration.

The configuration of parameters can be achieved at different times during the software process: before compile time, before link time or after build time. In the following, the term “configuration class” (of a parameter) is used in order to refer to a specific configuration point in time.

10.1.2 Containers

Containers structure the set of configuration parameters. This means:
- *all* configuration parameters are kept in containers.
- (sub-) containers can reference (sub-) containers. It is possible to assign a multiplicity to these references. The multiplicity then defines the possible number of instances of the contained parameters.

10.1.3 Specification template for configuration parameters

The following tables consist of three sections:
- the general section
- the configuration parameter section
- the section of included/referenced containers

Pre-compile time - specifies whether the configuration parameter is of configuration class *Pre-compile time* or not

Label	***Description***
X	The configuration parameter is of configuration class *Pre-compile time*.
--	The configuration parameter is never of configuration class *Pre-compile time*.

Link time - specifies whether the configuration parameter is of configuration class *Link time* or not

Label	*Description*
x	The configuration parameter is of configuration class *Link time*.
--	The configuration parameter is never of configuration class *Link time*.

Post Build - specifies whether the configuration parameter is of configuration class *Post Build* or not

Label	*Description*
x	The configuration parameter is of configuration class *Post Build* and no specific implementation is required.
L	*Loadable* - the configuration parameter is of configuration class *Post Build* and only one configuration parameter set resides in the ECU.
M	*Multiple* - the configuration parameter is of configuration class *Post Build* and is selected out of a set of multiple parameters by passing a dedicated pointer to the init function of the module.
--	The configuration parameter is never of configuration class *Post Build*.

10.2 Containers and configuration parameters

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

10.2.1 Variants

[EEP109] ⌈VARIANT-PRE-COMPILE
Only parameters with “Pre-compile time” configuration are allowed in this variant. ⌋(BSW00396, BSW00397)

[EEP110] ⌈VARIANT-POST-BUILD
Parameters with “Pre-compile time”, “Link time” and “Post-build time” are allowed in this variant. ⌋(BSW00404, BSW00396, BSW00398)

10.2.2 Eep

Module Name	*Eep*
Module 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.

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 CommonPublishedInformation container. Note that these parameters do not have any configuration class setting, since they are published information.

10.2.3 EepGeneral

SWS Item	**EEP085_Conf :**
Container Name	EepGeneral{EepGeneralConfiguration}
Description	Container for general configuration parameters of the EEPROM driver. These parameters are always pre-compile.
Configuration Parameters	

SWS Item	**EEP188_Conf :**
Name	EepDevErrorDetect {EEP_DEV_ERROR_DETECT}

Description	Pre-processor switch to enable and disable development error detection. true: Development error detection enabled. false: Development error detection disabled.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	true		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**EEP189_Conf :**		
Name	EepDriverIndex		
Description	Index of the driver, used by EA.		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 254		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**EEP163_Conf :**		
Name	EepUseInterrupts {EEP_USE_INTERRUPTS}		
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		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module dependency: Usually, this is only supported by some internal EEPROM peripherals.		

SWS Item	**EEP164_Conf :**		
Name	EepVersionInfoApi {EEP_VERSION_INFO_API}		
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		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**EEP165_Conf :**
Name	EepWriteCycleReduction {EEP_WRITE_CYCLE_REDUCTION}
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		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

No Included Containers

10.2.4 EepInitConfiguration

SWS Item	**EEP039_Conf :**
Container Name	EepInitConfiguration{EepInitConfiguration} [Multi Config Container]
Description	Container for runtime configuration parameters of the EEPROM driver. Implementation Type: Eep_ConfigType.
Configuration Parameters	

SWS Item	**EEP166_Conf :**		
Name	EepBaseAddress {EEP_BASE_ADDRESS}		
Description	This parameter is the EEPROM device base address. Implementation Type: Eep_AddressType.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: module		

SWS Item	**EEP167_Conf :**		
Name	EepDefaultMode {EEP_DEFAULT_MODE}		
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)
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: module		

SWS Item	EEP168_Conf :		
Name	EepFastReadBlockSize {EEP_FAST_READ_BLOCK_SIZE}		
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 EepNormalReadBlockSize. Implementation Type: Eep_LengthType.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: module		

SWS Item	EEP169_Conf :		
Name	EepFastWriteBlockSize {EEP_FAST_WRITE_BLOCK_SIZE}		
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 EepNormalWriteBlockSize. Implementation Type: Eep_LengthType.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: module 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	EEP170_Conf :		
Name	EepJobCallCycle {EEP_JOB_CALL_CYCLE}		
Description	Call cycle time of the EEPROM driver's main function. Unit: [s]		
Multiplicity	0..1		
Type	EcucFloatParamDef		
Range	`0 .. INF`		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: module		

SWS Item	EEP171_Conf :
Name	EepJobEndNotification {EEP_JOB_END_NOTIFICATION}
Description	This parameter is a reference to a callback function for positive job result (see EEP045).
Multiplicity	0..1
Type	EcucFunctionNameDef
Default value	--
maxLength	--
minLength	--

regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: module		

SWS Item	**EEP172_Conf :**		
Name	EepJobErrorNotification {EEP_JOB_ERROR_NOTIFICATION}		
Description	This parameter is a reference to a callback function for negative job result (see EEP046).		
Multiplicity	0..1		
Type	EcucFunctionNameDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: module		

SWS Item	**EEP173_Conf :**		
Name	EepNormalReadBlockSize {EEP_NORMAL_READ_BLOCK_SIZE}		
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	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: module		

SWS Item	**EEP174_Conf :**		
Name	EepNormalWriteBlockSize {EEP_NORMAL_WRITE_BLOCK_SIZE}		
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	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: module dependency: This parameter is optional and only available if the hardware allows configuration.		

SWS Item	**EEP175_Conf :**
Name	EepSize {EEP_SIZE}
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	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: module		

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_ReportErrorStatus API in case the corresponding error occurs. The EventId is taken from the referenced DemEventParameter's DemEventId value. The standardized errors are provided in the 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.

10.2.5 EepDemEventParameterRefs

SWS Item	**EEP200_Conf :**
Container Name	EepDemEventParameterRefs
Description	Container for the references to DemEventParameter elements which shall be invoked using the API Dem_ReportErrorStatus API in case the corresponding error occurs. The EventId is taken from the referenced DemEventParameter's DemEventId value. The standardized errors are provided in the container and can be extended by vendor specific error references.
Configuration Parameters	

SWS Item	**EEP204_Conf :**		
Name	EEP_E_COMPARE_FAILED		
Description	Reference to the DemEventParameter which shall be issued when the error "EEPROM compare failed (HW)" has occurred.		
Multiplicity	0..1		
Type	Reference to [DemEventParameter]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

SWS Item	**EEP201_Conf :**		
Name	EEP_E_ERASE_FAILED		
Description	Reference to the DemEventParameter which shall be issued when the error "EEPROM erase failed (HW)" has occurred.		
Multiplicity	0..1		
Type	Reference to [DemEventParameter]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	

	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

SWS Item	**EEP203_Conf :**		
Name	EEP_E_READ_FAILED		
Description	Reference to the DemEventParameter which shall be issued when the error "EEPROM read failed (HW)" has occurred.		
Multiplicity	0..1		
Type	Reference to [DemEventParameter]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

SWS Item	**EEP202_Conf :**		
Name	EEP_E_WRITE_FAILED		
Description	Reference to the DemEventParameter which shall be issued when the error "EEPROM write failed (HW)" has occurred.		
Multiplicity	0..1		
Type	Reference to [DemEventParameter]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

No Included Containers

10.2.6 EepExternalDriver

SWS Item	**EEP190_Conf :**
Container Name	EepExternalDriver
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	**EEP176_Conf :**		
Name	EepSpiReference		
Description	Reference to SPI sequence (required for external EEPROM drivers).		
Multiplicity	1..*		
Type	Reference to [SpiSequence]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

No Included Containers

10.2.7 SPI specific extension

[EEP094] ⌈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 EEP039_Conf). 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 EEP095).

- All required SPI channels
- All required SPI sequences
- All required SPI jobs ⌋(BSW00390, BSW00391, BSW00398)

10.3 Published parameters

10.3.1 Basic subset

[EEP240] ⌈The standardized common published parameters as required by BSW00402 in the General Requirements on Basic Software Modules [2] shall be published within the header file of this module and need to be provided in the BSW Module Description. The according module abbreviation can be found in the List of Basic Software Modules [1]. ⌋()

Additional module-specific published parameters are listed below if applicable.

10.3.2 SPI specific extension

[EEP095] ⌈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. ⌋(BSW00390, BSW00391, BSW00402)

10.3.3 EepPublishedInformation

SWS Item	**EEP111_Conf :**
Container Name	EepPublishedInformation
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	**EEP177_Conf :**		
Name	EepAllowedWriteCycles {EEP_ALLOWED_WRITE_CYCLES}		
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	--		
ConfigurationClass	***Published Information***	X	All Variants
Scope / Dependency	scope: module		

SWS Item	**EEP178_Conf :**		
Name	EepEraseTime {EEP_ERASE_TIME}		
Description	Maximum time for erasing one EEPROM data unit.		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	`0 .. INF`		
Default value	--		
ConfigurationClass	***Published Information***	X	All Variants
Scope / Dependency	scope: module		

SWS Item	**EEP179_Conf :**		
Name	EepEraseUnitSize {EEP_ERASE_UNIT_SIZE}		
Description	Size of smallest erasable EEPROM data unit in bytes.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	--		
ConfigurationClass	***Published Information***	X	All Variants
Scope / Dependency	scope: module		

SWS Item	**EEP180_Conf :**		
Name	EepEraseValue {EEP_ERASE_VALUE}		
Description	Value of an erased EEPROM cell.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 255		
Default value	--		
ConfigurationClass	***Published Information***	X	All Variants
Scope / Dependency	scope: module		

SWS Item	**EEP181_Conf :**		
Name	EepMinimumAddressType {EEP_MINIMUM_ADDRESS_TYPE}		
Description	Minimum expected size of Eep_AddressType.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	--		
ConfigurationClass	***Published Information***	X	All Variants
Scope / Dependency	scope: module		

SWS Item	**EEP182_Conf :**

Name	EepMinimumLengthType {EEP_MINIMUM_LENGTH_TYPE}		
Description	Minimum expected size of Eep_LengthType.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	--		
ConfigurationClass	***Published Information***	X	All Variants
Scope / Dependency	scope: module		

SWS Item	**EEP183_Conf :**		
Name	EepReadUnitSize {EEP_READ_UNIT_SIZE}		
Description	Size of smallest readable EEPROM data unit in bytes.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	--		
ConfigurationClass	***Published Information***	X	All Variants
Scope / Dependency	scope: module		

SWS Item	**EEP187_Conf :**		
Name	EepSpecifiedEraseCycles {EEP_SPECIFIED_ERASE_CYCLES}		
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	--		
ConfigurationClass	***Published Information***	X	All Variants
Scope / Dependency	scope: module		

SWS Item	**EEP184_Conf :**		
Name	EepTotalSize {EEP_TOTAL_SIZE}		
Description	Total size of EEPROM in bytes. Implementation Type: Eep_LengthType.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	--		
ConfigurationClass	***Published Information***	X	All Variants
Scope / Dependency	scope: module		

SWS Item	**EEP185_Conf :**		
Name	EepWriteTime {EEP_WRITE_TIME}		
Description	Maximum time for writing one EEPROM data unit.		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	`0 .. INF`		
Default value	--		
ConfigurationClass	***Published Information***	X	All Variants
Scope / Dependency	scope: module		

SWS Item	**EEP186_Conf :**
Name	EepWriteUnitSize {EEP_WRITE_UNIT_SIZE}
Description	Size of smallest writeable EEPROM data unit in bytes.
Multiplicity	1

Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	--		
ConfigurationClass	***Published Information***	X	All Variants
Scope / Dependency	scope: module		

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.[4]

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 SpiChannels 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.
    a. *EepChCommand*
    b. *EepChAddress*
    c. *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 SpiJobs 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 SpiJobs 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. The EEPROM driver is given access to these parameters by the means of C-defines contained in the configuration header file Spi_cfg.h:

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

The EEPROM driver should not directly include Spi_Cfg.h; it should rather include Spi.h, which in turn includes Spi_Cfg.

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.

11 Changes in Release 4.0

11.1 Deleted SWS Items

SWS Item	*Rationale*
EEP035	Changed into note as it was a requirement on the specification instead of the implementation
EEP099	Redundant to EEP038

11.2 Replaced SWS Items

11.3 Changed SWS Items

SWS Item	*Rationale*
EEP001	Rephrased requirement according to current version of SWS template
EEP003	Made hidden text visible
EEP027	Split second part of requirement into EEP216 to make them atomic
EEP028	Changed MEMIF_JOB_CANCELLED to MEMIF_JOB_CANCELED (use American English spelling)
EEP030	Removed dependency of operation on “hardware ready” requirement. Made hidden text visible
EEP036	Added note clarifying error handling
EEP047	Clarified optional callback notifications
EEP049	Clarified optional callback notifications
EEP053	Clarified the term “normal mode” by changing it to “normal SPI mode”
EEP056	Error handling consistency with Flash driver
EEP058	Split requirement into EEP058, EEP219, EEP220, EEP221
EEP068	Split requirement into EEP068, EEP222, EEP223, EEP224
EEP073	Clarified the term “burst mode” by changing it to “burst SPI mode”
EEP102	Added additional include requirements to EEP implementation
EEP106	Rephrased requirement according to current version of SWS template
EEP110	VARIANT-LINK-TIME changed to VARIANT-POST-BUILD
EEP113	For clarification of EepAddressType usage, split into EEP113 and EEP217
EEP128	Added a reference to configuration parameter definition EEP163_Conf. Made hidden text visible
EEP144	Replaced MEMIF_MODE_NORMAL by MEMIF_MODE_SLOW
EEP157	Updated signature of Eep_JobEndNotification
EEP158	Updated signature of Eep_JobErrorNotification
EEP226	Clarified range of Eep_AddresType

11.4 Added SWS Items

SWS Item	*Rationale*
EEP200	Error handling consistency with Flash driver
EEP201	Error handling consistency with Flash driver
EEP202	Error handling consistency with Flash driver
EEP203	Error handling consistency with Flash driver
EEP204	Error handling consistency with Flash driver
EEP205	Error handling consistency with Flash driver
EEP206	Error handling consistency with Flash driver
EEP207	Error handling consistency with Flash driver
EEP208	Support for Debugging

EEP209	Support for Debugging
EEP210	Support for Debugging
EEP211	Support for Debugging
EEP212	Support for Debugging
EEP213	Support for Debugging
EEP214	Support for Debugging
EEP215	Improved description of Eep_Cancel behavior
EEP216	Make SWS items atomic
EEP217	Clarify usage of Eep_AddressType
EEP218	Make SWS items atomic
EEP219	Make SWS item EEP058 atomic (write job)
EEP220	Make SWS item EEP058 atomic (erase job)
EEP221	Make SWS item EEP058 atomic (compare job)
EEP222	Make SWS item EEP068 atomic (write job)
EEP223	Make SWS item EEP068 atomic (erase job)
EEP224	Make SWS item EEP068 atomic (compare job)
EEP225	Added SWS item for Eep_ConfigType
EEP226	Added SWS item for Eep_AddressType
EEP227	Added SWS item for Eep_LengthType
EEP228	Added file structure existence requirement for Eep_Irq.c
EEP229	Added file structure include requirements for Eep.h
EEP230	Added file structure include requirements regarding Spi.h
EEP231	Added file structure include requirements for Eep_Irq.c
EEP232	Added file structure include requirements for Eep_Lcfg.c
EEP233	Added file structure include requirements for Eep_PBcfg.c
EEP200_Conf	EepDemEventParameterRefs
EEP201_Conf	EEP_E_ERASE_FAILED
EEP202_Conf	EEP_E_WRITE_FAILED
EEP203_Conf	EEP_E_READ_FAILED
EEP204_Conf	EEP_E_COMPARE_FAILED
EEP240	Rework of Published Information

12 Not applicable requirements

[EEP241] ⌈ These requirements are not applicable to this specification.⌋ (BSW170, BSW00399, BSW00400, BSW00375, BSW00416, BSW168, BSW00423, BSW00424, BSW00426, BSW00427, BSW00428, BSW00429, BSW00431, BSW00432, BSW00433, BSW00434, BSW00336, BSW00422, BSW00420, BSW00417, BSW161, BSW162, BSW00324, BSW005, BSW164, BSW00325, BSW00326, BSW00342, BSW00343, BSW007, BSW00413, BSW00347, BSW00307, BSW00301, BSW00302, BSW00328, BSW00312, BSW006, BSW00355, BSW00378, BSW00306, BSW00308, BSW00309, BSW00330, BSW00331, BSW009, BSW00401, BSW172, BSW010, BSW00341, BSW00334, BSW12267, BSW12163, BSW12068, BSW12069, BSW12063, BSW12129, BSW12067, BSW12077, BSW12078, BSW12092, BSW12265)