Document Title	Specification of Module EEPROM Driver
Document Owner	AUTOSAR GbR
Document Responsibility	AUTOSAR GbR
Document Version	2.1.0
Document Status	Draft
Part of Release	2.1
Revision	0014

Document Change History			
Date	**Version**	**Changed by**	**Change Description**
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 missinterpretation of the described "write-cycle-reduction" functionality
30.06.2005	1.0.0	AUTOSAR Administration	Initial Release

1 Release Notes

Errata and known deficiencies

There is an uncertainty upon the proper specification of the EEP_NUMBER_OF_WRITE_CYCLES parameter with respect to its usage as configuration or as published parameter.

Changes planned for next release

The above mentioned deficiencies will be resolved in the next release.

Disclaimer

Advice to users of AUTOSAR Specification Documents:

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
https://svn.autosar.org/repos/10Releases/
AUTOSAR_LayeredSoftwareArchitecture.pdf

[2] General Requirements on Basic Software Modules,
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SRS_General.pdf

[3] Specification of Memory Abstraction Interface
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SWS_Mem_AbstractionInterface.pdf

[4] Specification of SPI Handler/Driver
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SWS_SPI_HandlerDriver.pdf

[5] Specification of ECU Configuration
https://svn.autosar.org/repos/10Releases/
AUTOSAR_ECU_Configuration.pdf

[6] Requirements on EEPROM Driver
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SRS_EEPROM_Driver.pdf

[7] Specification of Development Error Tracer
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SWS_Development_Error_Tracer.pdf

[8] Specification of Diagnostics Event Manager
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SWS_DEM

[9] AUTOSAR Glossary
https://svn.autosar.org/repos/10Releases/
AUTOSAR_Glossary.pdf

[10] Specification of MCU Driver
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SWS_MCU_Driver

3.2 Related standards and norms

[11] 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

5.1.1 Code file structure

EEP101: The code file structure shall not be defined within this specification completely. At this point it shall be pointed out that the code-file structure shall include the following file named:

- Eep_Lcfg.c – for link time and for post-build configurable parameters and
- Eep_PBcfg.c – for post build time configurable parameters.

These files shall contain all link time and post-build time configurable parameters.

5.1.2 Header file structure

EEP083: The include file structure shall be as follows:

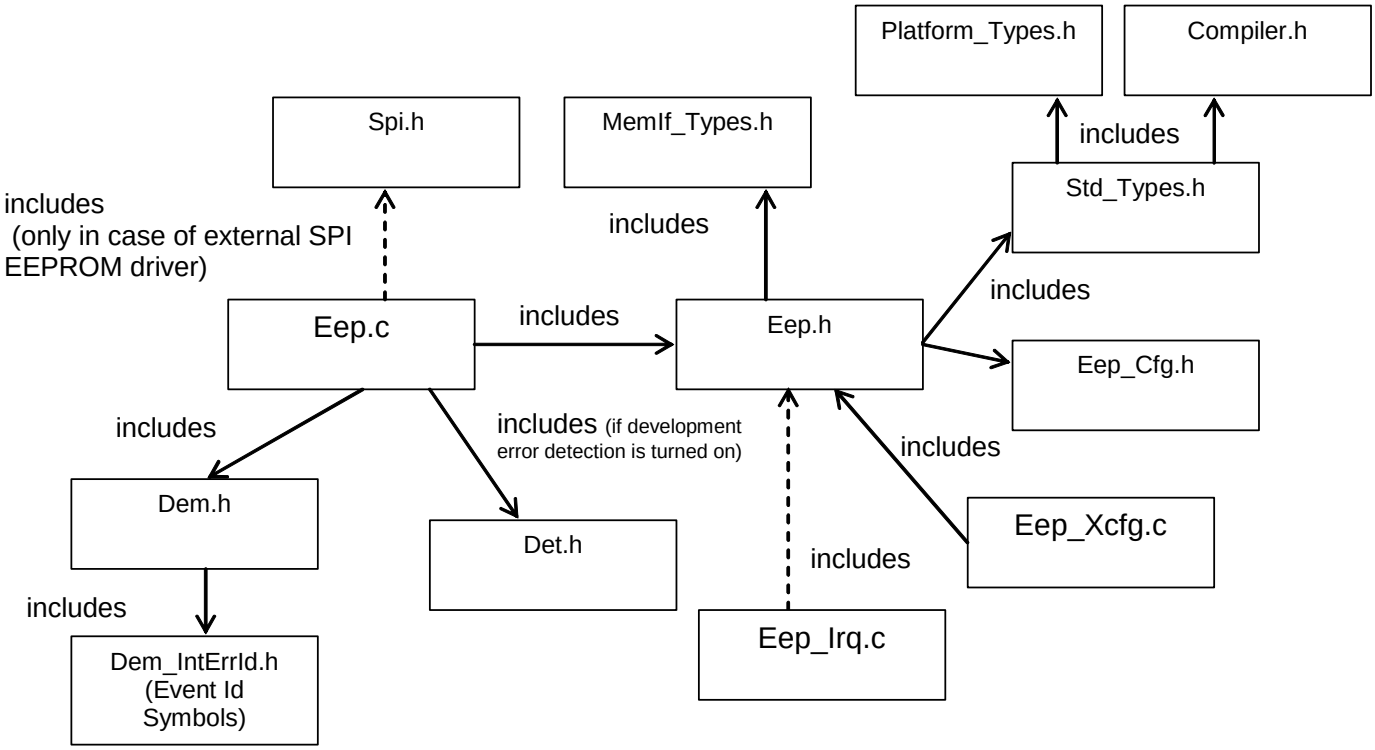

Figure 1

- `Eep.h` shall include `MemIf_Types.h`
- `Eep.c` shall include `Eep.h`
- `Eep.h` shall include `Eep_Cfg.h`
- In case of a driver for an external SPI EEPROM, `Eep.c` shall include `Spi.h`
- In case of a driver for an internal EEPROM, `Eep_Irq.c` this file could exist depending of implementation and also it could or not include `Eep.h`

EEP102: The module shall include the Dem.h file. By this inclusion 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.

6 Requirements traceability

Document: AUTOSAR Requirements on Basic Software, general

Requirement	***Satisfied by***
[BSW00344] Reference to link-time configuration	EEP039
[BSW00404] Reference to post build time configuration	EEP039, EEP110
[BSW00405] Reference to multiple configuration sets	Chapter 10.2
[BSW00345] Pre-compile-time configuration	EEP085
[BSW159] Tool-based configuration	Both static and runtime configuration parameters are located outside the source code of the module. This is the prerequisite for automatic configuration.
[BSW167] Static configuration checking	Requirement on configuration tool
[BSW171] Configurability of optional functionality	Conflicts partly with SPAL requirement [BSW12263] Configuration after compile time. EEP085
[BSW170] Data for reconfiguration of AUTOSAR SW-Components	Not applicable (requirement on SW Component)
[BSW00380] Separate C-Files for configuration parameters	EEP101
[BSW00419] Separate C-Files for pre-compile time configuration parameters	EEP101
[BSW00381] Separate configuration header file for pre-compile time parameters	EEP085
[BSW00412] Separate H-File for configuration parameters	EEP083
[BSW00383] List dependencies of configuration files	Chapters 5, 10.2.4 and 10.3.2
[BSW00384] List dependencies to other modules	Chapter 5, EEP102
[BSW00387] Specify the configuration class of callback function	Chapters 8.4 and 8.6.3
[BSW00388] Introduce containers	EEP085, EEP039
[BSW00389] Containers shall have names	EEP085, EEP039
[BSW00390] Parameter content shall be unique within the module	EEP085, EEP039, EEP094, EEP095, EEP111
[BSW00391] Parameter shall have unique names	EEP085, EEP039, EEP094, EEP095, EEP111
[BSW00392] Parameters shall have a type	EEP085, EEP039, EEP111
[BSW00393] Parameters shall have a range	EEP085, EEP039
[BSW00394] Specify the scope of the parameters	EEP085, EEP039
[BSW00395] List the required parameters (per parameter)	EEP085, EEP039
[BSW00396] Configuration classes	EEP085, EEP039, EEP109, EEP110
[BSW00397] Pre-compile-time parameters	EEP109, EEP085
[BSW00398] Link-time parameters	EEP110, EEP039, EEP094
[BSW00399] Loadable Post-build time parameters	Not applicable (Cannot be detailed at this point of time, because this depends on ECU integration.)
[BSW00400] Selectable Post-build time parameters	Not applicable (Cannot be detailed at this point of time, because this depends on ECU integration.)
[BSW00402] Published information	EEP099, EEP038, EEP111, EEP095
[BSW00375] Notification of wake-up reason	Not applicable (the EEPROM does not cause any wake-ups)
[BSW101] Initialization interface	EEP004

[BSW00416] Sequence of Initialization	Not applicable (this is a general software integration requirement)
[BSW00406] Check module initialization	EEP033, EEP006
[BSW168] Diagnostic Interface of SW components	Not applicable (no use case)
[BSW00407] Function to read out published parameters	EEP107, EEP108
[BSW00423] Usage of SW-C template to describe BSW modules with AUTOSAR Interfaces	Not applicable (EEPROM driver has no Autosar Interface)
[BSW00424] BSW main processing function task allocation	Not applicable (this is a general software integration requirement)
[BSW00425] Trigger conditions for schedulable objects	EEP039, Chapter 8.5
[BSW00426] Exclusive areas in BSW modules	Not applicable (Cannot be detailed at this point of time, because this depends on module implementation.)
[BSW00427] ISR description for BSW modules	Not applicable (Cannot be detailed at this point of time, because this depends on module implementation.)
[BSW00428] Execution order dependencies of main processing functions	Not applicable (Cannot be detailed at this point of time, because this depends on module implementation.)
[BSW00429] Restricted BSW OS functionality access	Not applicable (requirement on implementation, not on specification)
[BSW00431] The BSW Scheduler module implements task bodies	Not applicable EEPROM Module is not the BSW Scheduler)
[BSW00432] Modules should have separate main processing functions for read/receive and write/transmit data path	Not applicable (requirement on implementation, not on specification)
[BSW00433] Calling of main processing functions	Not applicable (this is a general software integration requirement)
[BSW00434] The Schedule Module shall provide an API for exclusive areas	Not applicable (EEPROM Module is not the BSW Scheduler)
[BSW00336] Shutdown interface	Not applicable (no use case)
[BSW00337] Classification of errors	EEP000, EEP003
[BSW00338] Reporting of development errors	EEP001
[BSW00369] Do not return development error codes via API	EEP001, EEP010, EEP033
[BSW00339] Reporting of production relevant error status	EEP002, Chapter 8.6.2
[BSW00421] Reporting of production relevant error events	EEP002, Chapter 8.6.1
[BSW00422] Debouncing of production relevant error status	Non applicable (applies only for DEM)
[BSW00420] Production relevant error event rate detection	Non applicable (applies only for DEM)
[BSW00417] Reporting of Error Events by Non-Basic Software	Non applicable (applies only for non BSW modules)
[BSW00323] API parameter checking	EEP005, EEP010
[BSW004] Version check	EEP091
[BSW00409] Header files for production code error IDs	EEP103
[BSW00385] List possible error notifications	EEP000, EEP003
[BSW00386] Configuration for detecting an error	EEP001, EEP105
[BSW161] Microcontroller abstraction	Not applicable (requirement on AUTOSAR architecture, not a single module)

[BSW162] ECU layout abstraction	Not applicable (requirement on AUTOSAR architecture, not a single module)
[BSW00324] Do not use HIS I/O Library	Not applicable (requirement on AUTOSAR architecture, not a single module)
[BSW005] No hard coded horizontal interfaces within MCAL	Not applicable (requirement on implementation, not on specification)
[BSW00415] User dependent include files	EEP083
[BSW164] Implementation of interrupt service routines	Not applicable (Cannot be detailed at this point of time, because this depends on module implementation.)
[BSW00325] Runtime of interrupt service routines	Not applicable (Cannot be detailed at this point of time, because this depends on module implementation.)
[BSW00326] Transition from ISRs to OS tasks	Not applicable (Cannot be detailed at this point of time, because this depends on module implementation.)
[BSW00342] Usage of source code and object code	Not applicable (requirement on implementation, not on specification)
[BSW00343] Specification and configuration of time	Not applicable (requirement on implementation, not on specification)
[BSW160] Human-readable configuration data	Requirement on configuration methodology and tools.
[BSW007] HIS MISRA C	Not applicable (requirement on implementation, not on specification)
[BSW00300] Module naming convention	Chapter 5.1
[BSW00413] Accessing instances of BSW modules	Not applicable (requirement on implementation, not on specification)
[BSW00347] Naming separation of different instances of BSW drivers	Not applicable (requirement on implementation, not on specification)
[BSW00305] Self-defined data types naming convention	Chapter 8.2
[BSW00307] Global variables naming convention	Not applicable (requirement on implementation, not on specification)
[BSW00310] API naming convention	Chapter 8.3
[BSW00373] Main processing function naming convention	Chapter 8.5
[BSW00327] Error values naming convention	Chapter 7.2
[BSW00335] Status values naming convention	Chapter 8.1, EEP080
[BSW00350] Development error detection keyword	EEP001
[BSW00408] Configuration parameter naming convention	Chapter 10.2
[BSW00410] Compiler switches shall have defined values	EEP085
[BSW00411] Get version info keyword	EEP108
[BSW00346] Basic set of module files	
[BSW158] Separation of configuration from implementation	EEP101, EEP039, EEP085
[BSW00314] Separation of interrupt frames and service routines	Chapter 5.1

[BSW00370] Separation of callback interface from API	Chapter 8.4
[BSW00348] Standard type header	Chapter 8.1
[BSW00353] Platform specific type header	Chapter 8.1
[BSW00361] Compiler specific language extension header	Chapter 5.1.2
[BSW00301] Limit imported information	Not applicable (requirement on implementation, not on specification)
[BSW00302] Limit exported information	Not applicable (requirement on implementation, not on specification)
[BSW00328] Avoid duplication of code	Not applicable (requirement on implementation, not on specification)
[BSW00312] Shared code shall be reentrant	Not applicable (requirement on implementation, not on specification)
[BSW006] Platform independency	Not applicable (requirement on implementation, not on specification)
[BSW00357] Standard API return type	EEP093, Chapter 8.3
[BSW00377] Module specific API return types	EEP080, Chapters 8.3.8 and 8.3.9
[BSW00304] AUTOSAR integer data types	Chapters 5.1.2, 8.2, 10.2 and 10.3
[BSW00355] Do not redefine AUTOSAR integer data types	Not applicable (requirement on implementation, not on specification)
[BSW00378] AUTOSAR boolean type	Not applicable (no use case)
[BSW00306] Avoid direct use of compiler and platform specific keywords	Not applicable (requirement on implementation, not on specification)
[BSW00308] Definition of global data	Not applicable (requirement on implementation, not on specification)
[BSW00309] Global data with read-only constraint	Not applicable (requirement on implementation, not on specification)
[BSW00371] Do not pass function pointers via API	Chapter 8.6.3 and 10.2.3
[BSW00358] Return type of init() functions	Chapter 8.3.1
[BSW00414] Parameter of init function	Chapter 8.3.1
[BSW00376] Return type and parameters of main processing functions	Chapter 8.5.1
[BSW00359] Return type of callback functions	EEP048
[BSW00360] Parameters of callback functions	EEP048
[BSW00329] Avoidance of generic interfaces	Chapter 8
[BSW00330] Usage of macros / inline functions instead of functions	Not applicable (requirement on implementation, not on specification)
[BSW00331] Separation of error and status values	Not applicable (requirement on implementation, not on specification)
[BSW009] Module User Documentation	Not applicable (requirement on implementation, not on specification)
[BSW00401] Documentation of multiple instances of configuration parameters	Not applicable (requirement on implementation, not on specification)

[BSW172] Compatibility and documentation of scheduling strategy	Not applicable (requirement on implementation, not on specification)
[BSW010] Memory resource documentation	Not applicable (requirement on implementation, not on specification)
[BSW00333] Documentation of callback function context	Chapters 8.6.3.1 and 8.6.3.2
[BSW00374] Module vendor identification	EEP099, EEP111
[BSW00379] Module identification	EEP099, EEP111
[BSW003] Version identification	EEP099, EEP111
[BSW00318] Format of module version numbers	EEP111
[BSW00321] Enumeration of module version numbers	EEP111
[BSW00341] Microcontroller compatibility documentation	Not applicable (requirement on implementation, not on specification)
[BSW00334] Provision of XML file	Not applicable (requirement on implementation, not on specification)

Document: AUTOSAR requirements on Basic Software, cluster SPAL, General

Requirement	***Satisfied by***
[BSW12263] Object code compatible configuration concept	EEP039
[BSW12056] Configuration of notification mechanisms	EEP039, EEP047, EEP049
[BSW12267] Configuration of wake-up sources	Not applicable (the EEPROM does not cause any wake-ups)
[BSW12057] Driver module initialization	EEP004
[BSW12125] Initialization of hardware resources	Cannot be detailed at this point of time, because this depends on EEPROM hardware.
[BSW12163] Driver module deinitialization	Not applicable (no use case)
[BSW12058] Individual initialization of overall registers	Cannot be detailed at this point of time, because this depends on EEPROM hardware.
[BSW12059] General initialization of overall registers	Cannot be detailed at this point of time (see above)
[BSW12461] Responsibility for register initialization	Cannot be detailed at this point of time, because this depends on EEPROM hardware.
[BSW12462] Provide settings for register initialization	Cannot be detailed at this point of time, because this depends on EEPROM hardware.
[BSW12463] Combine and forward settings for register initialization	Cannot be detailed at this point of time, because this depends on EEPROM hardware.
[BSW12060] Responsibility for initialization of one-time writable registers	Cannot be detailed at this point of time (see above)
[BSW12062] Selection of static configuration sets	EEP004
[BSW12068] MCAL initialization sequence	Not applicable (this is a general software integration requirement)
[BSW12069] Wake-up notification of ECU State Manager	Not applicable (the EEPROM does not cause any wake-ups)
[BSW157] Notification mechanisms of drivers and handlers	EEP029, EEP024, EEP047, EEP045, EEP046
[BSW12155] Prototypes of callback functions	EEP048
[BSW12169] Control of operation mode	Covered by [BSW12156] EEPROM mode selection function

Requirement	Satisfied by
[BSW12063] Raw value mode	Not applicable (no I/O functionality)
[BSW12075] Use of application buffers	EEP037
[BSW12129] Resetting of interrupt flags	Not applicable (requirement on implementation, not on specification)
[BSW12171] Support of synchronous and asynchronous SPI interface	Chapter 5
[BSW12064] Change of operation mode during running operation	EEP033
[BSW12448] Behavior after development error detection	EEP001, EEP005, EEP010, EEP033
[BSW12067] Setting of wake-up conditions	Not applicable (the EEPROM does not cause any wake-ups)
[BSW12077] Non-blocking implementation	Not applicable (requirement on implementation, not on specification)
[BSW12078] Runtime and memory efficiency	Not applicable (requirement on implementation, not on specification)
[BSW12092] Access to drivers	Not applicable (requirement on implementation, not on specification)
[BSW12265] Configuration data shall be kept constant	Not applicable (requirement on implementation, not on specification)
[BSW12264] Specification of configuration items	Chapter 10.2

Document: AUTOSAR requirements on Basic Software, cluster SPAL, EEPROM Driver

Requirement	Satisfied by
[BSW096] EEPROM driver static configuration	EEP039
[BSW12071] Publication of EEPROM properties	EEP038
[BSW087] EEPROM read function	EEP009, EEP013
[BSW088] EEPROM write function	EEP014, EEP015, EEP063, EEP090
[BSW089] EEPROM erase function	EEP019, EEP020, EEP070, EEP072
[BSW12091] EEPROM compare function	EEP025, EEP026
[BSW090] EEPROM cancel function	EEP021, EEP027, EEP028, EEP114
[BSW091] EEPROM status function	EEP029
[BSW12156] EEPROM mode selection function	EEP042, EEP050
[BSW092] EEPROM write cycle reduction	EEP060, EEP064
[BSW094] EEPROM segmentation handling	EEP063, EEP072, EEP070, EEP090
[BSW095] EEPROM job management	EEP036, EEP033
[BSW12047] EEPROM job processing function	EEP030
[BSW12157] Job processing – normal mode	EEP051, EEP052, EEP053
[BSW12072] Job processing – fast mode	EEP054, EEP055EEP055, EEP073
[BSW12050] EEPROM job processing execution time	EEP057, EEP069, EEP051, EEP054
[BSW12051] Functional scope	Chapter 0, EEP088
[BSW12164] SPI channel configuration	EEP039
[BSW12124] SPI access modes	EEP052, EEP053, EEP055, EEP073, EEP039

7 Functional specification

7.1 General behavior

EEP088: This specification shall be valid both for internal and external EEPROMs.

EEP035: The EEPROM driver shall offer asynchronous services for EEPROM operations (read/write/erase/compare).

EEP036: The EEPROM driver shall not buffer jobs. It shall accept only one job at a time. During job processing, no other job shall be accepted.

EEP037: The EEPROM driver shall not buffer data to be read or written. It shall use application data buffers that are referenced by a pointer passed via the API.

7.2 Error classification

EEP000: The following errors shall be detectable by the EEPROM driver depending on their build options (development/production mode):

EEP103: Values for production code Event Ids are assigned externally by the configuration of the Dem. They are published in the file Dem_IntErrId.h and included via Dem.h.

Type or error	Relevance	Related error code	Value

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 used without module initialization	Development	`EEP_E_UNINIT`	`0x20`
Read/write/erase/compare API service called while EEPROM is busy	Development	`EEP_E_BUSY`	`0x21`
External EEPROM driver: Communication with external device failed (e.g. no external EEPROM connected)	Production	`EEP_E_COM_FAILURE`	`Assigned externally`

Additional errors that are detected because of specific implementation and/or specific hardware properties shall be added in the EEPROM device specific implementation specification. The classification and enumeration shall be compatible to the errors listed above [EEP000].

EEP104: Development error values are of type `uint8`.

7.3 Error detection

EEP001: The detection of all development errors is configurable (On / Off) at pre-compile time. The switch EEP_DEV_ERROR_DETECT (see chapter 10.2.2) shall activate or deactivate the detection of all development errors.

EEP105: If the switch EEP_DEV_ERROR_DETECT is enabled API parameter checking is also enabled. The detailed description of the detected errors can be found in chapter 8.7.

EEP106: The detection of production code errors cannot be switched off.

7.4 Error notification

EEP002: Production errors shall be reported to the Diagnostic Event Manager.

EEP100: Detected development errors shall be reported to the *Det_ReportError* service of the Development Error Tracer (DET) if the pre-processor switch EEP_DEV_ERROR_DETECT is set (see chapter 10.2.2).

7.5 Processing of jobs – general requirements

EEP058: When a read/write/erase/compare job is finished successfully, the EEPROM driver shall set the EEPROM state to MEMIF_IDLE and the job result to MEMIF_JOB_OK. If configured, the notification Eep_JobEndNotification() shall be called.

EEP068: When an error is detected during read/write/erase/compare job processing, the EEPROM driver shall abort the job, set the EEPROM state to MEMIF_IDLE and the job result to the corresponding value: MEMIF_JOB_FAILED
If configured, the notification Eep_JobErrorNotification() shall be called.

EEP114: When a user cancels a job processing by using the Eep_Cancel() function, the EEPROM driver shall abort the job, set the EEPROM state to MEMIF_IDLE and the job result to the corresponding value: MEMIF_JOB_CANCELED, according to EEP028. If configured, the notification Eep_JobErrorNotification() shall be called directly inside the Eep_Cancel() function.

EEP084: The configuration parameter EEP_JOB_CALL_CYCLE (see EEP039) 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.

EEP086: The configuration parameter EEP_USE_INTERRUPTS (see EEP085) shall switch between interrupt and polling controlled job processing if this is supported by the EEPROM hardware. 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. 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: If the SPI access fails, the EEPROM driver shall behave as specified in EEP068. Additionally, the error `EEP_E_COM_FAILURE` shall be reported.

EEP052: In normal EEPROM mode, the EEPROM driver shall access the external EEPROM by usage of SPI channels that are configured for normal access to the SPI EEPROM.

EEP053: The parameter `EEP_NORMAL_READ_BLOCK_SIZE` shall fit to the number of bytes that are readable in normal mode.

EEP055: In fast EEPROM mode, the EEPROM driver shall access the external EEPROM by usage of SPI channels that are configured for burst access to the SPI EEPROM.

EEP073: The parameter `EEP_FAST_READ_BLOCK_SIZE` shall fit to the number of bytes that are readable in burst mode.

7.6 Processing of read jobs

EEP050: The EEPROM driver shall provide two different read modes:

- normal mode
- fast mode

This requirement is valid for all internal EEPROMs. This requirement is valid for those external EEPROMs that provide burst mode. If an external EEPROM does not support this feature, the EEPROM driver shall accept a selection of fast mode, but behave the same as in normal mode (don't care of mode parameter).

EEP051: In normal EEPROM mode, the EEPROM driver shall read within one job processing cycle a number of bytes specified by the parameter `EEP_NORMAL_READ_BLOCK_SIZE`.
Example:

- `EEP_NORMAL_READ_BLOCK_SIZE` = 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 EEPROM driver shall read within one job processing cycle a number of bytes specified by the parameter `EEP_FAST_READ_BLOCK_SIZE`.
Example:

- `EEP_FAST_READ_BLOCK_SIZE` = 32

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

For finishing or aborting read jobs see EEP058, EEP114 and EEP068.

7.7 Processing of write jobs

EEP057: The EEPROM driver 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.

EEP096: The EEPROM driver shall provide two different write modes:
- normal mode
- fast mode

This requirement is valid for those external EEPROMs that provide burst mode. If an EEPROM does not support this feature, the EEPROM driver shall accept a selection of fast mode, but behave the same as in normal mode (don't care of mode parameter).

EEP097: In normal EEPROM mode, the EEPROM driver shall write (and erase) within one job processing cycle a number of bytes specified by the parameter `EEP_NORMAL_WRITE_BLOCK_SIZE`.
Example:
- `EEP_NORMAL_WRITE_BLOCK_SIZE` = 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 EEPROM driver shall write (and erase) within one job processing cycle a number of bytes specified by the parameter `EEP_FAST_WRITE_BLOCK_SIZE`.
Example:
- `EEP_FAST_WRITE_BLOCK_SIZE` = 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 programming of that cell shall[1] be skipped. This feature shall be statically configurable via the configuration switch EEP_WRITE_CYCLE_REDUCTION.

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

EEP059: The EEPROM driver shall erase an EEPROM cell before writing to it if this is not done automatically by the EEPROM hardware.

EEP063: The EEPROM driver 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.

EEP090: The EEPROM driver 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.

EEP064: The number of read – modify – write operations during writing a data block shall be kept as small as possible.

For finishing or aborting write jobs see EEP058, EEP114 and EEP068.

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.

7.8 Processing of erase jobs

EEP069: The EEPROM driver shall erase only as many bytes to the EEPROM as supported by the EEPROM hardware within one job processing cycle.

EEP070: The EEPROM driver shall use block erase commands if supported by the EEPROM hardware and if the given parameters (`EepromAddress` and `Length`) are aligned to erasable blocks.

EEP072: The EEPROM driver 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.

For finishing or aborting erase jobs see EEP058, EEP114 and EEP068.

7.9 Processing of compare jobs

For processing of compare jobs, the following EEPROM mode related requirements are applicable: EEP050, EEP051, EEP054.

EEP075: For finishing or aborting compare jobs see EEP058, EEP114 and EEP068. When it is detected during compare job processing that the compared data are not equal, the EEPROM driver shall abort the job, set the EEPROM state to

`MEMIF_IDLE` and the job result to `MEMIF_COMPARE_UNEQUAL`. If configured, the notification `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: Eep.c shall check if the correct version of Eep.h is included. This shall be done by a preprocessor check.

8 API specification

8.1 Imported types

EEP093: Types `Std_VersionInfoType` and `Std_ReturnType` (defined in `Std_Types.h`) shall be used.

EEP080: The following types of the Memory Abstraction (defined in `MemIf_Types.h`) shall be used:

- `MemIf_StatusType`
- `MemIf_JobResultType`
- `MemIf_ModeType`

8.2 Type definitions

8.2.1 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

Type:	`uint8...uint32`	
Range:	`8..32 bit`	Size depends on target platform and EEPROM device.
Description:	This value is used as address offset for accessing EEPROM data. **EEP113**: Each EEPROM device shall have 0 as first address. An EEPROM base address shall added by the driver, if required.	

8.2.3 Eep_LengthType

Type:	`Eep_AddressType`	
Range:	Same as `Eep_AddressType`	Shall be 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

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 (out):	None	--
Return value:	None	--
Description:	**EEP004**: Service for EEPROM initialization. The Initialization function shall initialize all EEPROM relevant registers with the values of the structure referenced by the parameter `ConfigPtr.` **EEP005**: If development error detection is enabled, the parameter `ConfigPtr` shall be checked for not being a NULL pointer. If `ConfigPtr` is a NULL pointer this error shall be reported to the Development Error Tracer with error value `EEP_E_PARAM_CONFIG`. and the initialization shall be aborted. **EEP006**: After having finished the module initialization, the EEPROM state shall be set to `MEMIF_IDLE,` the job result shall be set to `MEMIF_JOB_OK.` **EEP044**: The EEPROM mode shall be set to the configured default mode.	
Caveats:	This service shall not be called during a running operation.	
Configuration:	--	

8.3.2 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_NORMAL`: Normal read access / normal SPI access. `MEMIF_MODE_FAST`: Fast read access / SPI burst access.
Parameters (out):	None	--
Return value:	None	--
Description:	**EEP042**: Service for EEPROM mode selection. This service shall set the EEPROM operation mode to the given mode parameter. The EEPROM state shall be checked according to requirement EEP033.	
Caveats:	This service shall not be called during a running operation.	

Configuration:	--

8.3.3 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 - EepromAddress`
Parameters (out):	`DataBufferPtr`	Pointer to destination data buffer in RAM
Return value:	`E_OK`	read command has been accepted
	`E_NOT_OK`	read command has not been accepted
Description:	**EEP009**: Service for reading from EEPROM. This service 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. **EEP013**: The read job shall be executed asynchronously within the EEPROM job processing function. During job processing a data block of size `Length` shall be read from `EepromAddress` + EEPROM base address to `*DataBufferPtr`. The API parameters shall be checked according to requirement EEP010. The EEPROM state shall be checked according to requirement EEP033.	
Caveats:	The EEPROM shall have been initialized before this service is called. Only one EEPROM job (read/write/erase/compare) can be accepted at the same time.	
Configuration:	--	

8.3.4 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 - EepromAddress`
Parameters (out):	None	--
Return value:	`E_OK`	write command has been accepted
	`E_NOT_OK`	write command has not been accepted
Description:	**EEP014**: Service for writing to EEPROM. This service 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. **EEP015**: The write job shall be executed asynchronously within the EEPROM job processing function. During job processing a data block of size `Length` shall be written from `*DataBufferPtr` to `EepromAddress` + EEPROM base address. The API parameters shall be checked according to requirement EEP010. The EEPROM state shall be checked according to requirement EEP033.	
Caveats:	The EEPROM shall have been initialized before this service is called. Only one EEPROM job (read/write/erase/compare) can be accepted at the same time.	
Configuration:	--	

8.3.5 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 - EepromAddress`
Parameters (out):	None	--
Return value:	`E_OK`	erase command has been accepted
	`E_NOT_OK`	erase command has not been accepted
Description:	**EEP019**: Service for erasing EEPROM sections. This service 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. **EEP020**: The erase job shall executed asynchronously within the EEPROM job processing function. An EEPROM block starting from `EepromAddress` + EEPROM base address of size `Length` shall be erased. The API parameters shall be checked according to requirement EEP010. The EEPROM state shall be checked according to requirement EEP033.
Caveats:	The EEPROM shall have been initialized before this service is called. Only one EEPROM job (read/write/erase/compare) can be accepted at the same time.
Configuration:	--

8.3.6 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 - EepromAddress`
Parameters (out):	None	--
Return value:	`E_OK`	compare command has been accepted
	`E_NOT_OK`	compare command has not been accepted
Description:	**EEP025**: Service for comparing a data block in EEPROM with an EEPROM block in the memory. This service 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. **EEP026**: The compare job shall be executed asynchronously within the EEPROM job processing function. During job processing an EEPROM data block at `EepromAddress` + EEPROM base address of size `Length` shall be compared with a data block at `*DataBufferPtr` of the same length. The API parameters shall be checked according to requirement EEP010. The EEPROM state shall be checked according to requirement EEP033.	
Caveats:	The EEPROM shall have been initialized before this service is called.	

	Only one EEPROM job (read/write/erase/compare) can be accepted at the same time.
Configuration:	--

8.3.7 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 (out):	None	--
Return value:	None	--
Description:	**EEP021**: Service for cancelling a running EEPROM job (read/write/erase/compare). This function shall abort a running job synchronously so that directly after returning from this function a new job (e.g. write) can be requested by the upper layer. **EEP027**: This function shall reset the module state to `MEMIF_IDLE`, and, if configured, it shall directly call the notification function `Eep_JobErrorNotification()` as well. **EEP028**: If the job result has the value `MEMIF_JOB_PENDING`, this function shall set the job result to `MEMIF_JOB_CANCELED`. Otherwise it shall leave the job result unchanged.	
Caveats:	The states and data of the affected EEPROM cells are undefined in case of canceling a write/erase job! Only the NVRAM Manager is authorized to use this function! Cancelling any job on-going in an external EEPROM device might set this one in a blocking state!	
Configuration:	--	

8.3.8 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 (out):	None	--
Return value:	MemIf_StatusType	See document [3]
Description:	**EEP029**: This service shall return the EEPROM status synchronously.	
Caveats:	--	
Configuration:	--	

8.3.9 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 (out):	None	--
Return value:	MemIf_JobResultType	See document [3]
Description:	**EEP024**: This service shall return the result of the last job synchronously.	
Caveats:	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`.	
Configuration:	--	

8.3.10 Eep_GetVersionInfo

Service name:	Eep_GetVersionInfo	
Syntax:	`void Eep_GetVersionInfo ( Std_VersionInfoType *versioninfo )`	
Service ID [hex]:	0x0A	
Sync/Async:	Synchronous	
Reentrancy:	non reentrant	
Parameters (in):	None	--
Parameters (out):	`versioninfo`	Pointer to where to store the version information of this

		module.
Return value:	None	--
Description:	**EEP107**: This service returns the version information of this module. The version information includes: - Module Id - Vendor Id - Vendor specific version numbers (BSW00407). Hint: If source code for caller and callee of this function is available this function should be realized as a macro. The macro should be defined in the modules header file.	
Caveats:	--	
Configuration:	EEP108: This function is pre compile time configurable `On/Off` by the configuration parameter: `EEP_VERSION_INFO_API`	

8.4 Callback notifications

EEP112: 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.

8.5 Scheduled functions

These functions are directly called by Basic Software Scheduler. The following functions shall have no return value and no parameter. All functions shall be non reentrant.

8.5.1 Eep_MainFunction

Service name:	Eep_MainFunction
Service ID [hex]:	0x09
Description:	This function shall perform the processing of the EEPROM jobs (read/write/erase/compare) as soon as the EEPROM hardware has finished the previous job actions (e.g. after a job cancelling, main function waits until the EEPROM hardware becomes available to perform the new job request). **EEP031**: When a job has been initiated, this function has to be called cyclically until the job is finished. **EEP032**: This function may also be called cyclically even if no job is pending. In this case the function returns without action.
Timing:	variable cyclic

Pre condition:	--
Configuration:	EEP_JOB_CALL_CYCLE

8.6 Expected Interfaces

8.6.1 Mandatory Interfaces

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

API function	*Module*	*Description*
`Dem_ReportErrorEvent`	Dem	Production error event reporting

8.6.2 Optional Interfaces

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

API function	*Module*	*Description*	*Configuration parameter (description see chapter 10)*
`Det_ReportError`	Det	Development error notification	`EEP_DEV_ERROR_DETECT`
`Dem_ReportErrorStatus`	Dem	Production error status reporting	Implementation and /or Hardware specific

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: The callback notifications shall be configurable as function pointers within the initialization data structure (`Eep_ConfigType`).

EEP048: The callback notifications shall have no parameters and no return value.

EEP049: If a callback notification is configured as null pointer, no callback shall be executed.

8.6.3.1 <End Job Notification>

EEP045: This callback function shall be called 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

Name:	`< End Job Notification >`	
Syntax:	`void (* Eep_JobEndNotification) (void)`	
Reentrancy:	`Non re-entrant`	
Parameters (in):	`None`	`--`
Parameters (out):	`None`	`--`
Return value:	`None`	`--`
Description:	Callback routine provided by the user to notify the caller that a job has been completed with a positive result.	
Caveats:	This routine might be called on interrupt level, depending on the calling function.	
Configuration:	--	

8.6.3.2 <Error Job Notification>

EEP046: This callback function shall be called 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.

Name:	`< Error Job Notification >`	
Syntax:	`void (* Eep_JobErrorNotification) (void)`	
Reentrancy:	`Non re-entrant`	
Parameters (in):	`None`	`--`
Parameters (out):	`None`	`--`
Return value:	`None`	`--`
Description:	Callback routine provided by the user to notify the caller that a job has been cancelled or aborted with a negative result.	
Caveats:	This routine might be called on interrupt level, depending on the calling function.	
Configuration:	--	

8.7 API parameter checking

EEP010: If the development error detection is enabled for this module, the following API parameter checkings shall be performed within the services `Eep_Read()`, `Eep_Write()`, `Eep_Compare()` and `Eep_Erase()`. Detected errors shall be reported to the Development Error Tracer (see EEP001). The called service shall be rejected with `E_NOT_OK`.

EEP016: `DataBufferPtr` shall not be a NULL pointer. Related error value: `EEP_E_PARAM_DATA`.

EEP017: `EepromAddress` shall be within the valid EEPROM address range. Related error value: `EEP_E_PARAM_ADDRESS`.

EEP018: `Length` shall be within the specified minimum and maximum values:

- Min.: 1
- Max.: EEP_SIZE – EepromAddress

Related error value: EEP_E_PARAM_LENGTH.

8.8 EEPROM state checking

EEP033: If the development error detection is enabled for this module, the services 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 idle, the called service shall

- report either the error EEP_E_BUSY or EEP_E_UNINIT according to the EEPROM state to the DET (see EEP001)
- reject the service with E_NOT_OK (except Eep_SetMode() because this service has no return value)

9 Sequence diagrams

9.1 Initialization

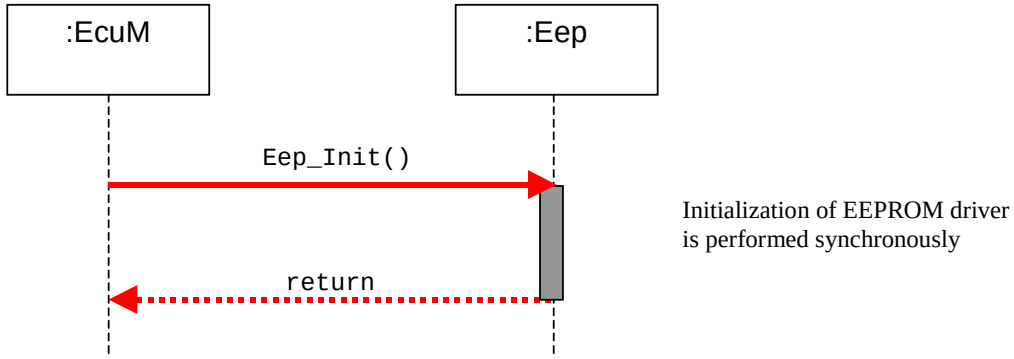

Figure 2

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.

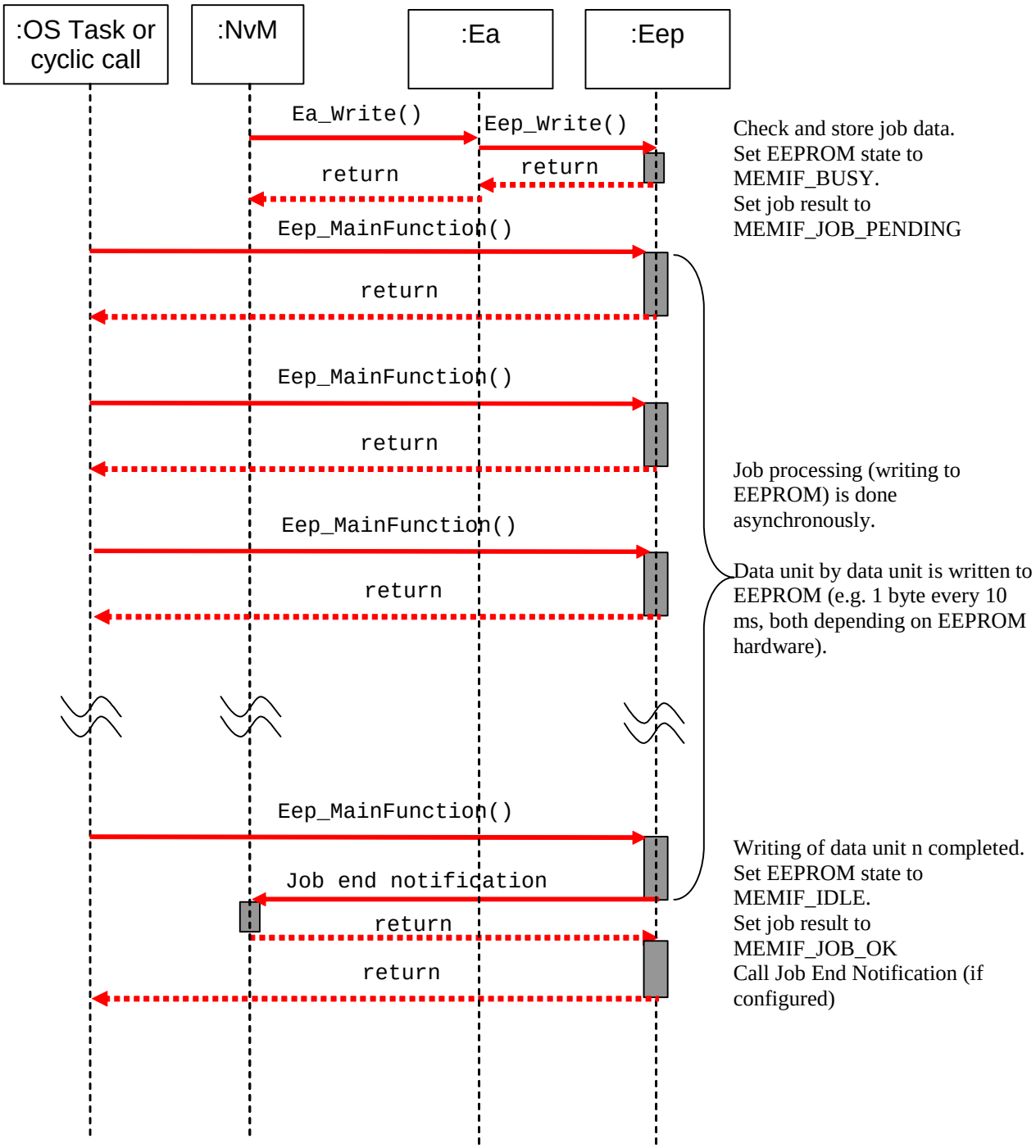
:OS Task or cyclic call
:NvM
:Ea
:Eep
Ea_Write()
Eep_Write()
return
return
Check and store job data.
Set EEPROM state to MEMIF_BUSY.
Set job result to MEMIF_JOB_PENDING
Eep_MainFunction()
return
Eep_MainFunction()
return
Job processing (writing to EEPROM) is done asynchronously.
Eep_MainFunction()
return
Data unit by data unit is written to EEPROM (e.g. 1 byte every 10 ms, both depending on EEPROM hardware).
Eep_MainFunction()
Job end notification
return
return
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

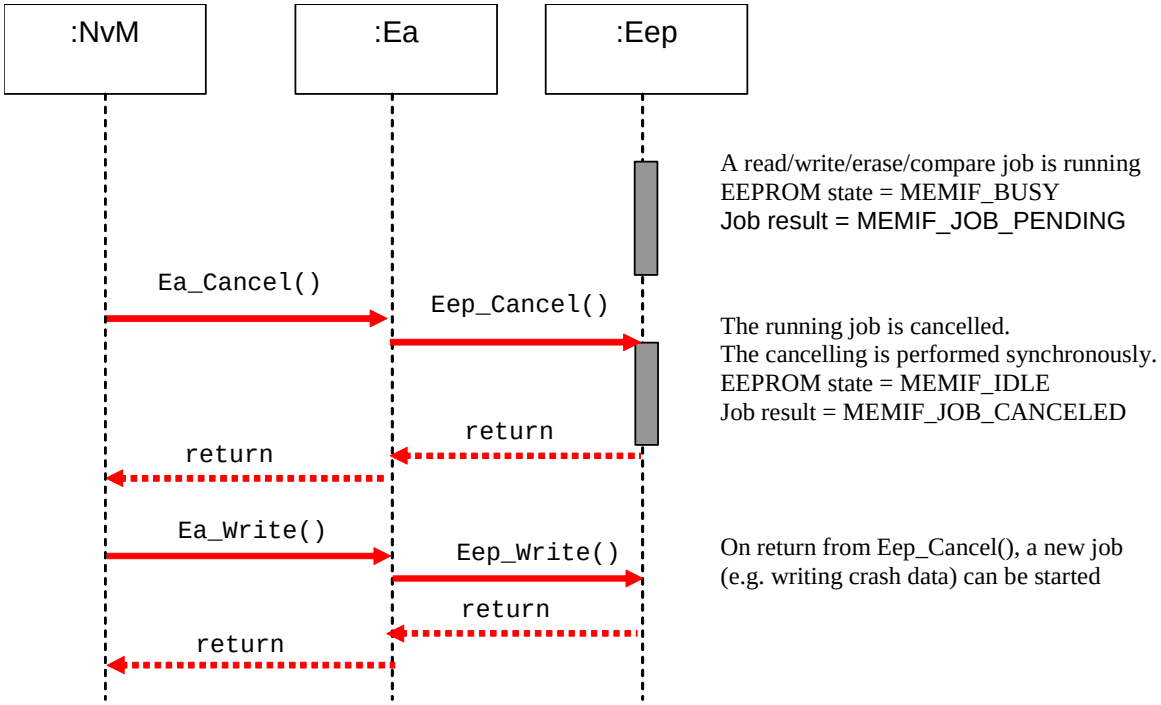

Figure 4

The `Eep_Cancel()` function shall not be called during a running `Eep_MainFunction()` function. This 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 is not interruptable by another task

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) shall be 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 shall be of configuration class *Pre-compile time* or not

Label	***Description***
x	The configuration parameter shall be of configuration class *Pre-compile time*.
--	The configuration parameter shall never be of configuration class *Pre-compile time*.

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

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

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

Label	***Description***
x	The configuration parameter shall be of configuration class *Post Build* and no specific implementation is required.
L	*Loadable* - the configuration parameter shall be of configuration class *Post Build* and only one configuration parameter set resides in the ECU.
M	*Multiple* - the configuration parameter shall be 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 shall never be of configuration class *Post Build*.

10.2 Containers and configuration parameters

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

10.2.1 Variants

EEP109: Variant PC: This variant is limited to pre-compile-configuration parameters only. The intention of this variant is to optimize the parameters configuration for a source code delivery.

EEP110: Variant LT: This variant allows a mix of pre-compile time-, link time-configuration parameters. The intention of this variant is to optimize the parameters configuration for an object code delivery.

10.2.2 EepGeneralConfiguration

SWS Item	EEP085:
Container Name	EepGeneralConfiguration
Description	This container contains the general configuration (parameters) of the EEPROM Driver software module. They shall be located in the module's configuration header file `Eep_Cfg.h`
Configuration Parameters	

Name	`EEP_DEV_ERROR_DETECT`		
Description	Switches the Development Error Detection and Notification `ON` or `OFF`.		
Type	`#define`		
Unit	`--`		
Range	`ON`	enabled	
	`OFF`	disabled	
Configuration Class	***Pre-compile***	x	all Variants
	Link time	--	--
	Post Build	--	--
Scope	Module		
Dependency	None		

Name	`EEP_VERSION_INFO_API`		
Description	Switches the Eep_GetVersionInfo function `ON` or `OFF`.		
Type	`#define`		
Unit	`--`		
Range	`ON`	enabled	
	`OFF`	disabled	
Configuration Class	***Pre-compile***	x	all Variants
	Link time	--	--
	Post Build	--	--
Scope	Module		
Dependency	None		

Name	`EEP_USE_INTERRUPTS`

Description	Switches to activate or deactivate interrupt controlled job processing.		
Type	#define		
Unit	--		
Range	ON	enabled	
	OFF	disabled	
Configuration Class	***Pre-compile***	x	all Variants
	Link time	--	--
	Post Build	--	--
Scope	Module		
Dependency	Usually, this is only supported by some internal EEPROM peripherals.		

Name	EEP_WRITE_CYCLE_REDUCTION		
Description	Switches to activate or deactivate write cycle reduction (EEPROM value is read and compared before being overwritten).		
Type	#define		
Unit	--		
Range	ON	enabled	
	OFF	disabled	
Configuration Class	***Pre-compile***	x	all Variants
	Link time	--	--
	Post Build	--	--
Scope	Module		
Dependency	None		

10.2.3 EepInitConfiguration

SWS Item	EEP039:
Container Name	EepInitConfiguration
Description	This container contains the initialization configuration parameters that shall be located in an external data structure of type Eep_ConfigType. The organization and location of this data structure is left up to the implementer.
Configuration Parameters	

Name	EEP_BASE_ADDRESS		
Description	This parameter is the EEPROM device base address.		
Type	Eep_AddressType		
Unit	--		
Range		--	
Configuration Class	***Pre-compile***	x	Variant PC
	Link time	x	Variant LT
	Post Build	--	--
Scope	Instance		
Dependency	None		

Name	EEP_SIZE		
Description	This parameter is the used size of EEPROM device in bytes.		
Type	Eep_LengthType		
Unit	Bytes		
Range		--	
Configuration Class	***Pre-compile***	x	Variant PC
	Link time	x	Variant LT
	Post Build	--	--
Scope	Instance		
Dependency	None		

Name	EEP_NORMAL_READ_BLOCK_SIZE		
Description	This parameter is the number of bytes read within one job processing cycle in normal mode.		
Type	Eep_LengthType		
Unit	Bytes		
Range		--	
Configuration Class	***Pre-compile***	x	Variant PC
	Link time	x	Variant LT
	Post Build	--	--
Scope	Instance		
Dependency	None		

Name	EEP_NORMAL_WRITE_BLOCK_SIZE		
Description	This parameter is the number of bytes written within one job processing cycle in normal mode.		
Type	Eep_LengthType		
Unit	Bytes		
Range		--	
Configuration Class	***Pre-compile***	x	Variant PC
	Link time	x	Variant LT
	Post Build	--	--
Scope	Instance		
Dependency	This parameter is optional and only available if the hardware allows configuration.		

Name	EEP_FAST_READ_BLOCK_SIZE		
Description	This parameter is the number of bytes read within one job processing cycle in fast mode.		
Type	Eep_LengthType		
Unit	Bytes		
Range		--	
Configuration Class	***Pre-compile***	x	Variant PC
	Link time	x	Variant LT
	Post Build	--	--
Scope	Instance		
Dependency	None		

Name	EEP_FAST_WRITE_BLOCK_SIZE		
Description	This parameter is the number of bytes written within one job processing cycle in fast mode.		
Type	Eep_LengthType		
Unit	Bytes		
Range		--	
Configuration Class	***Pre-compile***	x	Variant PC
	Link time	x	Variant LT
	Post Build	--	--
Scope	Instance		
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).		

Name	EEP_DEFAULT_MODE		
Description	This parameter is the default EEPROM device mode after initialization.		
Type	MemIf_ModeType		
Unit	--		
Range		--	
Configuration Class	***Pre-compile***	x	Variant PC

	Link time	x	Variant LT
	Post Build	--	--
Scope	Instance		
Dependency	None		

Name	`EEP_JOB_CALL_CYCLE`		
Description	This parameter is the call cycle of the job processing function during write/erase operations.		
Type	Integer		
Unit	`Milliseconds (ms)`		
Range		--	
Configuration Class	***Pre-compile***	x	Variant PC
	Link time	x	Variant LT
	Post Build	--	--
Scope	Instance		
Dependency	None		

Name	`EEP_DEFAULT_MODE`		
Description	This parameter is the default EEPROM device mode after initialization.		
Type	Integer		
Unit	`Milliseconds (ms)`		
Range		--	
Configuration Class	***Pre-compile***	x	Variant PC
	Link time	x	Variant LT
	Post Build	--	--
Scope	Instance		
Dependency	None		

Name	`EEP_JOB_END_NOTIFICATION`		
Description	This parameter is a reference to a callback function for positive job result (see EEP045).		
Type	`Function pointer`		
Unit	`--`		
Range	`--`	--	
Configuration Class	***Pre-compile***	x	Variant PC
	Link time	x	Variant LT
	Post Build	--	--
Scope	ECU		
Dependency	none		

Name	`EEP_JOB_ERROR_NOTIFICATION`		
Description	This parameter is a reference to a callback function for negative job result (see EEP046).		
Type	`Function pointer`		
Unit	`--`		
Range	`--`	--	
Configuration Class	***Pre-compile***	x	Variant PC
	Link time	x	Variant LT
	Post Build	--	--
Scope	ECU		
Dependency	none		

10.2.4 SPI specific extension

EEP094: In case of an external SPI EEPROM device, the following parameters shall also be located or referenced (according to the confguration methodology) in the

external data structure of type `Eep_ConfigType` (see EEP039). 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

10.3 Published parameters

10.3.1 Basic subset

EEP099: The following table specifies parameters that shall be published in the module's header file `Eep.h` or in the module's description file. Published information contains data defined by the implementer of the SW module that does not change when the module is adapted (i.e. configured) to the actual HW/SW environment. It thus contains version and manufacturer information.

EEP038: These published information are provided in the module's description for use by configuration tools. Further hardware / implementation specific parameters can be added if necessary.

SWS Item	EEP111:	
Information elements		
Information element name	***Type / Range***	***Information element description***
EEP_VENDOR_ID	#define/ uint16	Vendor ID of the dedicated implementation of this module according to the AUTOSAR vendor list
EEP_MODULE_ID	#define/uint8	Module ID of this module from Module List
EEP_AR_MAJOR_VERSION	#define/uint8	Major version number of AUTOSAR specification on which the appropriate implementation is based on.
EEP_AR_MINOR_VERSION	#define/uint8	Minor version number of AUTOSAR specification on which the appropriate implementation is based on.
EEP_AR_PATCH_VERSION	#define/uint8	Patch level version number of AUTOSAR specification on which the appropriate implementation is based on.
EEP_SW_MAJOR_VERSION	#define/uint8	Major version number of the vendor specific implementation of the module. The numbering is vendor specific.
EEP_SW_MINOR_VERSION	#define/uint8	Minor version number of the vendor specific implementation of the module. The numbering is vendor specific.
EEP_SW_PATCH_VERSION	#define/uint8	Patch level version number of the vendor specific implementation of the module. The numbering is vendor specific.
EEP_ERASE_TIME	µs	Time for erasing one EEPROM data unit.
EEP_WRITE_TIME	µs	Time for writing one EEPROM data unit.
EEP_READ_UNIT_SIZE	#define/uint8	Size of smallest readable EEPROM data unit in bytes
EEP_WRITE_UNIT_SIZE	#define/uint8	Size of smallest writeable EEPROM data unit in bytes
EEP_ERASE_UNIT_SIZE	#define/uint8	Size of smallest eraseable EEPROM data unit in bytes
EEP_ERASE_VALUE	EEP_READ_UNIT_SIZE	Value of an erased EEPROM cell
EEP_TOTAL_SIZE	Eep_LengthType	Total size of EEPROM in bytes
EEP_ALLOWED_WRITE_CYCLES	-- / --	Specified maximum number of write cycles under worst case conditions of specific EEPROM hardware (e.g. +90°C)
EEP_MINIMUM_ADDRESS_TYPE	-- / --	Minimum expected size of Eep_AddressType
EEP_MINIMUM_LENGTH_TYPE	-- / --	Minimum expected size of Eep_LengthType

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.

10.4 Configuration example

This chapter shall provide a better understanding of how and where configuration parameters are defined and used. Please be aware that the used API functions of the SPI are examples and may change. The valid reference is always the current SPI SWS. For the following use case a detailed implementation and configuration example is given:

Use case:

- Driver for external SPI EEPROM, supplied as object code
- Usage of SPI Handler/Driver IB and EB channels (internal buffers for commands and EEPROM addresses, external buffers for data)
- Configuration of SPI write command

10.4.1 Configuration of SPI parameters

1. Get SPI write command and EEPROM write sequence format from specification of external EEPROM device
2. Define SPI Channels for EEPROM write command, EEPROM address and data to be written, e.g.
    a. `EEP_SPI_CH_COMMAND`
    b. `EEP_SPI_CH_ADDRESS`
    c. `EEP_SPI_CH_DATA`
3. Define SPI Channel attributes based on SPI SWS (SPI Channel configuration)
4. Define SPI Jobs that contain the SPI Channels in the correct order (that is expected by the EEPROM device). One job is a list of SPI Channels (e.g. an array)
5. Define SPI Job attributes based on SPI SWS (SPI Job configuration) including symbolic job names, e.g. `EEP_SPI_JOB_WRITE`
6. Define SPI Sequences that contain SPI Jobs in the correct order. One sequence is a list of SPI Jobs (e.g. an array). In this example, the SPI Sequence contains only the EEPROM write job.
7. Define SPI Sequence attributes based on SPI SWS (SPI Job configuration) including symbolic sequence names, e.g. `EEP_SPI_SEQ_WRITE`
8. Publish all defined attributes for SPI usage in XML description file of EEPROM according to SPI SWS

10.4.2 Generation of SPI and EEPROM configuration data

Within the SPI configuration process, values for instantiation of the EEPROM configuration structure (Eep_ConfigType) are generated based on the attributes published by the EEPROM driver (and other SPI users).

Example:
In file Spi_Cfg.h:

```
#define EEP_SPI_CH_COMMAND      23
#define EEP_SPI_CH_ADDRESS      24
#define EEP_SPI_CH_WRITE_DATA   25
```

10.4.3 Instantiation of EEPROM configuration data

The file that contains the instantiation (=definition) of the EEPROM configuration structure includes Spi_Cfg.h and uses the defined values for initialization of structure elements.

Example:

```
const Eep_ConfigType EepConfigData =
{
   …
   EepCmdChannel    = EEP_SPI_CH_COMMAND,
   EepAdrChannel    = EEP_SPI_CH_ADDRESS,
   …
   EepWriteSequence = EEP_SPI_SEQ_WRITE,
   …
};
```

10.4.4 SPI API usage

During EEPROM initialization, a pointer to the configuration data structure in ROM is passed to the EEPROM:

```
Eep_Init(&EepConfigData);
```

Within the Eep_Init() function, the pointer is saved to a static variable, e.g. MyEepConfig. This is necessary to have access to the configuration after the EEPROM initialization has been finished.

For executing an EEPROM write job, the EEPROM driver first has to transfer the information for executing the EEPROM write command to the SPI Handler/Driver. The SPI Channel parameters are taken from the EEPROM configuration data structure in ROM. The parameters EepAddress, DataBufferPtr and Length are passed to the EEPROM driver via the Eep_Write() function.

```
(void)Spi_Write(MyEepConfig->EepCmdChannel,&EepWriteCommand);
(void)Spi_Write(MyEepConfig->EepAdrChannel,&EepAddress);
(void)Spi_SetupBuffers
(
    MyEepConfig->EepDataChannel,
    DataBufferPtr,
    NULL_PTR,
    Length
);
```

After that, the transmission of data to EEPROM can be started:

```
(void)Spi_Transmit(MyEepConfig->EepWriteSequence);
```

11 Changes to Release 1

11.1 Deleted SWS Items

SWS Item	*Rationale*
EEP061	Deleted to fix the Bugzilla issue #9919
EEP062	Deleted to fix the Bugzilla issue #9919

11.2 Replaced SWS Items

SWS Item of Release 1	*replaced by SWS Item*	*Rationale*
EEP001	EEP001, EEP100	To split the old requirement into two requirements to fit to the new SWS template.

11.3 Changed SWS Items

SWS Item	*Rationale*
EEP085	Change to fulfill the new SWS template.
EEP039	Change to fulfill the new SWS template.
EEP038	Change to fulfill the new SWS template.
EEP099	Change to fulfill the new SWS template.
EEP080	Change to fulfill the new Memory Abstraction architecture
EEP091	Change to independent of published parameters naming.
EEP033	Change to fix the bug #5621 in bugzilla.
EEP084	Change according to M. Huber feedbacks.
EEP030	Change according to the decision coming from Bugzilla issue #9474
EEP027	Change according to the decision coming from Bugzilla issue #9474

11.4 Added SWS Items

SWS Item	*Rationale*
EEP101	New item to fullfil the required code file structure.
EEP102	New item to describe the relationship with the Dem module.
EEP103	New item to describe Dem Ids allocation rules.
EEP104	New requirement to the production errors detection.
EEP105	New requirement to the production errors detection.
EEP106	New requirement to the production errors detection.
EEP107	New item for Eep_GetVersionInfo service description.
EEP108	New item for Eep_GetVersionInfo configuration rules.
EEP109	New requirement dedicated to variants.
EEP110	New requirement dedicated to variants.
EEP111	New requirement to published parameters.
EEP112	New requirement regarding to provide callback notifications to other modules.
EEP113	New requirement due to changes of Memory Abstraction architecture
EEP114	New requirement linked to the new behavior of cancel functionnality