

<b>Document Title</b>	Specification of Module EEPROM Driver
<b>Document Owner</b>	AUTOSAR GbR
<b>Document Responsibility</b>	AUTOSAR GbR
<b>Document Identification No</b>	021
<b>Document Classification</b>	Standard

<b>Document Version</b>	2.2.0
<b>Document Status</b>	Final
<b>Part of Release</b>	3.0
<b>Revision</b>	0001

<b>Document Change History</b>			
<b>Date</b>	<b>Version</b>	<b>Changed by</b>	<b>Change Description</b>
12.12.2007	2.2.0	AUTOSAR Administration	<ul style="list-style-type: none"> <li>• Minor rewording of requirement (EEP005).</li> <li>• Introduction of new requirements (EEP161 and EEP162) for NULL_PTR check.</li> <li>• Updates to EEP028 and Figure 4 to correct spelling of MEMIF_JOB_CANCELLED</li> <li>• Document meta information extended</li> <li>• Small layout adaptations made</li> </ul>
31.01.2007	2.1.0	AUTOSAR Administration	<ul style="list-style-type: none"> <li>• Constant name correction</li> <li>• Limitation of erase cycles</li> <li>• Link-time configuration versus config pointer check</li> <li>• Job result for compare jobs is not specified</li> <li>• Legal disclaimer revised</li> <li>• Release Notes added</li> <li>• "Advice for users" revised</li> <li>• "Revision Information" added</li> </ul>
25.04.2006	2.0.0	AUTOSAR Administration	<p>Document structure adapted to common Release 2.0 SWS Template.</p> <ul style="list-style-type: none"> <li>• adaptation to the new memory abstraction architecture</li> <li>• cancel function now asynchronous</li> </ul> <p>deletion of two specifications elements that could lead to a missinterpretation of the described "write-cycle-reduction" functionality</p>

30.06.2005	1.0.0	AUTOSAR Administration	Initial Release
------------	-------	---------------------------	-----------------

Page left intentionally blank

## Disclaimer

**Any use** of these specifications requires membership within the AUTOSAR Development Partnership or an agreement with the AUTOSAR Development Partnership. The AUTOSAR Development Partnership will not be liable for any use of these specifications.

Following the completion of the development of the AUTOSAR specifications commercial exploitation licenses will be made available to end users by way of written License Agreement only.

No part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from the publisher.

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

Copyright © 2004-2007 AUTOSAR Development Partnership. All rights reserved.

## 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 .....	7
2	Acronyms and abbreviations .....	8
3	Related documentation.....	9
3.1	Input documents.....	9
3.2	Related standards and norms .....	10
4	Constraints and assumptions .....	11
4.1	Limitations .....	11
4.2	Applicability to car domains.....	11
4.3	Applicability to safety related environments .....	11
5	Dependencies to other modules.....	12
5.1	File structure .....	12
6	Requirements traceability .....	14
7	Functional specification .....	20
7.1	General behavior.....	20
7.2	Error classification.....	20
7.3	Error detection.....	21
7.3.1	API parameter checking.....	21
7.3.2	EEPROM state checking.....	21
7.4	Error notification .....	22
7.5	Processing of jobs – general requirements .....	22
7.6	Processing of read jobs.....	23
7.7	Processing of write jobs .....	24
7.8	Processing of erase jobs .....	25
7.9	Processing of compare jobs .....	25
7.10	Version check.....	26
8	API specification.....	27
8.1	Imported types.....	27
8.2	Type definitions .....	27
8.2.1	Eep_ConfigType .....	27
8.2.2	Eep_AddressType.....	27
8.2.3	Eep_LengthType.....	27
8.3	Function definitions .....	28
8.3.1	Eep_Init.....	28
8.3.2	Eep_SetMode .....	28
8.3.3	Eep_Read .....	29
8.3.4	Eep_Write .....	30
8.3.5	Eep_Erase .....	31
8.3.6	Eep_Compare .....	32
8.3.7	Eep_Cancel.....	33
8.3.8	Eep_GetStatus.....	34
8.3.9	Eep_GetJobResult.....	34
8.3.10	Eep_GetVersionInfo.....	35

8.4	Callback notifications.....	35
8.5	Scheduled functions .....	36
8.5.1	Eep_MainFunction .....	36
8.6	Expected Interfaces.....	36
8.6.1	Mandatory Interfaces .....	37
8.6.2	Optional Interfaces .....	37
8.6.3	Configurable interfaces .....	37
8.6.3.1	<End Job Notification> .....	37
8.6.3.2	<Error Job Notification> .....	38
9	Sequence diagrams .....	39
9.1	Initialization .....	39
9.2	Read/write/erase/compare .....	39
9.3	Cancelation of a running job.....	41
10	Configuration specification .....	42
10.1	How to read this chapter .....	42
10.1.1	Configuration and configuration parameters .....	42
10.1.2	Containers.....	42
10.1.3	Specification template for configuration parameters .....	42
10.2	Containers and configuration parameters .....	44
10.2.1	Variants.....	44
10.2.2	Eep.....	44
10.2.3	EepGeneral.....	44
10.2.4	EepInitConfiguration.....	46
10.2.5	EepExternalDriver .....	48
10.2.6	SPI specific extension .....	49
10.3	Published parameters .....	49
10.3.1	Basic subset.....	49
10.3.2	SPI specific extension .....	50
10.3.3	EepPublishedInformation .....	50
10.4	Configuration example .....	52
10.4.1	Configuration of SPI parameters.....	52
10.4.2	Generation of SPI and EEPROM configuration data.....	53
10.4.3	Instantiation of EEPROM configuration data.....	53
10.4.4	SPI API usage.....	53
11	Changes to Release 1 .....	55
11.1	Deleted SWS Items .....	55
11.2	Replaced SWS Items .....	55
11.3	Changed SWS Items.....	55
11.4	Added SWS Items .....	55
12	Changes during SWS Improvements by Technical Office .....	56
12.1	Deleted SWS Items .....	56
12.2	Replaced SWS Items .....	56
12.3	Changed SWS Items.....	56
12.4	Added SWS Items .....	56

## 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.

<b>Acronym:</b>	<b>Description:</b>
Data block	<p>A data block may contain 1..n bytes and is used within the API of the EEPROM driver.</p> <p>Data blocks are passed with</p> <ul style="list-style-type: none"> <li>• Address offset in EEPROM</li> <li>• Pointer to memory location</li> <li>• Length</li> </ul> <p>to the EEPROM driver.</p>
Data unit	<p>The smallest data entity in EEPROM. The entities may differ for read/write/erase operation.</p> <p>Example 1: Motorola STAR12                      Read: 1 byte                      Write: 2 bytes                      Erase: 4 bytes</p> <p>Example 2: external SPI EEPROM device                      Read/Write/Erase: 1 byte</p>
Normal mode Burst mode	<p>Some external SPI EEPROM devices provide the possibility of different access modes:</p> <ul style="list-style-type: none"> <li>• 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.</li> <li>• 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.</li> </ul>
EEPROM cell	Smallest physical unit of an EEPROM device that holds the data. Usually 1 byte.

<b>Abbreviation:</b>	<b>Description:</b>
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://svn2.autosar.org/repos2/22\\_Releases](https://svn2.autosar.org/repos2/22_Releases)  
AUTOSAR\_LayeredSoftwareArchitecture.pdf
- [2] General Requirements on Basic Software Modules,  
[https://svn2.autosar.org/repos2/22\\_Releases](https://svn2.autosar.org/repos2/22_Releases)  
AUTOSAR\_SRS\_General.pdf
- [3] Specification of Memory Abstraction Interface  
[https://svn2.autosar.org/repos2/22\\_Releases](https://svn2.autosar.org/repos2/22_Releases)  
AUTOSAR\_SWS\_Mem\_AbstractionInterface.pdf
- [4] Specification of SPI Handler/Driver  
[https://svn2.autosar.org/repos2/22\\_Releases](https://svn2.autosar.org/repos2/22_Releases)  
AUTOSAR\_SWS\_SPI\_HandlerDriver.pdf
- [5] Specification of ECU Configuration  
[https://svn2.autosar.org/repos2/22\\_Releases](https://svn2.autosar.org/repos2/22_Releases)  
AUTOSAR\_ECU\_Configuration.pdf
- [6] Requirements on EEPROM Driver  
[https://svn2.autosar.org/repos2/22\\_Releases](https://svn2.autosar.org/repos2/22_Releases)  
AUTOSAR\_SRS\_EEPROM\_Driver.pdf
- [7] Specification of Development Error Tracer  
[https://svn2.autosar.org/repos2/22\\_Releases](https://svn2.autosar.org/repos2/22_Releases)  
AUTOSAR\_SWS\_Development\_Error\_Tracer.pdf
- [8] Specification of Diagnostics Event Manager  
[https://svn2.autosar.org/repos2/22\\_Releases](https://svn2.autosar.org/repos2/22_Releases)  
AUTOSAR\_SWS\_DEM
- [9] AUTOSAR Glossary  
[https://svn2.autosar.org/repos2/22\\_Releases](https://svn2.autosar.org/repos2/22_Releases)  
AUTOSAR\_Glossary.pdf
- [10] Specification of MCU Driver  
[https://svn2.autosar.org/repos2/22\\_Releases](https://svn2.autosar.org/repos2/22_Releases)  
AUTOSAR\_SWS\_MCU\_Driver
- [11] AUTOSAR Basic Software Module Description Template  
[https://svn2.autosar.org/repos2/22\\_Releases](https://svn2.autosar.org/repos2/22_Releases)  
AUTOSAR\_BSW\_Module\_Description.pdf

## 3.2 Related standards and norms

[12] 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).

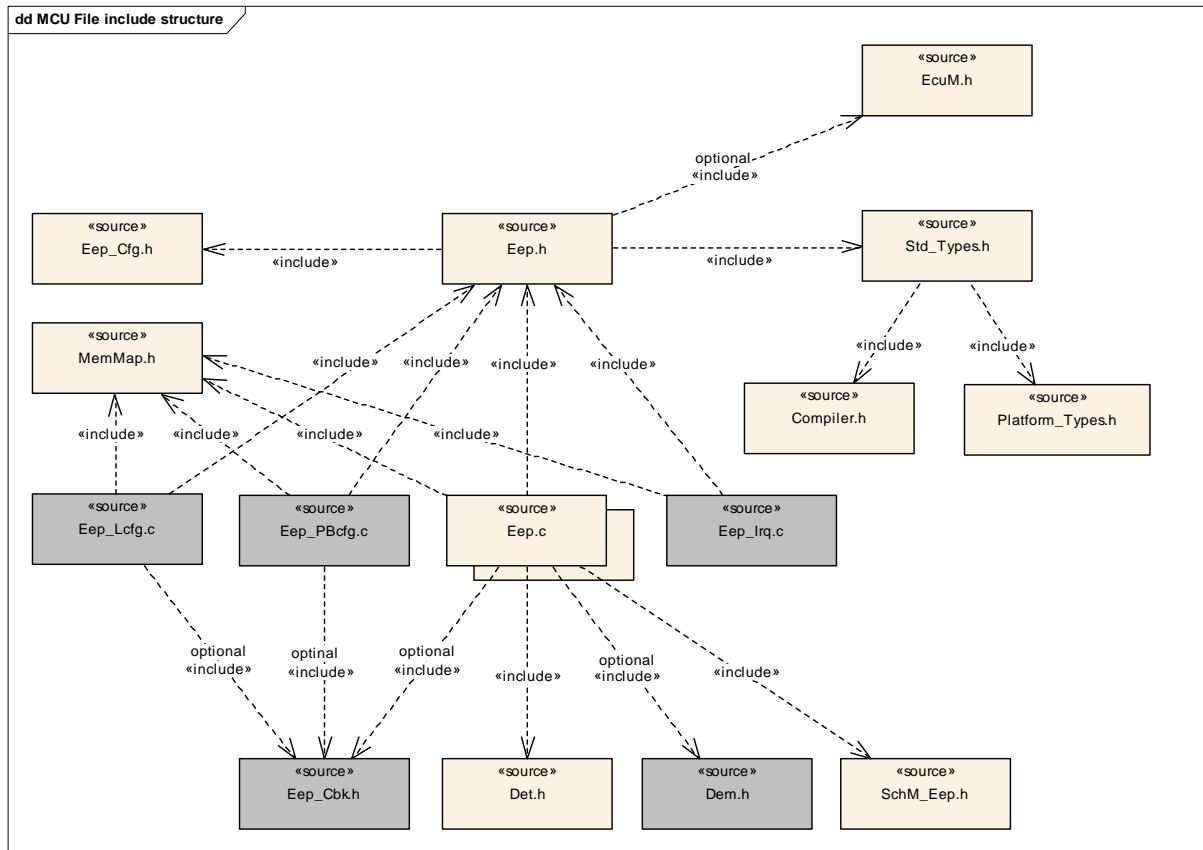
EEPROM driver is part of Memory Abstraction Architecture and for this reason some types depend on Memory Interface (MemIf) module.

### 5.1 File structure

**EEP159:** The module Eep shall provide a file Eep\_Lcfg.c containing all link time and parameters

**EEP160:** The module Eep shall provide a file Eep\_PBcfg.c containing all post build time configurable parameters.

**EEP083:** The module Eep shall adhere to the folloeing include file structure:



**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

**ECP102:** The Eep 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\_IntErrld.h.

## 6 Requirements traceability

Document: AUTOSAR Requirements on Basic Software, general

<b>Requirement</b>	<b>Satisfied by</b>
[BSW00344] Reference to link-time configuration	<a href="#">EEP039</a>
[BSW00404] Reference to post build time configuration	<a href="#">EEP039</a> , <a href="#">EEP110</a>
[BSW00405] Reference to multiple configuration sets	Chapter 10.2
[BSW00345] Pre-compile-time configuration	<a href="#">EEP085</a>
[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. <a href="#">EEP085</a>
[BSW170] Data for reconfiguration of AUTOSAR SW-Components	Not applicable (requirement on SW Component)
[BSW00380] Separate C-Files for configuration parameters	<a href="#">EEP101</a>
[BSW00419] Separate C-Files for pre-compile time configuration parameters	<a href="#">EEP101</a>
[BSW00381] Separate configuration header file for pre-compile time parameters	<a href="#">EEP085</a>
[BSW00412] Separate H-File for configuration parameters	<a href="#">EEP083</a>
[BSW00383] List dependencies of configuration files	Chapters 5, 10.2.6 and 10.3.2
[BSW00384] List dependencies to other modules	Chapter 5, <a href="#">EEP102</a>
[BSW00387] Specify the configuration class of callback function	Chapters 8.4 and 8.6.3
[BSW00388] Introduce containers	<a href="#">EEP085</a> , <a href="#">EEP039</a>
[BSW00389] Containers shall have names	<a href="#">EEP085</a> , <a href="#">EEP039</a>
[BSW00390] Parameter content shall be unique within the module	<a href="#">EEP085</a> , <a href="#">EEP039</a> , <a href="#">EEP094</a> , <a href="#">EEP095</a> , <a href="#">EEP111</a>
[BSW00391] Parameter shall have unique names	<a href="#">EEP085</a> , <a href="#">EEP039</a> , <a href="#">EEP094</a> , <a href="#">EEP095</a> , <a href="#">EEP111</a>
[BSW00392] Parameters shall have a type	<a href="#">EEP085</a> , <a href="#">EEP039</a> , <a href="#">EEP111</a>
[BSW00393] Parameters shall have a range	<a href="#">EEP085</a> , <a href="#">EEP039</a>
[BSW00394] Specify the scope of the parameters	<a href="#">EEP085</a> , <a href="#">EEP039</a>
[BSW00395] List the required parameters (per parameter)	<a href="#">EEP085</a> , <a href="#">EEP039</a>
[BSW00396] Configuration classes	<a href="#">EEP085</a> , <a href="#">EEP039</a> , <a href="#">EEP109</a> , <a href="#">EEP110</a>
[BSW00397] Pre-compile-time parameters	<a href="#">EEP109</a> , <a href="#">EEP085</a>
[BSW00398] Link-time parameters	<a href="#">EEP110</a> , <a href="#">EEP039</a> , <a href="#">EEP094</a>
[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	<a href="#">EEP099</a> , <a href="#">EEP038</a> , <a href="#">EEP111</a> , <a href="#">EEP095</a>
[BSW00375] Notification of wake-up reason	Not applicable (the EEPROM does not cause any wake-ups)
[BSW101] Initialization interface	<a href="#">EEP004</a>

[BSW00416] Sequence of Initialization	Not applicable (this is a general software integration requirement)
[BSW00406] Check module initialization	<a href="#">EEP033</a> , <a href="#">EEP006</a>
[BSW168] Diagnostic Interface of SW components	Not applicable (no use case)
[BSW00407] Function to read out published parameters	<a href="#">EEP107</a> , <a href="#">EEP108</a>
[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	<a href="#">EEP039</a> , 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	<a href="#">EEP000</a> , <a href="#">EEP003</a>
[BSW00338] Reporting of development errors	<a href="#">EEP001</a>
[BSW00369] Do not return development error codes via API	<a href="#">EEP001</a> , <a href="#">EEP033</a>
[BSW00339] Reporting of production relevant error status	<a href="#">EEP002</a> , Chapter 8.6.2
[BSW00421] Reporting of production relevant error events	<a href="#">EEP002</a> , 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	<a href="#">EEP005</a> , <a href="#">EEP016</a> , <a href="#">EEP017</a> , <a href="#">EEP018</a>
[BSW004] Version check	<a href="#">EEP091</a>
[BSW00409] Header files for production code error IDs	<a href="#">EEP103</a>
[BSW00385] List possible error notifications	<a href="#">EEP000</a> , <a href="#">EEP003</a>
[BSW00386] Configuration for detecting an error	<a href="#">EEP001</a>
[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	<a href="#">EEP083</a>
[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, <a href="#">EEP080</a>
[BSW00350] Development error detection keyword	<a href="#">EEP001</a>
[BSW00408] Configuration parameter naming convention	Chapter 10.2
[BSW00410] Compiler switches shall have defined values	<a href="#">EEP085</a>
[BSW00411] Get version info keyword	<a href="#">EEP108</a>
[BSW00346] Basic set of module files	
[BSW158] Separation of configuration from implementation	<a href="#">EEP101</a> , <a href="#">EEP039</a> , <a href="#">EEP085</a>
[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
[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	<a href="#">EEP093</a> , Chapter 8.3
[BSW00377] Module specific API return types	<a href="#">EEP080</a> , Chapters 8.3.8 and 8.3.9
[BSW00304] AUTOSAR integer data types	Chapters 5, 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
[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	Chapter 8.6.3.1 and 8.6.3.2
[BSW00360] Parameters of callback functions	Chapter 8.6.3.1 and 8.6.3.2
[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	<a href="#">EEP099</a> , <a href="#">EEP111</a>
[BSW00379] Module identification	<a href="#">EEP099</a> , <a href="#">EEP111</a>
[BSW003] Version identification	<a href="#">EEP099</a> , <a href="#">EEP111</a>
[BSW00318] Format of module version numbers	<a href="#">EEP111</a>
[BSW00321] Enumeration of module version numbers	<a href="#">EEP111</a>
[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

<b>Requirement</b>	<b>Satisfied by</b>
[BSW12263] Object code compatible configuration concept	<a href="#">EEP039</a>
[BSW12056] Configuration of notification mechanisms	<a href="#">EEP039</a> , <a href="#">EEP047</a> , <a href="#">EEP049</a>
[BSW12267] Configuration of wake-up sources	Not applicable (the EEPROM does not cause any wake-ups)
[BSW12057] Driver module initialization	<a href="#">EEP004</a>
[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	<a href="#">EEP004</a>
[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	<a href="#">EEP029</a> , <a href="#">EEP024</a> , <a href="#">EEP047</a> , <a href="#">EEP045</a> , <a href="#">EEP046</a>
[BSW12155] Prototypes of callback functions	Chapter 8.6.3.1 and 8.6.3.2
[BSW12169] Control of operation mode	Covered by [BSW12156] EEPROM mode selection function

<b>Requirement</b>	<b>Satisfied by</b>
[BSW12063] Raw value mode	Not applicable (no I/O functionality)
[BSW12075] Use of application buffers	<a href="#">EEP037</a>
[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	<a href="#">EEP033</a>
[BSW12448] Behavior after development error detection	<a href="#">EEP001</a> , <a href="#">EEP005</a> , <a href="#">EEP033</a> , <a href="#">EEP016</a> , <a href="#">EEP017</a> , <a href="#">EEP018</a>
[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

<b>Requirement</b>	<b>Satisfied by</b>
[BSW096] EEPROM driver static configuration	<a href="#">EEP039</a>
[BSW12071] Publication of EEPROM properties	<a href="#">EEP038</a>
[BSW087] EEPROM read function	<a href="#">EEP009</a> , <a href="#">EEP013</a>
[BSW088] EEPROM write function	<a href="#">EEP014</a> , <a href="#">EEP015</a> , <a href="#">EEP063</a> , <a href="#">EEP090</a>
[BSW089] EEPROM erase function	<a href="#">EEP019</a> , <a href="#">EEP020</a> , <a href="#">EEP070</a> , <a href="#">EEP072</a>
[BSW12091] EEPROM compare function	<a href="#">EEP025</a> , <a href="#">EEP026</a>
[BSW090] EEPROM cancel function	<a href="#">EEP021</a> , <a href="#">EEP027</a> , <a href="#">EEP028</a>
[BSW091] EEPROM status function	<a href="#">EEP029</a>
[BSW12156] EEPROM mode selection function	<a href="#">EEP042</a> , <a href="#">EEP130</a> , <a href="#">EEP132</a>
[BSW092] EEPROM write cycle reduction	<a href="#">EEP060</a> , <a href="#">EEP064</a>
[BSW094] EEPROM segmentation handling	<a href="#">EEP063</a> , <a href="#">EEP072</a> , <a href="#">EEP070</a> , <a href="#">EEP090</a>
[BSW095] EEPROM job management	<a href="#">EEP036</a> , <a href="#">EEP033</a>
[BSW12047] EEPROM job processing function	<a href="#">EEP030</a>
[BSW12157] Job processing – normal mode	<a href="#">EEP051</a> , <a href="#">EEP052</a> , <a href="#">EEP053</a>
[BSW12072] Job processing – fast mode	<a href="#">EEP054</a> , <a href="#">EEP055</a> , <a href="#">EEP055</a> , <a href="#">EEP073</a>
[BSW12050] EEPROM job processing execution time	<a href="#">EEP057</a> , <a href="#">EEP069</a> , <a href="#">EEP051</a> , <a href="#">EEP054</a>
[BSW12051] Functional scope	Chapter 1, <a href="#">EEP088</a>
[BSW12164] SPI channel configuration	<a href="#">EEP039</a>
[BSW12124] SPI access modes	<a href="#">EEP052</a> , <a href="#">EEP053</a> , <a href="#">EEP055</a> , <a href="#">EEP073</a> , <a href="#">EEP039</a>

## 7 Functional specification

### 7.1 General behavior

**EEP088:** The Eep SWS shall be valid both for internal and external EEPROMs.

**EEP035:** The Eep SWS shall define asynchronous services for EEPROM operations (read/write/erase/compare).

**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.

**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.

### 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`).

**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
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 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.

## 7.3 Error detection

**EEP001:** The Eep module shall allow the detection of development errors to be switched on or off at pre-compile time through the configuration parameter EepDevErrorDetect.

**EEP106:** The Eep module shall not allow the detection of production code errors to 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.

**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.

**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.

### 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)

## 7.4 Error notification

**EEP002:** The Eep module shall report production errors to the Diagnostic Event Manager.

**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 chapter 10.2.3).

## 7.5 Processing of jobs – general requirements

**EEP058:** When a read/write/erase/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`.

**EEP068:** When an error is detected during read/write/erase/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`.

**EEP084:** The configuration parameter `EepJobCallCycle` (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.

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`.

**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. Additionally, the error `EEP_E_COM_FAILURE` shall be reported.

**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.

**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 mode.

**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.

**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 mode.

## 7.6 Processing of read jobs

**EEP130:** The Eep module shall provide two different read modes:

- normal mode
- fast mode

**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).

**EEP051:** In normal EEPROM mode, the Eep module shall read within one job processing cycle a number of bytes specified by the parameter `EepNormalReadBlockSize`.

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`.

Example:

- `EepFastReadBlockSize = 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](#) and [EEP068](#).

## 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.

**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<sup>1</sup> skip the programming of that cell if it is configured to do so through the configuration parameter `EepWriteCycleReduction`.

---

<sup>1</sup> This feature is not mandatory but it depends on the EEPROM hardware manufacturer specification



**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.

**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.

**EEP064:** The Eep module shall keep the number of read – modify – write operations during writing a data block as small as possible.

For finishing or aborting write jobs see [EEP058](#) 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 Eep module shall erase only as many bytes to the EEPROM as supported by the EEPROM hardware within one job processing cycle.

**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.

**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.

For finishing or aborting erase jobs see [EEP058](#) and [EEP068](#).

## 7.9 Processing of compare jobs

For processing of compare jobs, the following EEPROM mode related requirements are applicable: [EEP130](#), [EEP132](#), [EEP051](#), [EEP054](#).

For finishing or aborting compare jobs see [EEP058](#) and [EEP068](#).

**EEP075:** When it is detected during compare job processing that the compared data are not equal, the Eep module shall abort the job, shall set the EEPROM state to MEMIF\_IDLE and shall the job result to MEMIF\_COMPARE\_UNEQUAL. If configured, the Eep module shall call the notification defined in the configuration parameter EepJobErrorNotification.

**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 make sure that the correct version of Eep.h is included in Eep.c through a preprocessor check.

## 8 API specification

### 8.1 Imported types

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

#### EEP138:

Header file	Imported Type
Memlf_Types.h	Memlf_JobResultType
	Memlf_ModeType
	Memlf_StatusType
Dem_Types.h	Dem_EventIdType
Std_Types.h	Std_ReturnType
	Std_VersionInfoType

### 8.2 Type definitions

#### 8.2.1 Eep\_ConfigType

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

#### 8.2.2 Eep\_AddressType

<b>Name:</b>	Eep_AddressType	
<b>Type:</b>	uint8, uint16, uint32	
<b>Range:</b>	8..32 bit	Size depends on target platform and EEPROM device.
<b>Description:</b>	This value is used as address offset for accessing EEPROM data.	

**EEP113:** Each EEPROM device has 0 as first address. The Eep module shall add an EEPROM base address, if required.

#### 8.2.3 Eep\_LengthType

<b>Name:</b>	Eep_LengthType	
<b>Type:</b>	uint8, uint16, uint32	
<b>Range:</b>	Same as Eep_AddressType	Is the same type as Eep_AddressType because of arithmetic operations. Size depends on target platform and EEPROM device.
<b>Description:</b>	Specifies the number of bytes to read/write/erase/compare.	

## 8.3 Function definitions

### 8.3.1 Eep\_Init

#### EEP143:

<b>Service name:</b>	Eep_Init	
<b>Syntax:</b>	void Eep_Init( const Eep_ConfigType* ConfigPtr )	
<b>Service ID[hex]:</b>	0x00	
<b>Sync/Async:</b>	Synchronous	
<b>Reentrancy:</b>	Non Reentrant	
<b>Parameters (in):</b>	ConfigPtr	Pointer to configuration set.
<b>Parameters (inout):</b>	None	
<b>Parameters (out):</b>	None	
<b>Return value:</b>	None	
<b>Description:</b>	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.

**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.

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 Gpt\_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.

**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:

<b>Service name:</b>	Eep_SetMode
<b>Syntax:</b>	void Eep_SetMode( MemIf_ModeType Mode )
<b>Service ID[hex]:</b>	0x01
<b>Sync/Async:</b>	Synchronous
<b>Reentrancy:</b>	Non Reentrant
<b>Parameters (in):</b>	Mode MEMIF_MODE_NORMAL: Normal read access / normal SPI access. MEMIF_MODE_FAST: Fast read access / SPI burst access.
<b>Parameters (inout):</b>	None
<b>Parameters (out):</b>	None
<b>Return value:</b>	None
<b>Description:</b>	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.

**EEP116:** The Eep's user shall not call the function Eep\_SetMode during a running operation.

### 8.3.3 Eep\_Read

**EEP145:**

<b>Service name:</b>	Eep_Read
<b>Syntax:</b>	Std_ReturnType Eep_Read( Eep_AddressType EepromAddress, uint8* DataBufferPtr, Eep_LengthType Length )
<b>Service ID[hex]:</b>	0x02
<b>Sync/Async:</b>	Asynchronous
<b>Reentrancy:</b>	Non Reentrant
<b>Parameters (in):</b>	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
<b>Parameters (inout):</b>	None
<b>Parameters (out):</b>	DataBufferPtr Pointer to destination data buffer in RAM
<b>Return value:</b>	Std_ReturnType E_OK: read command has been accepted E_NOT_OK: read command has not been accepted
<b>Description:</b>	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.

**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.

**EEP117:** The Eep's user shall only call Eep\_Read after the Eep module has 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:

<b>Service name:</b>	Eep_Write	
<b>Syntax:</b>	<pre>Std_ReturnType Eep_Write(     Eep_AddressType EepromAddress,     const uint8* DataBufferPtr,     Eep_LengthType Length )</pre>	
<b>Service ID[hex]:</b>	0x03	
<b>Sync/Async:</b>	Asynchronous	
<b>Reentrancy:</b>	Non Reentrant	
<b>Parameters (in):</b>	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
<b>Parameters (inout):</b>	None	
<b>Parameters (out):</b>	None	
<b>Return value:</b>	Std_ReturnType	E_OK: write command has been accepted E_NOT_OK: write command has not been accepted
<b>Description:</b>	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.

**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.

**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:

<b>Service name:</b>	Eep_Erase	
<b>Syntax:</b>	Std_ReturnType Eep_Erase( Eep_AddressType EepromAddress, Eep_LengthType Length )	
<b>Service ID[hex]:</b>	0x04	
<b>Sync/Async:</b>	Asynchronous	
<b>Reentrancy:</b>	Non Reentrant	
<b>Parameters (in):</b>	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
<b>Parameters (inout):</b>	None	
<b>Parameters (out):</b>	None	
<b>Return value:</b>	Std_ReturnType	E_OK: erase command has been accepted E_NOT_OK: erase command has not been accepted
<b>Description:</b>	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.

**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.

**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:

<b>Service name:</b>	Eep_Compare	
<b>Syntax:</b>	<pre>Std_ReturnType Eep_Compare(     Eep_AddressType EepromAddress,     const uint8* DataBufferPtr,     Eep_LengthType Length )</pre>	
<b>Service ID[hex]:</b>	0x05	
<b>Sync/Async:</b>	Asynchronous	
<b>Reentrancy:</b>	Non Reentrant	
<b>Parameters (in):</b>	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
<b>Parameters (inout):</b>	None	
<b>Parameters (out):</b>	None	
<b>Return value:</b>	Std_ReturnType	E_OK: compare command has been accepted E_NOT_OK: compare command has not been accepted
<b>Description:</b>	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.

**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.

**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:

<b>Service name:</b>	Eep_Cancel
<b>Syntax:</b>	void Eep_Cancel(  )
<b>Service ID[hex]:</b>	0x06
<b>Sync/Async:</b>	Synchronous
<b>Reentrancy:</b>	Non Reentrant
<b>Parameters (in):</b>	None
<b>Parameters (inout):</b>	None
<b>Parameters (out):</b>	None
<b>Return value:</b>	None
<b>Description:</b>	Cancels a running job.

**EEP021:** The function Eep\_Cancel 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:** The function Eep\_Cancel shall reset the module state to MEMIF\_IDLE, and, if configured, shall directly call the notification function defined in EepJobErrorNotification as well.

**EEP028:** The function Eep\_Cancel shall set the job result to MEMIF\_JOB\_CANCELLED, if the job result has the value MEMIF\_JOB\_PENDING. Otherwise it shall leave the job result unchanged.

**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 is not interruptable by another task

The states and data of the affected EEPROM cells are undefined in case of canceling a write/erase job using the service `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:**

<b>Service name:</b>	Eep_GetStatus	
<b>Syntax:</b>	MemIf_StatusType Eep_GetStatus( )	
<b>Service ID[hex]:</b>	0x07	
<b>Sync/Async:</b>	Synchronous	
<b>Reentrancy:</b>	Reentrant	
<b>Parameters (in):</b>	None	
<b>Parameters (inout):</b>	None	
<b>Parameters (out):</b>	None	
<b>Return value:</b>	MemIf_StatusType	See document [3]
<b>Description:</b>	Returns the EEPROM status.	

**EEP029:** The function `Eep_GetStatus` shall return the EEPROM status synchronously.

### 8.3.9 Eep\_GetJobResult

**EEP151:**

<b>Service name:</b>	Eep_GetJobResult	
<b>Syntax:</b>	MemIf_JobResultType Eep_GetJobResult( )	
<b>Service ID[hex]:</b>	0x08	
<b>Sync/Async:</b>	Synchronous	
<b>Reentrancy:</b>	Reentrant	
<b>Parameters (in):</b>	None	

<b>Parameters (inout):</b>	None	
<b>Parameters (out):</b>	None	
<b>Return value:</b>	MemIf_JobResultType	See document [3]
<b>Description:</b>	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.

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:

<b>Service name:</b>	Eep_GetVersionInfo	
<b>Syntax:</b>	<pre>void Eep_GetVersionInfo(     Std_VersionInfoType* versioninfo )</pre>	
<b>Service ID[hex]:</b>	0x0a	
<b>Sync/Async:</b>	Synchronous	
<b>Reentrancy:</b>	Non Reentrant	
<b>Parameters (in):</b>	None	
<b>Parameters (inout):</b>	None	
<b>Parameters (out):</b>	versioninfo	Pointer to where to store the version information of this module.
<b>Return value:</b>	None	
<b>Description:</b>	Service to get the version information of this module.	

**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).

**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.

## 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:**

<b>Service name:</b>	Eep_MainFunction
<b>Syntax:</b>	void Eep_MainFunction(  )
<b>Service ID[hex]:</b>	0x09
<b>Timing:</b>	VARIABLE_CYCLIC
<b>Description:</b>	Service to perform the processing of the EEPROM jobs (read/write/erase/compare) .

The function Eep\_MainFunction 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).

**EEP032:** The function Eep\_MainFunction shall return without action, if no job is pending.

**EEP031:** When a job has been initiated, the Eep's user shall call the Eep\_MainFunction cyclically until the job is finished.

## 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:

<i>API function</i>	<i>Description</i>
---------------------	--------------------

### 8.6.2 Optional Interfaces

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

#### EEP155:

<i>API function</i>	<i>Description</i>
Dem_ReportErrorStatus	Reports errors to the DEM.
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:** The callback notifications shall be configurable as function pointers within the initialization data structure (`Eep_ConfigType`).

**EEP049:** If a callback notification is configured as null pointer, the Eep module shall not execute the callback.

#### 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

#### EEP157:

<b>Service name:</b>	<code>(*Eep_JobEndNotification)</code>
<b>Syntax:</b>	<code>void (*Eep_JobEndNotification)(</code> <code>)</code>
<b>Sync/Async:</b>	true

<b>Reentrancy:</b>	Non Reentrant
<b>Parameters (in):</b>	None
<b>Parameters (inout):</b>	None
<b>Parameters (out):</b>	None
<b>Return value:</b>	None
<b>Description:</b>	Callback routine provided by the user to notify the caller that 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.

**EEP158:**

<b>Service name:</b>	(*Eep_JobErrorNotification)
<b>Syntax:</b>	void (*Eep_JobErrorNotification)(  )
<b>Sync/Async:</b>	true
<b>Reentrancy:</b>	Non Reentrant
<b>Parameters (in):</b>	None
<b>Parameters (inout):</b>	None
<b>Parameters (out):</b>	None
<b>Return value:</b>	None
<b>Description:</b>	Callback routine provided by the user to notify the caller that a job has been cancelled or aborted with a negative result.

**EEP127:** The callback function defined in the configuration parameter EepJobErrorNotification shall be callable on interrupt level.

## 9 Sequence diagrams

### 9.1 Initialization

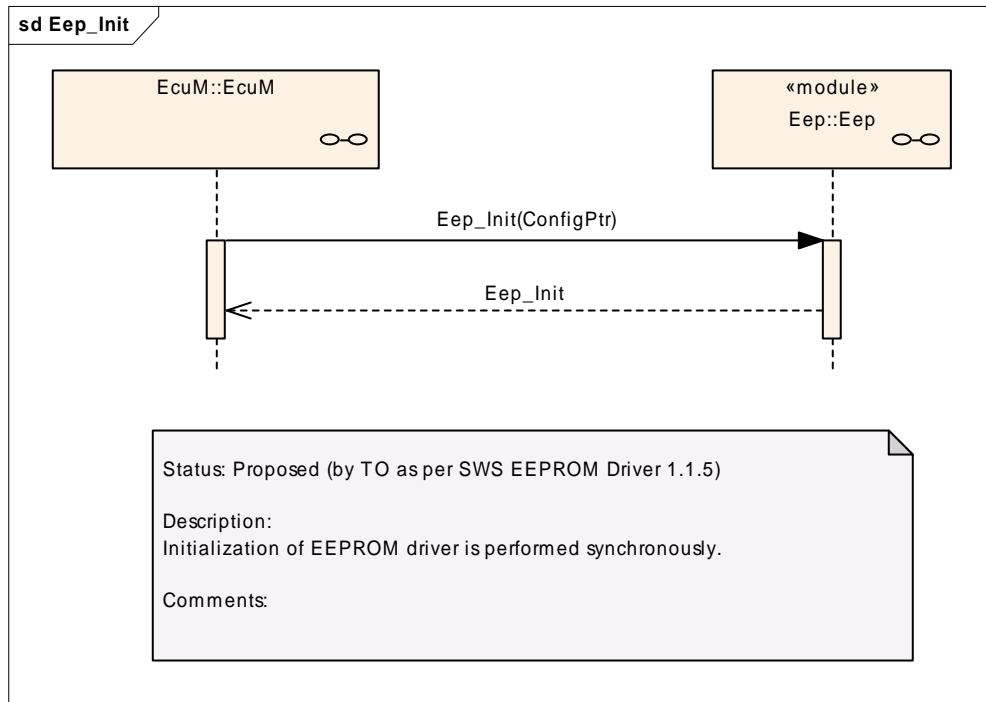
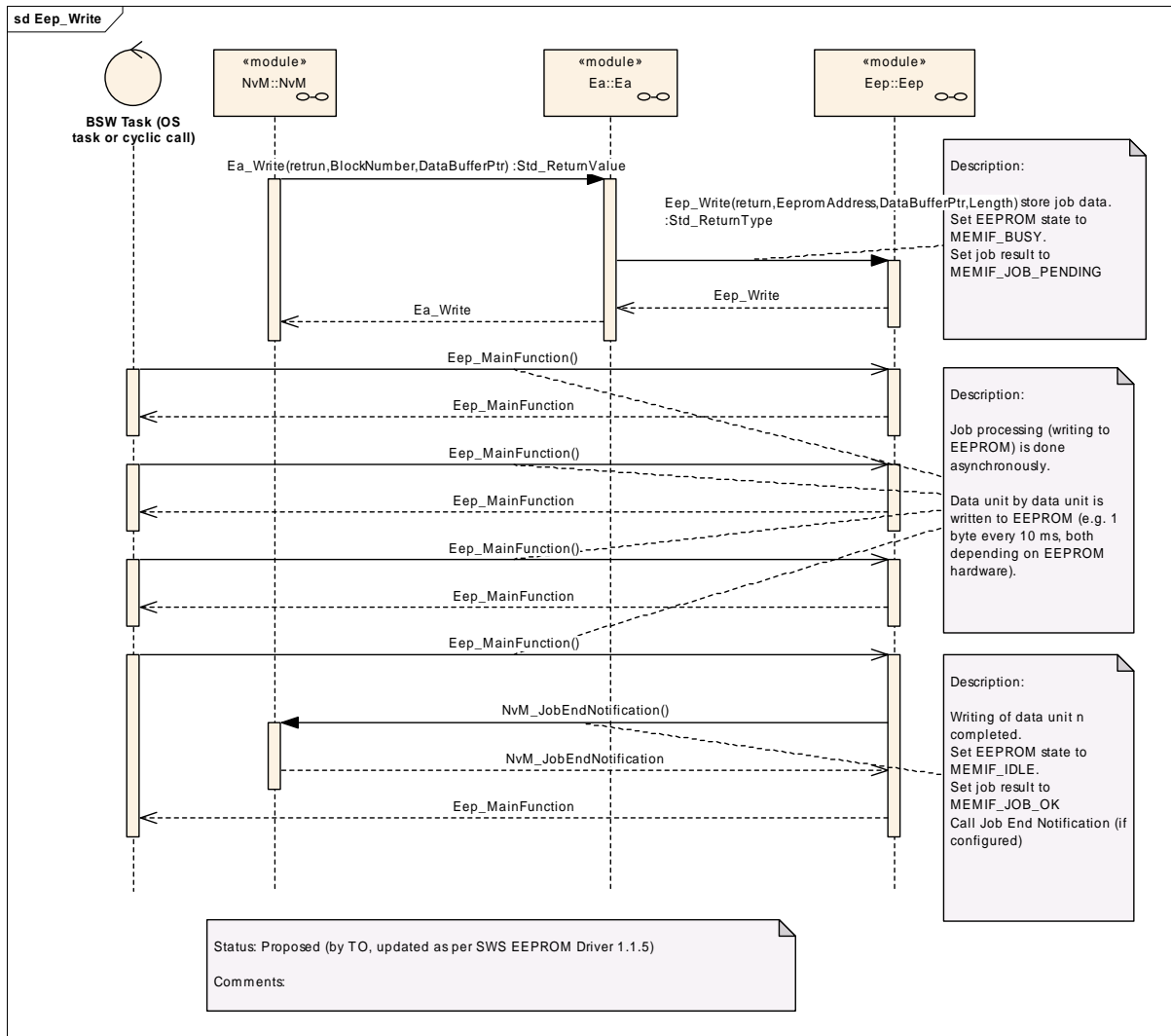


Figure 2

### 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.



**Figure 3**



### 9.3 Cancellation of a running job

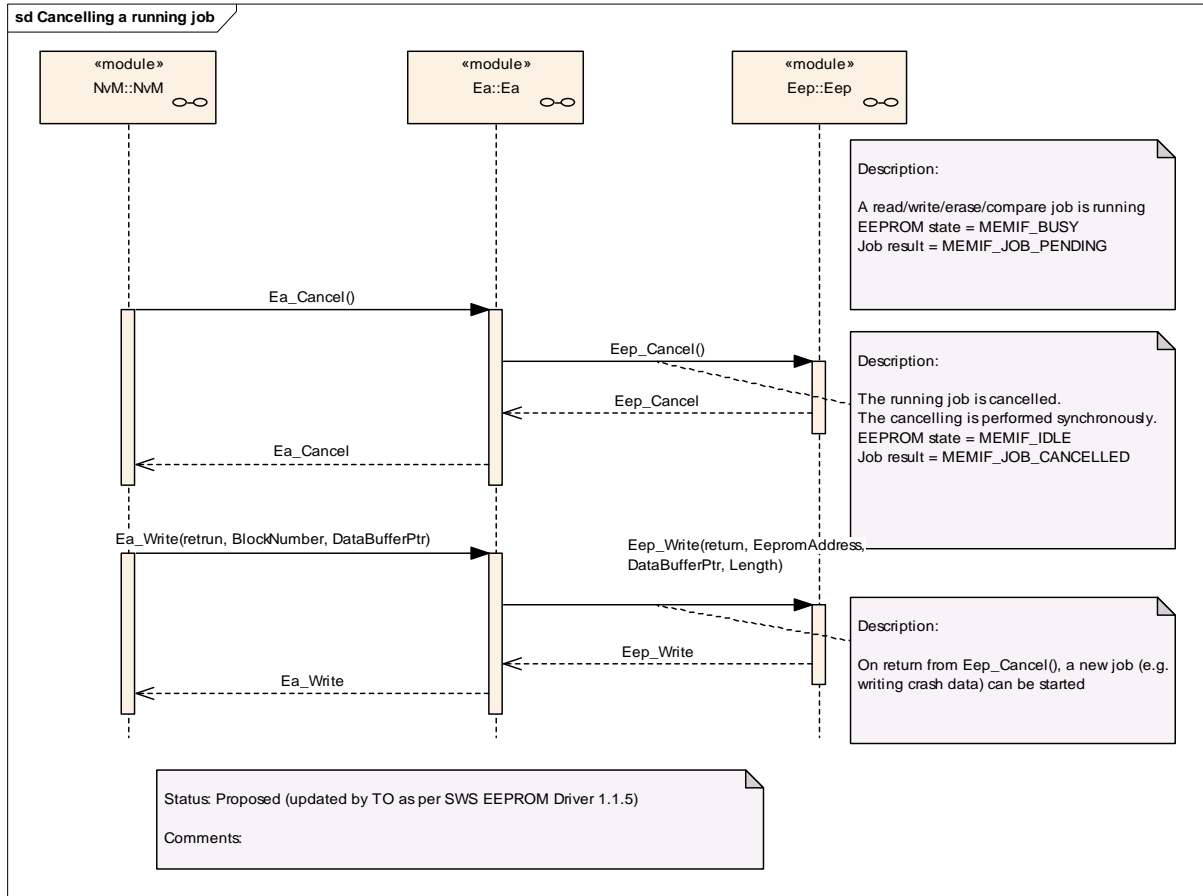


Figure 4

## 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

<b>Label</b>	<b>Description</b>
x	The configuration parameter is of configuration class <i>Pre-compile time</i> .
--	The configuration parameter is never of configuration class <i>Pre-compile time</i> .

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

<b>Label</b>	<b>Description</b>
x	The configuration parameter is of configuration class <i>Link time</i> .
--	The configuration parameter is never of configuration class <i>Link time</i> .

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

<b>Label</b>	<b>Description</b>
x	The configuration parameter is of configuration class <i>Post Build</i> and no specific implementation is required.
L	<i>Loadable</i> - the configuration parameter is of configuration class <i>Post Build</i> and only one configuration parameter set resides in the ECU.
M	<i>Multiple</i> - the configuration parameter is of configuration class <i>Post Build</i> 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 <i>Post Build</i> .

## 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 Eep

<b>Module Name</b>	<b>Eep</b>
<b>Module Description</b>	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.

<b>Included Containers</b>		
<b>Container Name</b>	<b>Multiplicity</b>	<b>Scope / Dependency</b>
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.
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.

### 10.2.3 EepGeneral

<b>SWS Item</b>	<b>EEP085 :</b>
<b>Container Name</b>	EepGeneral{EepGeneralConfiguration}
<b>Description</b>	Container for general configuration parameters of the EEPROM driver. These parameters are always pre-compile.
<b>Configuration Parameters</b>	

<b>SWS Item</b>	<b>EEP163 :</b>
<b>Name</b>	EepUseInterrupts {EEP_USE_INTERRUPTS}
<b>Description</b>	Switches to activate or deactivate interrupt controlled job processing. true: Interrupt controlled job processing enabled. false: Interrupt controlled job processing disabled.
<b>Multiplicity</b>	1

<b>Type</b>	BooleanParamDef		
<b>Default value</b>	false		
<b>ConfigurationClass</b>	<b>Pre-compile time</b>	X	All Variants
	<b>Link time</b>	--	
	<b>Post-build time</b>	--	
<b>Scope / Dependency</b>	scope: module dependency: Usually, this is only supported by some internal EEPROM peripherals.		

<b>SWS Item</b>	<b>EEP161 :</b>		
<b>Name</b>	EepDevErrorDetect {EEP_DEV_ERROR_DETECT}		
<b>Description</b>	Pre-processor switch to enable and disable development error detection. true: Development error detection enabled. false: Development error detection disabled.		
<b>Multiplicity</b>	1		
<b>Type</b>	BooleanParamDef		
<b>Default value</b>	true		
<b>ConfigurationClass</b>	<b>Pre-compile time</b>	X	All Variants
	<b>Link time</b>	--	
	<b>Post-build time</b>	--	
<b>Scope / Dependency</b>	scope: module		

<b>SWS Item</b>	<b>EEP164 :</b>		
<b>Name</b>	EepVersionInfoApi {EEP_VERSION_INFO_API}		
<b>Description</b>	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.		
<b>Multiplicity</b>	1		
<b>Type</b>	BooleanParamDef		
<b>Default value</b>	false		
<b>ConfigurationClass</b>	<b>Pre-compile time</b>	X	All Variants
	<b>Link time</b>	--	
	<b>Post-build time</b>	--	
<b>Scope / Dependency</b>	scope: module		

<b>SWS Item</b>	<b>EEP162 :</b>		
<b>Name</b>	EepDriverIndex		
<b>Description</b>	Index of the driver, used by EA.		
<b>Multiplicity</b>	1		
<b>Type</b>	IntegerParamDef (Symbolic Name generated for this parameter)		
<b>Range</b>	0 .. 254		
<b>Default value</b>	--		
<b>ConfigurationClass</b>	<b>Pre-compile time</b>	X	All Variants
	<b>Link time</b>	--	
	<b>Post-build time</b>	--	
<b>Scope / Dependency</b>	scope: module		

<b>SWS Item</b>	<b>EEP165 :</b>		
<b>Name</b>	EepWriteCycleReduction {EEP_WRITE_CYCLE_REDUCTION}		
<b>Description</b>	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.		
<b>Multiplicity</b>	1		
<b>Type</b>	BooleanParamDef		
<b>Default value</b>	false		

<b>ConfigurationClass</b>	<b>Pre-compile time</b>	X	All Variants
	<b>Link time</b>	--	
	<b>Post-build time</b>	--	
<b>Scope / Dependency</b>	scope: module		

**No Included Containers**

## 10.2.4 EepInitConfiguration

<b>SWS Item</b>	<b>EEP039 :</b>		
<b>Container Name</b>	EepInitConfiguration{EepInitConfiguration}		
<b>Description</b>	Container for runtime configuration parameters of the EEPROM driver. Implementation Type: Eep_ConfigType.		
<b>Configuration Parameters</b>			

<b>SWS Item</b>	<b>EEP173 :</b>		
<b>Name</b>	EepNormalReadBlockSize {EEP_NORMAL_READ_BLOCK_SIZE}		
<b>Description</b>	This parameter is the number of bytes read within one job processing cycle in normal mode. Implementation Type: Eep_LengthType.		
<b>Multiplicity</b>	1		
<b>Type</b>	IntegerParamDef		
<b>Default value</b>	--		
<b>ConfigurationClass</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	--	
<b>Scope / Dependency</b>	scope: module		

<b>SWS Item</b>	<b>EEP170 :</b>		
<b>Name</b>	EepJobCallCycle {EEP_JOB_CALL_CYCLE}		
<b>Description</b>	This parameter is the call cycle of the job processing function during write/erase operations. Unit: [s]		
<b>Multiplicity</b>	1		
<b>Type</b>	FloatParamDef		
<b>Default value</b>	--		
<b>ConfigurationClass</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	--	
<b>Scope / Dependency</b>	scope: module		

<b>SWS Item</b>	<b>EEP175 :</b>		
<b>Name</b>	EepSize {EEP_SIZE}		
<b>Description</b>	This parameter is the used size of EEPROM device in bytes. Implementation Type: Eep_LengthType.		
<b>Multiplicity</b>	1		
<b>Type</b>	IntegerParamDef		
<b>Default value</b>	--		
<b>ConfigurationClass</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	--	
<b>Scope / Dependency</b>	scope: module		

<b>SWS Item</b>	<b>EEP171 :</b>		
<b>Name</b>	EepJobEndNotification {EEP_JOB_END_NOTIFICATION}		

<b>Description</b>	This parameter is a reference to a callback function for positive job result (see EEP045).		
<b>Multiplicity</b>	1		
<b>Type</b>	FunctionNameDef		
<b>Default value</b>	--		
<b>ConfigurationClass</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	--	
<b>Scope / Dependency</b>	scope: module		

<b>SWS Item</b>	<b>EEP167 :</b>		
<b>Name</b>	EepDefaultMode {EEP_DEFAULT_MODE}		
<b>Description</b>	This parameter is the default EEPROM device mode after initialization. Implementation Type: MemIf_ModeType.		
<b>Multiplicity</b>	1		
<b>Type</b>	EnumerationParamDef		
<b>Range</b>	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.	
<b>ConfigurationClass</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	--	
<b>Scope / Dependency</b>	scope: module		

<b>SWS Item</b>	<b>EEP168 :</b>		
<b>Name</b>	EepFastReadBlockSize {EEP_FAST_READ_BLOCK_SIZE}		
<b>Description</b>	This parameter is the number of bytes read within one job processing cycle in fast mode. Implementation Type: Eep_LengthType.		
<b>Multiplicity</b>	1		
<b>Type</b>	IntegerParamDef		
<b>Default value</b>	--		
<b>ConfigurationClass</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	--	
<b>Scope / Dependency</b>	scope: module		

<b>SWS Item</b>	<b>EEP174 :</b>		
<b>Name</b>	EepNormalWriteBlockSize {EEP_NORMAL_WRITE_BLOCK_SIZE}		
<b>Description</b>	Number of bytes written within one job processing cycle in normal mode. Implementation Type: Eep_LengthType.		
<b>Multiplicity</b>	0..1		
<b>Type</b>	IntegerParamDef		
<b>Default value</b>	--		
<b>ConfigurationClass</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	--	
<b>Scope / Dependency</b>	scope: module dependency: This parameter is optional and only available if the hardware allows configuration.		

<b>SWS Item</b>	<b>EEP172 :</b>		
<b>Name</b>	EepJobErrorNotification {EEP_JOB_ERROR_NOTIFICATION}		
<b>Description</b>	This parameter is a reference to a callback function for negative job result (see EEP046).		

<b>Multiplicity</b>	1		
<b>Type</b>	FunctionNameDef		
<b>Default value</b>	--		
<b>ConfigurationClass</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	--	
<b>Scope / Dependency</b>	scope: module		

<b>SWS Item</b>	<b>EEP169 :</b>		
<b>Name</b>	EepFastWriteBlockSize {EEP_FAST_WRITE_BLOCK_SIZE}		
<b>Description</b>	This parameter is the number of bytes written within one job processing cycle in fast mode. Implementation Type: Eep_LengthType.		
<b>Multiplicity</b>	0..1		
<b>Type</b>	IntegerParamDef		
<b>Default value</b>	--		
<b>ConfigurationClass</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	--	
<b>Scope / Dependency</b>	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).		

<b>SWS Item</b>	<b>EEP166 :</b>		
<b>Name</b>	EepBaseAddress {EEP_BASE_ADDRESS}		
<b>Description</b>	This parameter is the EEPROM device base address. Implementation Type: Eep_AddressType.		
<b>Multiplicity</b>	1		
<b>Type</b>	IntegerParamDef		
<b>Default value</b>	--		
<b>ConfigurationClass</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	--	
<b>Scope / Dependency</b>	scope: module		

<b>Included Containers</b>		
<b>Container Name</b>	<b>Multiplicity</b>	<b>Scope / Dependency</b>
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 EepExternalDriver

<b>SWS Item</b>	<b>EEP094 :</b>	
<b>Container Name</b>	EepExternalDriver	
<b>Description</b>	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.	
<b>Configuration Parameters</b>		

<b>SWS Item</b>	<b>EEP176 :</b>	
<b>Name</b>	SpiReference	
<b>Description</b>	Reference to SPI sequence (required for external EEPROM drivers).	



<b>Multiplicity</b>	1..*		
<b>Type</b>	Reference to SpiSequence		
<b>ConfigurationClass</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	--	
<b>Scope / Dependency</b>			

**No Included Containers**

## 10.2.6 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). 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.

The standard common published information like

- `vendorId` (EEP\_VENDOR\_ID),
- `moduleId` (EEP\_MODULE\_ID),
- `arMajorVersion` (EEP\_AR\_MAJOR\_VERSION),
- `arMinorVersion` (EEP\_AR\_MINOR\_VERSION),
- `arPatchVersion` (EEP\_AR\_PATCH\_VERSION),
- `swMajorVersion` (EEP\_SW\_MAJOR\_VERSION),
- `swMinorVersion` (EEP\_SW\_MINOR\_VERSION),
- `swPatchVersion` (EEP\_SW\_PATCH\_VERSION),
- `vendorApiInfix` (EEP\_VENDOR\_API\_INFIX)

is provided in the BSW Module Description Template (see [11], Figure 4.1 and Figure 7.1). Additional published parameters are listed below if applicable for this module.

### 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.

### 10.3.3 EepPublishedInformation

<b>SWS Item</b>	<b>EEP111 :</b>		
<b>Container Name</b>	EepPublishedInformation		
<b>Description</b>	Additional published parameters not covered by CommonPublishedInformation container. Note that these parameters do not have any configuration class setting, since they are published information.		
<b>Configuration Parameters</b>			

<b>SWS Item</b>	<b>EEP184 :</b>		
<b>Name</b>	EepTotalSize {EEP_TOTAL_SIZE}		
<b>Description</b>	Total size of EEPROM in bytes. Implementation Type: Eep_LengthType.		
<b>Multiplicity</b>	1		
<b>Type</b>	IntegerParamDef		
<b>Default value</b>	--		
<b>ConfigurationClass</b>	<b>Published Information</b>	X	All Variants
<b>Scope / Dependency</b>	scope: module		

<b>SWS Item</b>	<b>EEP179 :</b>		
<b>Name</b>	EepEraseUnitSize {EEP_ERASE_UNIT_SIZE}		
<b>Description</b>	Size of smallest erasable EEPROM data unit in bytes.		
<b>Multiplicity</b>	1		
<b>Type</b>	IntegerParamDef		
<b>Default value</b>	--		
<b>ConfigurationClass</b>	<b>Published Information</b>	X	All Variants
<b>Scope / Dependency</b>	scope: module		

<b>SWS Item</b>	<b>EEP182 :</b>		
<b>Name</b>	EepMinimumLengthType {EEP_MINIMUM_LENGTH_TYPE}		
<b>Description</b>	Minimum expected size of Eep_LengthType.		
<b>Multiplicity</b>	1		
<b>Type</b>	IntegerParamDef		

<b>Default value</b>	--		
<b>ConfigurationClass</b>	<b>Published Information</b>	X	All Variants
<b>Scope / Dependency</b>	scope: module		

<b>SWS Item</b>	<b>EEP181 :</b>		
<b>Name</b>	EepMinimumAddressType {EEP_MINIMUM_ADDRESS_TYPE}		
<b>Description</b>	Minimum expected size of Eep_AddressType.		
<b>Multiplicity</b>	1		
<b>Type</b>	IntegerParamDef		
<b>Default value</b>	--		
<b>ConfigurationClass</b>	<b>Published Information</b>	X	All Variants
<b>Scope / Dependency</b>	scope: module		

<b>SWS Item</b>	<b>EEP186 :</b>		
<b>Name</b>	EepWriteUnitSize {EEP_WRITE_UNIT_SIZE}		
<b>Description</b>	Size of smallest writeable EEPROM data unit in bytes.		
<b>Multiplicity</b>	1		
<b>Type</b>	IntegerParamDef		
<b>Default value</b>	--		
<b>ConfigurationClass</b>	<b>Published Information</b>	X	All Variants
<b>Scope / Dependency</b>	scope: module		

<b>SWS Item</b>	<b>EEP180 :</b>		
<b>Name</b>	EepEraseValue {EEP_ERASE_VALUE}		
<b>Description</b>	Value of an erased EEPROM cell.		
<b>Multiplicity</b>	1		
<b>Type</b>	IntegerParamDef		
<b>Default value</b>	--		
<b>ConfigurationClass</b>	<b>Published Information</b>	X	All Variants
<b>Scope / Dependency</b>	scope: module		

<b>SWS Item</b>	--		
<b>Name</b>	EepSpecifiedEraseCycles {EEP_SPECIFIED_ERASE_CYCLES}		
<b>Description</b>	Number of erase cycles specified for the EEP device (usually given in the device data sheet).		
<b>Multiplicity</b>	1		
<b>Type</b>	IntegerParamDef		
<b>Default value</b>	--		
<b>ConfigurationClass</b>	<b>Published Information</b>	X	All Variants
<b>Scope / Dependency</b>	scope: module		

<b>SWS Item</b>	<b>EEP183 :</b>		
<b>Name</b>	EepReadUnitSize {EEP_READ_UNIT_SIZE}		
<b>Description</b>	Size of smallest readable EEPROM data unit in bytes.		
<b>Multiplicity</b>	1		
<b>Type</b>	IntegerParamDef		
<b>Default value</b>	--		
<b>ConfigurationClass</b>	<b>Published Information</b>	X	All Variants
<b>Scope / Dependency</b>	scope: module		

<b>SWS Item</b>	<b>EEP185 :</b>		
<b>Name</b>	EepWriteTime {EEP_WRITE_TIME}		
<b>Description</b>	Time for writing one EEPROM data unit.		
<b>Multiplicity</b>	1		

<b>Type</b>	FloatParamDef		
<b>Default value</b>	--		
<b>ConfigurationClass</b>	<b>Published Information</b>	X	All Variants
<b>Scope / Dependency</b>	scope: module		

<b>SWS Item</b>	<b>EEP178 :</b>		
<b>Name</b>	EepEraseTime {EEP_ERASE_TIME}		
<b>Description</b>	Time for erasing one EEPROM data unit.		
<b>Multiplicity</b>	1		
<b>Type</b>	FloatParamDef		
<b>Default value</b>	--		
<b>ConfigurationClass</b>	<b>Published Information</b>	X	All Variants
<b>Scope / Dependency</b>	scope: module		

<b>SWS Item</b>	<b>EEP177 :</b>		
<b>Name</b>	EepAllowedWriteCycles {EEP_ALLOWED_WRITE_CYCLES}		
<b>Description</b>	Specified maximum number of write cycles under worst case conditions of specific EEPROM hardware (e.g. +90°C)		
<b>Multiplicity</b>	1		
<b>Type</b>	IntegerParamDef		
<b>Default value</b>	--		
<b>ConfigurationClass</b>	<b>Published Information</b>	X	All Variants
<b>Scope / Dependency</b>	scope: module		

<b>No Included Containers</b>
-------------------------------

## 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

<i>SWS Item</i>	<i>Rationale</i>
EEP061	Deleted to fix the Bugzilla issue #9919
EEP062	Deleted to fix the Bugzilla issue #9919

### 11.2 Replaced SWS Items

<i>SWS Item of Release 1</i>	<i>replaced by SWS Item</i>	<i>Rationale</i>
EEP001	<a href="#">EEP001</a> , <a href="#">EEP100</a>	To split the old requirement into two requirements to fit to the new SWS template.

### 11.3 Changed SWS Items

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

### 11.4 Added SWS Items

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

## 12 Changes during SWS Improvements by Technical Office

### 12.1 Deleted SWS Items

<i>SWS Item</i>	<i>Rationale</i>
EEP105	redundant
EEP114	redundant with EEP021, EEP027, EEP028
EEP048	redundant with the API definitions of the callback functions
EEP010	now included in the existing requirements EEP016, EEP017, EEP018

### 12.2 Replaced SWS Items

<i>SWS Item of Release 1</i>	<i>replaced by SWS Item</i>	<i>Rationale</i>
EEP086	<a href="#">EEP128</a> , <a href="#">EEP129</a>	Made requirement atomic
EEP050	<a href="#">EEP130</a> , <a href="#">EEP132</a>	Made requirement atomic
EEP096	<a href="#">EEP133</a> , <a href="#">EEP134</a>	Made requirement atomic
EEP112	<a href="#">EEP137</a>	Formulated a requirement out of the paragraph; kept the paragraph as background information
EEP093, EEP080	<a href="#">EEP138</a>	Replaced by UML Model linking of Imported Types

### 12.3 Changed SWS Items

Many requirements have been changed to improve understandability without changing the technical contents.

### 12.4 Added SWS Items

<i>SWS Item</i>	<i>Rationale</i>
<a href="#">EEP115</a>	Caveat for Eep_Init
<a href="#">EEP116</a>	Caveat for Eep_SetMode
<a href="#">EEP117</a>	Caveat for Eep_Read
<a href="#">EEP118</a>	Caveat for Eep_Read
<a href="#">EEP119</a>	Caveat for Eep_Write
<a href="#">EEP120</a>	Caveat for Eep_Write
<a href="#">EEP121</a>	Caveat for Eep_Erase
<a href="#">EEP122</a>	Caveat for Eep_Erase
<a href="#">EEP123</a>	Caveat for Eep_Compare
<a href="#">EEP124</a>	Caveat for Eep_Compare
<a href="#">EEP126</a>	Caveat for <End Job Notification>
<a href="#">EEP127</a>	Caveat for <Error Job Notification>
<a href="#">EEP135</a>	Hint from Eep_GetVersionInfo
<a href="#">EEP136</a>	Requirement originally mentioned after the sequence diagram of Eep_Cancel
<a href="#">EEP143</a>	UML Model linking of Eep_Init
<a href="#">EEP144</a>	UML Model linking of Eep_SetMode



<a href="#">EEP145</a>	UML Model linking of Eep_Read
<a href="#">EEP146</a>	UML Model linking of Eep_Write
<a href="#">EEP147</a>	UML Model linking of Eep_Erase
<a href="#">EEP148</a>	UML Model linking of Eep_Compare
<a href="#">EEP149</a>	UML Model linking of Eep_Cancel
<a href="#">EEP150</a>	UML Model linking of Eep_GetStatus
<a href="#">EEP151</a>	UML Model linking of Eep_GetJobResult
<a href="#">EEP152</a>	UML Model linking of Eep_GetVersionInfo
<a href="#">EEP153</a>	UML Model linking of Eep_MainFunction
<a href="#">EEP154</a>	UML Model linking of the mandatory interfaces
<a href="#">EEP155</a>	UML Model linking of the optional interfaces
<a href="#">EEP157</a>	UML Model linking of configurable interface Eep_JobEndNotification
<a href="#">EEP158</a>	UML Model linking of configurable interface Eep_JobErrorNotification