Document Title	Specification of EEPROM Abstraction
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	287
Document Classification	Standard
Document Status	Final
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	4.3.0

Document Change History

Date	Release	Changed by	Change Description
2016-11-30	4.3.0	AUTOSAR Release Management	• Rules for request acceptance/rejection and related error reporting updated • Updated tracing information • Range / limits on main function changed
2015-07-31	4.2.2	AUTOSAR Release Management	• Error classification reworked • Debug support marked as obsolete • Parameter ranges corrected • Job result clarified if requested block can't be found
2014-10-31	4.2.1	AUTOSAR Release Management	• Requirements linked to BSW features, general and module specific requirements
2014-03-31	4.1.3	AUTOSAR Release Management	• Editorial changes
2013-10-31	4.1.2	AUTOSAR Release Management	• Timing requirement removed from module's main function • "const" qualifier Added to prototype of function Ea_Write • New configuration parameter EaMainFunctionPeriod • Fls_GetStatus returns MEMIF_UNINIT if module is not initialized • Editorial changes • Removed chapter(s) on change documentation

Document Change History			
Date	**Release**	**Changed by**	**Change Description**
2013-03-15	4.1.1	AUTOSAR Administration	• Reworked according to the new SWS_BSWGeneral • Scope attribute in tables in chapter 10 added • Published parameter EaMaximumBlockingTime deprecated • Configuration parameter EaIndex deprecated
2011-12-22	4.0.3	AUTOSAR Administration	• Introduced parameter checks and corresponding DET errors • Handling of internal management operations detailed • Module short name changed
2010-09-30	3.1.5	AUTOSAR Administration	• Check fpr NULL pointer added • Inter module checks detailed • Description of return values clarified
2010-02-02	3.1.4	AUTOSAR Administration	• Configuration variants clarified • Multiplicity of notification routines corected • Job result handling re-formulated • File include structure changed • Legal disclaimer revised
2008-08-13	3.1.1	AUTOSAR Administration	• Legal disclaimer revised
2007-12-21	3.0.1	AUTOSAR Administration	• EA_MAXIMUM_BLOCKING_TIME as published parameter • Small reformulations resulting from table generation • Tables in chapters 8 and 10 generated from UML model • Document meta information extended • Small layout adaptations made

Document Change History			
Date	**Release**	**Changed by**	**Change Description**
2007-01-24	2.1.15	AUTOSAR Administration	• File include structure updated • API of initialization function adapted • Range of EA block numbers adapted • Legal disclaimer revised • Release Notes added • “Advice for users” revised • “Revision Information” added
2006-05-16	2.0	AUTOSAR Administration	• Initial release

Disclaimer

This specification and the material contained in it, as released by AUTOSAR, is for the purpose of information only. AUTOSAR and the companies that have contributed to it shall not be liable for any use of the specification.

The material contained in this specification is protected by copyright and other types of Intellectual Property Rights. The commercial exploitation of the material contained in this specification requires a license to such Intellectual Property Rights.

This specification may be utilized or reproduced without any modification, in any form or by any means, for informational purposes only. For any other purpose, no part of the specification may be utilized or reproduced, in any form or by any means, without permission in writing from the publisher.

The AUTOSAR specifications have been developed for automotive applications only. They have neither been developed, nor tested for non-automotive applications.

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

Advice for users

AUTOSAR specifications 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 specifications for illustration purposes only, and they themselves are not part of the AUTOSAR Standard. Neither their presence in such specifications, 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 Content

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 9
3.3 Related specification 9
4 Constraints and assumptions 11
4.1 Limitations 11
4.2 Applicability to car domains 11
5 Dependencies to other modules 12
5.1 File structure 12
5.1.1 Header file structure 12
6 Requirements traceability 13
7 Functional specification 22
7.1 General behavior 22
7.1.1 Addressing scheme and segmentation 22
7.1.2 Address calculation 23
7.1.3 Limitation of erase / write cycles 25
7.1.4 Handling of “immediate” data 26
7.1.5 Managing block consistency information 26
7.2 Error classification 27
7.2.1 Development Errors 27
7.2.2 Runtime Errors 27
7.2.3 Transient Faults 27
7.2.4 Production Errors 27
7.2.5 Extended Production Errors 27
8 API specification 29
8.1 Imported Types 29
8.2 Type definitions 29
8.3 Function definitions 29
8.3.1 Ea_Init 29
8.3.2 Ea_SetMode 30
8.3.3 Ea_Read 31
8.3.4 Ea_Write 33
8.3.5 Ea_Cancel 34
8.3.6 Ea_GetStatus 35
8.3.7 Ea_GetJobResult 36
8.3.8 Ea_InvalidateBlock 37
8.3.9 Ea_GetVersionInfo 38
8.3.10 Ea_EraseImmediateBlock 39
8.4 Call-back notifications 40
8.4.1 Ea_JobEndNotification 41

8.4.2 Ea_JobErrorNotification 41
8.5 Scheduled functions 42
8.5.1 Ea_MainFunction 42
8.6 Expected Interfaces 43
8.6.1 Mandatory Interfaces 43
8.6.2 Optional Interfaces 44
8.6.3 Configurable interfaces 44
9 Sequence diagrams 47
9.1 Ea_Init 47
9.2 Ea_Write 48
9.3 Ea_Cancel 49
10 Configuration specification 52
10.1 Containers and configuration parameters 52
10.1.1 Ea 52
10.1.2 EaGeneral 53
10.1.3 EaBlockConfiguration 55
10.2 Published Information 57
10.2.1 EaPublishedInformation 57
11 Not applicable requirements 58

1 Introduction and functional overview

This specification describes the functionality, API and configuration of the EEPROM Abstraction Layer (see Figure 1).

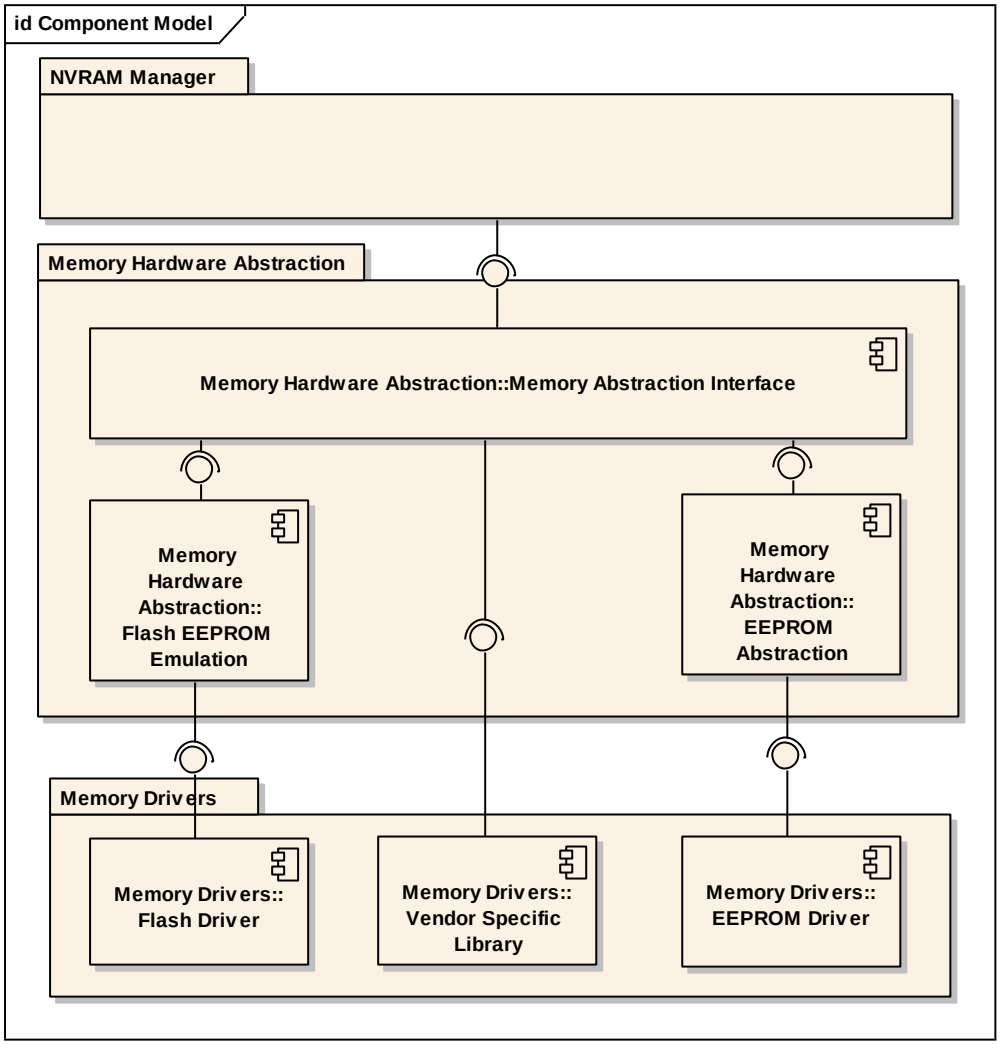


Figure 1: Module overview of memory hardware abstraction layer

The EEPROM Abstraction (EA) abstracts from the device specific addressing scheme and segmentation and provides the upper layers with a virtual addressing scheme and segmentation as well as a “virtually” unlimited number of erase cycles.

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.

Abbreviation / Acronym:	***Description:***
EA	EEPROM Abstraction
EEPROM	Electrically Erasable and Programmable ROM (Read Only Memory)
FEE	Flash EEPROM Emulation
LSB	Least significant bit / byte (depending on context). Here it's bit.
MemIf	Memory Abstraction Interface
MSB	Most significant bit / byte (depending on context). Here it's bit.
NvM	NVRAM Manager
NVRAM	Non-volatile RAM (Random Access Memory)
NVRAM block	Management unit as seen by the NVRAM Manager
(Logical) block	Smallest writable / erasable unit as seen by the modules user. Consists of one or more virtual pages.
Virtual page	May consist of one or several physical pages to ease handling of logical blocks and address calculation.
Internal residue	Unused space at the end of the last virtual page if the configured block size isn't an integer multiple of the virtual page size (see Figure 3).
Virtual address	Consisting of 16 bit block number and 16 bit offset inside the logical block.
Physical address	Address information in device specific format (depending on the underlying EEPROM driver and device) that is used to access a logical block.
Dataset	Concept of the NVRAM manager: A user addressable array of blocks of the same size. E.g. could be used to provide different configuration settings for the CAN driver (CAN IDs, filter settings, …) to an ECU which has otherwise identical application software (e.g. door module).
Redundant copy	Concept of the NVRAM manager: Storing the same information twice to enhance reliability of data storage.

3 Related documentation

3.1 Input documents

[1] List of Basic Software Modules
AUTOSAR_TR_BSWModuleList.pdf

[2] Layered Software Architecture
AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf

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

[4] General Requirements on SPAL
AUTOSAR_SRS_SPALGeneral.pdf

[5] Requirements on Memory Hardware Abstraction Layer
AUTOSAR_SRS_MemoryHWAbstractionLayer.doc

[6] Specification of Default Error Tracer
AUTOSAR_SWS_DefaultErrorTracer.pdf

[7] Specification of ECU Configuration,
AUTOSAR_TPS_ECUConfiguration.pdf

[8] Basic Software Module Description Template,
AUTOSAR_TPS_BSWModuleDescriptionTemplate.pd

[9] General Specification of Basic Software Modules
AUTOSAR_SWS_BSWGeneral.pdf

3.2 Related standards and norms

[7] Specification of NVRAM Manager
AUTOSAR_SWS_NVRAMManager.doc

[8] Specification of Memory Abstraction Interface
AUTOSAR_SWS_MemoryAbstractionInterface.pdf

[9] Specification of Flash EEPROM Emulation
AUTOSAR_SWS_FlashEEPROMEmulation.pdf

3.3 Related specification

AUTOSAR provides a General Specification on Basic Software modules [9] (SWS BSW General), which is also valid for EEPROM Abstraction.

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

4 Constraints and assumptions

4.1 Limitations

No limitations.

4.2 Applicability to car domains

No restrictions.

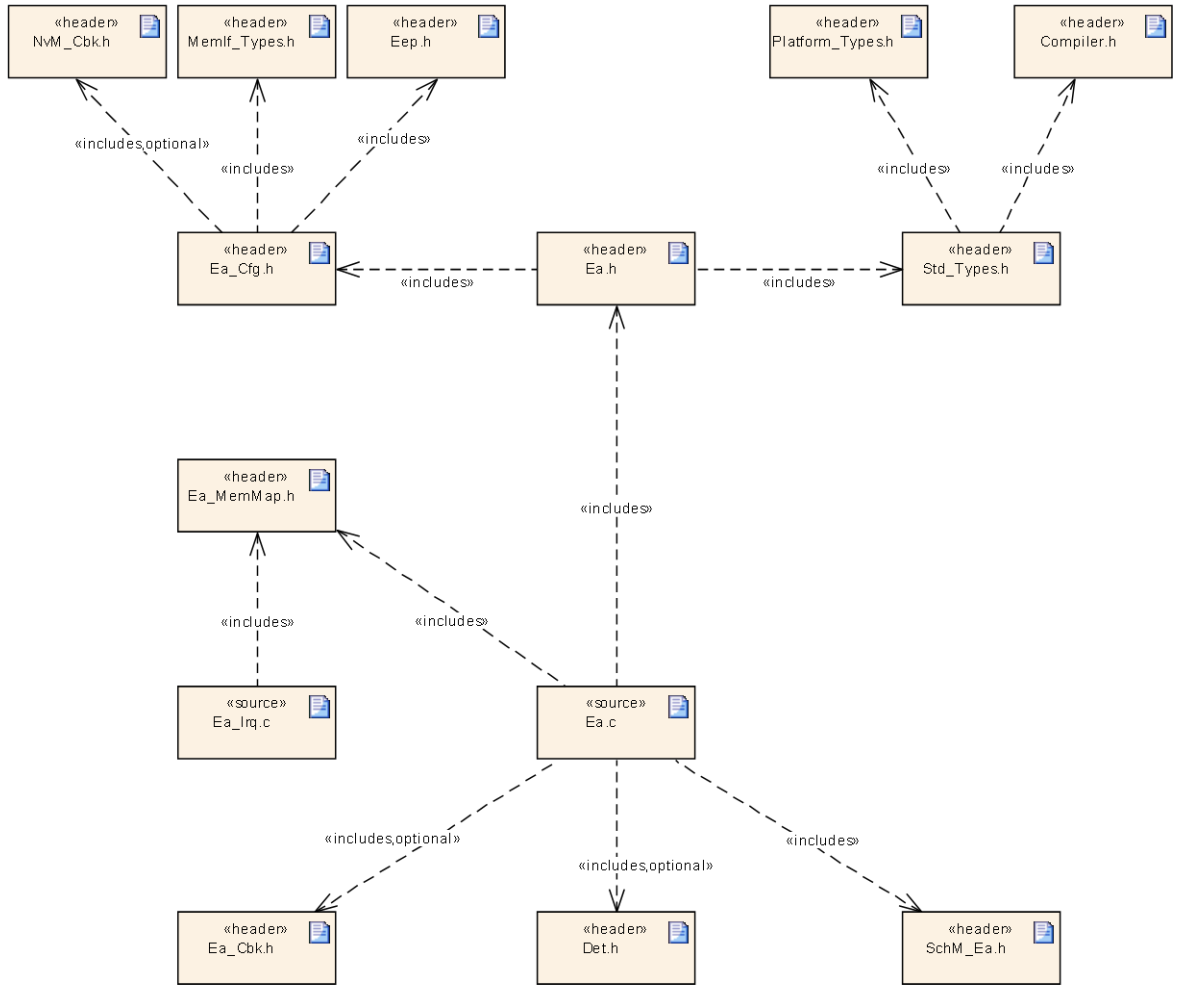
5 Dependencies to other modules

This module depends on the capabilities of the underlying EEPROM driver as well as the configuration of the NVRAM manager.

5.1 File structure

5.1.1 Header file structure

[SWS_Ea_00113] ⌈ The Ea module shall comply with the following file include structure:



⌋ (SRS_BSW_00346, SRS_BSW_00158, SRS_BSW_00301)

Figure 2: EEPROM Abstraction Layer File Include Structure

Note: Upper layer modules shall only include `Ea.h`

6 Requirements traceability

Requirement	Description	Satisfied by
RS_BRF_01048	AUTOSAR module design shall support modules to cooperate in a multitasking environment	SWS_Ea_00026, SWS_Ea_00056, SWS_Ea_00072, SWS_Ea_00089, SWS_Ea_00090, SWS_Ea_00174
RS_BRF_01056	AUTOSAR BSW modules shall provide standardized interfaces	SWS_Ea_00097, SWS_Ea_00098
RS_BRF_01064	AUTOSAR BSW shall provide callback functions in order to access upper layer modules	SWS_Ea_00051, SWS_Ea_00053, SWS_Ea_00054, SWS_Ea_00055, SWS_Ea_00094, SWS_Ea_00095, SWS_Ea_00141, SWS_Ea_00142, SWS_Ea_00143, SWS_Ea_00144, SWS_Ea_00145, SWS_Ea_00146, SWS_Ea_00153, SWS_Ea_00154
RS_BRF_01480	AUTOSAR shall support software component local modes, ECU global modes, and system wide modes	SWS_Ea_00085, SWS_Ea_00150, SWS_Ea_00192
RS_BRF_01812	AUTOSAR non-volatile memory functionality shall support the prioritization and asynchronous execution of jobs	SWS_Ea_00193, SWS_Ea_00195
SRS_BSW_00005	Modules of the µC Abstraction Layer (MCAL) may not have hard coded horizontal interfaces	SWS_Ea_00999
SRS_BSW_00006	The source code of software modules above the µC Abstraction Layer (MCAL) shall not be processor and compiler dependent.	SWS_Ea_00999
SRS_BSW_00007	All Basic SW Modules written in C language shall conform to the MISRA C 2012 Standard.	SWS_Ea_00999
SRS_BSW_00009	All Basic SW Modules shall be documented according to a common standard.	SWS_Ea_00999
SRS_BSW_00010	The memory consumption of all Basic SW Modules shall be documented for a defined configuration for all supported platforms.	SWS_Ea_00999
SRS_BSW_00101	The Basic Software Module shall be able to initialize variables and hardware in a separate initialization	SWS_Ea_00017, SWS_Ea_00084

	function	
SRS_BSW_00158	All modules of the AUTOSAR Basic Software shall strictly separate configuration from implementation	SWS_Ea_00113
SRS_BSW_00160	Configuration files of AUTOSAR Basic SW module shall be readable for human beings	SWS_Ea_00999
SRS_BSW_00161	The AUTOSAR Basic Software shall provide a microcontroller abstraction layer which provides a standardized interface to higher software layers	SWS_Ea_00999
SRS_BSW_00162	The AUTOSAR Basic Software shall provide a hardware abstraction layer	SWS_Ea_00999
SRS_BSW_00164	The Implementation of interrupt service routines shall be done by the Operating System, complex drivers or modules	SWS_Ea_00999
SRS_BSW_00168	SW components shall be tested by a function defined in a common API in the Basis-SW	SWS_Ea_00999
SRS_BSW_00172	The scheduling strategy that is built inside the Basic Software Modules shall be compatible with the strategy used in the system	SWS_Ea_00999
SRS_BSW_00300	All AUTOSAR Basic Software Modules shall be identified by an unambiguous name	SWS_Ea_00999
SRS_BSW_00301	All AUTOSAR Basic Software Modules shall only import the necessary information	SWS_Ea_00113
SRS_BSW_00302	All AUTOSAR Basic Software Modules shall only export information needed by other modules	SWS_Ea_00999
SRS_BSW_00304	All AUTOSAR Basic Software Modules shall use the following data types instead of native C data types	SWS_Ea_00999
SRS_BSW_00305	Data types naming convention	SWS_Ea_00999
SRS_BSW_00306	AUTOSAR Basic Software	SWS_Ea_00999

	Modules shall be compiler and platform independent	
SRS_BSW_00307	Global variables naming convention	SWS_Ea_00999
SRS_BSW_00308	AUTOSAR Basic Software Modules shall not define global data in their header files, but in the C file	SWS_Ea_00999
SRS_BSW_00309	All AUTOSAR Basic Software Modules shall indicate all global data with read-only purposes by explicitly assigning the const keyword	SWS_Ea_00999
SRS_BSW_00312	Shared code shall be reentrant	SWS_Ea_00999
SRS_BSW_00314	All internal driver modules shall separate the interrupt frame definition from the service routine	SWS_Ea_00999
SRS_BSW_00321	The version numbers of AUTOSAR Basic Software Modules shall be enumerated according specific rules	SWS_Ea_00999
SRS_BSW_00323	All AUTOSAR Basic Software Modules shall check passed API parameters for validity	SWS_Ea_00065, SWS_Ea_00135, SWS_Ea_00147, SWS_Ea_00148, SWS_Ea_00149, SWS_Ea_00152, SWS_Ea_00158, SWS_Ea_00159, SWS_Ea_00161, SWS_Ea_00162, SWS_Ea_00164, SWS_Ea_00165, SWS_Ea_00167, SWS_Ea_00168, SWS_Ea_00169, SWS_Ea_00170, SWS_Ea_00172, SWS_Ea_00173, SWS_Ea_00175, SWS_Ea_00176
SRS_BSW_00328	All AUTOSAR Basic Software Modules shall avoid the duplication of code	SWS_Ea_00999
SRS_BSW_00330	It shall be allowed to use macros instead of functions where source code is used and runtime is critical	SWS_Ea_00999
SRS_BSW_00333	For each callback function it shall be specified if it is called from interrupt context or not	SWS_Ea_00999
SRS_BSW_00334	All Basic Software Modules shall provide an XML file that contains the meta data	SWS_Ea_00999
SRS_BSW_00336	Basic SW module shall be able to shutdown	SWS_Ea_00999
SRS_BSW_00339	Reporting of production	SWS_Ea_00999

	relevant error status	
SRS_BSW_00341	Module documentation shall contains all needed informations	SWS_Ea_00999
SRS_BSW_00342	It shall be possible to create an AUTOSAR ECU out of modules provided as source code and modules provided as object code, even mixed	SWS_Ea_00999
SRS_BSW_00346	All AUTOSAR Basic Software Modules shall provide at least a basic set of module files	SWS_Ea_00113
SRS_BSW_00347	A Naming seperation of different instances of BSW drivers shall be in place	SWS_Ea_00999
SRS_BSW_00348	All AUTOSAR standard types and constants shall be placed and organized in a standard type header file	SWS_Ea_00999
SRS_BSW_00353	All integer type definitions of target and compiler specific scope shall be placed and organized in a single type header	SWS_Ea_00999
SRS_BSW_00361	All mappings of not standardized keywords of compiler specific scope shall be placed and organized in a compiler specific type and keyword header	SWS_Ea_00999
SRS_BSW_00371	The passing of function pointers as API parameter is forbidden for all AUTOSAR Basic Software Modules	SWS_Ea_00999
SRS_BSW_00373	The main processing function of each AUTOSAR Basic Software Module shall be named according the defined convention	SWS_Ea_00096
SRS_BSW_00378	AUTOSAR shall provide a boolean type	SWS_Ea_00999
SRS_BSW_00385	List possible error notifications	SWS_Ea_00099, SWS_Ea_00100
SRS_BSW_00392	Parameters shall have a type	SWS_Ea_00083, SWS_Ea_00117
SRS_BSW_00401	Documentation of multiple instances of configuration parameters shall be available	SWS_Ea_00999

SRS_BSW_00406	A static status variable denoting if a BSW module is initialized shall be initialized with value 0 before any APIs of the BSW module is called	SWS_Ea_00035, SWS_Ea_00128, SWS_Ea_00129, SWS_Ea_00130, SWS_Ea_00131, SWS_Ea_00132, SWS_Ea_00134, SWS_Ea_00136, SWS_Ea_00171, SWS_Ea_00178
SRS_BSW_00407	Each BSW module shall provide a function to read out the version information of a dedicated module implementation	SWS_Ea_00092
SRS_BSW_00414	Init functions shall have a pointer to a configuration structure as single parameter	SWS_Ea_00190, SWS_Ea_00191
SRS_BSW_00415	Interfaces which are provided exclusively for one module shall be separated into a dedicated header file	SWS_Ea_00999
SRS_BSW_00416	The sequence of modules to be initialized shall be configurable	SWS_Ea_00999
SRS_BSW_00417	Software which is not part of the SW-C shall report error events only after the DEM is fully operational.	SWS_Ea_00999
SRS_BSW_00422	Pre-de-bouncing of error status information is done within the DEM	SWS_Ea_00999
SRS_BSW_00423	BSW modules with AUTOSAR interfaces shall be describable with the means of the SW-C Template	SWS_Ea_00999
SRS_BSW_00424	BSW module main processing functions shall not be allowed to enter a wait state	SWS_Ea_00999
SRS_BSW_00425	The BSW module description template shall provide means to model the defined trigger conditions of schedulable objects	SWS_Ea_00999
SRS_BSW_00426	BSW Modules shall ensure data consistency of data which is shared between BSW modules	SWS_Ea_00999
SRS_BSW_00427	ISR functions shall be defined and documented in the BSW module description template	SWS_Ea_00999
SRS_BSW_00428	A BSW module shall state if its main processing	SWS_Ea_00999

	function(s) has to be executed in a specific order or sequence	
SRS_BSW_00429	BSW modules shall be only allowed to use OS objects and/or related OS services	SWS_Ea_00999
SRS_BSW_00432	Modules should have separate main processing functions for read/receive and write/transmit data path	SWS_Ea_00999
SRS_BSW_00433	Main processing functions are only allowed to be called from task bodies provided by the BSW Scheduler	SWS_Ea_00999
SRS_BSW_00456	- A Header file shall be defined in order to harmonize BSW Modules	SWS_Ea_00114
SRS_MemHwAb_14001	The FEE and EA modules shall allow the configuration of the alignment of the start and end addresses of logical blocks	SWS_Ea_00005, SWS_Ea_00068, SWS_Ea_00075, SWS_Ea_00137
SRS_MemHwAb_14002	The FEE and EA modules shall allow the configuration of a required number of write cycles for each logical block	SWS_Ea_00080
SRS_MemHwAb_14005	The FEE and EA modules shall provide upper layers with a virtual 32bit address space	SWS_Ea_00066, SWS_Ea_00075
SRS_MemHwAb_14006	The start address for a block erase or write operation shall always be aligned to the virtual 64K boundary	SWS_Ea_00024
SRS_MemHwAb_14007	The start address and length for reading a block shall not be limited to a certain alignment	SWS_Ea_00021
SRS_MemHwAb_14009	The FEE and EA modules shall provide a conversion between the logical linear addresses and the physical memory addresses	SWS_Ea_00007, SWS_Ea_00021, SWS_Ea_00024, SWS_Ea_00036, SWS_Ea_00063
SRS_MemHwAb_14010	The FEE and EA modules shall provide a write service that operates only on complete configured logical blocks	SWS_Ea_00087, SWS_Ea_00151, SWS_Ea_00159, SWS_EA_00181
SRS_MemHwAb_14012	Spreading of write access	SWS_Ea_00079
SRS_MemHwAb_14013	Writing of immediate data	SWS_Ea_00025

	shall not be delayed by internal management operations nor by erasing the memory area to be written to	
SRS_MemHwAb_14014	The FEE and EA modules shall detect possible data inconsistencies due to aborted / interrupted write operations	SWS_Ea_00046, SWS_Ea_00047, SWS_EA_00188, SWS_EA_00189
SRS_MemHwAb_14015	The FEE and EA modules shall report possible data inconsistencies	SWS_Ea_00104
SRS_MemHwAb_14016	The FEE and EA modules shall not return inconsistent data to the caller	SWS_Ea_00104
SRS_MemHwAb_14017	The EA module shall extend the functional scope of an EEPROM driver	SWS_Ea_00150
SRS_MemHwAb_14018	The FEE module shall extend the functional scope of an internal flash driver	SWS_Ea_00999
SRS_MemHwAb_14026	The block numbers 0x0000 and 0xFFFF shall not be used	SWS_Ea_00006
SRS_MemHwAb_14028	The FEE and EA modules shall provide a service to invalidate a logical block	SWS_Ea_00037, SWS_Ea_00074, SWS_Ea_00091, SWS_Ea_00194
SRS_MemHwAb_14029	The FEE and EA modules shall provide a read service that allows reading all or part of a logical block	SWS_Ea_00022, SWS_Ea_00086, SWS_Ea_00158, SWS_EA_00179
SRS_MemHwAb_14031	The FEE and EA modules shall provide a service that allows canceling an ongoing asynchronous operation	SWS_Ea_00077, SWS_Ea_00078, SWS_Ea_00088, SWS_Ea_00160
SRS_MemHwAb_14032	The FEE and EA modules shall provide an erase service that operates only on complete logical blocks containing immediate data	SWS_Ea_00063, SWS_Ea_00064, SWS_Ea_00065, SWS_Ea_00093, SWS_Ea_00104
SRS_SPAL_00157	All drivers and handlers of the AUTOSAR Basic Software shall implement notification mechanisms of drivers and handlers	SWS_Ea_00999
SRS_SPAL_12063	All driver modules shall only support raw value mode	SWS_Ea_00999
SRS_SPAL_12064	All driver modules shall raise an error if the change of the operation mode	SWS_Ea_00999

	leads to degradation of running operations	
SRS_SPAL_12067	All driver modules shall set their wake-up conditions depending on the selected operation mode	SWS_Ea_00999
SRS_SPAL_12068	The modules of the MCAL shall be initialized in a defined sequence	SWS_Ea_00999
SRS_SPAL_12069	All drivers of the SPAL that wake up from a wake-up interrupt shall report the wake-up reason	SWS_Ea_00999
SRS_SPAL_12077	All drivers shall provide a non blocking implementation	SWS_Ea_00999
SRS_SPAL_12078	The drivers shall be coded in a way that is most efficient in terms of memory and runtime resources	SWS_Ea_00999
SRS_SPAL_12092	The driver's API shall be accessed by its handler or manager	SWS_Ea_00999
SRS_SPAL_12125	All driver modules shall only initialize the configured resources	SWS_Ea_00999
SRS_SPAL_12129	The ISRs shall be responsible for resetting the interrupt flags and calling the according notification function	SWS_Ea_00999
SRS_SPAL_12163	All driver modules shall implement an interface for de-initialization	SWS_Ea_00999
SRS_SPAL_12263	The implementation of all driver modules shall allow the configuration of specific module parameter types at link time	SWS_Ea_00999
SRS_SPAL_12265	Configuration data shall be kept constant	SWS_Ea_00999
SRS_SPAL_12267	Wakeup sources shall be initialized by MCAL drivers and/or the MCU driver	SWS_Ea_00999
SRS_SPAL_12461	Specific rules regarding initialization of controller registers shall apply to all driver implementations	SWS_Ea_00999
SRS_SPAL_12462	The register initialization settings shall be published	SWS_Ea_00999
SRS_SPAL_12463	The register initialization settings shall be combined	SWS_Ea_00999

	and forwarded	

7 Functional specification

7.1 General behavior

[SWS_Ea_00137] ⌈ The EEPROM Abstraction (EA) shall only accept one job at a time, i.e. the module shall not provide a queue for pending jobs (that's the job of the NVRAM Manager). ⌋ (SRS_MemHwAb_14001)

Note: Since the NvM is the only caller for this module and in order to keep this module reasonably small, the modules functions shall not check, whether the module is currently busy or not. It is the responsibility of the NvM to serialize the pending jobs and only start a new job after the previous one has been finished or canceled.

7.1.1 Addressing scheme and segmentation

The EEPROM Abstraction (EA) provides upper layers with a 32bit virtual linear address space and uniform segmentation scheme. This virtual 32bit addresses consists of

- a 16bit block number – allowing a (theoretical) number of 65536 logical blocks
- a 16bit block offset – allowing a (theoretical) block size of 64Kbyte per block

The 16bit block number represents a configurable (virtual) paging mechanism. The values for this address alignment can be derived from that of the underlying EEPROM driver and device. This virtual paging is configurable via the parameter `EA_VIRTUAL_PAGE_SIZE`.

[SWS_Ea_00075] ⌈ The configuration of the Ea module shall be such that the virtual page size (defined in `EA_VIRTUAL_PAGE_SIZE`) is an integer multiple of the physical page size, i.e. it is not allowed to configure a smaller virtual page than the actual physical page size. ⌋ (SRS_MemHwAb_14001, SRS_MemHwAb_14005)

Example:
The size of a virtual page is configured to be eight bytes, thus the address alignment is eight bytes. The logical block with block number 1 is placed at physical address x. The logical block with the block number 2 then would be placed at x+8, block number 3 would be placed at x+16.

Note: This specification requirement allows the physical start address of a logical block to be calculated rather than making a lookup table necessary for the address mapping.

[SWS_Ea_00005] ⌈ Each configured logical block shall take up an integer multiple of the configured virtual page size (see also Chapter 10.2.3, configuration parameter `EA_VIRTUAL_PAGE_SIZE`). ⌋ (SRS_MemHwAb_14001)

Example: If the virtual page size is configured to be eight bytes, logical blocks can be of size 8, 16, 24, 32, … bytes but not e.g. 10, 20, 50, … bytes.

[SWS_Ea_00068] ⌈ Logical blocks must not overlap each other and must not be contained within one another. ⌋ (SRS_MemHwAb_14001)

Example: The address alignment / virtual paging is configured to be eight bytes by setting the parameter `EA_VIRTUAL_PAGE_SIZE` *accordingly. The logical block number 1 is configured to have a size of 32 bytes (see* Figure 3*). This logical block would use exactly 4 virtual pages. The next logical block thus would get the block number 5, since block numbers 2, 3 and 4 are "blocked" by the first logical block. This second block is configured to have a size of 100 bytes, taking up 13 virtual pages and leaving 4 bytes of the last page unused. The next available logical block number thus would be 17.*

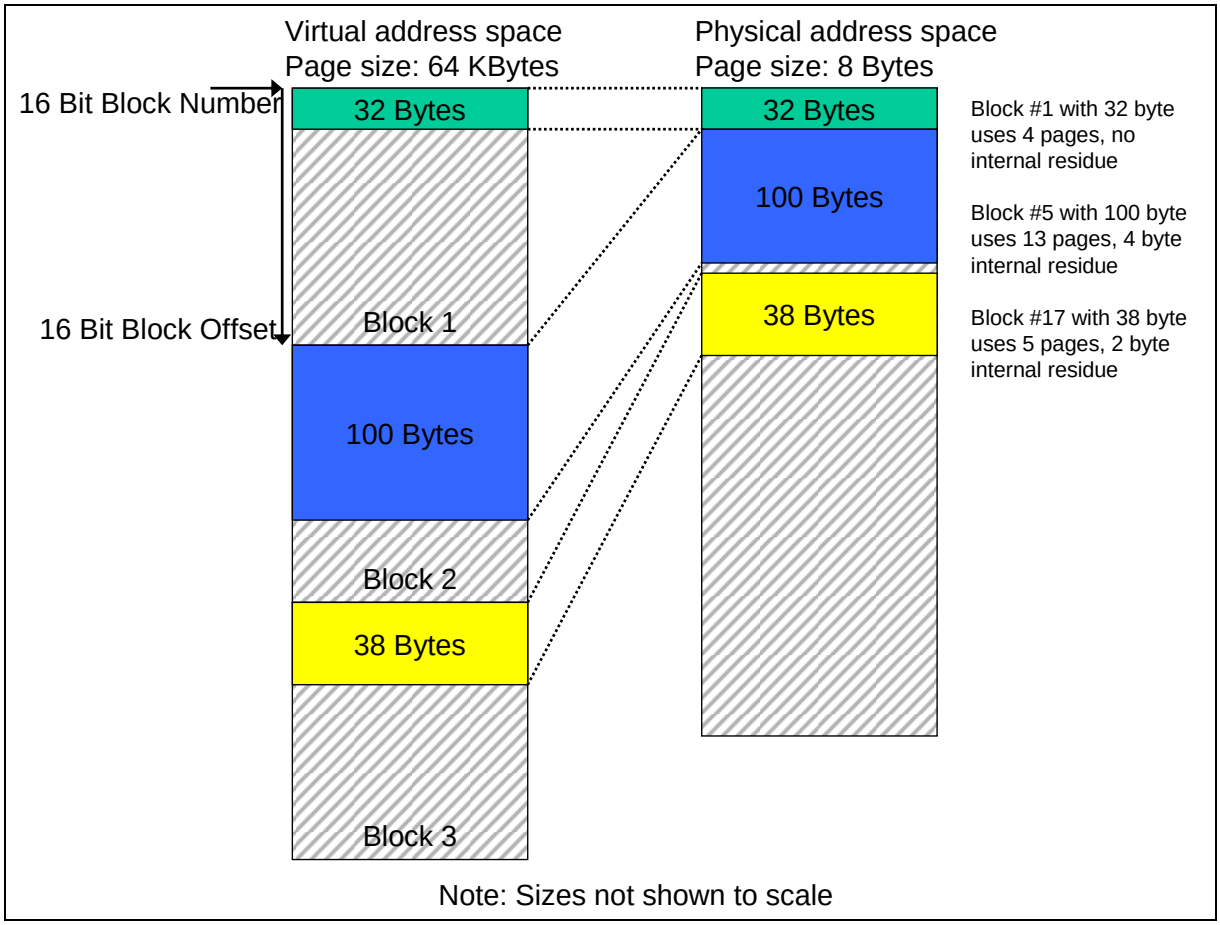


Figure 3: Virtual vs. physical memory layout

[SWS_Ea_00006] ⌈ The block numbers 0x0000 and 0xFFFF shall not be configurable for a logical block (see chapter 10.2.3, `EaBlockNumber` for details). ⌋ (SRS_MemHwAb_14026)

7.1.2 Address calculation

[SWS_Ea_00007] [Depending on the implementation of the EA module and the exact address format used, the functions of the EA module shall combine the 16bit block number and 16bit block offset to derive the physical EEPROM address needed for the underlying EEPROM driver.] (SRS_MemHwAb_14009)

Note: The exact address format needed by the underlying EEPROM driver and therefore the mechanism how to derive the physical EEPROM address from the given 16bit block number and 16bit block offset depends on the EEPROM device and the implementation of the EEPROM device driver and can therefore not be specified in this document.

[SWS_Ea_00066] [Only those bits of the 16bit block number, that do not denote a specific dataset or redundant copy shall be used for address calculation.] (SRS_MemHwAb_14005)

Note: Since this information is needed by the NVRAM manager, the number of bits to encode this can be configured for the NVRAM manager with the parameter `NVM_DATASET_SELECTION_BITS`*.*

*Example: Dataset information is configured to be encoded in the four LSB's of the 16bit block number (allowing for a maximum of 16 datasets per NVRAM block and a total of 4094 NVRAM blocks). An implementer decides to store all datasets of a logical block directly adjacent and using the length of the block and a pointer to access each dataset. To calculate the start address of the block (the address of the first dataset) she/he uses only the 12 MSB's, to access a specific dataset she/he adds the size of the block multiplied by the dataset index (the four MSB's) to this start address (*Figure 4*).*

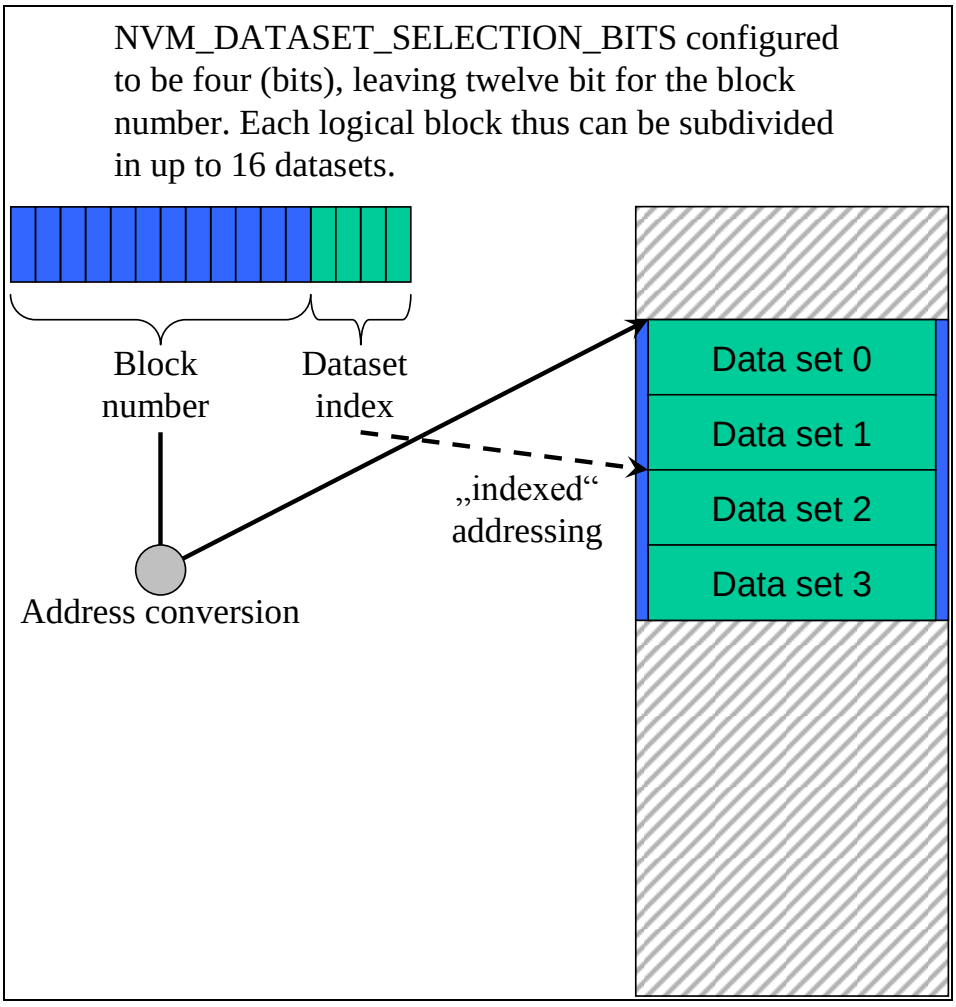


Figure 4: Block number and dataset index

7.1.3 Limitation of erase / write cycles

[SWS_Ea_00079] [The configuration of the Ea module shall define the expected number of erase/write cycles for each logical block in the configuration parameter `EaNumberOfWriteCycles`. **]** (SRS_MemHwAb_14012)

[SWS_Ea_00080] **[** If the underlying EEPROM device or device driver does not provide at least the configured number of erase/write cycles per physical memory cell (given in the parameter `EepAllowedWriteCycles`), the EA module shall provide mechanisms to spread the erase/ write access such that the physical device is not overstressed. This shall also apply to all management data used internally by the EA module. **]** (SRS_MemHwAb_14002)

Example: The logical block number 1 is configured for an expected 500.000 write cycles, the underlying EEPROM device and device driver are only specified for 100.000 erase cycles. In this case the EA module has to provide (at least) five separate memory areas and alternate the access between those areas internally, so that each physical memory location is only erased for a maximum of the specified 100.000 cycles.

7.1.4 Handling of "immediate" data

Blocks, containing immediate data, have to be written instantaneously, i.e. such blocks shall be writable without the need, to first erase the corresponding memory area (e.g. by using pre-erased memory). An ongoing lower priority read / erase / write or compare job shall be canceled by the NVRAM manager before immediate data is written.

Note: A running operation on the hardware (e.g. writing one page or erasing one sector) can usually not be aborted once it has been started. The maximum time of the longest hardware operation thus has to be accepted as delay even for immediate data.

Example: Three blocks with 10 bytes each have been configured for immediate data. The EA module / configuration tool reserves these 30 bytes (plus the implementation specific overhead per block / page if needed) for use by this immediate data only. That is this memory area shall not be used for storage of other data blocks.
Now, the NVRAM manager has requested the EA module to write a data block of 100 bytes. While this block is being written a situation occurs that one (or several) of the immediate data blocks need to be written. Therefore the NVRAM manager cancels the ongoing write request and subsequently issues the write request for the (first) block containing immediate data. The cancelation of the ongoing write request is performed synchronously by the EA module and the underlying EEPROM driver that is the write request for the immediate data can be started without any further delay. However, before the first bytes of immediate data can be written, the EA module respectively the underlying EEPROM driver have to wait for the end of an ongoing hardware access from the previous write request (e.g. writing of a page, erasing of a sector, transfer via SPI, …).

7.1.5 Managing block consistency information

[SWS_Ea_00046] [The Ea module shall manage for each block the information, whether this block is "correct" from the point of view of the EA module or not. This consistency information shall only concern the internal handling of the block, not the block's contents. **]** (SRS_MemHwAb_14014)

[SWS_Ea_00047] [When a block write operation is started the EA module shall mark the corresponding block as inconsistent[1]. Upon the successful end of the block

[1] This does not necessarily mean a write operation on the physical device. If there are other means to detect the consistency of a logical block, changing the management information stored with the block shall be avoided.

write operation, the EA module shall mark the block as consistent (again).] (SRS_MemHwAb_14014)

Note: This internal management information should not be mixed up with the validity information of a block which can be manipulated by using the Ea_InvalidateBlock service, i.e. the EA module shall be able to distinguish between an inconsistent block and a block that has been deliberately invalidated by the upper layer.

7.2 Error classification

7.2.1 Development Errors

[SWS_Ea_00196][The Ea module shall detect the following errors and exceptions depending on its configuration (development/production):

Type or error	*Relevance*	*Related error code*	*Value [hex]*
API service called while module is not (yet) initialized	Development	EA_E_UNINIT	0x01
API service called with invalid block number	Development	EA_E_INVALID_BLOCK_NO	0x02
API service called with invalid block offset	Development	EA_E_INVALID_BLOCK_OFS	0x03
API service called with invalid pointer argument	Development	EA_E_PARAM_POINTER	0x04
API service called with invalid block length information	Development	EA_E_INVALID_BLOCK_LEN	0x05
API service called while module is busy	Development	EA_E_BUSY	0x06
Ea_Cancel called while no job was pending	Development	EA_E_INVALID_CANCEL	0x08
Ea_Init failed	Development	EA_E_INIT_FAILED	0x09

] ()

7.2.2 Runtime Errors

There are no runtime errors.

7.2.3 Transient Faults

There are no transient faults.

7.2.4 Production Errors

There are no production errors.

7.2.5 Extended Production Errors

There are no extended production errors.

8 API specification

8.1 Imported Types

[SWS_Ea_00083] ⌈

Module	*Imported Type*
Eep	Eep_AddressType
	Eep_LengthType
MemIf	MemIf_JobResultType
	MemIf_ModeType
	MemIf_StatusType
Std_Types	Std_ReturnType
	Std_VersionInfoType

⌋ (SRS_BSW_00392)

[SWS_Ea_00117] ⌈ The types mentioned in SWS_Ea_00083 shall not be changed or extended for a specific EA module or hardware platform. ⌋ (SRS_BSW_00392)

8.2 Type definitions

[SWS_Ea_00190] ⌈

Name:	`Ea_ConfigType`	
Type:	`Structure`	
Range:	`implementation specific`	--
Description:	Configuration data structure of the Ea module.	

⌋ (SRS_BSW_00414)

8.3 Function definitions

8.3.1 Ea_Init

[SWS_Ea_00084] ⌈

Service name:	Ea_Init	
Syntax:	`void Ea_Init( const Ea_ConfigType* ConfigPtr )`	
Service ID[hex]:	0x00	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	ConfigPtr	Pointer to the selected configuration set.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Initializes the EEPROM abstraction module.	

⌋ (SRS_BSW_00101)

[**SWS_Ea_00191**]⌈ The configuration pointer `ConfigPtr` shall always have a `NULL_PTR` value.

] (SRS_BSW_00414)

Note: the Configuration pointer ConfigPtr is currently not used and shall therefore be set `NULL_PTR` value.

[SWS_Ea_00017] [The function `Ea_Init` shall shall set the module state from `MEMIF_UNINIT` to `MEMIF_BUSY_INTERNAL` once it starts the module's initialization.] (SRS_BSW_00101)

[SWS_Ea_00128] [If initialization is finished within `Ea_Init`, the function Ea_Init shall set the module state from `MEMIF_BUSY_INTERNAL` to `MEMIF_IDLE` once initialization has been successfully finished.] (SRS_BSW_00406)

Note: The Ea module's environment shall not call the function Ea_Init during a running operation of the EA module.

8.3.2 Ea_SetMode

[SWS_Ea_00085] [

Service name:	Ea_SetMode	
Syntax:	`void Ea_SetMode( MemIf_ModeType Mode )`	
Service ID[hex]:	0x01	
Sync/Async:	Asynchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	Mode	Desired mode for the underlying EEPROM driver
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Function to switch the mode of the underlying EEPROM Driver	

] (RS_BRF_01480)

[SWS_Ea_00150] [The function `Ea_SetMode` shall be enabled / disabled via the pre-compile time parameter `EaSetModeSupported` such that the function is completely removed from the code if it is disabled.] (SRS_MemHwAb_14017, RS_BRF_01480)

[SWS_Ea_00129] [If development error detection is enabled for the module: the function `Ea_SetMode` shall check if the module state is `MEMIF_UNINIT`. If this is the case, the function `Ea_SetMode` shall raise the development error `EA_E_UNINIT` and return to the caller without executing the mode switch.] (SRS_BSW_00406)

[SWS_Ea_00165] [If development error detection is enabled for the module: the function `Ea_SetMode` shall check if the module state is `MEMIF_BUSY`. If this is the case, the function `Ea_SetMode` shall raise the development error `EA_E_BUSY` and return to the caller without executing the mode switch.] (SRS_BSW_00323)

[**SWS_Ea_00192**][The function `Ea_SetMode` shall check if the module state is `MEMIF_IDLE` or `MEMIF_BUSY_INTERNAL`. If this is the case the module shall accept the mode change request. The mode change shall be executed asynchronously in the module's main function as soon as the module has finished the internal management operation.
] (RS_BRF_01480)

[**SWS_Ea_00193**][The Ea module shall execute the mode change request asynchronously within the Ea module's main function.] (RS_BRF_01812)

8.3.3 Ea_Read

[SWS_Ea_00086] [

Service name:	Ea_Read	

Syntax:	`Std_ReturnType Ea_Read(`
	`  uint16 BlockNumber,`
	`  uint16 BlockOffset,`
	`  uint8* DataBufferPtr,`
	`  uint16 Length`
	`)`	

Service ID[hex]:	0x02	
Sync/Async:	Asynchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	BlockNumber	Number of logical block, also denoting start address of that block in EEPROM.
	BlockOffset	Read address offset inside the block
	Length	Number of bytes to read
Parameters (inout):	None	
Parameters (out):	DataBufferPtr	Pointer to data buffer
Return value:	Std_ReturnType	E_OK: The requested job has been accepted by the module. E_NOT_OK: The requested job has not been accepted by the EA module.
Description:	Reads Length bytes of block Blocknumber at offset BlockOffset into the buffer DataBufferPtr.	

] (SRS_MemHwAb_14029)

[SWS_Ea_00021] [The function `Ea_Read` shall take the block number and offset and calculate the corresponding memory read address.] (SRS_MemHwAb_14007, SRS_MemHwAb_14009)

Note: The address offset and length parameter can take any value within the given types range, this allows reading of an arbitrary number of bytes from an arbitrary address inside a logical block.

[SWS_Ea_00072] [The EA module shall execute the read operation asynchronously within the EA module's main function.] (RS_BRF_01048)

[SWS_Ea_00022] [If the current module status is MEMIF_IDLE or if the current module status is `MEMIF_BUSY_INTERNAL`, the function `Ea_Read` shall accept the read request, copy the given / computed parameters to module internal variables,

initiate a read job, set the EA module status to MEMIF_BUSY, set the job result to MEMIF_JOB_PENDING and return with E_OK.](SRS_MemHwAb_14029)

[SWS_EA_00179] [If the current module status is MEMIF_UNINIT or MEMIF_BUSY, the function Ea_Read shall reject the job request and return with E_NOT_OK.] (SRS_MemHwAb_14029)

[SWS_Ea_00130] [If development error detection for the module EA is enabled: the function Ea_Read shall check if the module state is MEMIF_UNINIT. If this is the case, the function Ea_Read shall reject the read request, raise the development error EA_E_UNINIT and return with E_NOT_OK.] (SRS_BSW_00406)

[SWS_Ea_00167] [If development error detection is enabled for the module: the function Ea_Read shall check if the module state is MEMIF_BUSY. If this is the case, the function Ea_Read shall reject the read request, raise the development error EA_E_BUSY and return with E_NOT_OK.] (SRS_BSW_00323)

[SWS_Ea_00147] [If development error detection is enabled for the module: the function Ea_Read shall check whether the given block number is valid (i.e. inside the configured range). If this is not the case, the function Ea_Read shall reject the read request, raise the development error EA_E_INVALID_BLOCK_NO and return E_NOT_OK.] (SRS_BSW_00323)

[SWS_Ea_00168] [If development error detection is enabled for the module: the function Ea_Read shall check that the given block offset is valid (i.e. that it is less than the block length configured for this block). If this is not the case, the function Ea_Read shall reject the read request, raise the development error EA_E_INVALID_BLOCK_OFS and return with E_NOT_OK.] (SRS_BSW_00323)

[SWS_Ea_00169] [If development error detection is enabled for the module: the function Ea_Read shall check that the given length information is valid, i.e. that the requested length information plus the block offset do not exceed the block end address (block start address plus configured block length). If this is not the case, the function Ea_Read shall reject the read request, raise the development error EA_E_INVALID_BLOCK_LEN and return with E_NOT_OK.] (SRS_BSW_00323)

[SWS_Ea_00170] [If development error detection is enabled for the module: the function Ea_Read shall check that the given data pointer is valid (i.e. that it is not NULL). If this is not the case, the function Ea_Read shall reject the read request, raise the development error EA_E_PARAM_POINTER and return with E_NOT_OK.] (SRS_BSW_00323)

[SWS_Ea_00158] [If a read request is rejected by the function Ea_Read, i.e. requirements SWS_Ea_00130, SWS_Ea_00147, SWS_Ea_00167, SWS_Ea_00168, SWS_Ea_00169, SWS_Ea_00170, EA179 or EA180 apply, the function Ea_Read shall not change the current module status or job result.] (SRS_MemHwAb_14029, SRS_BSW_00323)

8.3.4 Ea_Write

[SWS_Ea_00087] ⌈

Service name:	Ea_Write	
Syntax:	`Std_ReturnType Ea_Write( uint16 BlockNumber, const uint8* DataBufferPtr )`	
Service ID[hex]:	0x03	
Sync/Async:	Asynchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	BlockNumber	Number of logical block, also denoting start address of that block in EEPROM.
	DataBufferPtr	Pointer to data buffer
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: The requested job has been accepted by the module. E_NOT_OK: The requested job has not been accepted by the EA module.
Description:	Writes the contents of the DataBufferPtr to the block BlockNumber.	

⌋ (SRS_MemHwAb_14010)

[SWS_Ea_00024] ⌈ The function `Ea_Write` shall take the block number and calculate the corresponding memory write address. The block offset shall be fixed to zero for this address calculation. ⌋ (SRS_MemHwAb_14006, SRS_MemHwAb_14009)

[SWS_Ea_00151] ⌈ The function `Ea_Write` shall set the length parameter for the write job to the length configured for this logical block. ⌋ (SRS_MemHwAb_14010)

[SWS_Ea_00025] ⌈ If the current module status is `MEMIF_IDLE` or if the current module status is `MEMIF_BUSY INTERNAL`, the function `Ea_Write` shall accept the write request, copy the given / computed parameters to module internal variables, initiate a write job, set the EA module status to `MEMIF_BUSY`, set the job result to `MEMIF_JOB_PENDING` and return with `E_OK`.⌋ (SRS_MemHwAb_14013)

[SWS_EA_00181] ⌈ If the current module status is `MEMIF_UNINIT` or `MEMIF_BUSY`, the function `Ea_Write` shall reject the job request and return with `E_NOT_OK`. ⌋ (SRS_MemHwAb_14010)

[SWS_Ea_00026] ⌈ The EA module shall execute the write job of the function `Ea_Write` asynchronously within the EA module's main function. ⌋ (RS_BRF_01048)

[SWS_Ea_00131] ⌈ If development error detection for the module EA is enabled: the function `Ea_Write` shall check if the module state is `MEMIF_UNINIT`. If this is the case, the function `Ea_Write` shall reject the write request, raise the development error `EA_E_UNINIT` and return with `E_NOT_OK`. ⌋ (SRS_BSW_00406)

[SWS_Ea_00171] ⌈ If development error detection is enabled for the module: the function `Ea_Write` shall check if the module state is `MEMIF_BUSY`. If this is the case, the function `Ea_Write` shall reject the write request, raise the development error `EA_E_BUSY` and return with `E_NOT_OK`. ⌋ (SRS_BSW_00406)

[SWS_Ea_00148] ⌈ If development error detection for the module EA is enabled: the function `Ea_Write` shall check whether the given block number is valid (i.e. inside the configured range). If this is not the case, the function `Ea_Write` shall reject the write request, raise the development error `EA_E_INVALID_BLOCK_NO` and return with `E_NOT_OK`. ⌋ (SRS_BSW_00323)

[SWS_Ea_00172] ⌈ If development error detection is enabled for the module: the function `Ea_Write` shall check that the given data pointer is valid (i.e. that it is not NULL). If this is not the case, the function `Ea_Write` shall reject the write request, raise the development error `EA_E_PARAM_POINTER` and return with `E_NOT_OK`. ⌋ (SRS_BSW_00323)

[SWS_Ea_00159] ⌈ If a write request is rejected by the function `Ea_Write`, i.e. requirements SWS_Ea_00131, SWS_Ea_00171, SWS_Ea_00148, SWS_Ea_00172, EA181 or EA183 apply, the function `Ea_Write` shall not change the current module status or job result. ⌋ (SRS_MemHwAb_14010, SRS_BSW_00323)

8.3.5 Ea_Cancel

[SWS_Ea_00088] ⌈

Service name:	Ea_Cancel

Syntax:	`void Ea_Cancel(`
	`  void`
	`)`

Service ID[hex]:	0x04
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	Cancels the ongoing asynchronous operation.

⌋ (SRS_MemHwAb_14031)

Note: The function `Ea_Cancel` and the cancel function of the underlying EEPROM driver are synchronous in their bsolete, i.e. their job is done once they return to the caller. On the other hand, they are asynchronous w.r.t. an ongoing read, erase or write job in the EEPROM memory. The cancel functions shall only reset their modules internal variables so that a new job can be accepted by the modules. They do not cancel an ongoing job in the hardware and they do not wait for an ongoing job to be finished by the hardware. This might lead to the situation in which the module's state is reported as IDLE while there is still an ongoing job being executed by the

hardware. Therefore, the EEPROM driver's main function shall check that the hardware is indeed free before starting a new job (see chapter 9.3 for a detailed sequence diagram).

Note: The function `Ea_Cancel` should only be used by the `NvM` to abort a read or write request for an NV block if higher priority data (i.e. immediate data) has to be written.

[SWS_Ea_00132] ⌈ If development error detection for the module EA is enabled: the function `Ea_Cancel` shall check if the module state is `MEMIF_UNINIT`. If this is the case, the function `Ea_Cancel` shall raise the development error `EA_E_UNINIT` and return to the caller without changing any internal variables. ⌋ (SRS_BSW_00406)

[SWS_Ea_00077] ⌈ If the current module status is `MEMIF_BUSY` (i.e. the request to cancel a pending job is accepted by the function `Ea_Cancel`), the function `Ea_Cancel` shall call the cancel function of the underlying EEPROM driver. ⌋ (SRS_MemHwAb_14031)

[SWS_Ea_00078] ⌈ If the current module status is `MEMIF_BUSY` (i.e. the request to cancel a pending job is accepted by the function `Ea_Cancel`), the function `Ea_Cancel` shall reset the EA module's internal variables to make the module ready for a new job request. I.e. the function `Ea_Cancel` shall set the job result to `MEMIF_JOB_CANCELED` and the module status to `MEMIF_IDLE`. ⌋ (SRS_MemHwAb_14031)

[SWS_Ea_00160] ⌈ If the current module status is not `MEMIF_BUSY` (i.e. the request to cancel a pending job is rejected by the function `Ea_Cancel`), the function Ea_Cancel shall not change the current module status or job result. ⌋ (SRS_MemHwAb_14031)

[SWS_Ea_00173] ⌈ If development error detection is enabled for the module: If the current module status is not `MEMIF_BUSY` (i.e. there is no job to cancel and therefore the request to cancel a pending job is rejected by the function `Ea_Cancel`), the function `Ea_Cancel` shall raise the development error `EA_E_INVALID_CANCEL`. ⌋ (SRS_BSW_00323)

8.3.6 Ea_GetStatus

[SWS_Ea_00089] ⌈

Service name:	Ea_GetStatus

Syntax:	`MemIf_StatusType Ea_GetStatus(`
	`  void`
	`)`

Service ID[hex]:	0x05
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters	None

(inout):		
Parameters (out):	None	
Return value:	MemIf_StatusType	MEMIF_UNINIT: The EA module has not been initialized (yet). MEMIF_IDLE: The EA module is currently idle. MEMIF_BUSY: The EA module is currently busy. MEMIF_BUSY_INTERNAL: The EA module is currently busy with internal management operations.
Description:	Service to return the Status.	

] (RS_BRF_01048)

[SWS_Ea_00034] [The function `Ea_GetStatus` shall return `MEMIF_UNINIT` if the module has not (yet) been initialized.] ()

[SWS_Ea_00156] [The function `Ea_GetStatus` shall return `MEMIF_IDLE` if the module is neither processing a request from the upper layer nor is it doing an internal management operation.] ()

[SWS_Ea_00157] [The function `Ea_GetStatus` shall return `MEMIF_BUSY` if it is currently processing a request from the upper layer.] ()

[SWS_Ea_00073] [The function `Ea_GetStatus` shall return `MEMIF_BUSY_INTERNAL`, if an internal management operation is currently ongoing.] ()

Note: Internal management operation may e.g. be a re-organization of the used EEPROM memory (garbage collection). This may imply that the underlying device driver is – at least temporarily – busy.

8.3.7 Ea_GetJobResult

[SWS_Ea_00090] [

Service name:	Ea_GetJobResult	
Syntax:	`MemIf_JobResultType Ea_GetJobResult( void )`	
Service ID[hex]:	0x06	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	None	
Return value:	MemIf_JobResultType	MEMIF_JOB_OK: The last job has been finished successfully. MEMIF_JOB_PENDING: The last job is waiting for execution or currently being executed. MEMIF_JOB_CANCELED: The last job has been canceled (which means it failed). MEMIF_JOB_FAILED: The last job was not finished successfully (it failed). MEMIF_BLOCK_INCONSISTENT: The requested block is inconsistent, it may contain corrupted data.

		MEMIF_BLOCK_INVALID: The requested block has been invalidated, the requested operation can not be performed.
Description:	Service to return the JobResult.	

] (RS_BRF_01048)

[SWS_Ea_00134] [If development error detection for the module EA is enabled: the function `Ea_GetJobResult` shall check if the module state is `MEMIF_UNINIT`. If this is the case, the function `Ea_GetJobResult` shall raise the development error `EA_E_UNINIT` and return with `MEMIF_JOB_FAILED`.] (SRS_BSW_00406)

[SWS_Ea_00035] [The function `Ea_GetJobResult` shall return the status of the last job requested by the NVRAM manager.] (SRS_BSW_00406)

[SWS_Ea_00174] [Only those jobs which have been requested directly by the upper layer shall have influence on the job result returned by the function `Ea_GetJobResult`. I.e. jobs which are issued by the EA module itself in the course of internal management operations shall not alter the job result.] (RS_BRF_01048)

Note: To facilitate this, the EA module may have to implement a second set of local variables to store the data for internal jobs.

Note: Internal management operations (e.g. "garbage collection") will only be invoked in the context of jobs requested from the NvM. Whether they have to be done before or after the requested job is the decision of the modules implementor and shall not be detailed in this specification.

8.3.8 Ea_InvalidateBlock

[SWS_Ea_00091] [

Service name:	Ea_InvalidateBlock	
Syntax:	`Std_ReturnType Ea_InvalidateBlock( uint16 BlockNumber )`	
Service ID[hex]:	0x07	
Sync/Async:	Asynchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	BlockNumber	Number of logical block, also denoting start address of that block in EEPROM.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: The requested job has been accepted by the module. E_NOT_OK - only if DET is enabled: The requested job has not been accepted by the EA module.
Description:	Invalidates the block BlockNumber.	

] (SRS_MemHwAb_14028)

[SWS_Ea_00036] [The function `Ea_InvalidateBlock` shall take the block number and calculate the corresponding memory block address.] (SRS_MemHwAb_14009)

[SWS_Ea_00037] ⌈ Depending on implementation, the function `Ea_InvalidateBlock` shall invalidate the block `<BlockNumber>` by either calling the erase function of the underlying device driver or changing some module internal management information accordingly. ⌋ (SRS_MemHwAb_14028)

Note: How exactly the requested block is invalidated depends on the module's implementation and will not be further detailed in this specification. The internal management information has to be stored in NV memory since it has to be resistant against resets. What this information is and how it is stored is not further detailed by this specification.

[SWS_Ea_00135] ⌈ If development error detection for the module Ea is enabled: the function `Ea_InvalidateBlock` shall check if the module state is `MEMIF_UNINIT`. If this is the case, the function `Ea_InvalidateBlock` shall reject the invalidation request, raise the development error `EA_E_UNINIT` and return with `E_NOT_OK`. ⌋ (SRS_BSW_00323)

[SWS_Ea_00175] ⌈ If development error detection is enabled for the module: the function `Ea_InvalidateBlock` shall check if the module state is `MEMIF_BUSY`. If this is the case, the function `Ea_InvalidateBlock` shall reject the invalidation request, raise the development error `EA_E_BUSY` and return with `E_NOT_OK`. ⌋ (SRS_BSW_00323)

[**SWS_Ea_00194**]⌈ The function `Ea_InvalidateBlock` shall check if the module state is `MEMIF_IDLE` or `MEMIF_BUSY_INTERNAL`. If this is the case the module shall accept the invalidation request and shall return `E_OK` to the caller. The block invalidation shall be executed asynchronously in the module's main function as soon as the module has finished the internal management operation.⌋ (SRS_MemHwAb_14028)

[**SWS_Ea_00195**]⌈ The Ea module shall execute the block invalidation request asynchronously within the Ea module's main function.⌋ (RS_BRF_01812)

[SWS_Ea_00149] ⌈ If development error detection for the module EA is enabled: the function `Ea_InvalidateBlock` shall check whether the given block number is valid (i.e. it has been configured). If this is not the case, the function `Ea_InvalidateBlock` shall reject the request, raise the development error `EA_E_INVALID_BLOCK_NO` and return with `E_NOT_OK`. ⌋ (SRS_BSW_00323)

[SWS_Ea_00161] ⌈ If an invalidation request is rejected by the function `Ea_InvalidateBlock`, i.e. requirements SWS_Ea_00135, SWS_Ea_00149, SWS_Ea_00175 or EA184 apply, the function `Ea_InvalidateBlock` shall not change the current module status or job result. ⌋ (SRS_BSW_00323)

8.3.9 Ea_GetVersionInfo

[SWS_Ea_00092] ⌈

Service name:	Ea_GetVersionInfo	
Syntax:	`void Ea_GetVersionInfo( Std_VersionInfoType* VersionInfoPtr )`	
Service ID[hex]:	0x08	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	VersionInfoPtr	Pointer to standard version information structure.
Return value:	None	
Description:	Service to get the version information of this module.	

] (SRS_BSW_00407)

[SWS_Ea_00164] [If development error detection for the module `EA` is enabled: the function `EA_GetVersionInfo` shall check that the given data pointer is valid (i.e. that it is not NULL). If this is not the case, the function `Ea_GetVersionInfo` shall raise the development error `EA_E_PARAM_POINTER`.] (SRS_BSW_00323)

8.3.10 Ea_EraseImmediateBlock

[SWS_Ea_00093] [

Service name:	Ea_EraseImmediateBlock	
Syntax:	`Std_ReturnType Ea_EraseImmediateBlock( uint16 BlockNumber )`	
Service ID[hex]:	0x09	
Sync/Async:	Asynchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	BlockNumber	Number of logical block, also denoting start address of that block in EEPROM.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: The requested job has been accepted by the module. E_NOT_OK - only if DET is enabled: The requested job has not been accepted by the EA module.
Description:	Erases the block BlockNumber.	

] (SRS_MemHwAb_14032)

Note: The function `Ea_EraseImmediateBlock` shall only be called by e.g. diagnostic or similar system services to pre-erase the area for immediate data if necessary.

[SWS_Ea_00063] [The function `Ea_EraseImmediateBlock` shall take the block number and calculate the corresponding memory block address. The block offset shall be fixed to zero for this address calculation.] (SRS_MemHwAb_14009, SRS_MemHwAb_14032)

[SWS_Ea_00064] ⌈ The function `Ea_EraseImmediateBlock` shall ensure that the EA module can write immediate data. Whether this involves physically erasing a memory area and therefore calling the erase function of the underlying driver depends on the implementation. ⌋ (SRS_MemHwAb_14032)

[SWS_Ea_00136] ⌈ If development error detection for the module EA is enabled: the function `Ea_EraseImmediateBlock` shall check if the module state is `MEMIF_UNINIT`. If this is the case, the function `Ea_EraseImmediateBlock` shall reject the erase request, raise the development error `EA_E_UNINIT` and return with `E_NOT_OK`. ⌋ (SRS_BSW_00406)

[SWS_Ea_00176] ⌈ If development error detection is enabled for the module: the function `Ea_EraseImmediateBlock` shall check if the module state is `MEMIF_BUSY`. If this is the case, the function `Ea_EraseImmediateBlock` shall reject the erase request, raise the development error `EA_E_BUSY` and return with `E_NOT_OK`. ⌋ (SRS_BSW_00323)

[SWS_Ea_00152] ⌈ If development error detection for the module EA is enabled: the function `Ea_EraseImmediateBlock` shall check whether the given block number is valid (i.e. it has been configured). If this is not the case, the function `Ea_EraseImmediateBlock` shall reject the erase request, raise the development error `EA_E_INVALID_BLOCK_NO` and return with `E_NOT_OK`. ⌋ (SRS_BSW_00323)

[SWS_Ea_00065] ⌈ If development error detection for the EA module is enabled, the function `Ea_EraseImmediateBlock` shall check whether the addressed logical block is configured as containing immediate data (configuration parameter `EaImmediateData == TRUE`). If not, the function `Ea_EraseImmediateBlock` shall reject the erase request, raise the deleopment error `EA_E_INVALID_BLOCK_NO` and return with `E_NOT_OK`. ⌋ (SRS_BSW_00323, SRS_MemHwAb_14032)

[SWS_Ea_00162] ⌈ If an erase request for an immediate block is rejected by the function `Ea_EraseImmediateBlock`, i.e. requirements SWS_Ea_00136, SWS_Ea_00176, SWS_Ea_00152, SWS_Ea_00065 or EA185 apply, the function `Ea_EraseImmediateBlock` shall not change the current module status or job result. ⌋ (SRS_BSW_00323)

8.4 Call-back notifications

This chaper lists all functions provided by the Ea module to lower layer modules.

[SWS_Ea_00114] ⌈ The Ea module shall provide function prototypes of the callback functions in the file `Ea_Cbk.h` ⌋ (SRS_BSW_00456)

Note: Depending on the implementation of the modules making up the NV memory stack, callback routines provided by the EA module may be called on interrupt level.

The implementation of the EA module therefore has to make sure that the runtime of those routines is reasonably short, i.e. since callbacks may be propagated upward through several software layers. Whether callback routines are allowable / feasible on interrupt level depends on the project specific needs (reaction time) and limitations (runtime in interrupt context). Therefore system design has to make sure that the configuration of the involved modules meets those requirements.

8.4.1 Ea_JobEndNotification

[SWS_Ea_00094] [

Service name:	Ea_JobEndNotification

Syntax:	`void Ea_JobEndNotification(`
	`  void`
	`)`

Service ID[hex]:	0x10
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	Service to report to this module the successful end of an asynchronous operation.

] (RS_BRF_01064)

The underlying EEPROM driver shall call the function `Ea_JobEndNotification` to report the successful end of an asynchronous operation.

[SWS_Ea_00153] [If the job result is currently `MEMIF_JOB_PENDING`, the function Ea_JobEndNotification shall set the job result to `MEMIF_JOB_OK`, else it shall leave the job result untouched.] (RS_BRF_01064)

[SWS_Ea_00051] [The function `Ea_JobEndNotification` shall perform any necessary block management operations and shall call the corresponding callback routine of the upper layer module (`Ea_NvMJobEndNotification`).] (RS_BRF_01064)

Note: The function `Ea_JobEndNotification` shall be callable on interrupt level.

8.4.2 Ea_JobErrorNotification

[SWS_Ea_00095] [

Service name:	Ea_JobErrorNotification

Syntax:	`void Ea_JobErrorNotification(`
	`  void`
	`)`

Service ID[hex]:	0x11
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant

Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	Service to report to this module the failure of an asynchronous operation.

] (RS_BRF_01064)

The underlying EEPROM driver shall call the function `Ea_JobErrorNotification` to report the failure of an asynchronous operation.

[SWS_Ea_00154] [If the job result is currently `MEMIF_JOB_PENDING`, the function Ea_JobErrorNotification shall set the job result to `MEMIF_JOB_FAILED`, else it shall leave the job result untouched.] (RS_BRF_01064)

[SWS_Ea_00053] [The function `Ea_JobErrorNotification` shall perform any necessary block management and error handling operations and shall call the corresponding callback routine of the upper layer module (`Ea_NvMJobErrorNotification`).] (RS_BRF_01064)

Note: The function `Ea_JobErrorNotification` shall be callable on interrupt level.

8.5 Scheduled functions

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

8.5.1 Ea_MainFunction

[SWS_Ea_00096] [

Service name:	Ea_MainFunction
Syntax:	`void Ea_MainFunction( void )`
Service ID[hex]:	0x12
Description:	Service to handle the requested jobs and the internal management operations.

] (SRS_BSW_00373)

Note: The cycle time for the function `Ea_MainFunction` should be the same as that configured for the underlying EEPROM driver.

[SWS_Ea_00178] [If the module initialization (started in the function `Ea_Init`) is completed in the module's main function, the function `Ea_MainFunction` shall set the module status from `MEMIF_BUSY_INTERNAL` to `MEMIF_IDLE` once initialization of the module has been successfully finished.] (SRS_BSW_00406)

[SWS_Ea_00056] ⌈ The function `Ea_MainFunction` shall asynchronously handle the read / write / erase / invalidate jobs requested by the upper layer and internal management operations. ⌋ (RS_BRF_01048)

[SWS_Ea_00074] ⌈ The function `Ea_MainFunction` shall check, whether the block requested for reading has been invalidated by the upper layer module. If so, the function `Ea_MainFunction` shall set the job result to `MEMIF_BLOCK_INVALID` and call the job error notification function if configured. ⌋ (SRS_MemHwAb_14028)

[SWS_Ea_00104] ⌈ The function `Ea_MainFunction` shall check the consistency of the logical block being read before notifying the caller. If an inconsistency of the block is detected (see SWS_Ea_00046 and SWS_Ea_00047) or if the requested block can't be found, the function `Ea_MainFunction` shall set the job result to `MEMIF_BLOCK_INCONSISTENT` and call the error notification routine of the upper layer if configured. ⌋ (SRS_MemHwAb_14032, SRS_MemHwAb_14015, SRS_MemHwAb_14016)

Note: In this case the upper layer shall not use the contents of the data buffer.

[SWS_EA_00188]: ⌈ If an internal management operation has been suspended because of a job request from the upper layer, the function `Ea_MainFunction` shall resume this internal management operation once the job requested by the upper layer has been finished. ⌋ (SRS_MemHwAb_14014)

[SWS_EA_00189]: ⌈ If an internal management operation has been aborted because of a job request from the upper layer, the function `Ea_MainFunction` shall restart this internal management operation once the job requested by the upper layer has been finished. ⌋ (SRS_MemHwAb_14014)

8.6 Expected Interfaces

In this chapter all interfaces required from other modules are listed.

8.6.1 Mandatory Interfaces

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

[SWS_Ea_00097] ⌈

API function	*Description*
Eep_Cancel	Cancels a running job.
Eep_Erase	Service for erasing EEPROM sections.
Eep_GetJobResult	This service returns the result of the last job.
Eep_GetStatus	Returns the EEPROM status.
Eep_Read	Reads from EEPROM.
Eep_SetMode	Sets the mode.

Eep_Write	Writes to EEPROM.

] (RS_BRF_01056)

8.6.2 Optional Interfaces

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

[SWS_Ea_00098] [

API function	*Description*
Det_ReportError	Service to report development errors.

] (RS_BRF_01056)

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 callback function. The names of this kind of interfaces are not fixed because they are configurable.

Note: Depending on the implementation of the modules making up the NV memory stack, callback routines invoked by the EA module may be called on interrupt level. The implementor of the module providing these routines therefore has to make sure that their runtime is reasonably short, i.e. since callbacks may be propagated upward through several software layers. Whether callback routines are allowable / feasible on interrupt level depends on the project specific needs (reaction time) and limitations (runtime in interrupt context). Therefore system design has to make sure that the configuration of the involved modules meets those requirements.

[SWS_Ea_00099] [

Service name:	NvM_JobEndNotification

Syntax:	`void NvM_JobEndNotification(`
	`  void`
	`)`

Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	Function to be used by the underlying memory abstraction to signal end of job without error.

] (SRS_BSW_00385)

[SWS_Ea_00054] [The Ea module shall call the function defined in the configuration parameter `EaNvMJobEndNotification` upon successful end of an asynchronous read operation after performing all necessary internal management operations. Successful end of an asynchronous read operation implies the read job is finished and the result is OK.] (RS_BRF_01064)

[SWS_Ea_00141] [The Ea module shall call the function defined in the configuration parameter `EaNvMJobEndNotification` upon successful end of an asynchronous write operation after performing all necessary internal management operations. Successful end of an asynchronous write operation implies the write job is finished, the result is OK and the block has been marked as valid.] (RS_BRF_01064)

[SWS_Ea_00142] [The Ea module shall call the function defined in the configuration parameter `EaNvMJobEndNotification` upon successful end of an asynchronous erase operation after performing all necessary internal management operations. Successful end of an asynchronous erase operation implies the erase job for immediate data is finished and the result is OK (see SWS_Ea_00064).] (RS_BRF_01064)

[SWS_Ea_00143] [The Ea module shall call the function defined in the configuration parameter `EaNvMJobEndNotification` upon successful end of an asynchronous block invalidation operation after performing all necessary internal management operations. Successful end of an asynchronous block invalidation operation implies the block invalidation job is finished and the result is OK (i.e. the block has been marked as invalid).] (RS_BRF_01064)

[SWS_Ea_00100] [

Service name:	NvM_JobErrorNotification

Syntax:	`void NvM_JobErrorNotification(`
	`  void`
	`)`

Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	Function to be used by the underlying memory abstraction to signal end of job with error.

] (SRS_BSW_00385)

[SWS_Ea_00055] [The Ea module shall call the function defined in the configuration parameter `EaNvMJobErrorNotification` upon failure of an asynchronous read operation after performing all necessary internal management and error handling operations. Failure of an asynchronous read operation implies the read job is finished and has failed (i.e. block invalid or inconsistent).] (RS_BRF_01064)

[SWS_Ea_00144] [The Ea module shall call the function defined in the configuration parameter `EaNvMJobErrorNotification` upon failure of an asynchronous write operation after performing all necessary internal management and error handling operations. Failure of an asynchronous write operation implies the write job is finished and has failed and block has been marked as inconsistent.] (RS_BRF_01064)

[SWS_Ea_00145] [The Ea module shall call the function defined in the configuration parameter `EaNvMJobErrorNotification` upon failure of an asynchronous erase operation after performing all necessary internal management and error handling operations. Failure of an asynchronous erase operation implies the erase job for immediate data is finished and has failed (see SWS_Ea_00064).] (RS_BRF_01064)

[SWS_Ea_00146] [The Ea module shall call the function defined in the configuration parameter `EaNvMJobErrorNotification` upon failure of an asynchronous block invalidation operation after performing all necessary internal management and error handling operations. Failure of an asynchronous block invalidation operation implies the block invalidation job is finished and has failed.] (RS_BRF_01064)

9 Sequence diagrams

Note: For a vendor specific library the following sequence diagrams are valid only insofar as they show the relation to the calling modules (Ecu_StateManager resp. memory abstraction interface). The calling relations from a memory abstraction module to an underlying driver are not relevant / binding for a vendor specific library.

9.1 Ea_Init

The following figure shows the call sequence for the Ea_Init routine. It is different from that of all other services of this module as it is not called by the NVRAM manager and not called via the memory abstraction interface.

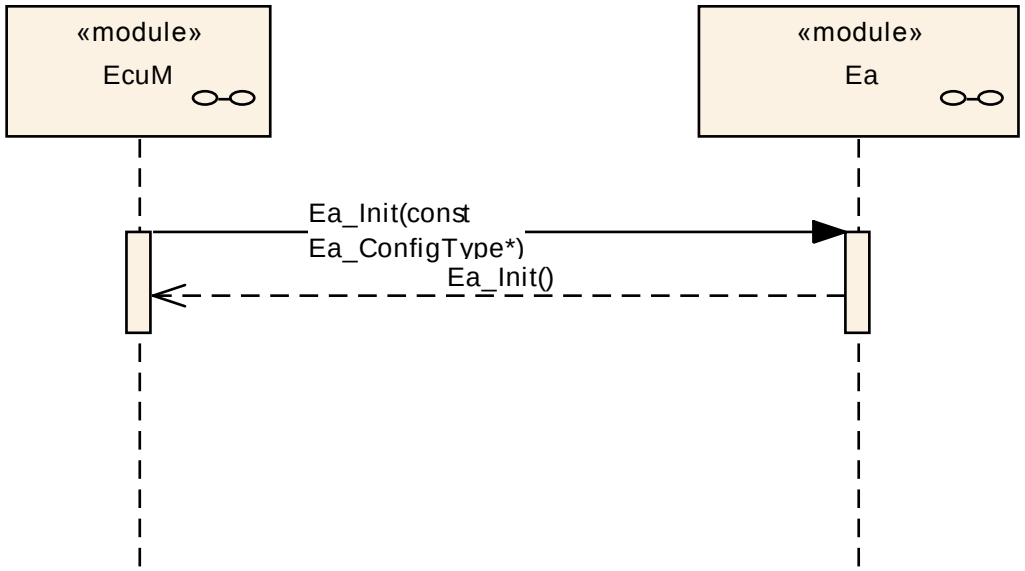


Figure 5: Sequence diagram of “Ea_Init” service

9.2 Ea_Write

The following figure shows as an example the call sequence for the Ea_Write service. This sequence diagram also applies to the other asynchronous services of this module.

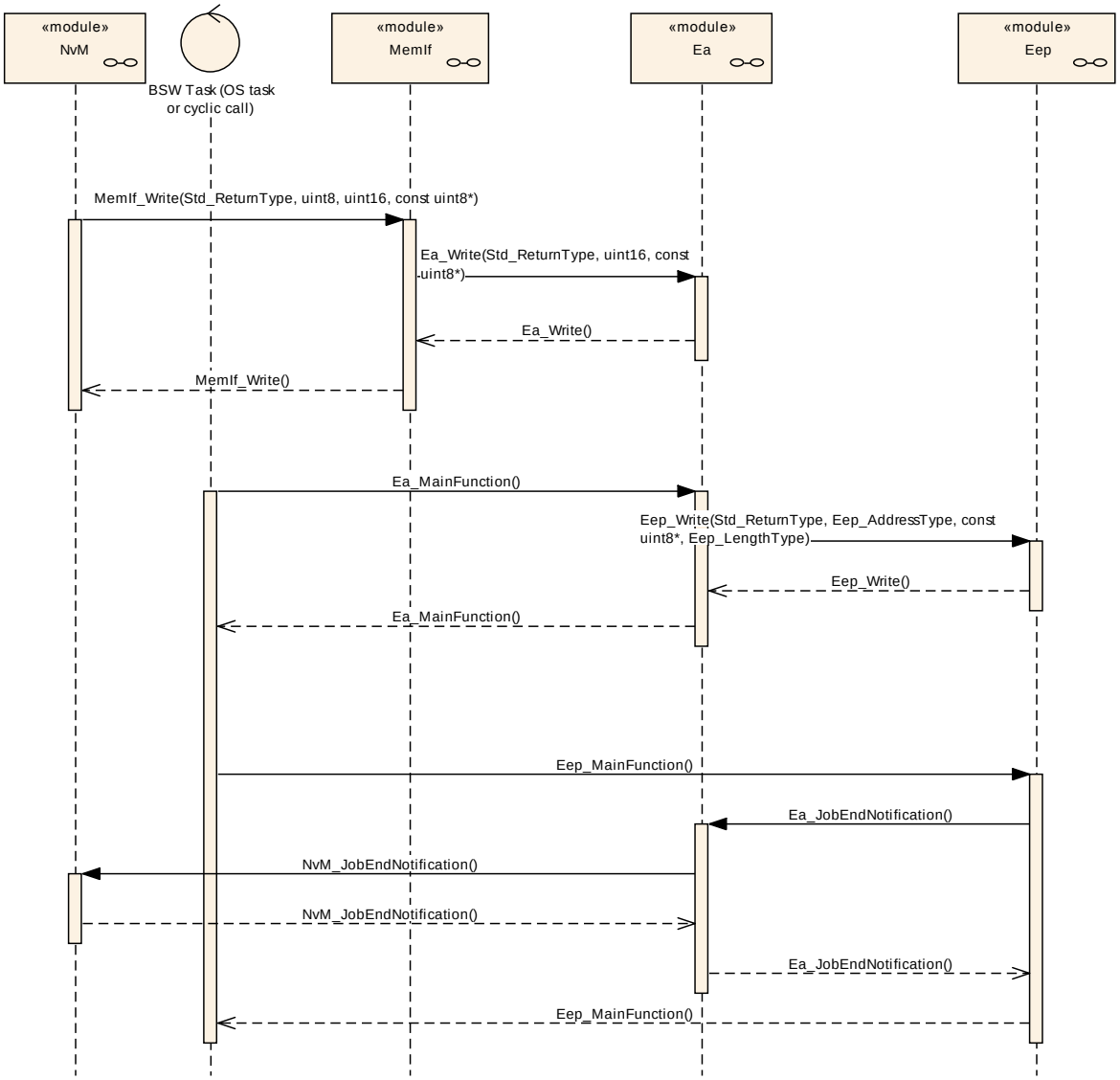


Figure 6: Sequence diagram "Ea_Write"

9.3 Ea_Cancel

The following figure shows as an example the call sequence for a canceled `Ea_Write` service. This sequence diagram shows that `Ea_Cancel` is asynchronous w.r.t. the underlying hardware while itself being synchronous.

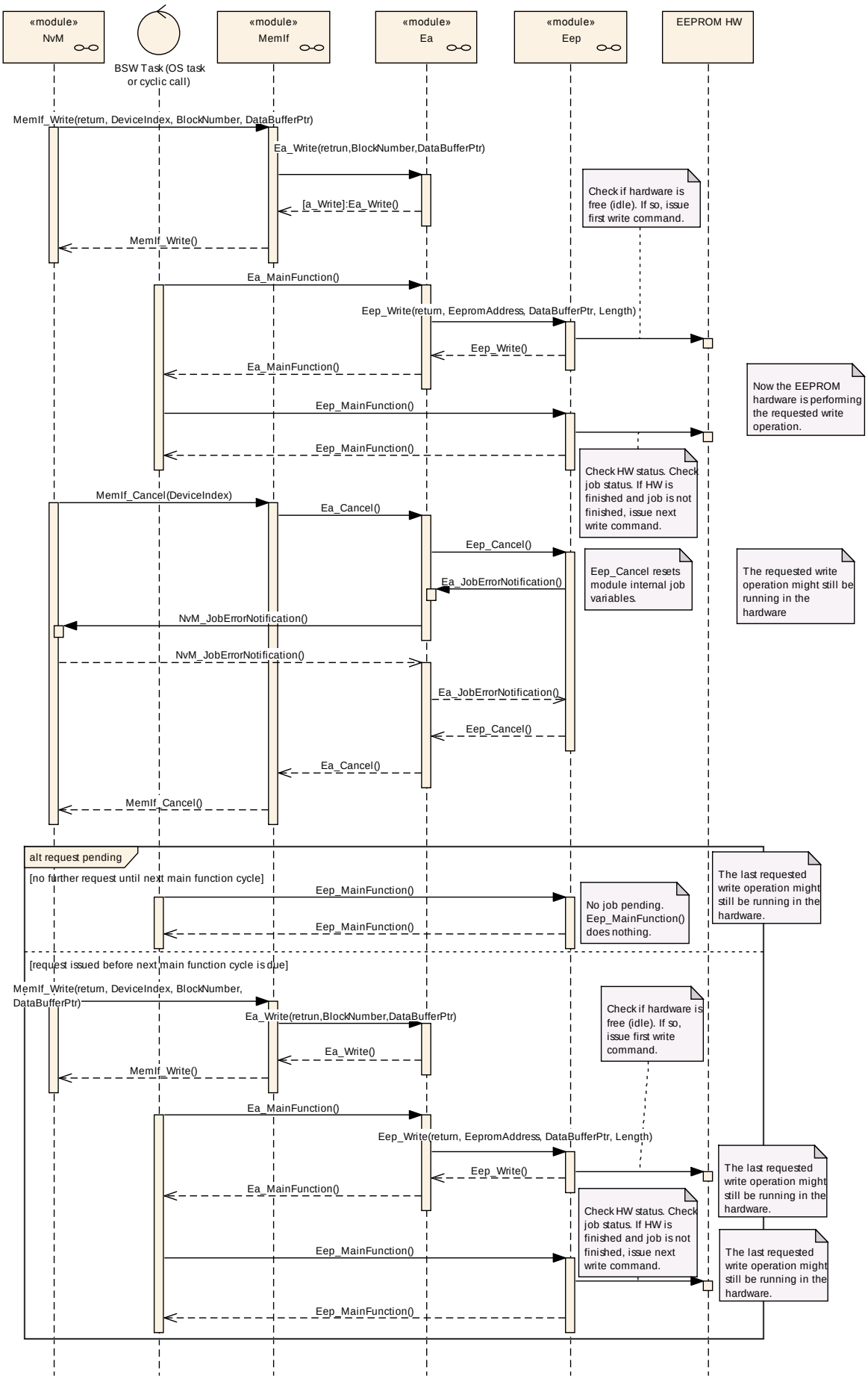

«module»
NvM
BSW Task (OS task
or cyclic call)
«module»
MemIf
«module»
Ea
«module»
Eep
EEPROM HW
MemIf_Write(return, DeviceIndex, BlockNumber, DataBufferPtr)
Ea_Write(retrun,BlockNumber,DataBufferPtr)
[a_Write]:Ea_Write()
MemIf_Write()
Check if hardware is free (idle). If so, issue first write command.
Ea_MainFunction()
Eep_Write(return, EepromAddress, DataBufferPtr, Length)
Eep_Write()
Ea_MainFunction()
Now the EEPROM hardware is performing the requested write operation.
Eep_MainFunction()
Eep_MainFunction()
Check HW status. Check job status. If HW is finished and job is not finished, issue next write command.
MemIf_Cancel(DeviceIndex)
Ea_Cancel()
Eep_Cancel()
Eep_Cancel resets module internal job variables.
Ea_JobErrorNotification()
The requested write operation might still be running in the hardware
NvM_JobErrorNotification()
NvM_JobErrorNotification()
Ea_JobErrorNotification()
Eep_Cancel()
Ea_Cancel()
MemIf_Cancel()
alt request pending
[no further request until next main function cycle]
Eep_MainFunction()
No job pending. Eep_MainFunction() does nothing.
Eep_MainFunction()
The last requested write operation might still be running in the hardware.
[request issued before next main function cycle is due]
MemIf_Write(return, DeviceIndex, BlockNumber, DataBufferPtr)
Ea_Write(retrun,BlockNumber,DataBufferPtr)
Ea_Write()
MemIf_Write()
Check if hardware is free (idle). If so, issue first write command.
Ea_MainFunction()
Eep_Write(return, EepromAddress, DataBufferPtr, Length)
Eep_Write()
Ea_MainFunction()
The last requested write operation might still be running in the hardware.
Check HW status. Check job status. If HW is finished and job is not finished, issue next write command.
Eep_MainFunction()
The last requested write operation might still be running in the hardware.
Eep_MainFunction()

Figure 7: Sequence diagram „Ea_Cancel“

10 Configuration specification

10.1 Containers and configuration parameters

The following chapters summarize all configuration parameters. The detailed meanings of the parameters are described in Chapters 7 and Chapter 8.

10.1.1 Ea

SWS Item	**ECUC_Ea_00133 :**
Module Name	*Ea*
Module Description	Configuration of the Ea (EEPROM Abstraction) module. The module shall abstract from the device specific addressing scheme and segmentation and provide the upper layers with a virtual addressing scheme and segmentation as well as a 'virtually' unlimited number of erase cycles.
Post-Build Variant Support	false
Supported Config Variants	VARIANT-PRE-COMPILE

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
EaBlockConfiguration	1..*	Configuration of block specific parameters for the EEPROM abstraction module.
EaGeneral	1	General configuration of the EEPROM abstraction module. This container lists block independent configuration parameters.
EaPublishedInformation	1	Additional published parameters not covered by CommonPublishedInformation container. Note that these parameters do not have any configuration class setting, since they are published information.

10.1.2 EaGeneral

SWS Item	**ECUC_Ea_00039 :**
Container Name	EaGeneral
Description	General configuration of the EEPROM abstraction module. This container lists block independent configuration parameters.
Configuration Parameters	

SWS Item	**ECUC_Ea_00120 :**		
Name	EaDevErrorDetect		
Description	Switches the development error detection and notification on or off. • true: detection and notification is enabled. • false: detection and notification is disabled.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Ea_00132 :**		
Name	EaMainFunctionPeriod		
Description	The period between successive calls to the main function in seconds.		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	]0 .. INF[		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

SWS Item	**ECUC_Ea_00121 :**		
Name	EaNvmJobEndNotification		
Description	Mapped to the job end notification routine provided by the upper layer module (NvM_JobEndNotification).		
Multiplicity	0..1		
Type	EcucFunctionNameDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	

Scope / Dependency	scope: local

SWS Item	**ECUC_Ea_00122 :**		
Name	EaNvmJobErrorNotification		
Description	Mapped to the job error notification routine provided by the upper layer module (NvM_JobErrorNotification).		
Multiplicity	0..1		
Type	EcucFunctionNameDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Ea_00123 :**		
Name	EaPollingMode		
Description	Pre-processor switch to enable and disable the polling mode for this module. true: Polling mode enabled, callback functions (provided to EEP module) disabled. false: Polling mode disabled, callback functions (provided to EEP module) enabled.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Ea_00001 :**		
Name	EaSetModeSupported		
Description	Compile switch to enable / disable the function Ea_SetMode.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	true		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Ea_00124 :**
Name	EaVersionInfoApi
Description	Pre-processor switch to enable / disable the API to read out the modules

	version information. true: Version info API enabled. false: Version info API disabled.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Ea_00125 :**		
Name	EaVirtualPageSize		
Description	The size in bytes to which logical blocks shall be aligned.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers

10.1.3 EaBlockConfiguration

SWS Item	**ECUC_Ea_00040 :**
Container Name	EaBlockConfiguration
Description	Configuration of block specific parameters for the EEPROM abstraction module.
Configuration Parameters	

SWS Item	**ECUC_Ea_00130 :**		
Name	EaBlockNumber		
Description	Block identifier (handle). 0x0000 and 0xFFFF shall not be used for block numbers (see EA006). Range: min = 2^NVM_DATASET_SELECTION_BITS max = 0xFFFF -2^NVM_DATASET_SELECTION_BITS Note: Depending on the number of bits set aside for dataset selection several other block numbers shall also be left out to ease implementation.		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	1 .. 65534		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

SWS Item	**ECUC_Ea_00128 :**

Name	EaBlockSize		
Description	Size of a logical block in bytes.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 65535		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

SWS Item	**ECUC_Ea_00131 :**		
Name	EaImmediateData		
Description	Marker for high priority data. true: Block contains immediate data. false: Block does not contain immediate data.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

SWS Item	**ECUC_Ea_00119 :**		
Name	EaNumberOfWriteCycles		
Description	Number of write cycles required for this block.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Ea_00129 :**		
Name	EaDeviceIndex		
Description	Reference to the device this block is stored in.		
Multiplicity	1		
Type	Symbolic name reference to [EepGeneral]		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local dependency: This information is needed by the NVRAM manager respectively the Memory Abstraction Interface to address a certain logical block. It is listed in this specification to give a complete overview over all block related configuration parameters.		

No Included Containers

10.2 Published Information

10.2.1 EaPublishedInformation

SWS Item	**ECUC_Ea_00043 :**
Container Name	EaPublishedInformation
Description	Additional published parameters not covered by CommonPublishedInformation container. Note that these parameters do not have any configuration class setting, since they are published information.
Configuration Parameters	

SWS Item	**ECUC_Ea_00126 :**		
Name	EaBlockOverhead		
Description	Management overhead per logical block in bytes. Note: If the management overhead depends on the block size or block location a formula has to be provided that allows the configurator to calculate the management overhead correctly.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Published Information***	X	All Variants
Scope / Dependency	scope: local		

SWS Item	**ECUC_Ea_00127 :**		
Name	EaPageOverhead		
Description	Management overhead per page in bytes. Note: If the management overhead depends on the block size or block location a formula has to be provided that allows the configurator to calculate the management overhead correctly.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Published Information***	X	All Variants
Scope / Dependency	scope: local		

No Included Containers

11 Not applicable requirements

[SWS_Ea_00999] ⌈ These requirements are not applicable to this specification. ⌋ (SRS_BSW_00416, SRS_BSW_00168, SRS_BSW_00423, SRS_BSW_00424, SRS_BSW_00425, SRS_BSW_00426, SRS_BSW_00427, SRS_BSW_00428, SRS_BSW_00429, SRS_BSW_00432, SRS_BSW_00433, SRS_BSW_00336, SRS_BSW_00339, SRS_BSW_00422, SRS_BSW_00417, SRS_BSW_00161, SRS_BSW_00162, SRS_BSW_00005, SRS_BSW_00415, SRS_BSW_00164, SRS_BSW_00342, SRS_BSW_00160, SRS_BSW_00007, SRS_BSW_00300, SRS_BSW_00347, SRS_BSW_00305, SRS_BSW_00307, SRS_BSW_00314, SRS_BSW_00348, SRS_BSW_00353, SRS_BSW_00361, SRS_BSW_00302, SRS_BSW_00328, SRS_BSW_00312, SRS_BSW_00006, SRS_BSW_00304, SRS_BSW_00378, SRS_BSW_00306, SRS_BSW_00308, SRS_BSW_00309, SRS_BSW_00371, SRS_BSW_00330, SRS_BSW_00009, SRS_BSW_00401, SRS_BSW_00172, SRS_BSW_00010, SRS_BSW_00333, SRS_BSW_00321, SRS_BSW_00341, SRS_BSW_00334, SRS_SPAL_12263, SRS_SPAL_12267, SRS_SPAL_12125, SRS_SPAL_12163, SRS_SPAL_12461, SRS_SPAL_12462, SRS_SPAL_12463, SRS_SPAL_12068, SRS_SPAL_12069, SRS_SPAL_00157, SRS_SPAL_12063, SRS_SPAL_12129, SRS_SPAL_12064, SRS_SPAL_12067, SRS_SPAL_12077, SRS_SPAL_12078, SRS_SPAL_12092, SRS_SPAL_12265, SRS_MemHwAb_14018)