Document Title	Specification of EEPROM Abstraction
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	287

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

Document Change History

Date	Release	Changed by	Description
2023-11-23	R23-11	AUTOSAR Release Management	• Editorial Changes
2022-11-24	R22-11	AUTOSAR Release Management	• Removal of obsolete items from R21-11 • Editorial changes
2021-11-25	R21-11	AUTOSAR Release Management	• Ea_SetMode() service is removed. • Ea_Cancel() service is now asynchronous. • Added support for buffer alignment for read and write operations. • Replaced Eep by MemAcc module as lower layer API interface to Ea.
2020-11-30	R20-11	AUTOSAR Release Management	• EA_E_INIT_FAILED is removed
2019-11-28	R19-11	AUTOSAR Release Management	• Configuration layouts added • Changed Document Status from Final to published
2018-10-31	4.4.0	AUTOSAR Release Management	• Editorial changes
2017-12-08	4.3.1	AUTOSAR Release Management	• Introduction of runtime errors • Set MEMIF_BUSY in Ea_InvalidateBlock and in Ea_EraseImmediateBlock

△

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
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
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

Contents

1 Introduction and functional overview

This specification describes the functionality, API and configuration of the EEPROM Abstraction Layer (see Figure 1.1).

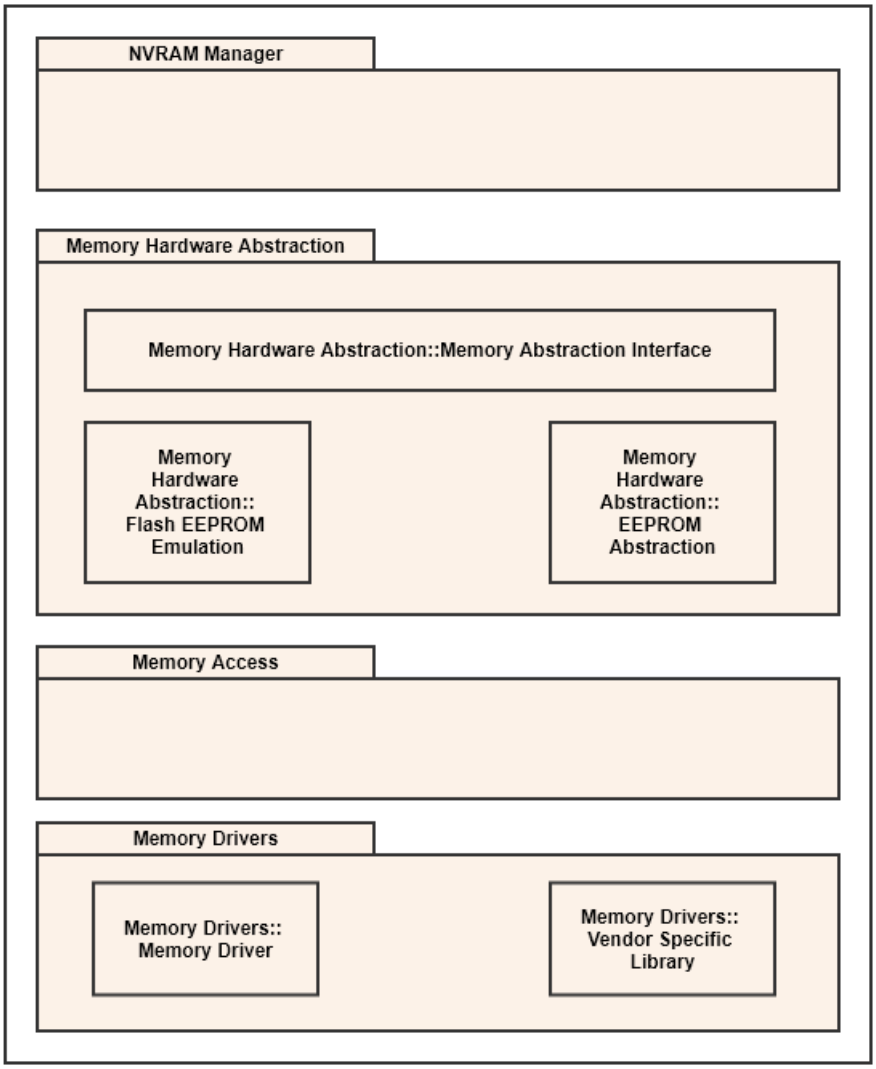

Figure 1.1: Module overview of memory hardware abstraction layer

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:
Address Area	Contiguous memory area in the logical address space Typically multiple physical memory sectors are combined to one logical address area.
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.
Mem	Memory Driver
MemAcc	Memory Access
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 7.1).
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.

Table 2.1: Acronyms and abbreviations used in the scope of this Document

3 Related documentation

3.1 Input documents & related standards and norms

[1] General Specification of Basic Software Modules
AUTOSAR_CP_SWS_BSWGeneral

[2] Requirements on Memory Hardware Abstraction Layer
AUTOSAR_CP_SRS_MemoryHWAbstractionLayer

3.2 Related specification

AUTOSAR provides a General Specification on Basic Software modules [1], which is also valid for EEEPROM Abstraction.

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

6 Requirements Tracing

The following tables reference the requirements specified in [1] and [2], and links to the fulfillment of these. Please note that if column “Satisfied by” is empty for a specific requirement this means that this requirement is not fulfilled by this document.

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_00054] [SWS_Ea_00055] [SWS_Ea_00094] [SWS_Ea_00141] [SWS_Ea_00142] [SWS_Ea_00143] [SWS_Ea_00144] [SWS_Ea_00145] [SWS_Ea_00146] [SWS_Ea_00153]
[RS_BRF_01812]	AUTOSAR non-volatile memory functionality shall support the prioritization and asynchronous execution of jobs	[SWS_Ea_00195]
[SRS_BSW_00101]	The Basic Software Module shall be able to initialize variables and hardware in a separate initialization function	[SWS_Ea_00017] [SWS_Ea_00084]
[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_00167] [SWS_Ea_00168] [SWS_Ea_00169] [SWS_Ea_00170] [SWS_Ea_00172] [SWS_Ea_00173] [SWS_Ea_00175] [SWS_Ea_00176]
[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_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_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_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_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]

△

Requirement	Description	Satisfied by
[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 layer modules 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 shall not be delayed by internal management operations nor by erasing the memory area to be written to	[SWS_Ea_00025]
[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_-14026]	The block numbers 0x0000 and 0x FFFF 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]

Table 6.1: RequirementsTracing

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 `EaVirtualPageSize`.⌋*(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 7.1). 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.

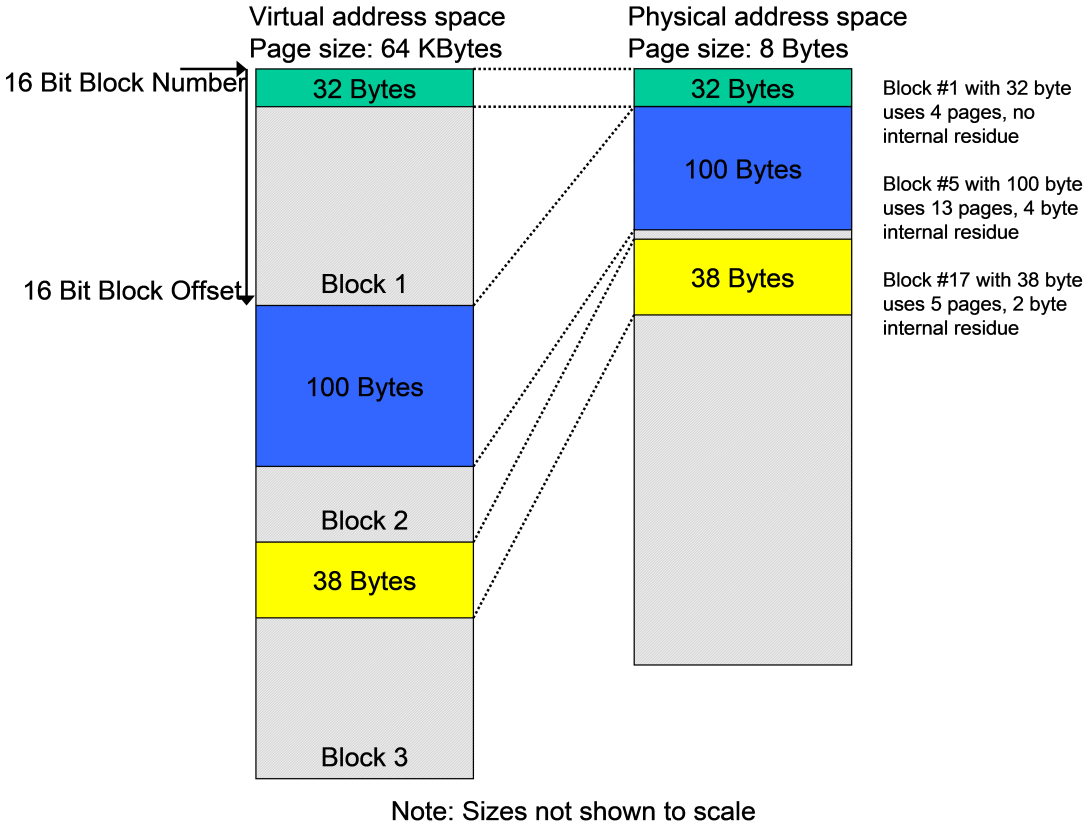

Figure 7.1: Virtual vs. physical memory layout

[SWS_Ea_00006] ⌈The block numbers 0x0000 and 0xFFFF shall not be configurable for a logical block `EaBlockNumber`.⌋ *(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 LSB's) to this start address (Figure 7.2).

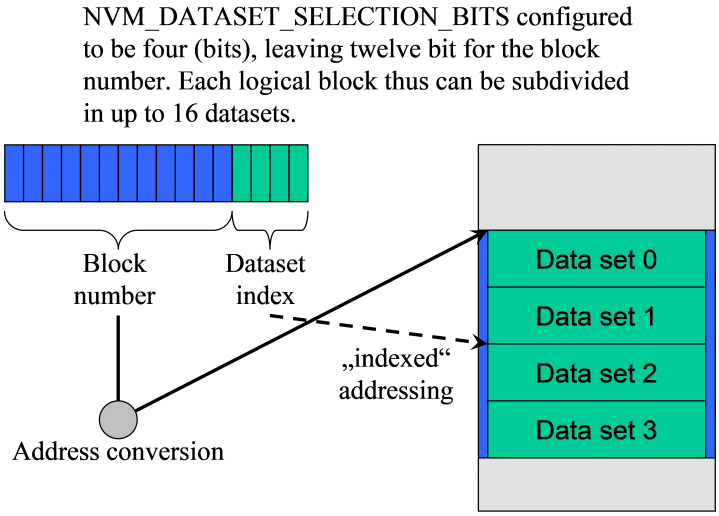

Figure 7.2: Block number and dataset index

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 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.

7.1.6 Buffer Alignment

[SWS_Ea_00197] ⌈The Ea shall align internal buffers to the EaBufferAlignmentValue Ref.⌋*()*

[SWS_Ea_00198] ⌈The Ea shall align read request to the EaMinimumReadPageSize.⌋*()*

7.2 Error Classification

7.2.1 Development Errors

[SWS_Ea_91001] Definiton of development errors in module Ea ⌈

Type of error	*Related error code*	*Error value*
API service called while module is not (yet) initialized	EA_E_UNINIT	0x01
API service called with invalid block number	EA_E_INVALID_BLOCK_NO	0x02
API service called with invalid block offset	EA_E_INVALID_BLOCK_OFS	0x03
API service called with invalid pointer argument	EA_E_PARAM_POINTER	0x04
API service called with invalid block length information	EA_E_INVALID_BLOCK_LEN	0x05

⌋*()*

7.2.2 Runtime Errors

[SWS_Ea_91002] Definiton of runtime errors in module Ea ⌈

Type of error	*Related error code*	*Error value*
API service called while module is busy	EA_E_BUSY	0x06
Ea_Cancel called while no job was pending	EA_E_INVALID_CANCEL	0x08

[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.

⌋()

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] Definition of imported datatypes of module Ea ⌈

Module	*Header File*	*Imported Type*
MemAcc	MemAcc_GeneralTypes.h	MemAcc_AddressAreaIdType
	MemAcc_GeneralTypes.h	MemAcc_AddressType
	MemAcc_GeneralTypes.h	MemAcc_DataType
	MemAcc_GeneralTypes.h	MemAcc_JobResultType
	MemAcc_GeneralTypes.h	MemAcc_LengthType
MemIf	MemIf.h	MemIf_JobResultType
	MemIf.h	MemIf_StatusType
Std	Std_Types.h	Std_ReturnType
	Std_Types.h	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] Definition of datatype Ea_ConfigType ⌈

Name	Ea_ConfigType	
Kind	Structure	
Elements	implementation specific	
	Type	–
	Comment	–
Description	Configuration data structure of the Ea module.	
Available via	Ea.h	

⌋(*SRS_BSW_00414*)

8.3 Function definitions

8.3.1 Ea_Init

[SWS_Ea_00084] Definition of API function Ea_Init ⌈

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.	
Available via	Ea.h	

⌋*(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_Read

[SWS_Ea_00086] Definition of API function Ea_Read ⌈

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.	
Available via	Ea.h	

⌋*(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] ⌈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 runtime 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 or SWS_Ea_00179 apply, the function Ea_Read shall not change the current module status or job result.⌋*(SRS_MemHwAb_14029, SRS_BSW_00323)*

8.3.3 Ea_Write

[SWS_Ea_00087] Definition of API function Ea_Write ⌈

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.	
Available via	Ea.h	

⌋*(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] ⌈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 runtime 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 de-

velopment 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 or SWS_Ea_00181 apply, the function Ea_Write shall not change the current module status or job result.⌋(*SRS_MemHwAb_14010, SRS_BSW_00323*)

8.3.4 Ea_Cancel

[SWS_Ea_00088] Definition of API function Ea_Cancel ⌈

Service Name	Ea_Cancel
Syntax	`void Ea_Cancel ( void )`
Service ID [hex]	0x04
Sync/Async	Asynchronous
Reentrancy	Non Reentrant
Parameters (in)	None
Parameters (inout)	None
Parameters (out)	None
Return value	None
Description	Cancels the ongoing asynchronous operation.
Available via	Ea.h

⌋(*SRS_MemHwAb_14031*)

[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 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 func-

tion Ea_Cancel), the function Ea_Cancel shall raise the runtime error EA_E_INVALID_CANCEL.⌋(*SRS_BSW_00323*)

8.3.5 Ea_GetStatus

[SWS_Ea_00089] Definition of API function Ea_GetStatus ⌈

Service Name	Ea_GetStatus	
Syntax	`MemIf_StatusType Ea_GetStatus ( void )`	
Service ID [hex]	0x05	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
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.	
Available via	Ea.h	

⌋(*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.

8.3.6 Ea_GetJobResult

[SWS_Ea_00090] Definition of API function Ea_GetJobResult ⌈

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.	
Available via	Ea.h	

⌋*(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.

8.3.7 Ea_InvalidateBlock

[SWS_Ea_00091] Definition of API function Ea_InvalidateBlock ⌈

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.	
Available via	Ea.h	

⌋(*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] ⌈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 runtime 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, set the Ea module status to MEMIF_BUSY, set the job result to MEMIF_JOB_PENDING and return E_OK to the caller.⌋(*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_Invalidate Block, i.e. requirements SWS_Ea_00135, SWS_Ea_00149 or SWS_Ea_00175 apply, the function Ea_InvalidateBlock shall not change the current module status or job result.⌋*(SRS_BSW_00323)*

8.3.8 Ea_GetVersionInfo

[SWS_Ea_00092] Definition of API function Ea_GetVersionInfo ⌈

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.	
Available via	Ea.h	

⌋*(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)*

8.3.9 Ea_EraseImmediateBlock

[SWS_Ea_00093] Definition of API function Ea_EraseImmediateBlock ⌈

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.	
Available via	Ea.h	

⌋*(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] ⌈The function Ea_EraseImmediateBlock shall check if the module state is MEMIF_BUSY. If this is the case, the function shall reject the erase request, raise the runtime 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 or SWS_Ea_00065 apply, the function Ea_EraseImmediateBlock shall not change the current module status or job result.⌋ *(SRS_BSW_00323)*

8.4 Callback notifications

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

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] Definition of callback function Ea_JobEndNotification ⌈

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.
Available via	Ea.h

⌋ *(RS_BRF_01064)* The underlying EEPROM driver shall call the function Ea_JobEnd Notification 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)*

[SWS_Ea_00200] ⌈The function Ea_JobEndNotification shall perform any necessary block management and error handling operations and shall call the corresponding callback routine of the upper layer module (Ea_NvMJobErrorNotification).⌋*()*

Note: The function Ea_JobEndNotification 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] Definition of scheduled function Ea_MainFunction ⌈

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.
Available via	SchM_Ea.h

⌋*(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] Definition of mandatory interfaces in module Ea ⌈

API Function	*Header File*	*Description*
Det_ReportRuntimeError	Det.h	Service to report runtime errors. If a callout has been configured then this callout shall be called.
Eep_GetStatus	Eep.h	Returns the EEPROM status.
MemAcc_Cancel	MemAcc.h	Triggers a cancel operation of the pending job for the address area referenced by the addressAreaId. Cancelling affects only jobs in pending state. For any other states, the request will be ignored.
MemAcc_Erase	MemAcc.h	Triggers an erase job of the given area. Triggers an erase job of the given area defined by targetAddress and length. The result of this service can be retrieved using the Mem_GetJobResult API. If the erase operation was successful, the result of the job is MEM_JOB_OK. If the erase operation failed, e.g. due to a hardware issue, the result of the job is MEM_JOB_FAILED.

▽

△

API Function	Header File	Description
MemAcc_GetJobResult	MemAcc.h	Returns the consolidated job result of the address area referenced by addressAreaId.
MemAcc_Read	MemAcc.h	Triggers a read job to copy data from the source address into the referenced destination data buffer. The result of this service can be retrieved using the MemAcc_GetJobResult API. If the read operation was successful, the result of the job is MEMACC_OK. If the read operation failed, the result of the job is either MEMACC_FAILED in case of a general error or MEMACC_ECC_CORRECTED/MEMACC_ECC_UNCORRECTED in case of a correctable/uncorrectable ECC error.
MemAcc_Write	MemAcc.h	Triggers a write job to store the passed data to the provided address area with given address and length. The result of this service can be retrieved using the MemAcc_GetJobResult API. If the write operation was successful, the job result is MEMACC_OK. If there was an issue writing the data, the result is MEMACC_FAILED.

⌋(*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] Definition of optional interfaces in module Ea ⌈

API Function	Header File	Description
Det_ReportError	Det.h	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] Definition of configurable interface NvM_JobEndNotification ⌈

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.
Available via	NvM_MemIf.h

⌋*(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] Definition of configurable interface NvM_JobErrorNotification ⌈

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.
Available via	NvM_MemIf.h

⌋*(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.

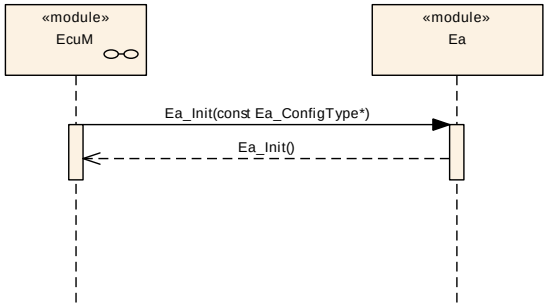

Figure 9.1: Sequence diagram of "Ea_Init" service

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.

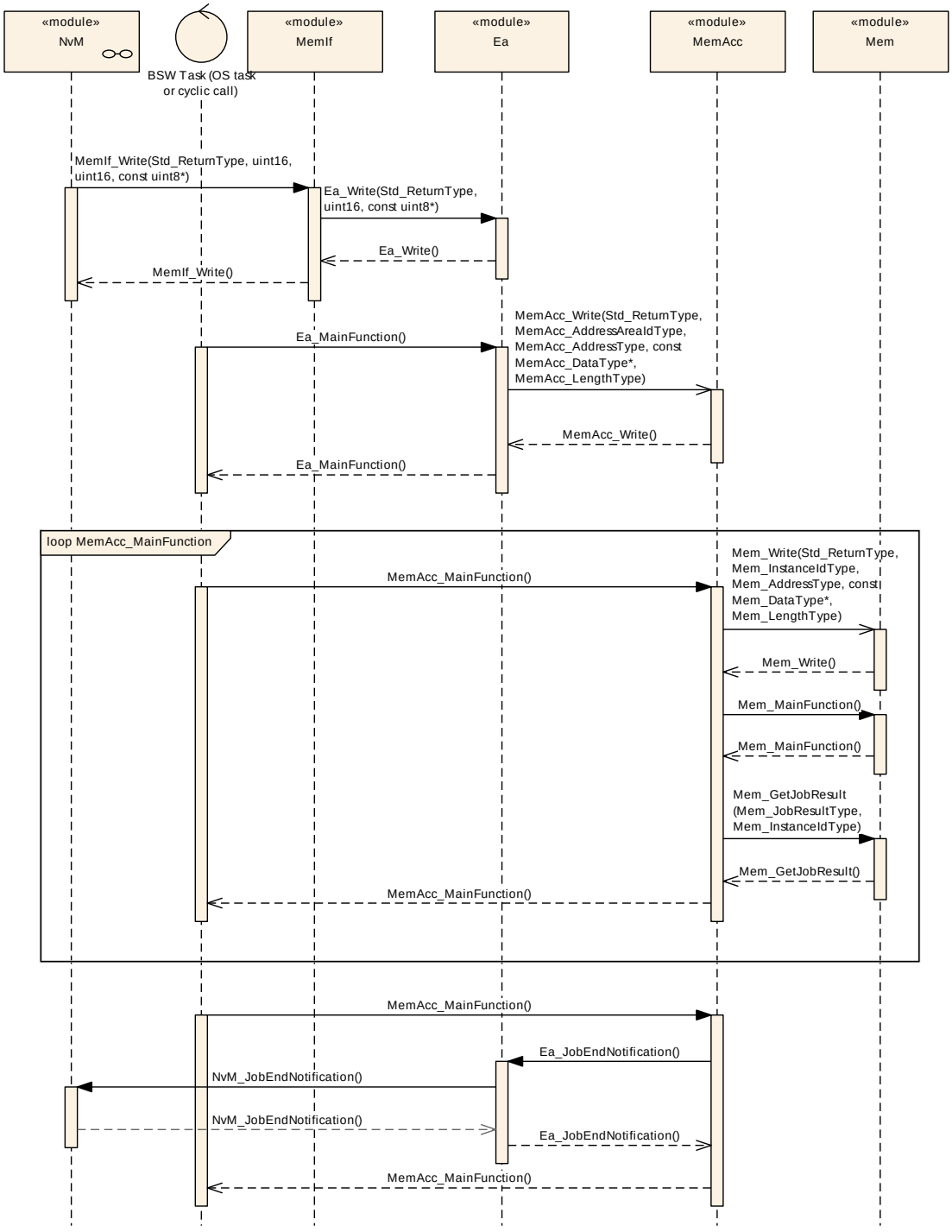

Figure 9.2: Sequence diagram of "Ea_Write" service

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.

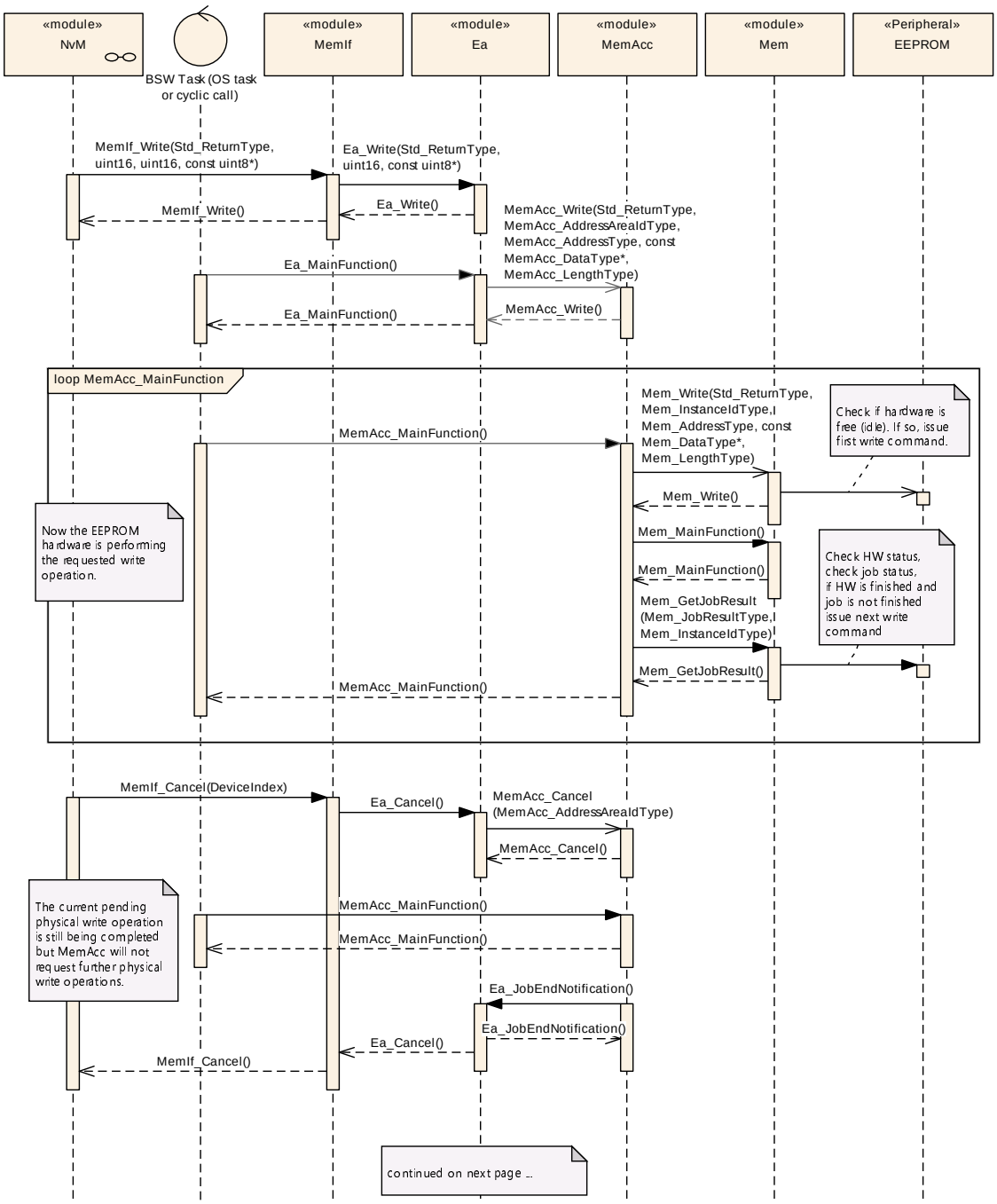

Figure 9.3: Sequence diagram of "`Ea_Cancel`" service (Part 1)

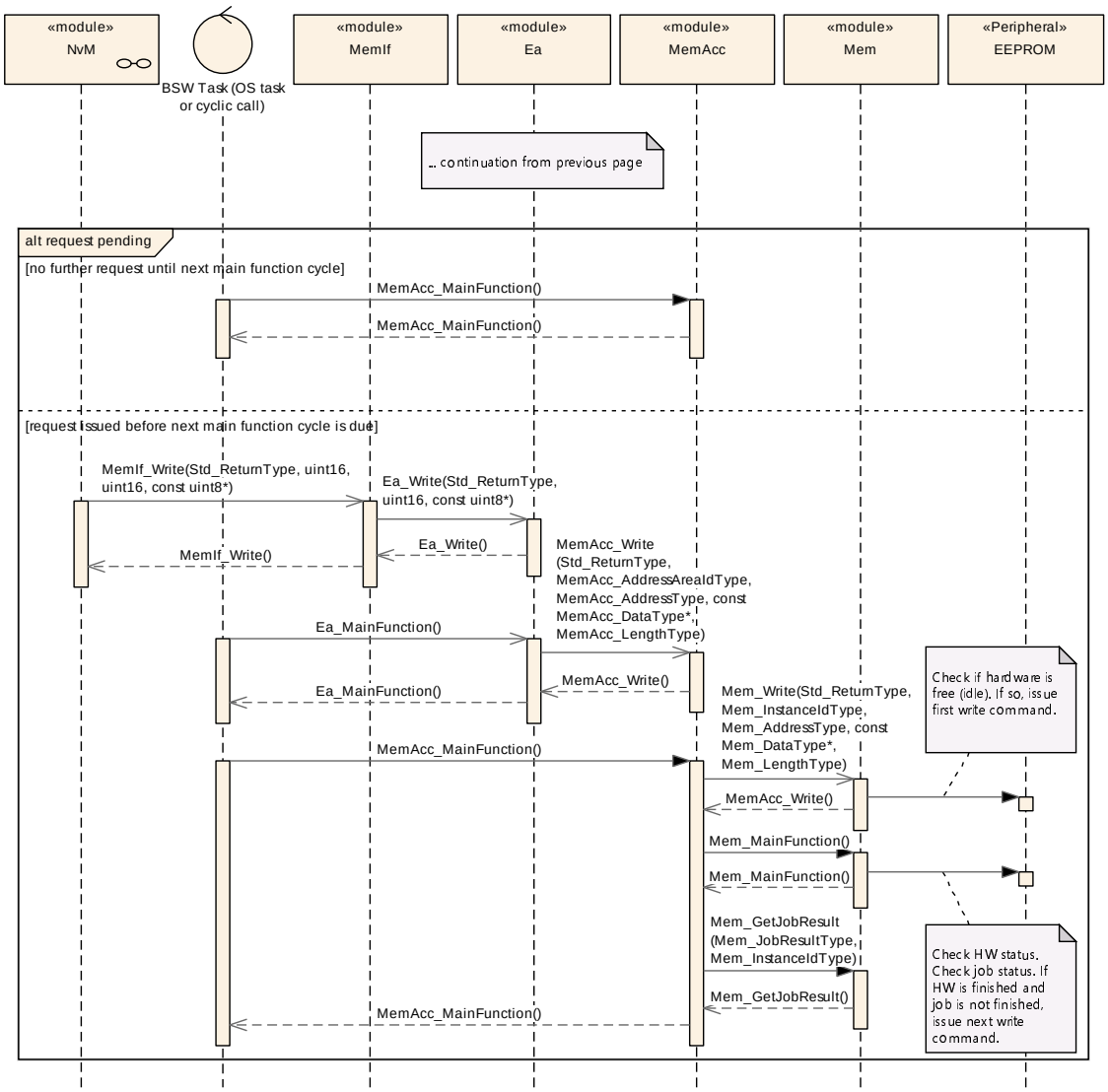

Figure 9.4: Sequence diagram of "`Ea_Cancel`" service (Part 2)

10 Configuration specification

10.1 Containers and configuration parameters

The following chapters summarize all configuration parameters.

10.1.1 Ea

SWS Item	[ECUC_Ea_00133]
Module Name	Ea
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 Common PublishedInformation container. Note that these parameters do not have any configuration class setting, since they are published information.

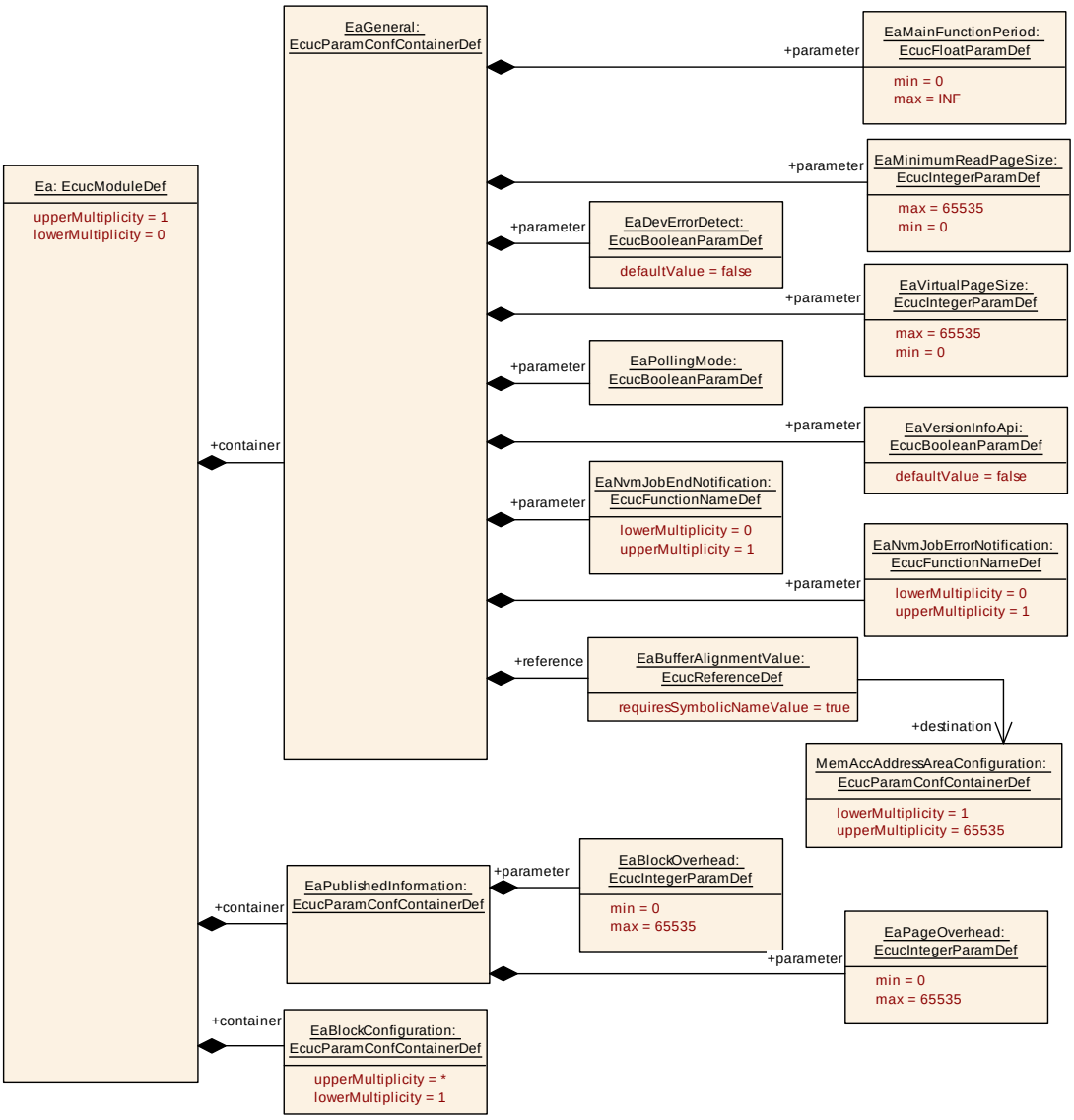

Figure 10.1: Ea Configuration Layout

10.1.2 EaGeneral

SWS Item	[ECUC_Ea_00039]
Container Name	EaGeneral
Parent Container	Ea
Description	General configuration of the EEPROM abstraction module. This container lists block independent configuration parameters.
Configuration Parameters	

SWS Item	[ECUC_Ea_00120]		
Parameter Name	EaDevErrorDetect		
Parent Container	EaGeneral		
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]		
Parameter Name	EaMainFunctionPeriod		
Parent Container	EaGeneral		
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_00135]		
Parameter Name	EaMinimumReadPageSize		
Parent Container	EaGeneral		
Description	Minimum Page size will be a multiple of the minimum page size. Ea shall align read requests to this size. **Tags:** atp.Status=draft		
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		

SWS Item	[ECUC_Ea_00121]		
Parameter Name	EaNvmJobEndNotification		
Parent Container	EaGeneral		
Description	Mapped to the job end notification routine provided by the upper layer module (NvM_JobEndNotification).		
Multiplicity	0..1		
Type	EcucFunctionNameDef		
Default value	–		
Regular Expression	–		
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]		
Parameter Name	EaNvmJobErrorNotification		
Parent Container	EaGeneral		
Description	Mapped to the job error notification routine provided by the upper layer module (NvM_JobErrorNotification).		
Multiplicity	0..1		
Type	EcucFunctionNameDef		
Default value	–		
Regular Expression	–		
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]
Parameter Name	EaPollingMode
Parent Container	EaGeneral
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_00124]**		
Parameter Name	EaVersionInfoApi		
Parent Container	EaGeneral		
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]**		
Parameter Name	EaVirtualPageSize		
Parent Container	EaGeneral		
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		

SWS Item	**[ECUC_Ea_00136]**		
Parameter Name	EaBufferAlignmentValue		
Parent Container	EaGeneral		
Description	Parameter determines the alignment of the start address that Ea buffers need to have. Value shall be inherited from MemAccBufferAlignmentValue. **Tags:** atp.Status=draft		
Multiplicity	1		
Type	Symbolic name reference to MemAccAddressAreaConfiguration		
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
Parent Container	Ea
Description	Configuration of block specific parameters for the EEPROM abstraction module.
Configuration Parameters	

SWS Item	**[ECUC_Ea_00130]**		
Parameter Name	EaBlockNumber		
Parent Container	EaBlockConfiguration		
Description	Block identifier (handle). 0x0000 and 0xFFFF shall not be used for block numbers (see SWS_Ea_00006). 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]**		
Parameter Name	EaBlockSize		
Parent Container	EaBlockConfiguration		
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]		
Parameter Name	EaImmediateData		
Parent Container	EaBlockConfiguration		
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]		
Parameter Name	EaNumberOfWriteCycles		
Parent Container	EaBlockConfiguration		
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]		
Parameter Name	EaDeviceIndex		
Parent Container	EaBlockConfiguration		
Description	Reference to the device this block is stored in. This reference is mutually exclusive to EaMemAccAddressArea.		
Multiplicity	0..1		
Type	Symbolic name reference to EepGeneral		
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 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. This reference is mutually exclusive to EaMemAccAddressArea.

SWS Item	**[ECUC_Ea_00134]**		
Parameter Name	EaMemAccAddressArea		
Parent Container	EaBlockConfiguration		
Description	Reference to the MemAccAddressAreaConfiguration. This reference is mutually exclusive to EaDeviceIndex. **Tags:** atp.Status=draft		
Multiplicity	0..1		
Type	Symbolic name reference to MemAccAddressAreaConfiguration		
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 dependency: This reference is mutually exclusive to EaDeviceIndex.		

No Included Containers

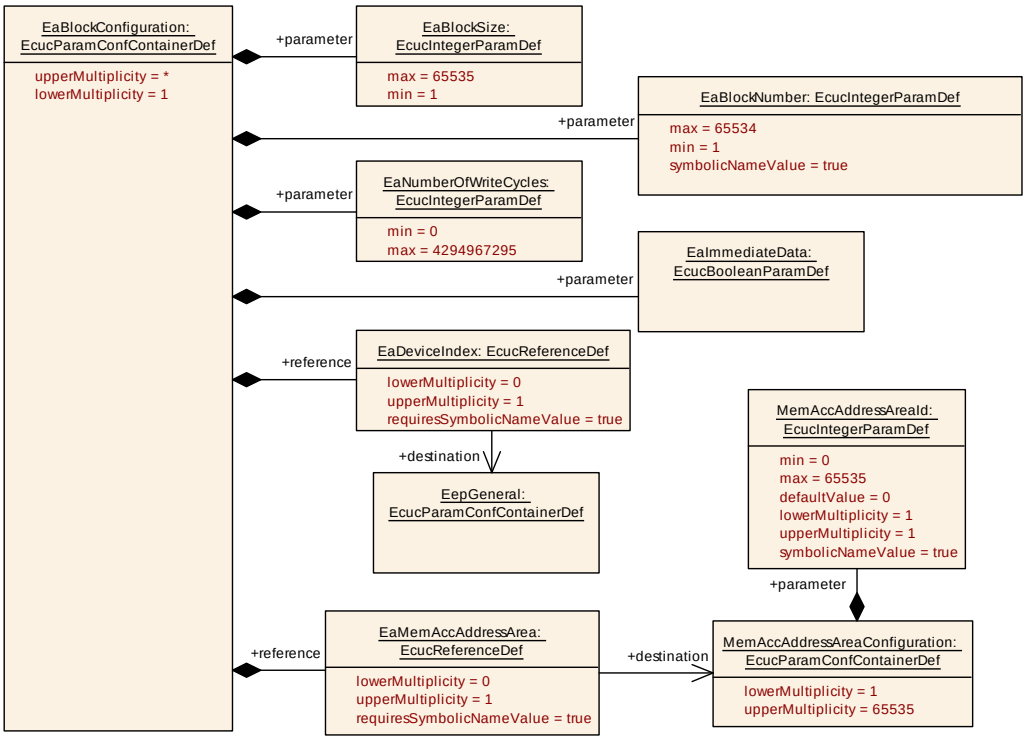

Figure 10.2: Ea Block Configuration Layout

10.2 Published Information

10.2.1 EaPublishedInformation

SWS Item	[ECUC_Ea_00043]
Container Name	EaPublishedInformation
Parent Container	Ea
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]		
Parameter Name	EaBlockOverhead		
Parent Container	EaPublishedInformation		
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]		
Parameter Name	EaPageOverhead		
Parent Container	EaPublishedInformation		
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

A Not applicable requirements

[SWS_Ea_NA_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_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_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)*