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

Document Version	2.0.0
Document Status	Final
Part of Release	4.0
Revision	3

Document Change History

Date	Version	Changed by	Change Description
03.11.2011	2.0.0	AUTOSAR Administration	• Introduced parameter checks and corresponding DET errors • Handling of internal management operations detailed • Module short name changed
13.10.2010	1.4.0	AUTOSAR Administration	• Check fpr NULL pointer added • Inter module checks detailed • Description of return values clarified
03.12.2009	1.3.0	AUTOSAR Administration	• Configuration variants clarified • Multiplicity of notification routines corected • Job result handling re-formulated • File include structure changed • Legal disclaimer revised
23.06.2008	1.2.1	AUTOSAR Administration	• Legal disclaimer revised
10.12.2007	1.2.0	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
14.02.2007	1.1.0	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

Document Change History

Date	Version	Changed by	Change Description
23.03.2006	1.0.0	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 Specification Documents may contain exemplary items (exemplary reference models, "use cases", and/or references to exemplary technical solutions, devices, processes or software).

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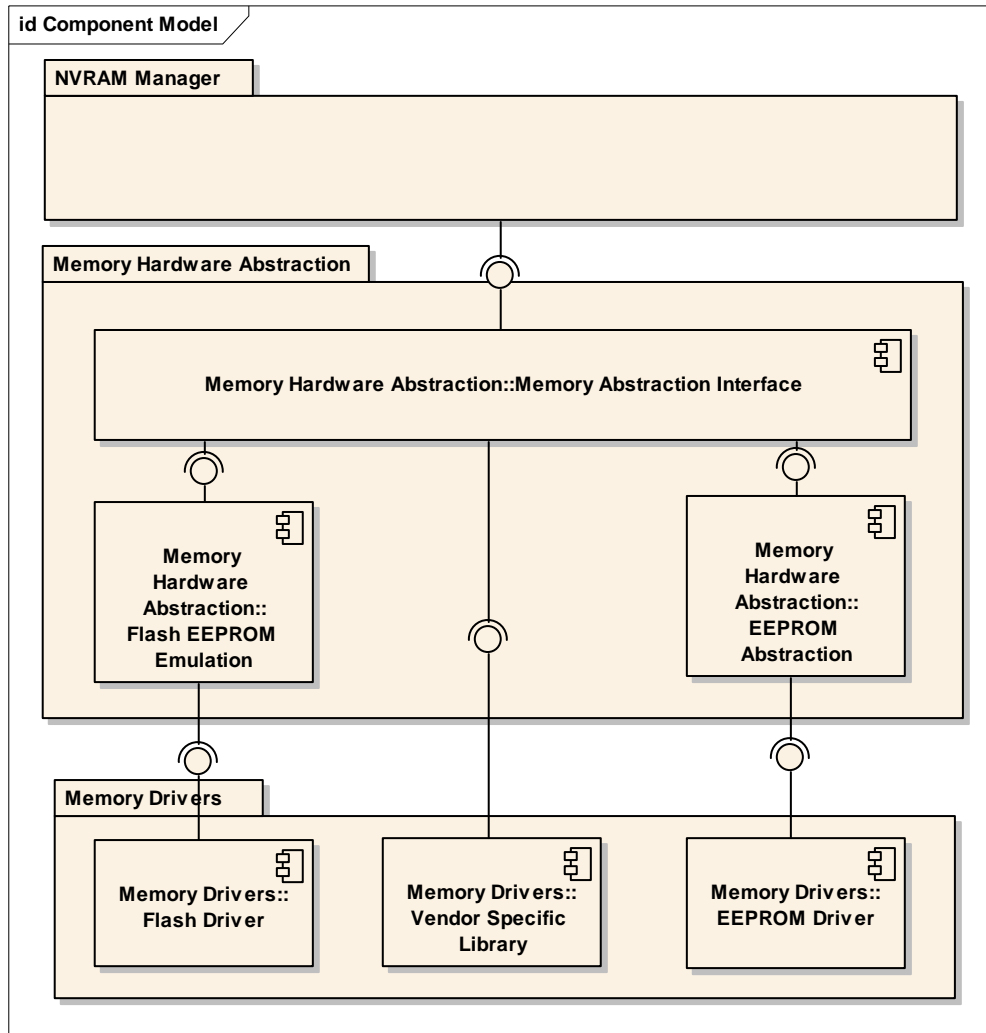
Table of Content

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.

Document ID 287: AUTOSAR_SWS_EEPROMAbstraction

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 Development Error Tracer
AUTOSAR_SWS_DevelopmentErrorTracer.pdf

[7] Specification of ECU Configuration,
AUTOSAR_TPS_ECUConfiguration.pdf

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

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

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 Code file structure

[EA057] ⌈ The code file structure shall not be defined within this specification. ⌋()

5.1.2 Header file structure

[EA113] ⌈ The Ea module shall comply with the following file include structure:
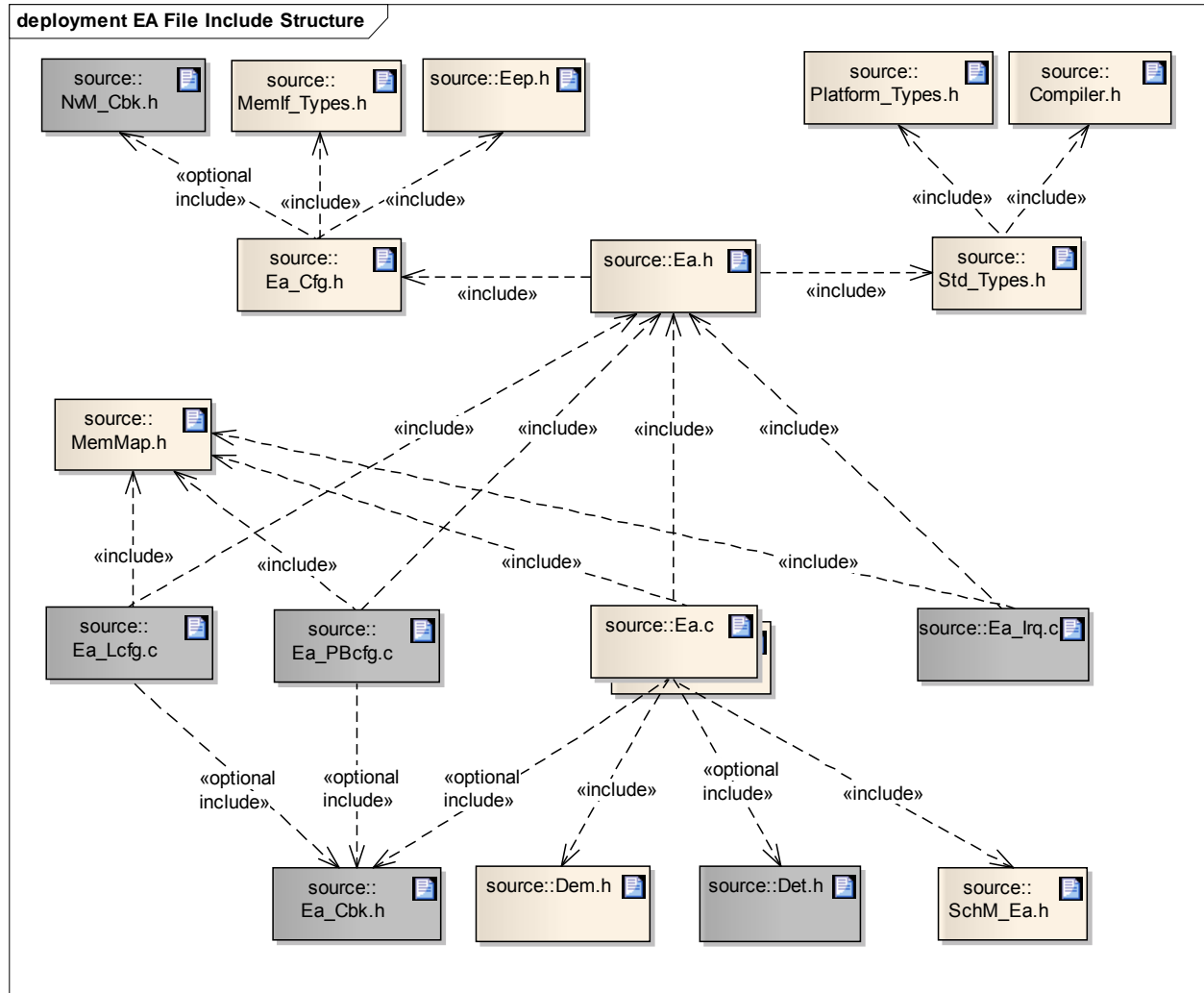
⌋(BSW00346, BSW158, BSW00370, BSW00301)

Figure 2: EEPROM Abstraction Layer File Include Structure

Note: Files which are optional (depending on implementation / configuration) are shown in grey.

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

[EA058] ⌈ The EA module shall include the Dem.h file. By this inclusion the APIs to report errors as well as the required Event Id symbols are included. This specification defines the name of the Event Id symbols which are provided by XML to the DEM configuration tool. The DEM configuration tool assigns ECU dependent values to the Event Id symbols and publishes the symbols in Dem_IntErrId.h. ⌋()

6 Requirements traceability

Requirement	Satisfied by
-	EA068
-	EA095
-	EA167
-	EA150
-	EA037
-	EA161
-	EA025
-	EA074
-	EA078
-	EA174
-	EA137
-	EA162
-	EA061
-	EA094
-	EA086
-	EA158
-	EA066
-	EA056
-	EA082
-	EA091
-	EA075
-	EA173
-	EA157
-	EA049
-	EA135
-	EA090
-	EA088
-	EA171
-	EA034
-	EA154
-	EA020
-	EA054
-	EA160
-	EA077
-	EA178
-	EA156
-	EA092
-	EA168

-	EA172
-	EA175
-	EA022
-	EA073
-	EA141
-	EA051
-	EA072
-	EA097
-	EA159
-	EA151
-	EA146
-	EA079
-	EA081
-	EA058
-	EA144
-	EA053
-	EA164
-	EA153
-	EA169
-	EA155
-	EA166
-	EA062
-	EA089
-	EA057
-	EA083
-	EA084
-	EA055
-	EA098
-	EA087
-	EA145
-	EA114
-	EA060
-	EA005
-	EA165
-	EA026
-	EA117
-	EA142
-	EA143
-	EA176
-	EA170
BSW00300	EA999
BSW00301	EA113

BSW00302	EA999
BSW00304	EA999
BSW00305	EA999
BSW00306	EA999
BSW00307	EA999
BSW00308	EA999
BSW00309	EA999
BSW00312	EA999
BSW00314	EA999
BSW00321	EA999
BSW00323	EA148, EA147, EA149, EA065, EA152
BSW00324	EA999
BSW00326	EA999
BSW00328	EA999
BSW00330	EA999
BSW00331	EA045
BSW00333	EA999
BSW00334	EA999
BSW00336	EA999
BSW00338	EA045, EA011
BSW00339	EA999
BSW00341	EA999
BSW00342	EA999
BSW00346	EA113
BSW00347	EA999
BSW00348	EA999
BSW00350	EA059, EA011
BSW00353	EA999
BSW00355	EA999
BSW00361	EA999
BSW00369	EA045
BSW00370	EA113
BSW00371	EA999
BSW00373	EA096
BSW00378	EA999
BSW00385	EA100, EA099
BSW00386	EA059, EA045, EA011
BSW004	EA013
BSW00401	EA999
BSW00406	EA128, EA129, EA131, EA130, EA134, EA132, EA136, EA035
BSW00409	EA048
BSW00415	EA999

BSW00416	EA999
BSW00417	EA999
BSW00420	EA999
BSW00421	EA999
BSW00422	EA999
BSW00423	EA999
BSW00424	EA999
BSW00425	EA999
BSW00426	EA999
BSW00427	EA999
BSW00428	EA999
BSW00429	EA999
BSW00431	EA999
BSW00432	EA999
BSW00433	EA999
BSW00434	EA999
BSW005	EA999
BSW006	EA999
BSW007	EA999
BSW009	EA999
BSW010	EA999
BSW12057	EA017
BSW12058	EA999
BSW12059	EA999
BSW12060	EA999
BSW12062	EA999
BSW12063	EA999
BSW12064	EA999
BSW12067	EA999
BSW12068	EA999
BSW12069	EA999
BSW12077	EA999
BSW12078	EA999
BSW12081	EA999
BSW12092	EA999
BSW12125	EA999
BSW12129	EA999
BSW12155	EA999
BSW12163	EA999
BSW12169	EA085
BSW12263	EA999
BSW12265	EA999

BSW12267	EA999
BSW12461	EA999
BSW12462	EA999
BSW12463	EA999
BSW14002	EA080
BSW14006	EA024
BSW14007	EA021
BSW14009	EA063, EA036, EA024, EA021, EA007
BSW14014	EA047, EA046
BSW14015	EA104
BSW14016	EA104
BSW14018	EA999
BSW14026	EA006
BSW14032	EA104, EA065, EA064, EA063, EA093
BSW157	EA999
BSW158	EA113
BSW160	EA999
BSW161	EA999
BSW162	EA999
BSW164	EA999
BSW168	EA999
BSW172	EA999

Document: General Requirements on Basic Software Modules

Requirement	Satisfied by
[BSW00344] Reference to link-time configuration	Not applicable (this module does not provide any link-time parameters)
[BSW00404] Reference to post build time configuration	Not applicable (this module does not provide post build time configuration)
[BSW00405] Reference to multiple configuration sets	Not applicable (this module does not support multiple configuration sets)
[BSW00345] Pre-compile-time configuration	EA039, EA040_Conf
[BSW159] Tool-based configuration	EA039, EA040_Conf
[BSW167] Static configuration checking	EA013, EA038
[BSW171] Configurability of optional functionality	EA150
[BSW170] Data for reconfiguration of AUTOSAR SW-Components	Not applicable (no reconfiguration supported)
[BSW00380] Separate C-File for configuration parameters	Not applicable (no link-time or post build time configuration parameters)
[BSW00381] Separate configuration header file for pre-compile time parameters	EA113
[BSW00412] Separate H-File for configuration	Not applicable

parameters	(no link-time or post build time configuration parameters)
[BSW00383] List dependencies of configuration files	EA113
[BSW00384] List dependencies to other modules	Chapter 5
[BSW00387] Specify the configuration class of callback function	Chapter 8.6
[BSW00388] Introduce containers	Chapter 10.2
[BSW00389] Containers shall have names	Chapter 10.2
[BSW00390] Parameter content shall be unique within the module	Chapter 8, Chapter 10.2.3, Chapter 10.2.4
[BSW00391] Parameter shall have unique names	Chapter 8, Chapter 10.2.3, Chapter 10.2.4
[BSW00392] Parameters shall have a type	Chapter 8, Chapter 10.2.3, Chapter 10.2.4
[BSW00393] Parameters shall have a range	Chapter 8, Chapter 10.2.3, Chapter 10.2.4
[BSW00394] Specify the scope of the parameters	Chapter 8, Chapter 10.2.3, Chapter 10.2.4
[BSW00395] List the required parameters (per parameter)	Chapter 8, Chapter 10.2.3, Chapter 10.2.4
[BSW00396] Configuration classes	Chapter 8, Chapter 10.2.3, Chapter 10.2.4
[BSW00397] Pre-compile-time parameters	Chapter 8, Chapter 10.2.3, Chapter 10.2.4
[BSW00398] Link-time parameters	Not applicable (no link-time configuration parameters)
[BSW00399] Loadable Post-build time parameters	Not applicable (no post build time configuration parameters)
[BSW00400] Selectable Post-build time parameters	Not applicable (no post build time configuration parameters)
[BSW00402] Published information	Chapter 10.3
[BSW00375] Notification of wake-up reason	Not applicable (this module does not provide wakeup capabilities)
[BSW101] Initialization interface	EA017
[BSW00416] Sequence of Initialization	Not applicable (requirement on system design, not a single module)
[BSW00406] Check module initialization	EA139, EA128, EA129, EA130, EA131, EA132, EA133, EA134, EA135, EA136
[BSW168] Diagnostic Interface of SW components	Not applicable (this module does not provide special diagnostics support)
[BSW00407] Function to read out published parameters	Chapter 8.3.9, EA043_Conf
[BSW00423] Usage of SW-C template to describe BSW modules with AUTOSAR Interfaces	Not applicable (this module does not provide an AUTOSAR interface)
[BSW00424] BSW main processing function task allocation	Not applicable (requirement on system design, not on a single module)
[BSW00425] Trigger conditions for schedulable objects	Not applicable (requirement on the BSW module description template)
[BSW00426] Exclusive areas in BSW modules	Not applicable (no exclusive areas defined in this module)
[BSW00427] ISR description for BSW modules	Not applicable (this module does not directly implement any ISRs)
[BSW00428] Execution order dependencies of main processing functions	Not applicable (only one main processing function in this module)
[BSW00429] Restricted BSW OS functionality access	Not applicable (this module does not use any OS functionality)
[BSW00431] The BSW Scheduler module	Not applicable

implements task bodies	(requirement on the BSW scheduler)
[BSW00432] Modules should have separate main processing functions for read/receive and write/transmit data path	Not applicable (only one main processing function in this module)
[BSW00433] Calling of main processing functions	Not applicable (requirement on system design, not on a single module)
[BSW00434] The Schedule Module shall provide an API for exclusive areas	Not applicable (requirement on the –BSW scheduler, not this module)
[BSW00336] Shutdown interface	Not applicable (this module does not provide shutdown capabilities)
[BSW00337] Classification of errors	EA139, EA140
[BSW00338] Detection and Reporting of development errors	EA011, EA045
[BSW00369] Do not return development error codes via API	EA045
[BSW00339] Reporting of production relevant error status	Not applicable (no production relevant errors)
[BSW00421] Reporting of production relevant error events	Not applicable (no production relevant errors)
[BSW00422] Debouncing of production relevant error status	Not applicable (requirement on the DEM, not this module)
[BSW00420] Production relevant error event rate detection	Not applicable (requirement on the DEM, not this module)
[BSW00417] Reporting of Error Events by Non-Basic Software	Not applicable (requirement on non BSW modules)
[BSW00323] API parameter checking	EA038, EA065, EA147, EA148, EA149, EA152
[BSW004] Version check	EA013
[BSW00409] Header files for production code error IDs	EA048
[BSW00385] List possible error notifications	EA099, EA100
[BSW00386] Configuration for detecting an error	EA011, EA045, EA059
[BSW161] Microcontroller abstraction	Not applicable (requirement on AUTOSAR architecture, not a single module)
[BSW162] ECU layout abstraction	Not applicable (requirement on AUTOSAR architecture, not a single module)
[BSW00324] Do not use HIS I/O Library	Not applicable (requirement on AUTOSAR architecture, not a single module)
[BSW005] No hard coded horizontal interfaces within MCAL	Not applicable (requirement on AUTOSAR architecture, not a single module)
[BSW00415] User dependent include files	Not applicable (only one user for this module)
[BSW164] Implementation of interrupt service routines	Not applicable (this module does not directly implement any ISRs)
[BSW00325] Runtime of interrupt service routines	See note in chapter 8.4 and chapter 8.6.3
[BSW00326] Transition from ISRs to OS tasks	Not applicable (requirement on implementation, not on specification)
[BSW00342] Usage of source code and object code	Not applicable (requirement on AUTOSAR architecture, not a single module)
[BSW00343] Specification and configuration of	EA070_Conf

time	
[BSW160] Human-readable configuration data	Not applicable (requirement on documentation, not on specification)
[BSW007] HIS MISRA C	Not applicable (requirement on implementation, not on specification)
[BSW00300] Module naming convention	Not applicable (requirement on implementation, not on specification)
[BSW00413] Accessing instances of BSW modules	Requirement can not be implemented in R2.0 timeframe.
[BSW00347] Naming separation of different instances of BSW drivers	Not applicable (requirement on the implementation, not on the specification)
[BSW00305] Self-defined data types naming convention	Not applicable (this module does not define any data types)
[BSW00307] Global variables naming convention	Not applicable (requirement on the implementation, not on the specification)
[BSW00310] API naming convention	Chapter 8.3
[BSW00373] Main processing function naming convention	Chapter 8.5.1 (EA096)
[BSW00327] Error values naming convention	EA139, EA140
[BSW00335] Status values naming convention	Chapter 8.1
[BSW00350] Development error detection keyword	EA011, EA059, EA039
[BSW00408] Configuration parameter naming convention	Chapter 10.2
[BSW00410] Compiler switches shall have defined values	Chapter 10.2
[BSW00411] Get version info keyword	Chapter 10.2.3
[BSW00346] Basic set of module files	EA113
[BSW158] Separation of configuration from implementation	EA113
[BSW00314] Separation of interrupt frames and service routines	Not applicable (this module does not directly implement any ISRs)
[BSW00370] Separation of callback interface from API	EA113, Chapter 8.4
[BSW00348] Standard type header	Not applicable (requirement on the standard header file)
[BSW00353] Platform specific type header	Not applicable (requirement on the platform specific header file)
[BSW00361] Compiler specific language extension header	Not applicable (requirement on the compiler specific header file)
[BSW00301] Limit imported information	EA113
[BSW00302] Limit exported information	Not applicable (requirement on the implementation, not on the specification)
[BSW00328] Avoid duplication of code	Not applicable (requirement on the implementation, not on the specification)
[BSW00312] Shared code shall be reentrant	Not applicable (requirement on the implementation, not on the specification)
[BSW006] Platform independency	Not applicable (this is a module of the microcontroller abstraction layer)

[BSW00357] Standard API return type	Chapter 8.3.3, Chapter 8.3.4. Chapter 8.3.8, Chapter 8.3.10
[BSW00377] Module specific API return types	Chapter 8.3.6, Chapter 8.3.7
[BSW00304] AUTOSAR integer data types	Not applicable (requirement on implementation, not for specification)
[BSW00355] Do not redefine AUTOSAR integer data types	Not applicable (requirement on implementation, not for specification)
[BSW00378] AUTOSAR boolean type	Not applicable (requirement on implementation, not for specification)
[BSW00306] Avoid direct use of compiler and platform specific keywords	Not applicable (requirement on implementation, not for specification)
[BSW00308] Definition of global data	Not applicable (requirement on implementation, not for specification)
[BSW00309] Global data with read-only constraint	Not applicable (requirement on implementation, not for specification)
[BSW00371] Do not pass function pointers via API	Not applicable (no function pointers in this specification)
[BSW00358] Return type of init() functions	Chapter 8.3.1
[BSW00414] Parameter of init function	Chapter 8.3.1
[BSW00376] Return type and parameters of main processing functions	Chapter 8.5.1
[BSW00359] Return type of callback functions	Chapter 8.4
[BSW00360] Parameters of callback functions	Chapter 8.4
[BSW00329] Avoidance of generic interfaces	Chapter 8.3 (explicit interfaces defined)
[BSW00330] Usage of macros / inline functions instead of functions	Not applicable (requirement on implementation, not for specification)
[BSW00331] Separation of error and status values	EA045, EA139, EA140
[BSW009] Module User Documentation	Not applicable (requirement on documentation, not on specification)
[BSW00401] Documentation of multiple instances of configuration parameters	Not applicable (all configuration parameters are single instance only)
[BSW172] Compatibility and documentation of scheduling strategy	Not applicable (no internal scheduling policy)
[BSW010] Memory resource documentation	Not applicable (requirement on documentation, not on specification)
[BSW00333] Documentation of callback function context	Not applicable (requirement on documentation, not for specfciation)
[BSW00374] Module vendor identification	EA043_Conf
[BSW00379] Module identification	EA043_Conf
[BSW003] Version identification	EA043_Conf
[BSW00318] Format of module version numbers	EA043_Conf
[BSW00321] Enumeration of module version numbers	Not applicable (requirement on implementation, not for specification)
[BSW00341] Microcontroller compatibility documentation	Not applicable (requirement on documentation, not on specification)

[BSW00334] Provision of XML file	Not applicable (requirement on documentation, not on specification)

Document: General Requirements on SPAL

Requirement	Satisfied by
[BSW12263] Object code compatible configuration concept	Not applicable (this module does not provide any post-build parameters)
[BSW12056] Configuration of notification mechanisms	Chapter 8.6.3
[BSW12267] Configuration of wake-up sources	Not applicable (this module does not provide any wakeup capabilities)
[BSW12057] Driver module initialization	EA017
[BSW12125] Initialization of hardware resources	Not applicable (this module has no direct hardware access)
[BSW12163] Driver module de-initialization	Not applicable (this module does not provide any shutdown capabilities)
[BSW12058] Individual initialization of overall registers	Not applicable (this module has no direct hardware access)
[BSW12059] General initialization of overall registers	Not applicable (this module has no direct hardware access)
[BSW12060] Responsibility for initialization of one-time writable registers	Not applicable (this module has no direct hardware access)
[BSW12461] Responsibility for register initialization	Not applicable (this module has no direct hardware access)
[BSW12462] Provide settings for register initialization	Not applicable (this module has no direct hardware access)
[BSW12463] Combine and forward settings for register initialization	Not applicable (this module has no direct hardware access)
[BSW12062] Selection of static configuration sets	Not applicable (this module does not have configuration data)
[BSW12068] MCAL initialization sequence	Not applicable (this module belongs to the ECU abstraction layer)
[BSW12069] Wake-up notification of ECU State Manager	Not applicable (this module does not provide any wakeup capabilities)
[BSW157] Notification mechanisms of drivers and handlers	Not applicable (this module does not provide any notification mechanisms)
[BSW12155] Prototypes of callback functions	Not applicable (this module does not implement any callback routines)
[BSW12169] Control of operation mode	EA085
[BSW12063] Raw value mode	Not applicable (this module does not handle or mishandle any data)
[BSW12075] Use of application buffers	Chapters 8.3.3, and 8.3.4
[BSW12129] Resetting of interrupt flags	Not applicable (this module does not directly implement any ISRs)
[BSW12064] Change of operation mode during running operation	Not applicable (this module has no internal operation mode)
[BSW12448] Behavior after development error detection	Chapter 7.4

Document ID 287: AUTOSAR_SWS_EEPROMAbstraction

[BSW12067] Setting of wake-up conditions	Not applicable (this module does not provide any wakeup capabilities)
[BSW12077] Non-blocking implementation	Not applicable (this module does not implement any schedulable services)
[BSW12078] Runtime and memory efficiency	Not applicable (requirement on implementation, not on specification)
[BSW12092] Access to drivers	Not applicable (this module is the EEPROM driver's "manager")
[BSW12265] Configuration data shall be kept constant	Not applicable (this module does not have configuration data)
[BSW12264] Specification of configuration items	EA039, EA040_Conf, EA043_Conf
[BSW12081] Use HIS requirements as input	Not applicable (no corresponding HIS requirements available)

Document: Requirements on Memory Hardware Abstraction Layer

Requirement	Satisfied by
BSW14001 Configuration of address alignment	EA004, EA039
BSW14002 Configuration of number of required write cycles	EA079_Conf, EA080, EA040_Conf
BSW14003 Configuration of maximum blocking time	EA070_Conf
BSW14004 Configuration of "immediate" data blocks	EA040_Conf, EA131_Conf
BSW14026 Don't use certain block numbers	EA006
BSW14027 Publish overhead for internal management data per block	EA043_Conf, EA126_Conf
BSW14005 Virtual linear address space and segmentation	EA003
BSW14006 Alignment of block erase / write addresses	EA004, EA024
BSW14007 Alignment of block read addresses	Note below EA021
BSW14008 Checking block read addresses	EA038
BSW14009 Conversion of logical to physical addresses	EA007, EA021, EA024, EA036, EA063
BSW14010 Block-wise write service	Chapter 8.3.4
BSW14029 Block-wise read service	Chapter 8.3.3
BSW14031 Service to cancel an ongoing asynchronous operation	Chapter 8.3.5
BSW14028 Service to invalidate a memory block	Chapter 8.3.8
BSW14012 Spreading of write access	EA079, EA080
BSW14013 Writing of "immediate" data must not be delayed	7.1.4
BSW14032 Block-wise erase service for immediate data	EA093, EA063, EA064, EA065
BSW14014 Detection of data inconsistencies	EA104, EA046, EA047
BSW14015 Reporting of data inconsistencies	EA104
BSW14016 Don't return inconsistent data to the caller	EA104
BSW14017 Scope of EEPROM Abstraction Layer	Chapter 1
BSW14018 Scope of Flash EEPROM Emulation	Not applicable (this is the EA modules specification)

7 Functional specification

7.1 General behavior

[EA137] ⌈ 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). ⌋()

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`.

[EA075] ⌈ 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. ⌋()

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.

[EA005] ⌈ 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`). ⌋()

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.

[EA068] ⌈ Logical blocks must not overlap each other and must not be contained within one another. ⌋()

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.*
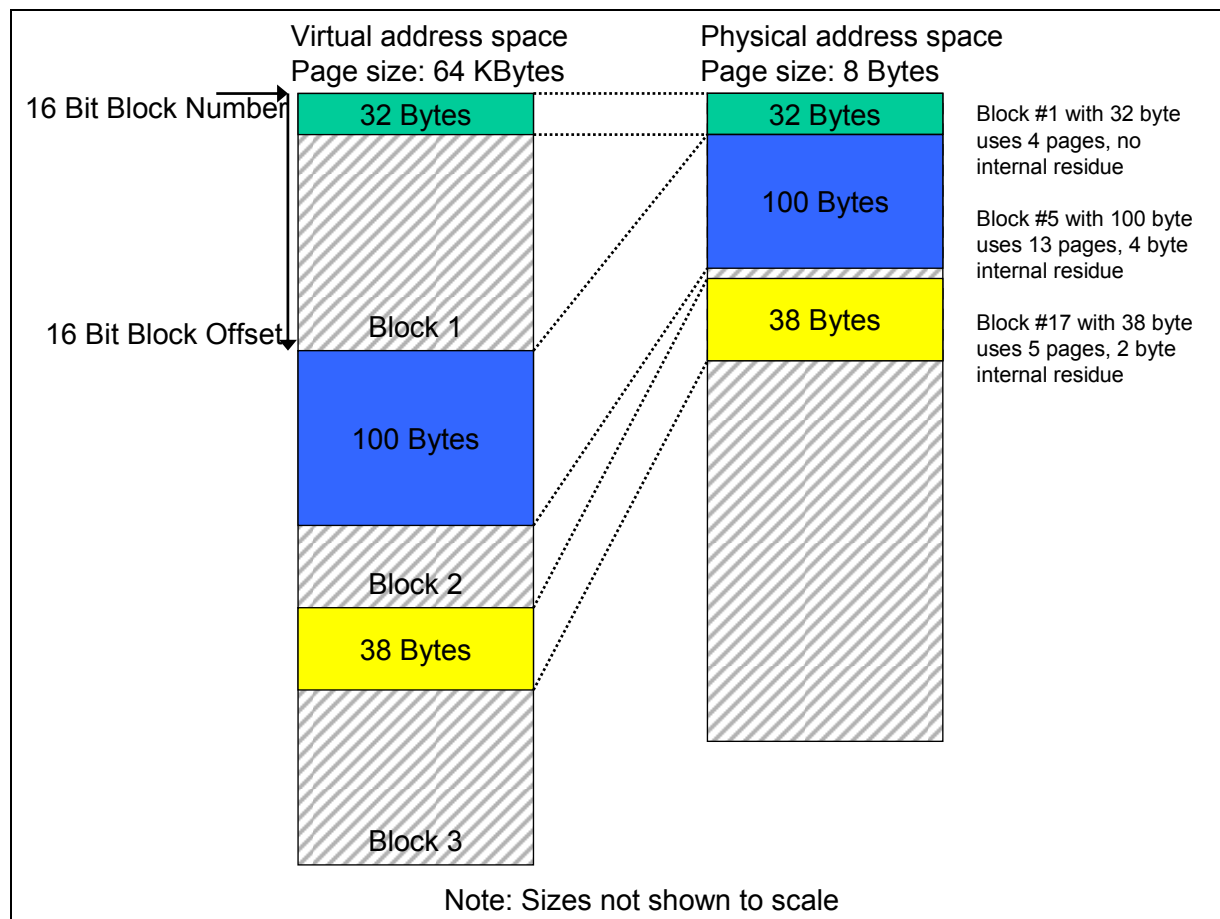
Figure 3: Virtual vs. physical memory layout

[EA006] ⌈ The block numbers 0x0000 and 0xFFFF shall not be configurable for a logical block (see chapter 10.2.3, `EaBlockNumber` for details). ⌋(BSW14026)

7.1.2 Address calculation

[EA007] ⌈ 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. ⌋(BSW14009)

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.

[EA066] ⌈ Only those bits of the 16bit block number, that do not denote a specific dataset or redundant copy shall be used for address calculation. ⌋()

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).
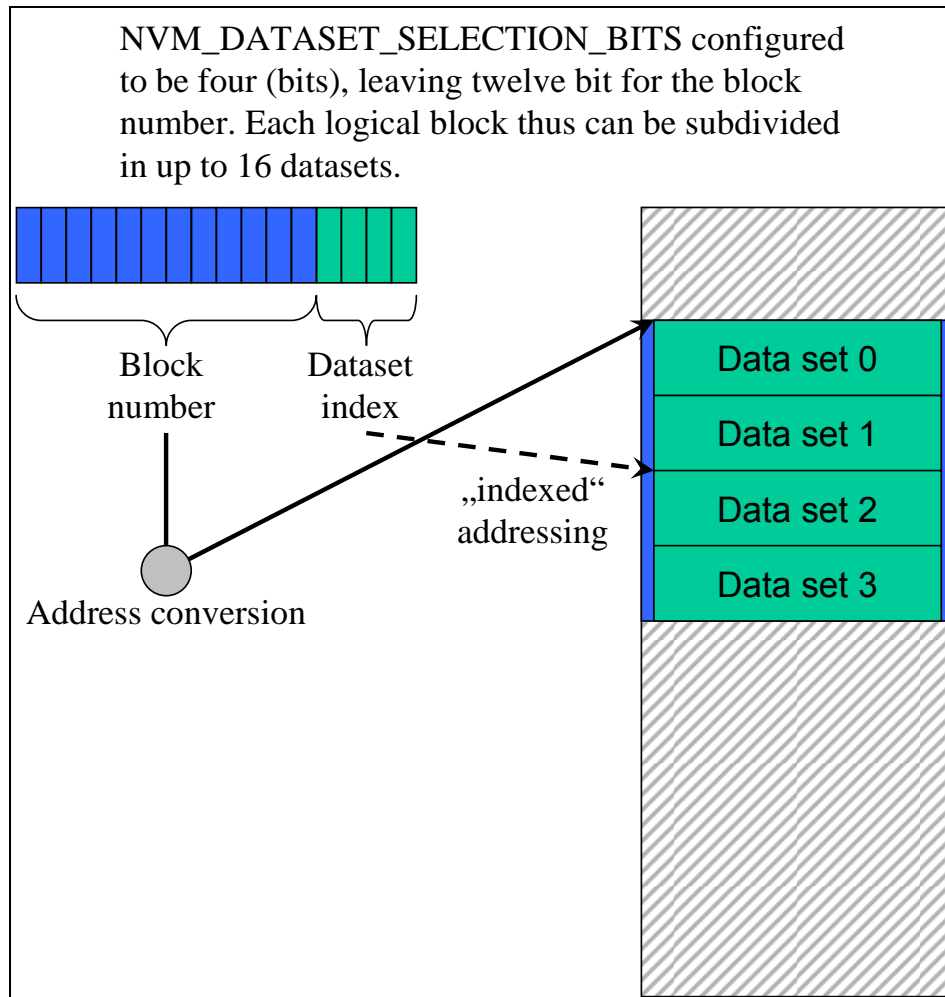
Figure 4: Block number and dataset index

7.1.3 Limitation of erase / write cycles

[EA079] ⌈ The configuration of the Ea module shall define the expected number of erase/write cycles for each logical block in the configuration parameter `EaNumberOfWriteCycles`. ⌋()

[EA080] ⌈ 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. ⌋(BSW14002)

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

[EA046] ⌈ 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. ⌋(BSW14014)

[EA047] ⌈ 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). ⌋(BSW14014)

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

[EA048] ⌈ Values for production code Event Ids are assigned externally by the configuration of the Dem. They are published in the file Dem_IntErrId.h and included via Dem.h. ⌋(BSW00409)

[EA049] ⌈ Development error values are of type uint8. ⌋()

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_INVALID_DATA_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`
API service called while module is busy doing internal management operations	Development	`EA_E_BUSY_INTERNAL`	`0x07`
Ea_Cancel called while no job was pending	Development	`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.

Document ID 287: AUTOSAR_SWS_EEPROMAbstraction

7.3 Error detection

[EA011] ⌈ The detection of development errors shall be configurable (on/off) at pre-compile time. The switch `EA_DEV_ERROR_DETECT` shall activate or deactivate the detection of all development errors. ⌋(BSW00338, BSW00386, BSW00350)

[EA059] ⌈ If the *EA_DEV_ERROR_DETECT* switch is enabled, API parameter checking is enabled. The detailed description of the detected errors can be found in chapter 7.2 and chapter 8. ⌋(BSW00386, BSW00350)

[EA060] ⌈ The detection of production code errors cannot be switched off. ⌋()

7.4 Error notification

[EA045] ⌈ Detected development errors shall be reported to the `Det_ReportError` service of the Development Error Tracer (DET) if the pre-processor switch `EA_DEV_ERROR_DETECT` is set (see chapter 10.2). ⌋(BSW00338, BSW00369, BSW00386, BSW00331)

[EA081] ⌈ Production errors shall be reported to Diagnostic Event Manager. ⌋()

7.5 Consistency checks

[EA013] ⌈ The EA module shall perform inter module checks to avoid integration of incompatible files: all included header files shall be checked by pre-processing directives. The EA module shall thereby verify that `<MODULENAME>_AR_RELEASE_MAJOR_VERSION` and `<MODULENAME>_AR_RELEASE_MINOR_VERSION` are identical to the expected values, where `<MODULENAME>` is the module abbreviation of the external module, which provides the included header file. If the values are not identical, an error shall be raised at compile time. ⌋(BSW004)

Note: The configuration tool shall check all configuration parameters for being within the expected bounds. Also the dependencies between configuration parameters shall be checked by the configuration tool during system generation or during the build process (for details see chapter 10).

7.6 Debugging support

[EA155] ⌈ The module's job result, the status and the variables used for job control (for externally requested jobs as well as for internal management operations) shall be made globally accessible. ⌋()

8 API specification

8.1 Imported Types

[EA083]
⌈

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

⌋()

[EA117] ⌈ The types mentioned in EA083 shall not be changed or extended for a specific EA module or hardware platform. ⌋()

8.2 Type definitions

This module does not define any module specific types.

8.3 Function definitions

8.3.1 Ea_Init

[EA084]
⌈

Service name:	Ea_Init

Syntax:	`void Ea_Init(`
	`    void`
	`)`

Service ID[hex]:	0x00
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	Initializes the EEPROM abstraction module.

⌋()

[EA017]⌈ The function `Ea_Init` shall shall set the module state from `MEMIF_UNINIT` to `MEMIF_BUSY_INTERNAL` once it starts the module's initialization. ⌋(BSW12057)

[EA128]⌈ 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. ⌋(BSW00406)

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

[EA085]

⌈

Service name:	Ea_SetMode

Syntax:	`void Ea_SetMode(`
	`    MemIf_ModeType Mode`
	`)`

Service ID[hex]:	0x01
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	Mode Desired mode for the underlying EEPROM driver
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	Sets the mode.

⌋(BSW12169)

Example: During normal operation of an ECU the EA module and underlying device driver shall use as few (runtime) resources as possible, therefore the EEPROM driver is switched to "slow" mode. During startup and especially during shutdown it might be desirable to read / write the NV memory blocks as fast as possible, therefore the EA module and the underlying device driver could be switched into "fast" mode.

[EA020]⌈ If the current module state is `MEMIF_IDLE` and if supported by the underlying hardware and device driver, the function `Ea_SetMode` shall call the function "`Eep_SetMode`" of the underlying EEPROM driver with the given "`Mode`" parameter. ⌋()

[EA150]⌈ 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. ⌋()

[EA129] ⌈ 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. ⌋(BSW00406)

[EA165] ⌈ 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. ⌋()

[EA166] ⌈ If development error detection is enabled for the module: the function `Ea_SetMode` shall check if the module state is `MEMIF_BUSY_INTERNAL`. If this is the case, the function `Ea_SetMode` shall raise the development error `EA_E_BUSY_INTERNAL` and return to the caller without executing the mode switch. ⌋()

8.3.3 Ea_Read

[EA086]

⌈

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.	

⌋()

[EA021] ⌈ The function `Ea_Read` shall take the block number and offset and calculate the corresponding memory read address. ⌋(BSW14007, BSW14009)

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.

[EA072] ⌈ The EA module shall execute the read operation asynchronously within the EA module's main function. ⌋()

[EA022] ⌈ If the current module status is `MEMIF_IDLE` or if the current module status is `MEMIF_BUSY INTERNAL` and the internal management operation can be suspended or aborted, 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`. ⌋()

EA179: ⌈ If the current module status is `MEMIF_UNINIT` or `MEMIF_BUSY` or `MEMIF_BUSY_INTERNAL` and the internal management operation can't be suspended or aborted, the function `Ea_Read` shall reject the job request and return with `E_NOT_OK`. ⌋()

[EA130] ⌈ 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`. ⌋(BSW00406)

[EA167] ⌈ 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`. ⌋()

EA180: If development error detection is enabled for the module: if the current module status is `MEMIF_BUSY_INTERNAL` and if it is not possible to suspend or abort the internal management operation (because of data consistency / module implementation / hardware restrictions), the function `Ea_Read` shall reject the read request, raise the development error `EA_E_BUSY_INTERNAL` and return with `E_NOT_OK`. ()

[EA147] ⌈ 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`. ⌋(BSW00323)

[EA168] ⌈ 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`. ⌋()

[EA169] ⌈ 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`. ⌋()

[EA170] ⌈ 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_INVALID_DATA_PTR` and return with `E_NOT_OK`. ⌋()

[EA158] ⌈ If a read request is rejected by the function `Ea_Read`, i.e. requirements EA130, EA147, EA167, EA168, EA169, EA170, EA179 or EA180 apply, the function `Ea_Read` shall not change the current module status or job result. ⌋()

8.3.4 Ea_Write

[EA087]

⌈

Service name:	Ea_Write	

Syntax:	`Std_ReturnType Ea_Write(`
	`    uint16 BlockNumber,`
	`    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.	

⌋()

Document ID 287: AUTOSAR_SWS_EEPROMAbstraction

[EA024] ⌈ 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. ⌋(BSW14006, BSW14009)

[EA151] ⌈ The function `Ea_Write` shall set the length parameter for the write job to the length configured for this logical block. ⌋()

[EA025] ⌈ If the current module status is `MEMIF_IDLE` or if the current module status is `MEMIF_BUSY INTERNAL` and the internal management operation can be suspended or aborted, 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`. ⌋()

EA181: If the current module status is `MEMIF_UNINIT` or `MEMIF_BUSY` or `MEMIF_BUSY_INTERNAL` and the internal management operation can't be suspended or aborted, the function `Ea_Write` shall reject the job request and return with `E_NOT_OK`. ⌋()

EA182: If the write request addresses a block containing immediate data, the function `Ea_Write` shall accept the write request, even if the current module status is `MEMIF_BUSY_INTERNAL` and the internal management operation can't be suspended or aborted. ⌋()

Note: In this case the internal management operation shall be aborted without the chance to restart it and with the risk of unrecoverable errors for the "normal" data.

[EA026] ⌈ The EA module shall execute the write job of the function `Ea_Write` asynchronously within the EA module's main function. ⌋()

[EA131] ⌈ 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`. ⌋(BSW00406)

[EA171] ⌈ 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`. ⌋()

EA183: If development error detection is enabled for the module: if the current module status is `MEMIF_BUSY_INTERNAL` and if it is not possible to suspend or

abort the internal management operation (because of data consistency / module implementation / hardware restrictions), the function `Ea_Write` shall reject the write request, raise the development error `FEE_E_BUSY_INTERNAL` and return with `E_NOT_OK`. ⌋()

[EA148] ⌈ 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`. ⌋(BSW00323)

[EA172] ⌈ 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_INVALID_DATA_PTR` and return with `E_NOT_OK`. ⌋()

[EA159] ⌈ If a write request is rejected by the function `Ea_Write`, i.e. requirements EA131, EA171, EA148, EA172, EA181 or EA183 apply, the function `Ea_Write` shall not change the current module status or job result. ⌋()

8.3.5 Ea_Cancel

[EA088]

⌈

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.

⌋()

Note: The function `Ea_Cancel` and the cancel function of the underlying EEPROM driver are synchronous in their behaviour, 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 9.4 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.

[EA132] ⌈ 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. ⌋(BSW00406)

[EA077] ⌈ 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. ⌋()

[EA078] ⌈ 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`. ⌋()

[EA160] ⌈ 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. ⌋()

[EA173] ⌈ 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`. ⌋()

8.3.6 Ea_GetStatus

[EA089]

⌈

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.

⌋()

[EA034] ⌈ The function `Ea_GetStatus` shall return `MEMIF_UNINIT` if the module has not (yet) been initialized. ⌋()

[EA156] ⌈ 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. ⌋()

[EA157] ⌈ The function `Ea_GetStatus` shall return `MEMIF_BUSY` if it is currently processing a request from the upper layer. ⌋()

[EA073] ⌈ 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

[EA090]

⌈

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.	

⌋()

[EA134] ⌈ 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`. ⌋(BSW00406)

[EA035] ⌈ The function `Ea_GetJobResult` shall return the status of the last job requested by the NVRAM manager. ⌋(BSW00406)

[EA174] ⌈ 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. ⌋()

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

[EA091]

⌈

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	None	

(inout):	
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.

⌋()

[EA036] ⌈ The function `Ea_InvalidateBlock` shall take the block number and calculate the corresponding memory block address. ⌋(BSW14009)

[EA037] ⌈ 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. ⌋()

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.

[EA135] ⌈ 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`. ⌋()

[EA175] ⌈ 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`. ⌋()

EA184: If development error detection is enabled for the module: if the current module status is `MEMIF_BUSY_INTERNAL` and if it is not possible to suspend or abort the internal management operation (because of data consistency / module implementation / hardware restrictions), the function `Ea_InvalidateBlock` shall reject the invalidation request, raise the development error `EA_E_BUSY_INTERNAL` and return with `E_NOT_OK`. ⌋()

[EA149] ⌈ 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`. ⌋(BSW00323)

[EA161] ⌈ If an invalidation request is rejected by the function `Ea_InvalidateBlock`, i.e. requirements [EA135](), [EA149](), [EA175]() or [EA184]() apply, the function `Ea_InvalidateBlock` shall not change the current module status or job result. ⌋()

8.3.9 Ea_GetVersionInfo

[EA092]
⌈

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.	

⌋()

[EA061] ⌈ The function Ea_GetVersionInfo shall return the version information of this module. The version information includes:
- Module Id
- Vendor Id
- Vendor specific version numbers (BSW00407). ⌋()

[EA062] ⌈ The function `Ea_GetVersionInfo` shall be pre compile time configurable On/Off by the configuration parameter `EaVersionInfoApi`. ⌋()

[EA082] ⌈ If source code for caller and callee of the function `Ea_GetVersionInfo` is available, the Ea module should realize this function as a macro, defined in the modules header file. ⌋()

[EA164] ⌈ 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_INVALID_DATA_PTR`. ⌋()

8.3.10 Ea_EraseImmediateBlock

[EA093]

⌈

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.	

⌋(BSW14032)

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.*

[EA063] ⌈ 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. ⌋(BSW14009, BSW14032)

[EA064] ⌈ 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. ⌋(BSW14032)

[EA136] ⌈ 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`. ⌋(BSW00406)

[EA176] ⌈ 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`. ⌋()

EA185:⌈ If development error detection is enabled for the module: if the current module status is `MEMIF_BUSY_INTERNAL` and if it is not possible to suspend or abort the internal management operation (because of data consistency / module implementation / hardware restrictions), the function `Ea_EraseImmediateBlock` shall reject the request, raise the development error `EA_E_BUSY_INTERNAL` and return with `E_NOT_OK`. ⌋()

[EA152] ⌈ 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`. ⌋(BSW00323)

[EA065] ⌈ 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`. ⌋(BSW00323, BSW14032)

[EA162] ⌈ If an erase request for an immediate block is rejected by the function `Ea_EraseImmediateBlock`, i.e. requirements [EA136](EA136), [EA176](EA176), [EA152](EA152), [EA065](EA065) or [EA185](EA185) apply, the function `Ea_EraseImmediateBlock` shall not change the current module status or job result. ⌋()

8.4 Call-back notifications

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

[EA114] ⌈ The Ea module shall provide function prototypes of the callback functions in the file `Ea_Cbk.h` ⌋()

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

[EA094]

⌈

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.

⌋()

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

[EA153] ⌈ 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. ⌋()

[EA051] ⌈ 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`). ⌋()

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

8.4.2 Ea_JobErrorNotification

[EA095]

⌈

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.

⌋()

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

[EA154] ⌈ 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. ⌋()

[EA053] ⌈ 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`). ⌋()

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

[EA096]

⌈

Service name:	Ea_MainFunction

Syntax:	`void Ea_MainFunction(`
	`    void`
	`)`

Service ID[hex]:	0x12
Timing:	VARIABLE_CYCLIC
Description:	Service to handle the requested jobs and the internal management operations.

⌋(BSW00373)

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

[EA178] ⌈ 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. ⌋()

[EA056] ⌈ The function `Ea_MainFunction` shall asynchronously handle the read / write / erase / invalidate jobs requested by the upper layer and internal management operations. ⌋()

[EA074] ⌈ 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. ⌋()

[EA104] ⌈ 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 EA046 and EA047), 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. ⌋(BSW14032, BSW14015, BSW14016)

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

EA186: ⌈ If the current module status is `MEMIF_BUSY_INTERNAL` and if the internal management operation can be suspended without jeopardizing the data consistency: the function `Ea_MainFunction` shall save all information which is necessary to resume the internal management operation, suspend the internal management operation and start processing the job requested by the upper layer. ⌋()

EA187: ⌈ If the current module status is `MEMIF_BUSY_INTERNAL` and if the internal management operation can be aborted without jeopardizing the data consistency: the function `Ea_MainFunction` shall save all information which is necessary to restart the internal management operation, abort the internal management operation and start processing the job requested by the upper layer. ⌋()

Note: Whether an internal management operation can be suspended or aborted depends on the type of management operation, the implementation of the EA module and the capabilities of the underlying hardware and thus cannot be determined in this document.

EA188: ⌈ 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. ⌋()

EA189: ⌈ 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. ⌋()

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.

[EA097]
⌈

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.

⌋()

8.6.2 Optional Interfaces

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

[EA098]
⌈

API function	Description
Det_ReportError	Service to report development errors.

⌋()

8.6.3 Configurable interfaces

In this chapter all interfaces are listed where the target function could be configured. The target function is usually a 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.

[EA099]

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.

(BSW00385)

[EA054] ⌈ 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. ⌋()

[EA141] ⌈ 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. ⌋()

[EA142] ⌈ 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 `EA064`). ⌋()

[EA143] ⌈ 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). ⌋()

[EA100]

⌈

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.

⌋(BSW00385)

[EA055] ⌈ 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). ⌋()

[EA144] ⌈ 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. ⌋()

[EA145] ⌈ 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 EA064). ⌋()

[EA146] ⌈ 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. ⌋()

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.
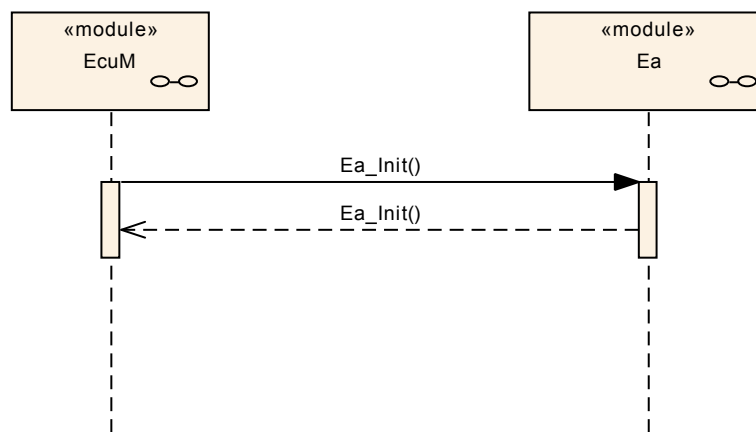
Figure 5: Sequence diagram of "Ea_Init" service

9.2 Ea_SetMode

The following figure shows as an example the call sequence for the Ea_SetMode service. This sequence diagram also applies to the other synchronous services of this module with exception of the Ea_Init routine (see above).
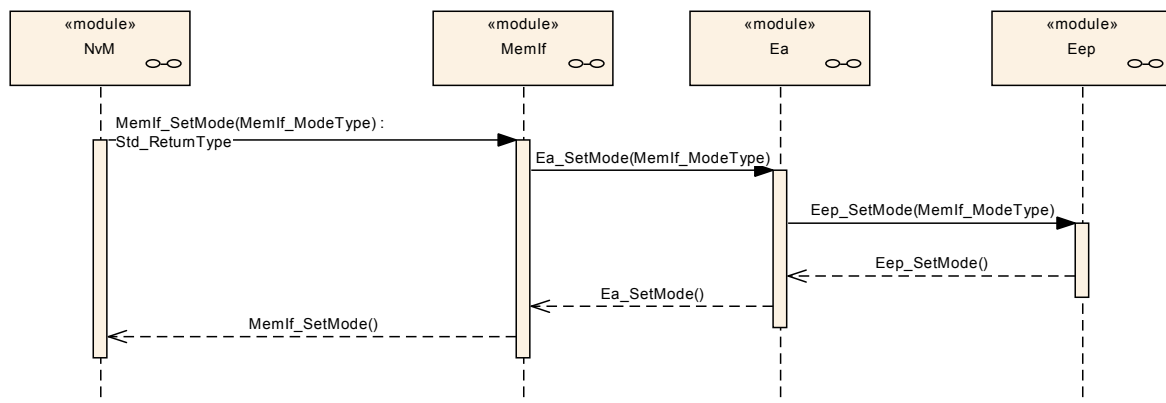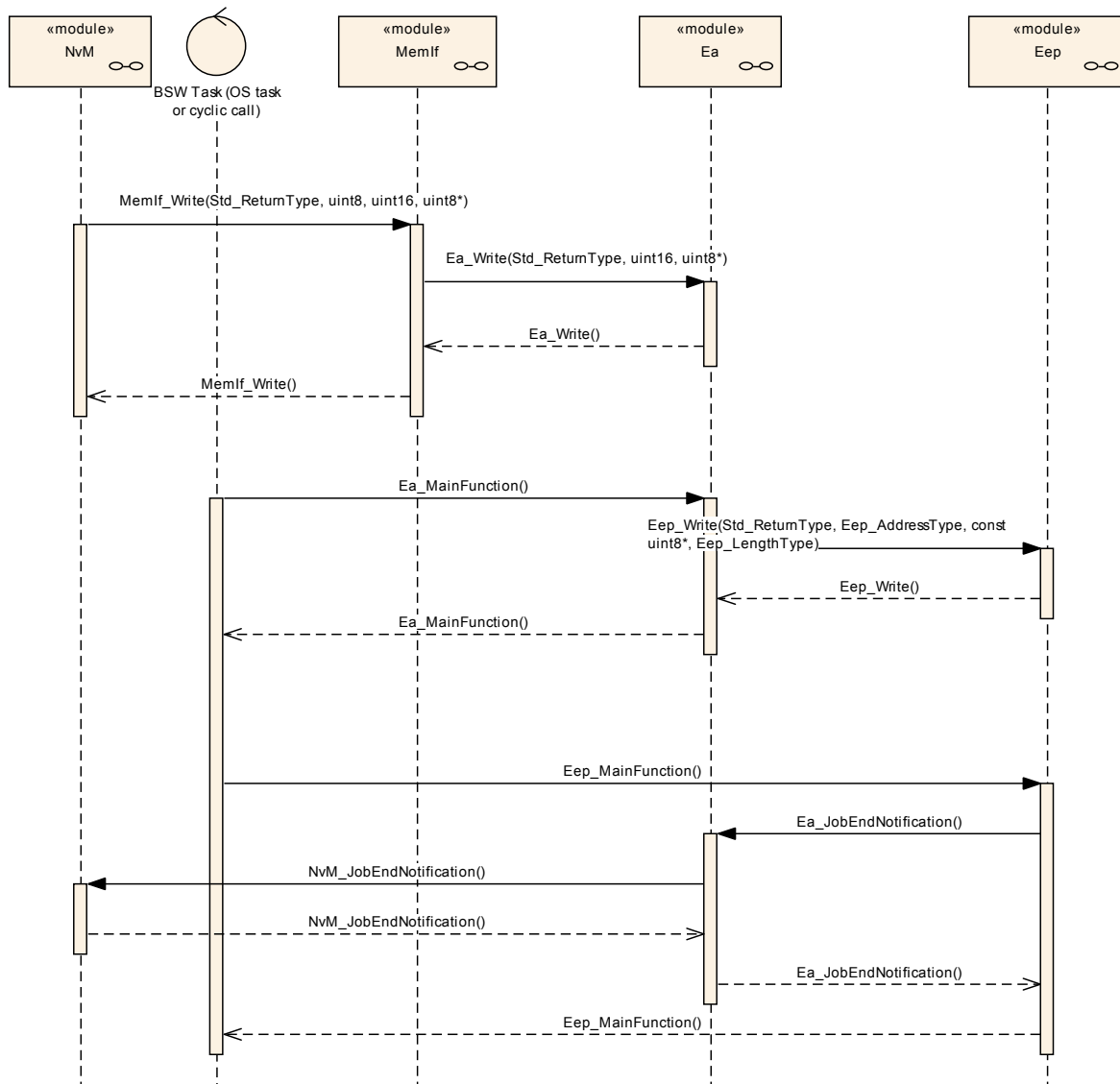
Figure 6: Sequence diagram of the "Ea_SetMode" service

9.3 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 7: Sequence diagram "Ea_Write"

9.4 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.
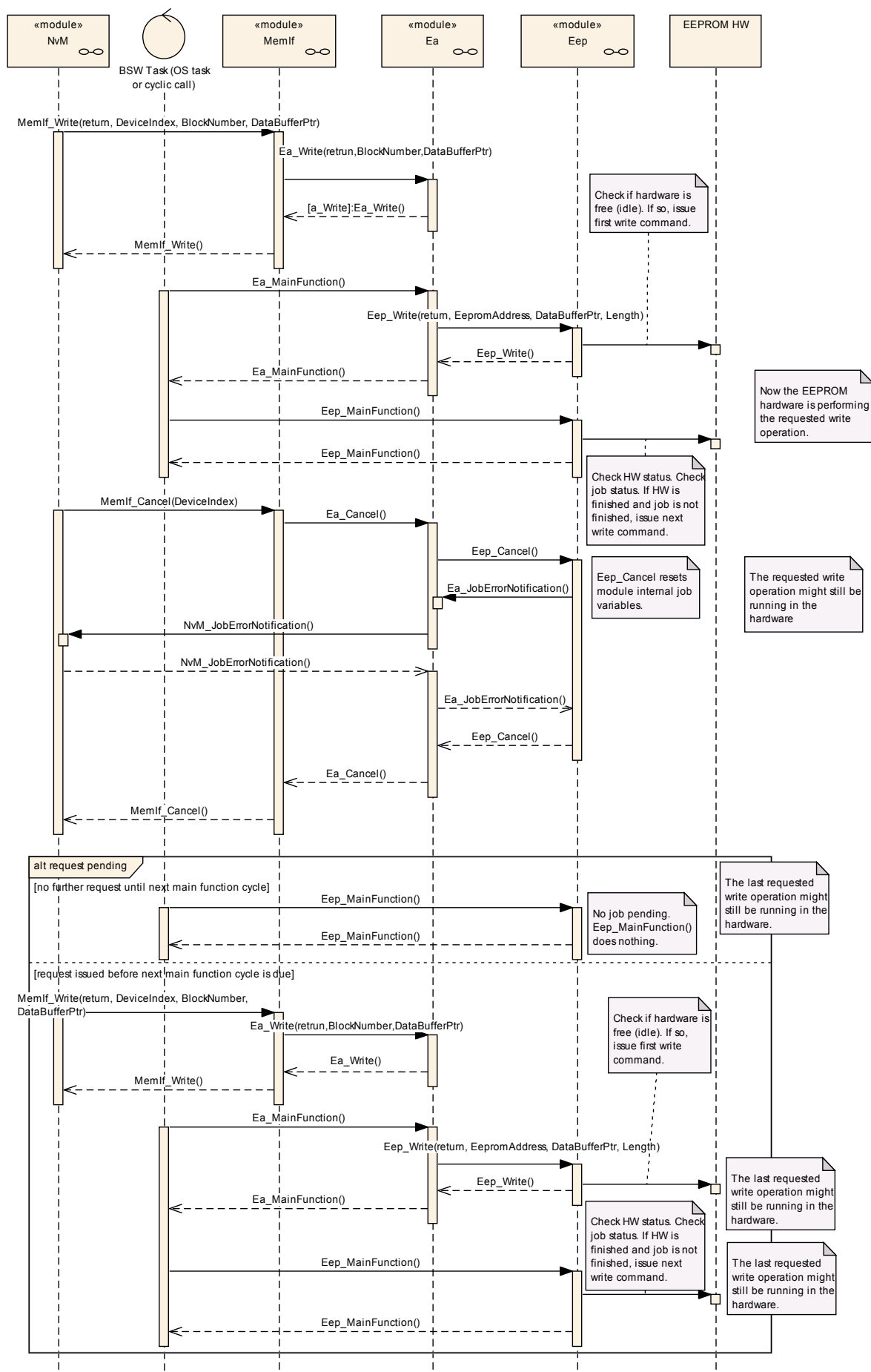
Figure 8: Sequence diagram „Ea_Cancel"

10 Configuration specification

10.1 How to read this chapter

In addition to this section, it is highly recommended to read the documents:
- AUTOSAR Layered Software Architecture [2]
- AUTOSAR ECU Configuration Specification [7]
  This document describes the AUTOSAR configuration methodology and the AUTOSAR configuration metamodel in detail.

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

10.1.1 Configuration and configuration parameters

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

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

10.1.2 Containers

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

10.1.3 Specification template for configuration parameters

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

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

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

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

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

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

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

10.2 Containers and configuration parameters

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

10.2.1 Variants

[EA163] ⌈ The EA module shall support (only) the following configuration variants:

- VARIANT-PRE-COMPILE
  Only parameters with "Pre-compile time" configuration are allowed in this variant. ⌋()

10.2.2 Ea

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.

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.
EaPublishedInformatio	1	Additional published parameters not covered by

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

Document ID 287: AUTOSAR_SWS_EEPROMAbstraction

10.2.3 EaGeneral

SWS Item	EA039_Conf :
Container Name	EaGeneral{EA_ModuleConfiguration}
Description	General configuration of the EEPROM abstraction module. This container lists block independent configuration parameters.
Configuration Parameters	

SWS Item	EA120_Conf :		
Name	EaDevErrorDetect {EA_DEV_ERROR_DETECT}		
Description	Pre-processor switch to enable and disable development error detection. true: Development error detection enabled. false: Development error detection disabled.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	EA118_Conf :		
Name	EaIndex		
Description	Specifies the InstanceId of this module instance. If only one instance is present it shall have the Id 0.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 254		
Default value	--		
ConfigurationClass	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	EA121_Conf :		
Name	EaNvmJobEndNotification {EA_NVM_JOB_END_NOTIFICATION}		
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	--		
ConfigurationClass	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	EA122_Conf :
Name	EaNvmJobErrorNotification {EA_NVM_JOB_ERROR_NOTIFICATION}
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	--		
ConfigurationClass	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	EA123_Conf :		
Name	EaPollingMode {EA_POLLING_MODE}		
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	--		
ConfigurationClass	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	EA001_Conf :		
Name	EaSetModeSupported {EA_SET_MODE_SUPPORTED}		
Description	Compile switch to enable / disable the function Ea_SetMode.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	true		
ConfigurationClass	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	EA124_Conf :		
Name	EaVersionInfoApi {EA_VERSION_INFO_API}		
Description	Pre-processor switch to enable / disable the API to read out the modules version information. true: Version info API enabled. false: Version info API disabled.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	EA125_Conf :		
Name	EaVirtualPageSize {EA_VIRTUAL_PAGE_SIZE}		
Description	The size in bytes to which logical blocks shall be aligned.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	--		
ConfigurationClass	Pre-compile time	X	All Variants
	Link time	--	

	Post-build time	--	
Scope / Dependency	scope: module		

No Included Containers

10.2.4 EaBlockConfiguration

SWS Item	**EA040_Conf :**
Container Name	EaBlockConfiguration{EA_BlockConfiguration}
Description	Configuration of block specific parameters for the EEPROM abstraction module.
Configuration Parameters	

SWS Item	**EA130_Conf :**		
Name	EaBlockNumber {EA_BLOCK_NUMBER}		
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	--		
ConfigurationClass	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**EA128_Conf :**		
Name	EaBlockSize {EA_BLOCK_SIZE}		
Description	Size of a logical block in bytes.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 65535		
Default value	--		
ConfigurationClass	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**EA131_Conf :**		
Name	EaImmediateData {EA_IMMEDIATE_DATA}		
Description	Marker for high priority data. true: Block contains immediate data. false: Block does not contain immediate data.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**EA119_Conf :**
Name	EaNumberOfWriteCycles {EA_NUMBER_OF_WRITE_CYCLES}

Description	Number of write cycles required for this block.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	--		
ConfigurationClass	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	EA129_Conf :		
Name	EaDeviceIndex {EA_DEVICE_INDEX}		
Description	Device index (handle). Range: 0 .. 254 (0xFF reserved for broadcast call to GetStatus function).		
Multiplicity	1		
Type	Reference to [EepGeneral]		
ConfigurationClass	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module 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

10.3 Published Information

[[**EA177**]] ⌈ The standardized common published parameters as required by BSW00402 in the General Requirements on Basic Software Modules [3] shall be published within the header file of this module and need to be provided in the BSW Module Description. The according module abbreviation can be found in the List of Basic Software Modules [1]. ⌋()

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

10.3.1 EaPublishedInformation

SWS Item	EA043_Conf :
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	EA126_Conf :
Name	EaBlockOverhead {EA_BLOCK_OVERHEAD}

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	--		
ConfigurationClass	Published Information	X	All Variants
Scope / Dependency	scope: module		

SWS Item	EA070_Conf :		
Name	EaMaximumBlockingTime {EA_MAXIMUM_BLOCKING_TIME}		
Description	The maximum time the EA module's API routines shall be blocked (delayed) by internal operations. (EA070) Note: Internal operations in that case means operations that are not explicitly invoked from the upper layer module but need to be handled for proper operation of this module or the underlying memory driver.		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	0 .. INF		
Default value	--		
ConfigurationClass	Published Information	X	All Variants
Scope / Dependency	scope: module		

SWS Item	EA127_Conf :		
Name	EaPageOverhead {EA_PAGE_OVERHEAD}		
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	--		
ConfigurationClass	Published Information	X	All Variants
Scope / Dependency	scope: module		

No Included Containers

11 Changes to R3.x

11.1 Deleted SWS Items

SWS Item	Rationale
EA106	No requirement, statement left in
EA107	No requirement, statement left in
EA112	No requirement, statement left in
EA076	No requirement, statement left in
EA010	Requirement split into EA139 and EA140
EA050	No requirement, statement left in
EA052	No requirement, statement left in
EA012	Requirement deleted (confusing)
EA110	Requirement deleted (confusing)
EA116	Requirement deleted (redundant)
EA009	No requirement, statement left in (re-formulated)
EA115	Requirement deleted (redundant with EA074)

11.2 Replaced SWS Items

SWS Item of Release	replaced by SWS Item	Rationale

11.3 Changed SWS Items

SWS Item	Rationale
EA113	Picture extended and replaced
EA051	EA callback routine renamed to avoid naming conflicts
EA101	EA callback routine renamed to avoid naming conflicts
EA053	EA callback routine renamed to avoid naming conflicts
EA102	EA callback routine renamed to avoid naming conflicts
EA054	Configuruation parameter for NvM callback routine renamed to avoid naming conflicts
EA055	Configuruation parameter for NvM callback routine renamed to avoid naming conflicts
EA025	Copy-paste-error with FEE / EA module status fixed.
EA045	Copy-paste-error with development error switch fixed.
EA010	Development error EA_E_NOT_INITIALIZED added.
EA054	Callback notification for block invalidation (now EA143)
EA055	Callback notification for block invalidation (now EA146)
EA054	Req. split up to make it atomic: EA141, EA142, EA143
EA055	Req. split up to make it atomic: EA144, EA145, EA146
EA038	Check for invalid block number excluded from requirement
EA024	Setting for block offset restricted to address calculation.
EA063	Setting for block offset added to req. on address calculation.
EA037	Requirement refined (dependency added)
EA078	Requirement detailed (job result, module status)
EA118_Conf	Range restricted to meaningful min. & max. values.
EA054, EA141, EA142, EA143, EA055, EA144, EA145, EA146	Wrong naming convention for configuration parameters EaNvMJobEndNotification and EaNvmJobErrorNotification corrected.
EA034, EA073	Description of Ea_GetStatus behaviour corrected.
EA121_Conf,	Multiplicity of configuration parameters EaNvmJobEndNotification and

Document ID 287: AUTOSAR_SWS_EEPROMAbstraction

- AUTOSAR confidential -

EA122_Conf	EaNvmJobErrorNotification adapted.

11.4 Added SWS Items

SWS Item	Rationale
EA128	Check for initialization status
EA129	Check for initialization status
EA130	Check for initialization status
EA131	Check for initialization status
EA132	Check for initialization status
EA133	Check for initialization status
EA134	Check for initialization status
EA135	Check for initialization status
EA136	Check for initialization status
EA137	Module shall not provide a job queue (explcit statement)
EA139	Req. EA010 split up to make it atomic.
EA140	Req. EA010 split up to make it atomic.
EA141	Req. EA054 split up to make it atomic (this is EA054b)
EA142	Req. EA054 split up to make it atomic (this is EA054c)
EA143	Req. EA054 split up to make it atomic (this is EA054d)
EA144	Req. EA055 split up to make it atomic (this is EA055b)
EA145	Req. EA055 split up to make it atomic (this is EA055c)
EA146	Req. EA055 split up to make it atomic (this is EA055d)
EA147	Reformulation of Ea_Read description & functionality.
EA148	Clarification for Ea_Write, analogue to Ea_Read
EA149	Check for valid block no. added (was missing)
EA150	Compile switch to enable / disable Ea_SetMode added.
EA151	Setting of length parameter for write job added (was missing)
EA152	Check for valid block no. added (was missing)
EA153	Changing job result from job end notification routine.
EA154	Changing job result from job error notification routine.
EA155	Support of debugging concept added
EA156, EA157	Description of Ea_GetStatus behaviour corrected.
EA158, EA159, EA160, EA161, EA162	Job result (and module status) shall not be affected by rejected job requests.
EA163	Clarification of configuration variant description
EA001_PI	Rework of Published Information

12 Changes to R4.x

12.1 Deleted SWS Items

SWS Item	Rationale
EA038, EA139, EA140	
EA133	
EA101, EA102	

12.2 Replaced SWS Items

SWS Item of Release	replaced by SWS Item	Rationale

12.3 Changed SWS Items

SWS Item	Rationale
EA113	NvM_Cbk.h included optionally
EA013	Inter module checks detailed
EA086, EA087, EA089, EA091, EA093	Job result descriptions updated
EA013	Inter module checks clarified
EA013	
EA156, EA157	
EA037, EA158, EA159, EA160, EA161, EA162	
EA013	
EA049	`EA_E_PARAM_POINTER`
EA017, EA020, EA021, EA022, EA056, EA065, EA072, EA077, EA078, EA128, EA129, EA130, EA131, EA132, EA133, EA134, EA135, EA136, EA147, EA148, EA149, EA152, EA156, EA157, EA158, EA159, EA160, EA161, EA162, EA164	
EA022, EA025, EA149, EA158, EA159, EA161, EA162	

12.4 Added SWS Items

SWS Item	Rationale
EA164	DET error if NULL pointer is passed as an argument
EA165, EA166, EA167, EA168, EA169, EA170; EA171, EA172, EA173, EA174, EA175, EA176, EA178	

Document ID 287: AUTOSAR_SWS_EEPROMAbstraction

EA179, EA180, EA181, EA182, EA183, EA184, EA185, EA186, EA187, EA188, EA189	

13 Not applicable requirements

[EA999] 「 These requirements are not applicable to this specification. 」(BSW00416, BSW168, BSW00423, BSW00424, BSW00425, BSW00426, BSW00427, BSW00428, BSW00429, BSW00431, BSW00432, BSW00433, BSW00434, BSW00336, BSW00339, BSW00421, BSW00422, BSW00420, BSW00417, BSW161, BSW162, BSW00324, BSW005, BSW00415, BSW164, BSW00326, BSW00342, BSW160, BSW007, BSW00300, BSW00347, BSW00305, BSW00307, BSW00314, BSW00348, BSW00353, BSW00361, BSW00302, BSW00328, BSW00312, BSW006, BSW00304, BSW00355, BSW00378, BSW00306, BSW00308, BSW00309, BSW00371, BSW00330, BSW009, BSW00401, BSW172, BSW010, BSW00333, BSW00321, BSW00341, BSW00334, BSW12263, BSW12267, BSW12125, BSW12163, BSW12058, BSW12059, BSW12060, BSW12461, BSW12462, BSW12463, BSW12062, BSW12068, BSW12069, BSW157, BSW12155, BSW12063, BSW12129, BSW12064, BSW12067, BSW12077, BSW12078, BSW12092, BSW12265, BSW12081, BSW14018)