Document Title	Specification of Memory Abstraction Interface
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	285
Document Status	published
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	R20-11

Document Change History

Date	Release	Changed by	Change Description
2020-11-30	R20-11	AUTOSAR Release Management	• Chapter “7.1 Error classification” was reshaped
2019-11-28	R19-11	AUTOSAR Release Management	• Configuration layout added • Changed Document Status from Final to published
2018-10-31	4.4.0	AUTOSAR Release Management	• Editorial changes
2017-12-08	4.3.1	AUTOSAR Release Management	• Editorial changes
2016-11-30	4.3.0	AUTOSAR Release Management	• Updated tracing information • Editorial changes
2015-07-31	4.2.2	AUTOSAR Release Management	• Block result MEMIF_BLOCK_INCONSISTENT extended to blocks which can’t be foundError classification reworked • Links to requirements added
2014-10-31	4.2.1	AUTOSAR Release Management	• Requirements linked to features, general and module specific requirements
2014-03-31	4.1.3	AUTOSAR Release Management	• Editorial changes

Document Change History			
Date	**Release**	**Changed by**	**Change Description**
2013-10-31	4.1.2	AUTOSAR Release Management	• Timing requirement removed from • module's main function • "const" qualifier added to prototype of function Fee_Write • New configuration parameter FeeMainFunctionPeriod • Editorial changes • Removed chapter(s) on change documentation
2013-03-15	4.1.1	AUTOSAR Administration	• Reworked according to the new SWS_BSWGeneral • Scope attribute in tables in chapter 10 added • Changes in file include structure (clean-up) • Requirement IDs for type definitions added
2011-12-22	4.0.3	AUTOSAR Administration	• Module short name changed • Consistency checking reformulated
2010-09-30	3.1.5	AUTOSAR Administration	• Check for NULL pointer added • Inter module checks detailed
2010-02-02	3.1.4	AUTOSAR Administration	• Description of return values extended • File include structure changed • Variant requirement description added • Legal disclaimer revised
2008-08-13	3.1.1	AUTOSAR Administration	• Legal disclaimer revised
2007-12-21	3.0.1	AUTOSAR Administration	• File include structure updated • Return types of various APIs adapted • Ranges of configuration parameters adjusted • Legal disclaimer revised • Release Notes added • "Advice for users" revised • "Revision Information" added

Document Change History			
Date	**Release**	**Changed by**	**Change Description**
2006-05-16	2.0	AUTOSAR Administration	• Initial Release

Disclaimer

Table of Contents

1 Introduction and functional overview

This specification describes the functionality, API and configuration of the AUTOSAR Basic Software Module “Memory Abstraction Interface” (MemIf). This module allows the NVRAM manager to access several memory abstraction modules (FEE or EA modules) (see Figure 1).

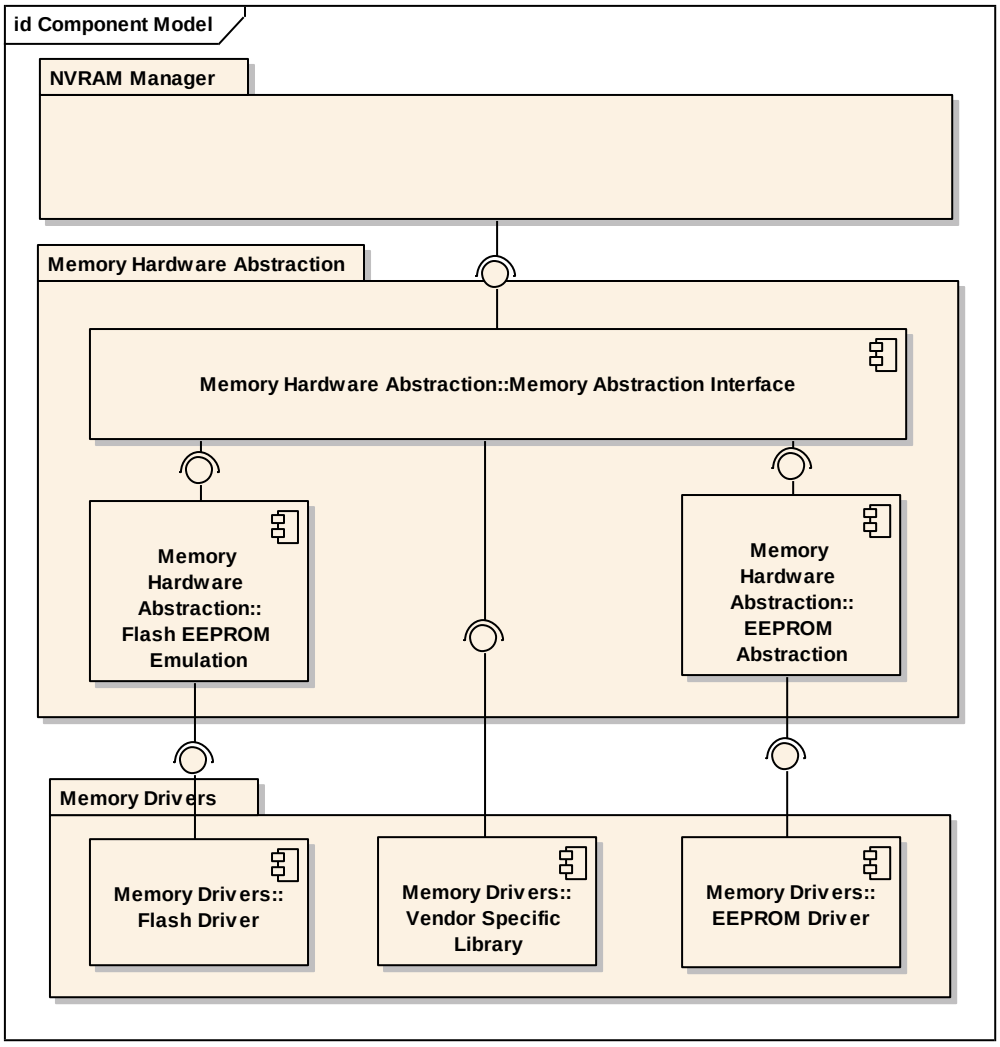

Figure 1: Module overview of memory hardware abstraction layer

The Memory Abstraction Interface (MemIf) shall abstract from the number of underlying FEE or EA modules and provide upper layers with a virtual segmentation on a uniform linear address space.

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)
Fast Mode	E.g. during startup / shutdown the underlying driver may be switched into fast mode in order to allow for fast reading / writing in those phases. *Note: Whether this is possible depends on the implementation of the driver and the capabilities of the underlying device. Whether it is done depends on the configuration of the NVRAM manager and thus on the needs of a specific project.*
Slow Mode	During normal operation the underlying driver may be used in slow mode in order to reduce the resource usage in terms of runtime or blocking time of the underlying device / communication media. *Note: Whether this is possible depends on the implementation of the driver and the capabilities of the underlying device. Whether it is done depends on the configuration of the NVRAM manager and thus on the needs of a specific project.*
Vendor specific library	A vendor specific library is an ICC-2 implementation of the FEE/FLS and EA/EEP modules respectively. It provides the same upper layer interface (API) and functionality as the corresponding ICC-3 implementation.

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.pdf

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

[4] General Requirements on SPAL
AUTOSAR_SRS_SPALGeneral.pdf

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

[6] Specification of Default Error Tracer
AUTOSAR_SWS_DefaultErrorTracer.pdf

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

3.2 Related standards and norms

[7] Specification of NVRAM Manager
AUTOSAR_SWS_NVRAMManager.doc

[8] Specification of Flash EEPROM Emulation
AUTOSAR_SWS_FlashEEPROMEmulation.pdf

[9] Specification of EEPROM Abstraction
AUTOSAR_SWS_EEPROMAbstraction.pdf

3.3 Related specification

4 Constraints and assumptions

4.1 Limitations

No limitations.

4.2 Applicability to car domains

No restrictions.

5 Dependencies to other modules

6 Requirements traceability

Requirement	Description	Satisfied by
RS_BRF_01472	AUTOSAR shall support modes	SWS_MemIf_00038
RS_BRF_02272	AUTOSAR shall offer tracing of application software behavior	SWS_MemIf_00042
SRS_BSW_00005	Modules of the µC Abstraction Layer (MCAL) may not have hard coded horizontal interfaces	SWS_MemIf_00999
SRS_BSW_00006	The source code of software modules above the µC Abstraction Layer (MCAL) shall not be processor and compiler dependent.	SWS_MemIf_00999
SRS_BSW_00007	All Basic SW Modules written in C language shall conform to the MISRA C 2012 Standard.	SWS_MemIf_00999
SRS_BSW_00009	All Basic SW Modules shall be documented according to a common standard.	SWS_MemIf_00999
SRS_BSW_00010	The memory consumption of all Basic SW Modules shall be documented for a defined configuration for all supported platforms.	SWS_MemIf_00999
SRS_BSW_00101	The Basic Software Module shall be able to initialize variables and hardware in a separate initialization function	SWS_MemIf_00999
SRS_BSW_00159	All modules of the AUTOSAR Basic Software shall support a tool based configuration	SWS_MemIf_00999
SRS_BSW_00160	Configuration files of AUTOSAR Basic SW module shall be readable for human beings	SWS_MemIf_00999
SRS_BSW_00161	The AUTOSAR Basic Software shall provide a microcontroller abstraction layer which provides a standardized interface to higher software layers	SWS_MemIf_00999
SRS_BSW_00162	The AUTOSAR Basic Software shall provide a hardware abstraction layer	SWS_MemIf_00999
SRS_BSW_00164	The Implementation of interrupt service routines shall be done by the Operating System, complex drivers or modules	SWS_MemIf_00999

SRS_BSW_00168	SW components shall be tested by a function defined in a common API in the Basis-SW	SWS_MemIf_00999
SRS_BSW_00170	The AUTOSAR SW Components shall provide information about their dependency from faults, signal qualities, driver demands	SWS_MemIf_00999
SRS_BSW_00172	The scheduling strategy that is built inside the Basic Software Modules shall be compatible with the strategy used in the system	SWS_MemIf_00999
SRS_BSW_00300	All AUTOSAR Basic Software Modules shall be identified by an unambiguous name	SWS_MemIf_00999
SRS_BSW_00302	All AUTOSAR Basic Software Modules shall only export information needed by other modules	SWS_MemIf_00999
SRS_BSW_00304	All AUTOSAR Basic Software Modules shall use the following data types instead of native C data types	SWS_MemIf_00999
SRS_BSW_00306	AUTOSAR Basic Software Modules shall be compiler and platform independent	SWS_MemIf_00999
SRS_BSW_00307	Global variables naming convention	SWS_MemIf_00999
SRS_BSW_00308	AUTOSAR Basic Software Modules shall not define global data in their header files, but in the C file	SWS_MemIf_00999
SRS_BSW_00309	All AUTOSAR Basic Software Modules shall indicate all global data with read-only purposes by explicitly assigning the const keyword	SWS_MemIf_00999
SRS_BSW_00312	Shared code shall be reentrant	SWS_MemIf_00999
SRS_BSW_00314	All internal driver modules shall separate the interrupt frame definition from the service routine	SWS_MemIf_00999
SRS_BSW_00321	The version numbers of AUTOSAR Basic Software Modules shall be enumerated according specific rules	SWS_MemIf_00999
SRS_BSW_00323	All AUTOSAR Basic Software Modules shall check passed API parameters for validity	SWS_MemIf_00022
SRS_BSW_00325	The runtime of interrupt service routines and functions that are	SWS_MemIf_00999

	running in interrupt context shall be kept short	
SRS_BSW_00327	Error values naming convention	SWS_MemIf_00006
SRS_BSW_00328	All AUTOSAR Basic Software Modules shall avoid the duplication of code	SWS_MemIf_00999
SRS_BSW_00330	It shall be allowed to use macros instead of functions where source code is used and runtime is critical	SWS_MemIf_00999
SRS_BSW_00333	For each callback function it shall be specified if it is called from interrupt context or not	SWS_MemIf_00999
SRS_BSW_00334	All Basic Software Modules shall provide an XML file that contains the meta data	SWS_MemIf_00999
SRS_BSW_00336	Basic SW module shall be able to shutdown	SWS_MemIf_00999
SRS_BSW_00337	Classification of development errors	SWS_MemIf_00006
SRS_BSW_00339	Reporting of production relevant error status	SWS_MemIf_00999
SRS_BSW_00341	Module documentation shall contains all needed informations	SWS_MemIf_00999
SRS_BSW_00342	It shall be possible to create an AUTOSAR ECU out of modules provided as source code and modules provided as object code, even mixed	SWS_MemIf_00999
SRS_BSW_00343	The unit of time for specification and configuration of Basic SW modules shall be preferably in physical time unit	SWS_MemIf_00999
SRS_BSW_00347	A Naming seperation of different instances of BSW drivers shall be in place	SWS_MemIf_00999
SRS_BSW_00348	All AUTOSAR standard types and constants shall be placed and organized in a standard type header file	SWS_MemIf_00999
SRS_BSW_00353	All integer type definitions of target and compiler specific scope shall be placed and organized in a single type header	SWS_MemIf_00999
SRS_BSW_00358	The return type of init() functions implemented by AUTOSAR Basic Software Modules shall be void	SWS_MemIf_00999
SRS_BSW_00359	All AUTOSAR Basic Software	SWS_MemIf_00999

	Modules callback functions shall avoid return types other than void if possible	
SRS_BSW_00360	AUTOSAR Basic Software Modules callback functions are allowed to have parameters	SWS_MemIf_00999
SRS_BSW_00361	All mappings of not standardized keywords of compiler specific scope shall be placed and organized in a compiler specific type and keyword header	SWS_MemIf_00999
SRS_BSW_00369	All AUTOSAR Basic Software Modules shall not return specific development error codes via the API	SWS_MemIf_00024
SRS_BSW_00371	The passing of function pointers as API parameter is forbidden for all AUTOSAR Basic Software Modules	SWS_MemIf_00999
SRS_BSW_00373	The main processing function of each AUTOSAR Basic Software Module shall be named according the defined convention	SWS_MemIf_00999
SRS_BSW_00375	Basic Software Modules shall report wake-up reasons	SWS_MemIf_00999
SRS_BSW_00378	AUTOSAR shall provide a boolean type	SWS_MemIf_00999
SRS_BSW_00380	Configuration parameters being stored in memory shall be placed into separate c-files	SWS_MemIf_00999
SRS_BSW_00384	The Basic Software Module specifications shall specify at least in the description which other modules they require	SWS_MemIf_00047
SRS_BSW_00385	List possible error notifications	SWS_MemIf_00048
SRS_BSW_00386	The BSW shall specify the configuration for detecting an error	SWS_MemIf_00006, SWS_MemIf_00023
SRS_BSW_00392	Parameters shall have a type	SWS_MemIf_00037, SWS_MemIf_00064, SWS_MemIf_00065, SWS_MemIf_00066
SRS_BSW_00398	The link-time configuration is achieved on object code basis in the stage after compiling and before linking	SWS_MemIf_00999
SRS_BSW_00399	Parameter-sets shall be located in a separate segment and shall be loaded after the code	SWS_MemIf_00999
SRS_BSW_00400	Parameter shall be selected from multiple sets of parameters after code has	SWS_MemIf_00999

	been loaded and started	
SRS_BSW_00401	Documentation of multiple instances of configuration parameters shall be available	SWS_MemIf_00999
SRS_BSW_00404	BSW Modules shall support post-build configuration	SWS_MemIf_00999
SRS_BSW_00405	BSW Modules shall support multiple configuration sets	SWS_MemIf_00999
SRS_BSW_00406	A static status variable denoting if a BSW module is initialized shall be initialized with value 0 before any APIs of the BSW module is called	SWS_MemIf_00999
SRS_BSW_00407	Each BSW module shall provide a function to read out the version information of a dedicated module implementation	SWS_MemIf_00045
SRS_BSW_00412	-	SWS_MemIf_00999
SRS_BSW_00413	An index-based accessing of the instances of BSW modules shall be done	SWS_MemIf_00999
SRS_BSW_00414	Init functions shall have a pointer to a configuration structure as single parameter	SWS_MemIf_00999
SRS_BSW_00415	Interfaces which are provided exclusively for one module shall be separated into a dedicated header file	SWS_MemIf_00999
SRS_BSW_00416	The sequence of modules to be initialized shall be configurable	SWS_MemIf_00999
SRS_BSW_00417	Software which is not part of the SW-C shall report error events only after the DEM is fully operational.	SWS_MemIf_00999
SRS_BSW_00422	Pre-de-bouncing of error status information is done within the DEM	SWS_MemIf_00999
SRS_BSW_00423	BSW modules with AUTOSAR interfaces shall be describable with the means of the SW-C Template	SWS_MemIf_00999
SRS_BSW_00424	BSW module main processing functions shall not be allowed to enter a wait state	SWS_MemIf_00999
SRS_BSW_00425	The BSW module description template shall provide means to model the defined trigger conditions of schedulable objects	SWS_MemIf_00999
SRS_BSW_00426	BSW Modules shall ensure	SWS_MemIf_00999

	data consistency of data which is shared between BSW modules	
SRS_BSW_00427	ISR functions shall be defined and documented in the BSW module description template	SWS_MemIf_00999
SRS_BSW_00428	A BSW module shall state if its main processing function(s) has to be executed in a specific order or sequence	SWS_MemIf_00999
SRS_BSW_00429	Access to OS is restricted	SWS_MemIf_00999
SRS_BSW_00432	Modules should have separate main processing functions for read/receive and write/transmit data path	SWS_MemIf_00999
SRS_BSW_00433	Main processing functions are only allowed to be called from task bodies provided by the BSW Scheduler	SWS_MemIf_00999
SRS_MemHwAb_14010	The FEE and EA modules shall provide a write service that operates only on complete configured logical blocks	SWS_MemIf_00040
SRS_MemHwAb_14019	The Memory Abstraction Interface shall provide uniform access to the API services of the underlying memory abstraction modules	SWS_MemIf_00017
SRS_MemHwAb_14020	The Memory Abstraction Interface shall allow the selection of an underlying memory abstraction module by using a device index	SWS_MemIf_00011, SWS_MemIf_00018, SWS_MemIf_00035
SRS_MemHwAb_14021	The Memory Abstraction Interface shall allow the pre-compile time configuration of the number of underlying memory abstraction modules	SWS_MemIf_00018, SWS_MemIf_00019, SWS_MemIf_00020, SWS_MemIf_00022
SRS_MemHwAb_14022	The Memory Abstraction Interface shall preserve the functionality of the underlying memory abstraction module	SWS_MemIf_00010, SWS_MemIf_00017, SWS_MemIf_00038, SWS_MemIf_00039, SWS_MemIf_00040, SWS_MemIf_00041, SWS_MemIf_00042, SWS_MemIf_00043, SWS_MemIf_00044, SWS_MemIf_00046
SRS_MemHwAb_14023	The Memory Abstraction Interface shall only check those parameters that are used within the interface itself	SWS_MemIf_00022
SRS_MemHwAb_14028	The FEE and EA modules shall provide a service to invalidate a logical block	SWS_MemIf_00044
SRS_MemHwAb_14029	The FEE and EA modules shall provide a read service that allows reading all or part of a	SWS_MemIf_00039

	logical block	
SRS_MemHwAb_14031	The FEE and EA modules shall provide a service that allows canceling an ongoing asynchronous operation	SWS_MemIf_00041
SRS_MemHwAb_14032	The FEE and EA modules shall provide an erase service that operates only on complete logical blocks containing immediate data	SWS_MemIf_00046
SRS_SPAL_00157	All drivers and handlers of the AUTOSAR Basic Software shall implement notification mechanisms of drivers and handlers	SWS_MemIf_00999
SRS_SPAL_12056	All driver modules shall allow the static configuration of notification mechanism	SWS_MemIf_00999
SRS_SPAL_12057	All driver modules shall implement an interface for initialization	SWS_MemIf_00999
SRS_SPAL_12063	All driver modules shall only support raw value mode	SWS_MemIf_00999
SRS_SPAL_12064	All driver modules shall raise an error if the change of the operation mode leads to degradation of running operations	SWS_MemIf_00999
SRS_SPAL_12067	All driver modules shall set their wake-up conditions depending on the selected operation mode	SWS_MemIf_00999
SRS_SPAL_12068	The modules of the MCAL shall be initialized in a defined sequence	SWS_MemIf_00999
SRS_SPAL_12069	All drivers of the SPAL that wake up from a wake-up interrupt shall report the wake-up reason	SWS_MemIf_00999
SRS_SPAL_12075	All drivers with random streaming capabilities shall use application buffers	SWS_MemIf_00999
SRS_SPAL_12077	All drivers shall provide a non blocking implementation	SWS_MemIf_00999
SRS_SPAL_12078	The drivers shall be coded in a way that is most efficient in terms of memory and runtime resources	SWS_MemIf_00019, SWS_MemIf_00020
SRS_SPAL_12092	The driver's API shall be accessed by its handler or manager	SWS_MemIf_00999
SRS_SPAL_12125	All driver modules shall only	SWS_MemIf_00999

	initialize the configured resources	
SRS_SPAL_12129	The ISRs shall be responsible for resetting the interrupt flags and calling the according notification function	SWS_MemIf_00999
SRS_SPAL_12163	All driver modules shall implement an interface for de-initialization	SWS_MemIf_00999
SRS_SPAL_12263	The implementation of all driver modules shall allow the configuration of specific module parameter types at link time	SWS_MemIf_00999
SRS_SPAL_12265	Configuration data shall be kept constant	SWS_MemIf_00999
SRS_SPAL_12267	Wakeup sources shall be initialized by MCAL drivers and/or the MCU driver	SWS_MemIf_00999
SRS_SPAL_12448	All driver modules shall have a specific behavior after a development error detection	SWS_MemIf_00023
SRS_SPAL_12461	Specific rules regarding initialization of controller registers shall apply to all driver implementations	SWS_MemIf_00999
SRS_SPAL_12462	The register initialization settings shall be published	SWS_MemIf_00999
SRS_SPAL_12463	The register initialization settings shall be combined and forwarded	SWS_MemIf_00999

7 Functional specification

7.1 Error classification

The section 7.2 "Error Handling" of the document "General Specification of Basic Software Modules" describes the error handling of the Basic Software in detail. Above all, it constitutes a classification scheme consisting of five error types which may occur in BSW modules.
Based on this foundation, the following section specifies particular errors arranged in the respective subsections below.

7.1.1 Development Errors

[SWS_MemIf_00006]⌈

Type of error	*Related error code*	*Error value*
API service called with wrong device index parameter	MEMIF_E_PARAM_DEVICE	0x01
API service called with NULL pointer argument	MEMIF_E_PARAM_POINTER	0x02

⌋(SRS_BSW_00337, SRS_BSW_00386, SRS_BSW_00327)

7.1.2 Runtime Errors

There are no runtime errors.

7.1.3 Transient Faults

There are no transient faults.

7.1.4 Production Errors

There are no production errors.

7.1.5 Extended Production Errors

There are no extended production errors.

8 API specification

8.1 Imported types

8.1.1 Standard types

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

[SWS_MemIf_00037]⌈

Module	*Header File*	*Imported Type*
Std	Std_Types.h	Std_ReturnType
	Std_Types.h	Std_VersionInfoType

⌋(SRS_BSW_00392)

8.2 Type definitions

[SWS_MemIf_00010] ⌈The types specified in this chapter shall not be changed or extended for a specific memory abstraction module or hardware platform. ⌋ (SRS_MemHwAb_14022)

[SWS_MemIf_00011] ⌈The data type for the memory device index shall be uint8. The lowest value to be used for this device index shall be 0. The allowed range of indices thus shall be `0..MEMIF_NUMBER_OF_DEVICES-1`. ⌋ (SRS_MemHwAb_14020)

8.2.1 MemIf_StatusType

[SWS_MemIf_00064]⌈

Name	MemIf_StatusType		
Kind	Enumeration		
Range	MEMIF_UNINIT	--	The underlying abstraction module or device driver has not been initialized (yet).
	MEMIF_IDLE	--	The underlying abstraction module or device driver is currently idle.
	MEMIF_BUSY	--	The underlying abstraction module or device driver is currently busy.
	MEMIF_BUSY_INTERNAL	--	The underlying abstraction module is busy with internal management operations. The underlying device driver can be busy or idle.
Description	Denotes the current status of the underlying abstraction module and device drive.		

Available via	MemIf.h

](SRS_BSW_00392)

8.2.2 MemIf_JobResultType

[SWS_MemIf_00065][

Name	MemIf_JobResultType		
Kind	Enumeration		
Range	MEMIF_JOB_OK	--	The job has been finished successfully.
	MEMIF_JOB_FAILED	--	The job has not been finished successfully.
	MEMIF_JOB_PENDING	--	The job has not yet been finished.
	MEMIF_JOB_ CANCELED	--	The job has been canceled.
	MEMIF_BLOCK_ INCONSISTENT	--	1. The requested block is inconsistent, it may contain corrupted data. 2. Block is NOT found.
	MEMIF_BLOCK_ INVALID	--	The requested block has been marked as invalid, the requested operation can not be performed.
Description	Denotes the result of the last job.		
Available via	MemIf.h		

](SRS_BSW_00392)

8.2.3 MemIf_ModeType

[SWS_MemIf_00066][

Name	MemIf_ModeType		
Kind	Enumeration		
Range	MEMIF_MODE_ SLOW	--	The underlying memory abstraction modules and drivers are working in slow mode.
	MEMIF_MODE_ FAST	--	The underlying memory abstraction modules and drivers are working in fast mode.
Description	Denotes the operation mode of the underlying abstraction modules and device drivers.		
Available via	MemIf.h		

](SRS_BSW_00392)

8.3 Function definitions

[SWS_MemIf_00017] ⌈The API specified in this chapter shall be mapped to the API of the underlying memory abstraction modules. For functional behavior refer to the specification of those modules respectively to that of the underlying memory drivers. ⌋ (SRS_MemHwAb_14019, SRS_MemHwAb_14022)

[SWS_MemIf_00018] ⌈The parameter `DeviceIndex` shall be used for selection of memory abstraction modules (and thus memory devices). If only one memory abstraction module is configured, the parameter `DeviceIndex` shall be ignored. ⌋ (SRS_MemHwAb_14020, SRS_MemHwAb_14021)

[SWS_MemIf_00019] ⌈If only one memory abstraction module is configured, the Memory Abstraction Interface shall be implemented as a set of macros mapping the Memory Abstraction Interface API to the API of the corresponding memory abstraction module. ⌋ (SRS_SPAL_12078, SRS_MemHwAb_14021)

Example:
```
#define MemIf_Write(DeviceIndex, BlockNumber, DataPtr) \
      Fee_Write(BlockNumber, DataPtr)
```

[SWS_MemIf_00020] ⌈If more than one memory abstraction module is configured, the Memory Abstraction Interface shall use efficient mechanisms to map the API calls to the appropriate memory abstraction module. ⌋ (SRS_SPAL_12078, SRS_MemHwAb_14021)

Note: One solution is to use tables of pointers to functions where the parameter `DeviceIndex` is used as array index.

Example:
```
#define MemIf_Write(DeviceIndex, BlockNumber, DataPtr) \
      MemIf_WriteFctPtr[DeviceIndex](BlockNumber,DataPtr)
```

Note: The service IDs given in this interface specification are related to the service IDs of the underlying memory abstraction module(s). For that reason, they may not start with 0.

[SWS_MemIf_00022] ⌈If more than one memory abstraction module is configured and development error detection is enabled for this module, the functions of the Memory Abstraction Interface API shall check the parameter `DeviceIndex` for being an existing device or the broadcast identifier within the module's services. ⌋ (SRS_BSW_00323, SRS_MemHwAb_14021, SRS_MemHwAb_14023)

[SWS_MemIf_00023] ⌈The functions of the Memory Abstraction Interface API shall report detected errors attributed to an illegal parameter `DeviceIndex` to the Default Error Tracer (DET) with the error code `MEMIF_E_PARAM_DEVICE` and the called service shall not be executed. ⌋ (SRS_BSW_00386, SRS_SPAL_12448)

[SWS_MemIf_00024] ⌈If a called function of the Memory Abstraction Interface API has detected an error attributed to an illegal parameter `DeviceIndex` and has a return value, it shall be set as follows:

MemIf_GetStatus: `MEMIF_UNINIT`

MemIf_GetJobResult: `MEMIF_JOB_FAILED`
All other functions: `E_NOT_OK`.](SRS_BSW_00369)

8.3.1 MemIf_SetMode

[SWS_MemIf_00038][

Service Name	MemIf_SetMode	
Syntax	`void MemIf_SetMode ( MemIf_ModeType Mode )`	
Service ID [hex]	0x01	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	Mode	--
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Invokes the "SetMode" functions of all underlying memory abstraction modules.	
Available via	MemIf.h	

](RS_BRF_01472, SRS_MemHwAb_14022)
Note: The device index was intentionally left out in the above function, that is the Memory Interface shall switch all underlying modules into the requested mode. An extra “broadcast” parameter is not needed in this case since the devices shall not be switched to different modes individually.

8.3.2 MemIf_Read

[SWS_MemIf_00039][

Service Name	MemIf_Read
Syntax	`Std_ReturnType MemIf_Read ( uint8 DeviceIndex, uint16 BlockNumber, uint16 BlockOffset, uint8* DataBufferPtr, uint16 Length )`
Service ID [hex]	0x02
Sync/Async	Synchronous

Reentrancy	Non Reentrant	
Parameters (in)	Device Index	--
	Block Number	--
	Block Offset	--
	Length	--
Parameters (inout)	None	
Parameters (out)	Data BufferPtr	--
Return value	Std_-Return-Type	In case development error detection is enabled for the Memory Abstraction Interface and a development error is detected according to SWS_MemIf_00022 the function shall return E_NOT_OK else it shall return the value of the called function of the underlying module.
Description	Invokes the "Read" function of the underlying memory abstraction module selected by the parameter DeviceIndex.	
Available via	MemIf.h	

](SRS_MemHwAb_14029, SRS_MemHwAb_14022)

8.3.3 MemIf_Write

[SWS_MemIf_00040][

Service Name	MemIf_Write	
Syntax	`Std_ReturnType MemIf_Write ( uint8 DeviceIndex, uint16 BlockNumber, const uint8* DataBufferPtr )`	
Service ID [hex]	0x03	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	Device Index	--
	Block Number	--
	Data BufferPtr	--
Parameters	None	

(inout)		
Parameters (out)	None	
Return value	Std_-Return-Type	In case development error detection is enabled for the Memory Abstraction Interface and a development error is detected according to SWS_MemIf_00022 the function shall return E_NOT_OK else it shall return the value of the called function of the underlying module.
Description	Invokes the "Write" function of the underlying memory abstraction module selected by the parameter DeviceIndex.	
Available via	MemIf.h	

](SRS_MemHwAb_14010, SRS_MemHwAb_14022)

8.3.4 MemIf_Cancel

[SWS_MemIf_00041][

Service Name	MemIf_Cancel	
Syntax	`void MemIf_Cancel ( uint8 DeviceIndex )`	
Service ID [hex]	0x04	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	DeviceIndex	--
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Invokes the "Cancel" function of the underlying memory abstraction module selected by the parameter DeviceIndex.	
Available via	MemIf.h	

](SRS_MemHwAb_14031, SRS_MemHwAb_14022)

8.3.5 MemIf_GetStatus

[SWS_MemIf_00042][

Service Name	MemIf_GetStatus
Syntax	`MemIf_StatusType MemIf_GetStatus ( uint8 DeviceIndex )`

Service ID [hex]	0x05	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	DeviceIndex	--
Parameters (inout)	None	
Parameters (out)	None	
Return value	MemIf_StatusType	--
Description	Invokes the "GetStatus" function of the underlying memory abstraction module selected by the parameter DeviceIndex.	
Available via	MemIf.h	

](RS_BRF_02272, SRS_MemHwAb_14022)
[SWS_MemIf_00035] [If the function MemIf_GetStatus is called with the device index denoting a broadcast to all configured devices (`MEMIF_BROADCAST_ID`), the Memory Abstraction Interface module shall call the “GetStatus” functions of all underlying devices in turn. It shall return the value

- `MEMIF_IDLE` – if all underlying devices have returned this state
- `MEMIF_UNINIT` – if at least one device returned this state, all other returned states shall be ignored
- `MEMIF_BUSY` – if at least one configured device returned this state and no other device returned `MEMIF_UNINIT`
- `MEMIF_BUSY_INTERNAL` – if at least one configured device returned this state and no other device returned `MEMIF_BUSY` or `MEMIF_UNINIT`] (SRS_MemHwAb_14020)

Note: *The special “broadcast” device ID in the call to MemIf_GetStatus is used to query whether all devices are idle in order to shut down the ECU.*

8.3.6 MemIf_GetJobResult

[SWS_MemIf_00043][

Service Name	MemIf_GetJobResult
Syntax	`MemIf_JobResultType MemIf_GetJobResult ( uint8 DeviceIndex )`
Service ID [hex]	0x06
Sync/Async	Synchronous

Reentrancy	Non Reentrant	
Parameters (in)	Device Index	--
Parameters (inout)	None	
Parameters (out)	None	
Return value	MemIf_-Job-Result-Type	In case development error detection is enabled for the Memory Abstraction Interface and a development error is detected according to SWS_MemIf_00022 the function shall return MEMIF_JOB_FAILED else it shall return the value of the called function of the underlying module.
Description	Invokes the "GetJobResult" function of the underlying memory abstraction module selected by the parameter DeviceIndex.	
Available via	MemIf.h	

](SRS_MemHwAb_14022)

8.3.7 MemIf_InvalidateBlock

[SWS_MemIf_00044][

Service Name	MemIf_InvalidateBlock	

Syntax	`Std_ReturnType MemIf_InvalidateBlock (`
	`  uint8 DeviceIndex,`
	`  uint16 BlockNumber`
	`)`	

Service ID [hex]	0x07	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	Device Index	--
	Block Number	--
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_-Return-Type	In case development error detection is enabled for the Memory Abstraction Interface and a development error is detected according to SWS_MemIf_00022 the function shall return E_NOT_OK else it shall return the value of the called function of the underlying module.
Description	Invokes the "InvalidateBlock" function of the underlying memory abstraction module	

	selected by the parameter DeviceIndex.
Available via	MemIf.h

](SRS_MemHwAb_14028, SRS_MemHwAb_14022)

8.3.8 MemIf_GetVersionInfo

[SWS_MemIf_00045][

Service Name	MemIf_GetVersionInfo	
Syntax	`void MemIf_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	Returns version information.	
Available via	MemIf.h	

](SRS_BSW_00407)

8.3.9 MemIf_EraseImmediateBlock

[SWS_MemIf_00046][

Service Name	MemIf_EraseImmediateBlock	
Syntax	`Std_ReturnType MemIf_EraseImmediateBlock ( uint8 DeviceIndex, uint16 BlockNumber )`	
Service ID [hex]	0x09	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	Device Index	--

	Block Number	--
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_-Return-Type	In case development error detection is enabled for the Memory Abstraction Interface and a development error is detected according to SWS_MemIf_00022 the function shall return E_NOT_OK else it shall return the value of the called function of the underlying module.
Description	Invokes the "EraseImmediateBlock" function of the underlying memory abstraction module selected by the parameter DeviceIndex.	
Available via	MemIf.h	

](SRS_MemHwAb_14032, SRS_MemHwAb_14022)

8.4 Call-back notifications

None, the NVRAM manager shall provide the callback routines for the underlying memory abstraction modules.

8.5 Scheduled functions

None, there are no asynchronous functions in this module.

8.6 Expected Interfaces

8.6.1 Mandatory Interfaces

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

[SWS_MemIf_00047][

API Function	***Header File***	***Description***
Ea_Cancel	Ea.h	Cancels the ongoing asynchronous operation.
Ea_EraseImmediate-Block	Ea.h	Erases the block BlockNumber.
Ea_GetJobResult	Ea.h	Service to return the JobResult.
Ea_GetStatus	Ea.h	Service to return the Status.
Ea_InvalidateBlock	Ea.h	Invalidates the block BlockNumber.

Ea_Read	Ea.h	Reads Length bytes of block Blocknumber at offset BlockOffset into the buffer DataBufferPtr.
Ea_SetMode	Ea.h	Function to switch the mode of the underlying EEPROM Driver
Ea_Write	Ea.h	Writes the contents of the DataBufferPtr to the block Block Number.
Fee_Cancel	Fee.h	Service to call the cancel function of the underlying flash driver.
Fee_Erase-ImmediateBlock	Fee.h	Service to erase a logical block.
Fee_GetJobResult	Fee.h	Service to query the result of the last accepted job issued by the upper layer software.
Fee_GetStatus	Fee.h	Service to return the status.
Fee_InvalidateBlock	Fee.h	Service to invalidate a logical block.
Fee_Read	Fee.h	Service to initiate a read job.
Fee_SetMode	Fee.h	Function to switch the mode of the underlying Flash Driver
Fee_Write	Fee.h	Service to initiate a write job.

](SRS_BSW_00384)

8.6.2 Optional Interfaces

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

[SWS_MemIf_00048][

API Function	*Header File*	*Description*
Det_ReportError	Det.h	Service to report development errors.

](SRS_BSW_00385)

8.6.3 Configurable interfaces

In this chapter all interfaces are listed where the target function could be configured. The target function is usually a call-back function. The names of these kind of interfaces is not fixed because they are configurable.

There are no configurable interfaces for this module.

9 Sequence diagrams

Refer to the specifications of the memory abstraction modules.

10 Configuration specification

10.1 Containers and configuration parameters

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

10.1.1 MemIf

SWS Item	**ECUC_MemIf_00025 :**
Module Name	*MemIf*
Module Description	Configuration of the MemIf (Memory Abstraction Interface) module.
Post-Build Variant Support	false
Supported Config Variants	VARIANT-PRE-COMPILE

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
MemIfGeneral	1	Configuration of the memory abstraction interface (Memif) module.

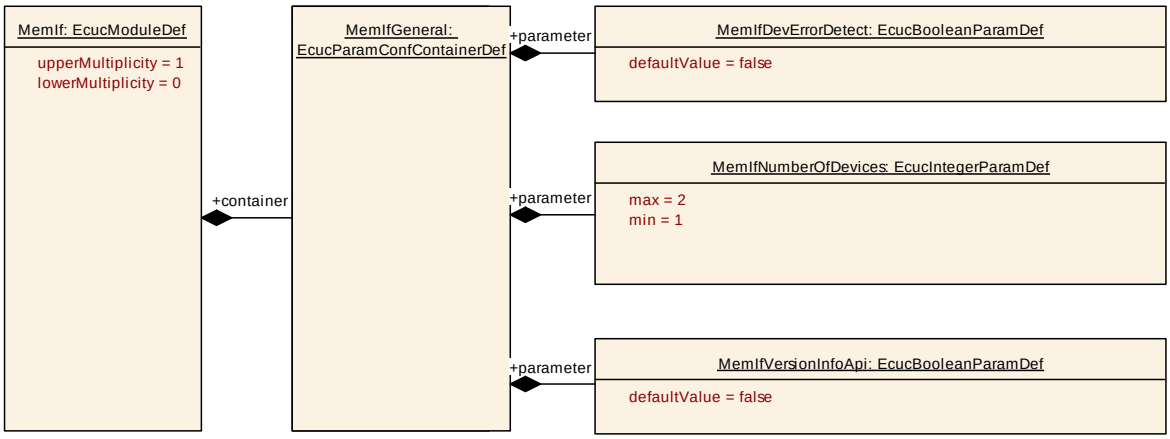

Figure 2: MemIf Configuration Layout

10.1.2 MemIfGeneral

SWS Item	**ECUC_MemIf_00034 :**
Container Name	MemIfGeneral
Parent Container	MemIf
Description	Configuration of the memory abstraction interface (Memif) module.
Configuration Parameters	

SWS Item	**ECUC_MemIf_00035 :**
Name	MemIfDevErrorDetect
Parent Container	MemIfGeneral
Description	Switches the development error detection and notification on or off. • true: detection and notification is enabled.

	• false: detection and notification is disabled.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_MemIf_00033 :**		
Name	MemIfNumberOfDevices		
Parent Container	MemIfGeneral		
Description	Concrete number of underlying memory abstraction modules. Calculation Formula: Count number of configured EA and FEE modules.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 2		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_MemIf_00032 :**		
Name	MemIfVersionInfoApi		
Parent Container	MemIfGeneral		
Description	Pre-processor switch to enable / disable the API to read out the modules version information. true: Version info API enabled. false: Version info API disabled.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers

11 Not applicable requirements

[SWS_MemIf_00999] ⌈These requirements are not applicable to this specification. ⌋ (SRS_BSW_00404, SRS_BSW_00405, SRS_BSW_00159, SRS_BSW_00170, SRS_BSW_00380, SRS_BSW_00412, SRS_BSW_00398, SRS_BSW_00399, SRS_BSW_00400, SRS_BSW_00375, SRS_BSW_00101, SRS_BSW_00416, SRS_BSW_00406, SRS_BSW_00168, SRS_BSW_00423, SRS_BSW_00424, SRS_BSW_00425, SRS_BSW_00426, SRS_BSW_00427, SRS_BSW_00428, SRS_BSW_00429, SRS_BSW_00432, SRS_BSW_00433, SRS_BSW_00336, SRS_BSW_00339, SRS_BSW_00422, SRS_BSW_00417, SRS_BSW_00161, SRS_BSW_00162, SRS_BSW_00005, SRS_BSW_00415, SRS_BSW_00164, SRS_BSW_00325, SRS_BSW_00342, SRS_BSW_00343, SRS_BSW_00160, SRS_BSW_00007, SRS_BSW_00300, SRS_BSW_00413, SRS_BSW_00347, SRS_BSW_00307, SRS_BSW_00373, SRS_BSW_00314, SRS_BSW_00348, SRS_BSW_00353, SRS_BSW_00361, SRS_BSW_00302, SRS_BSW_00328, SRS_BSW_00312, SRS_BSW_00006, SRS_BSW_00304, SRS_BSW_00378, SRS_BSW_00306, SRS_BSW_00308, SRS_BSW_00309, SRS_BSW_00371, SRS_BSW_00358, SRS_BSW_00414, SRS_BSW_00359, SRS_BSW_00360, SRS_BSW_00330, SRS_BSW_00009, SRS_BSW_00401, SRS_BSW_00172, SRS_BSW_00010, SRS_BSW_00333, SRS_BSW_00321, SRS_BSW_00341, SRS_BSW_00334, SRS_SPAL_12263, SRS_SPAL_12056, SRS_SPAL_12267, SRS_SPAL_12057, SRS_SPAL_12125, SRS_SPAL_12163, SRS_SPAL_12461, SRS_SPAL_12462, SRS_SPAL_12463, SRS_SPAL_12068, SRS_SPAL_12069, SRS_SPAL_00157, SRS_SPAL_12063, SRS_SPAL_12075, SRS_SPAL_12129, SRS_SPAL_12064, SRS_SPAL_12067, SRS_SPAL_12077, SRS_SPAL_12092, SRS_SPAL_12265)