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	R24-11

Document Change History

Date	Release	Changed by	Description
2024-11-27	R24-11	AUTOSAR Release Management	• Editorial changes
2023-11-23	R23-11	AUTOSAR Release Management	• Updated SWS_MemIf_00047 • Removed Obsolete status of SWS_MemIf_00065 • Editorial changes
2022-11-24	R22-11	AUTOSAR Release Management	• Changed SWS_MemIf_00999 to SWS_MemIf_NA_00999
2021-11-25	R21-11	AUTOSAR Release Management	• Improve the structure of the 'error sections' • Cleanup diagrams in chapter 10
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
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
2006-05-16	2.0.0	AUTOSAR Administration	• Initial Release

Disclaimer

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 [1] NVRAM manager to access several memory abstraction modules (FEE or EA modules) (see Figure 1.1).

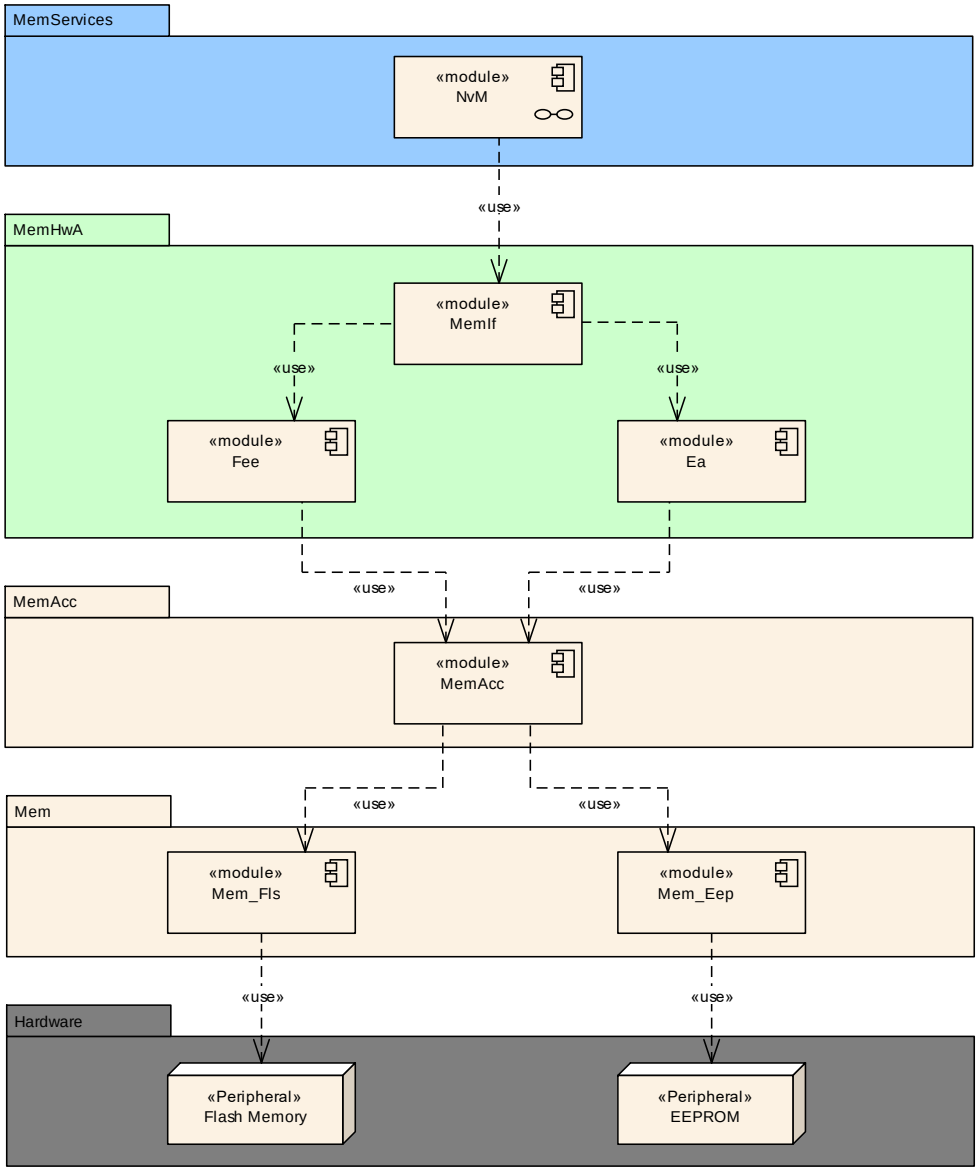

Figure 1.1: Module overview of memory hardware abstraction layer

The Memory Abstraction Interface (MemIf) shall abstract from the number of underlying [2] FEE or [3] EA modules and provide upper layers with a virtual segmentation on a uniform linear address space.

2 Acronyms and Abbreviations

The glossary below includes acronyms and abbreviations relevant to the MemIf module that are not included in the [4, AUTOSAR 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.
Mem	AUTOSAR Basic Software Module Memory Driver
MemAcc	AUTOSAR Basic Software Module Memory Access
MemIf	Memory Abstraction Interface
MSB	Most significant bit / byte (depending on context). Here it's bit.
NvM	NVRAM Manager
NVRAM	Non-volatile RAM (Random Access Memory)
Address area	Contiguous memory area in the logical address space. Typically, multiple physical memory sectors are combined to one logical address area.
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.

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

3 Related documentation

3.1 Input documents & related standards and norms

[1] Specification of NVRAM Manager
AUTOSAR_CP_SWS_NVRAMManager

[2] Specification of Flash EEPROM Emulation
AUTOSAR_CP_SWS_FlashEEPROMEmulation

[3] Specification of EEPROM Abstraction
AUTOSAR_CP_SWS_EEPROMAbstraction

[4] Glossary
AUTOSAR_FO_TR_Glossary

[5] General Specification of Basic Software Modules
AUTOSAR_CP_SWS_BSWGeneral

[6] General Requirements on SPAL
AUTOSAR_CP_RS_SPALGeneral

[7] General Requirements on Basic Software Modules
AUTOSAR_CP_RS_BSWGeneral

[8] Requirements on Memory Hardware Abstraction Layer
AUTOSAR_CP_RS_MemoryHWAbstractionLayer

3.2 Related specification

AUTOSAR provides a General Specification on Basic Software modules [5, SWS BSW General], which is also valid for Memory Abstraction Interface.

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

4 Constraints and assumptions

4.1 Limitations

No limitations.

4.2 Applicability to car domains

No restrictions.

5 Dependencies to other modules

6 Requirements Tracing

The following tables reference the requirements specified in [6, SRS SPALGeneral], [7, SRS BSWGeneral], [8, SRS MemoryHWAbstraction] and links to the fulfillment of these. Please note that if column ”Satisfied by” is empty for a specific requirement this means that this requirement is not fulfilled by this document.

Requirement	Description	Satisfied by
[RS_BRF_02272]	AUTOSAR shall offer tracing of application software behavior	[SWS_MemIf_00042]
[SRS_BSW_00323]	All AUTOSAR Basic Software Modules shall check passed API parameters for validity	[SWS_MemIf_00022]
[SRS_BSW_00327]	Error values naming convention	[SWS_MemIf_00006]
[SRS_BSW_00337]	Classification of development errors	[SWS_MemIf_00006]
[SRS_BSW_00369]	All AUTOSAR Basic Software Modules shall not return specific development error codes via the API	[SWS_MemIf_00024]
[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 and conditions 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]
[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_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_00039] [SWS_MemIf_00040] [SWS_MemIf_00041] [SWS_MemIf_00042] [SWS_MemIf_00043] [SWS_MemIf_00044] [SWS_MemIf_00046]

▽

△

Requirement	Description	Satisfied by
[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 logical block	[SWS_MemIf_00039]
[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_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_12448]	All driver modules shall have a specific behavior after a development error detection	[SWS_MemIf_00023]

Table 6.1: Requirements Tracing

7 Functional specification

7.1 Error Classification

Section "Error Handling" of the document [5] "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] Definiton of development errors in module MemIf

Upstream requirements: SRS_BSW_00337, SRS_BSW_00386, SRS_BSW_00327

⌈

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

⌋

7.1.2 Runtime Errors

There are no runtime errors.

7.1.3 Production Errors

There are no production errors.

7.1.4 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] Definition of imported datatypes of module MemIf

Upstream requirements: SRS_BSW_00392

⌈

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

⌋

8.2 Type definitions

[SWS_MemIf_00010]

Upstream requirements: SRS_MemHwAb_14022

⌈The types specified in this chapter shall not be changed or extended for a specific memory abstraction module or hardware platform.⌋

[SWS_MemIf_00011]

Upstream requirements: SRS_MemHwAb_14020

⌈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.⌋

8.2.1 MemIf_StatusType

[SWS_MemIf_00064] Definition of datatype MemIf_StatusType

Upstream requirements: SRS_BSW_00392

⌈

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		

⌋

8.2.2 MemIf_JobResultType

[SWS_MemIf_00065] Definition of datatype MemIf_JobResultType

Upstream requirements: SRS_BSW_00392

⌈

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		

⌋

8.3 Function definitions

[SWS_MemIf_00017]

Upstream requirements: SRS_MemHwAb_14019, SRS_MemHwAb_14022

⌈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.⌋

[SWS_MemIf_00018]

Upstream requirements: SRS_MemHwAb_14020, SRS_MemHwAb_14021

⌈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.⌋

[SWS_MemIf_00019]

Upstream requirements: SRS_SPAL_12078, SRS_MemHwAb_14021

⌈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.⌋

Example:

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

[SWS_MemIf_00020]

Upstream requirements: SRS_SPAL_12078, SRS_MemHwAb_14021

⌈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.⌋

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]

Upstream requirements: SRS_BSW_00323, SRS_MemHwAb_14021, SRS_MemHwAb_14023

⌈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.⌋

[SWS_MemIf_00023]

Upstream requirements: SRS_BSW_00386, SRS_SPAL_12448

⌈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.⌋

[SWS_MemIf_00024]

Upstream requirements: SRS_BSW_00369

⌈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`⌋

8.3.1 MemIf_Read

[SWS_MemIf_00039] Definition of API function MemIf_Read

Upstream requirements: SRS_MemHwAb_14029, SRS_MemHwAb_14022

⌈

Service Name	MemIf_Read

Syntax	`Std_ReturnType MemIf_Read (`
	`  uint16 DeviceIndex,`
	`  uint16 BlockNumber,`
	`  uint16 BlockOffset,`
	`  uint8* DataBufferPtr,`
	`  uint16 Length`
	`)`

Service ID [hex]	0x02
Sync/Async	Synchronous

▽

△

Reentrancy	Non Reentrant	
Parameters (in)	DeviceIndex	–
	BlockNumber	–
	BlockOffset	–
	Length	–
Parameters (inout)	None	
Parameters (out)	DataBufferPtr	–
Return value	Std_ReturnType	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	

⌋

8.3.2 MemIf_Write

[SWS_MemIf_00040] Definition of API function MemIf_Write

Upstream requirements: SRS_MemHwAb_14010, SRS_MemHwAb_14022

⌈

Service Name	MemIf_Write	
Syntax	`Std_ReturnType MemIf_Write ( uint16 DeviceIndex, uint16 BlockNumber, const uint8* DataBufferPtr )`	
Service ID [hex]	0x03	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	DeviceIndex	–
	BlockNumber	–
	DataBufferPtr	–
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	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	

⌋

8.3.3 MemIf_Cancel

[SWS_MemIf_00041] Definition of API function MemIf_Cancel

Upstream requirements: SRS_MemHwAb_14031, SRS_MemHwAb_14022

⌈

Service Name	MemIf_Cancel	
Syntax	`void MemIf_Cancel ( uint16 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	

⌋

8.3.4 MemIf_GetStatus

[SWS_MemIf_00042] Definition of API function MemIf_GetStatus

Upstream requirements: RS_BRF_02272, SRS_MemHwAb_14022

⌈

Service Name	MemIf_GetStatus	
Syntax	`MemIf_StatusType MemIf_GetStatus ( uint16 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	

⌋

[SWS_MemIf_00035]

Upstream requirements: SRS_MemHwAb_14020

⌈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

⌋

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.*

8.3.5 MemIf_GetJobResult

[SWS_MemIf_00043] Definition of API function MemIf_GetJobResult

Upstream requirements: SRS_MemHwAb_14022

⌈

Service Name	MemIf_GetJobResult	
Syntax	`MemIf_JobResultType MemIf_GetJobResult ( uint16 DeviceIndex )`	
Service ID [hex]	0x06	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	DeviceIndex	–
Parameters (inout)	None	
Parameters (out)	None	
Return value	MemIf_JobResultType	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

⌋

8.3.6 MemIf_InvalidateBlock

[SWS_MemIf_00044] Definition of API function MemIf_InvalidateBlock

Upstream requirements: SRS_MemHwAb_14028, SRS_MemHwAb_14022

⌈

Service Name	MemIf_InvalidateBlock	
Syntax	`Std_ReturnType MemIf_InvalidateBlock ( uint16 DeviceIndex, uint16 BlockNumber )`	
Service ID [hex]	0x07	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	DeviceIndex	–
	BlockNumber	–
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	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	

⌋

8.3.7 MemIf_GetVersionInfo

[SWS_MemIf_00045] Definition of API function MemIf_GetVersionInfo

Upstream requirements: SRS_BSW_00407

⌈

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	

⌋

8.3.8 MemIf_EraseImmediateBlock

[SWS_MemIf_00046] Definition of API function MemIf_EraseImmediateBlock

Upstream requirements: SRS_MemHwAb_14032, SRS_MemHwAb_14022

⌈

Service Name	MemIf_EraseImmediateBlock	

Syntax	`Std_ReturnType MemIf_EraseImmediateBlock (`
	`  uint16 DeviceIndex,`
	`  uint16 BlockNumber`
	`)`	

Service ID [hex]	0x09	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	DeviceIndex	–
	BlockNumber	–
Parameters (inout)	None	
Parameters (out)	None	

▽

△

Return value	Std_ReturnType	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	

⌋

8.4 Callback 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] Definition of mandatory interfaces required by module Mem If

Upstream requirements: SRS_BSW_00384

⌈

API Function	***Header File***	***Description***
Ea_EraseImmediateBlock	Ea.h	Erases the block BlockNumber.
Ea_GetStatus	Ea.h	Service to return the Status.
Ea_InvalidateBlock	Ea.h	Invalidates the block BlockNumber.
Fee_EraseImmediateBlock	Fee.h	Service to erase a logical block.
Fee_GetStatus	Fee.h	Service to return the status.
Fee_InvalidateBlock	Fee.h	Service to invalidate a logical block.

⌋

8.6.2 Optional Interfaces

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

[SWS_MemIf_00048] Definition of optional interfaces requested by module Mem If

Upstream requirements: SRS_BSW_00385

⌈

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

⌋

8.6.3 Configurable interfaces

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

[ECUC_MemIf_00025] Definition of EcucModuleDef MemIf ⌈

Module Name	MemIf
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.

⌋

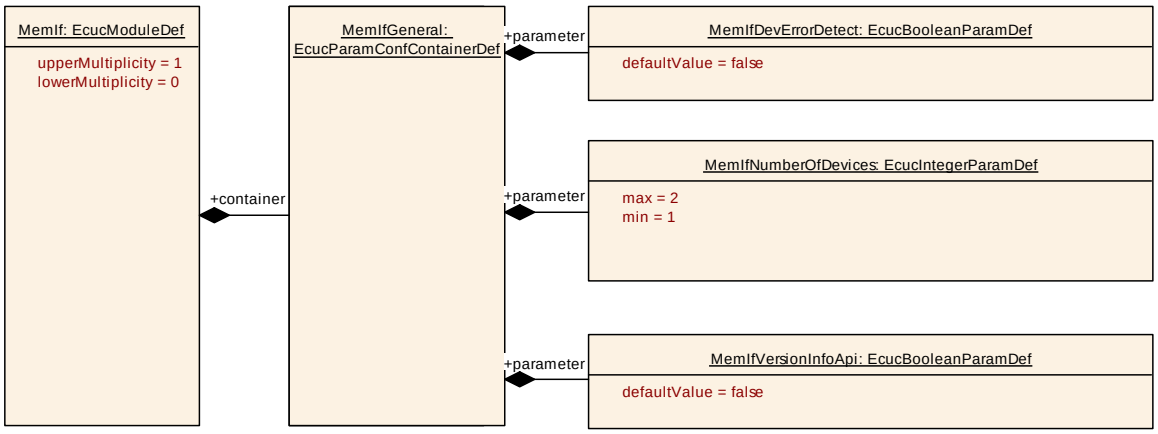

Figure 10.1: Configuration of the MemIf

10.1.2 MemIfGeneral

[ECUC_MemIf_00034] Definition of EcucParamConfContainerDef MemIfGeneral ⌈

Container Name	MemIfGeneral
Parent Container	MemIf
Description	Configuration of the memory abstraction interface (Memif) module.
Configuration Parameters	

Included Parameters		
Parameter Name	Multiplicity	ECUC ID
MemIfDevErrorDetect	1	[ECUC_MemIf_00035]
MemIfNumberOfDevices	1	[ECUC_MemIf_00033]
MemIfVersionInfoApi	1	[ECUC_MemIf_00032]

No Included Containers

⌋

[ECUC_MemIf_00035] Definition of EcucBooleanParamDef MemIfDevErrorDetect ⌈

Parameter 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		

⌋

[ECUC_MemIf_00033] Definition of EcucIntegerParamDef MemIfNumberOfDevices ⌈

Parameter 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		

⌋

[ECUC_MemIf_00032] Definition of EcucBooleanParamDef MemIfVersionInfoApi

⌈

Parameter 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		

⌋

A Change history of AUTOSAR traceable items

A.1 Traceable item history of this document according to AUTOSAR Release R24-11

A.1.1 Added Specification Items in R24-11

none

A.1.2 Changed Specification Items in R24-11

none

A.1.3 Deleted Specification Items in R24-11

none

B Not applicable requirements

[SWS_MemIf_NA_00999]

Upstream requirements: SRS_BSW_00404, SRS_BSW_00101, SRS_BSW_00159, SRS_BSW_00168, SRS_BSW_00170, SRS_BSW_00330, SRS_BSW_00336, SRS_BSW_00339, SRS_BSW_00343, SRS_BSW_00375, SRS_BSW_00380, SRS_BSW_00398, SRS_BSW_00399, SRS_BSW_00400, SRS_BSW_00405, SRS_BSW_00406, SRS_BSW_00416, SRS_BSW_00417, SRS_BSW_00422, 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_SPAL_00157, SRS_SPAL_12056, SRS_SPAL_12057, SRS_SPAL_12063, SRS_SPAL_12064, SRS_SPAL_12067, SRS_SPAL_12068, SRS_SPAL_12069, SRS_SPAL_12075, SRS_SPAL_12077, SRS_SPAL_12092, SRS_SPAL_12125, SRS_SPAL_12129, SRS_SPAL_12163, SRS_SPAL_12263, SRS_SPAL_12265, SRS_SPAL_12267, SRS_SPAL_12461, SRS_SPAL_12462, SRS_SPAL_12463

⌈These requirements are not applicable to this specification.⌋