Document Title	Specification of Module Memory Abstraction Interface
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	285
Document Classification	Standard

Document Version	1.4.0
Document Status	Final
Part of Release	3.2
Revision	3

Document Change History

Date	Version	Changed by	Change Description
28.02.2014	1.4.0	AUTOSAR Release Management	• Const qualifier added to MemIf_Write prototype (MemIf040) • Duplicate requirement IDs split up, new requirement IDs added (ECUC_MemIf_00032, ECUC_MemIf_00033, ECUC_MemIf_00034, ECUC_MemIf_00035) • Editorial changes
27.04.2011	1.3.1	AUTOSAR Administration	• Update chapter 8 and 11 • Legal disclaimer revised
23.06.2008	1.2.1	AUTOSAR Administration	Legal disclaimer revised
13.11.2007	1.2.0	AUTOSAR Administration	• Broadcast identifier properly explained • 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 • Return types of various APIs adapted • Ranges of configuration parameters adjusted • Legal disclaimer revised • Release Notes added • “Advice for users” revised • “Revision Information” added
27.04.2006	1.0.0	AUTOSAR Administration	Initial Release

Disclaimer

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 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).

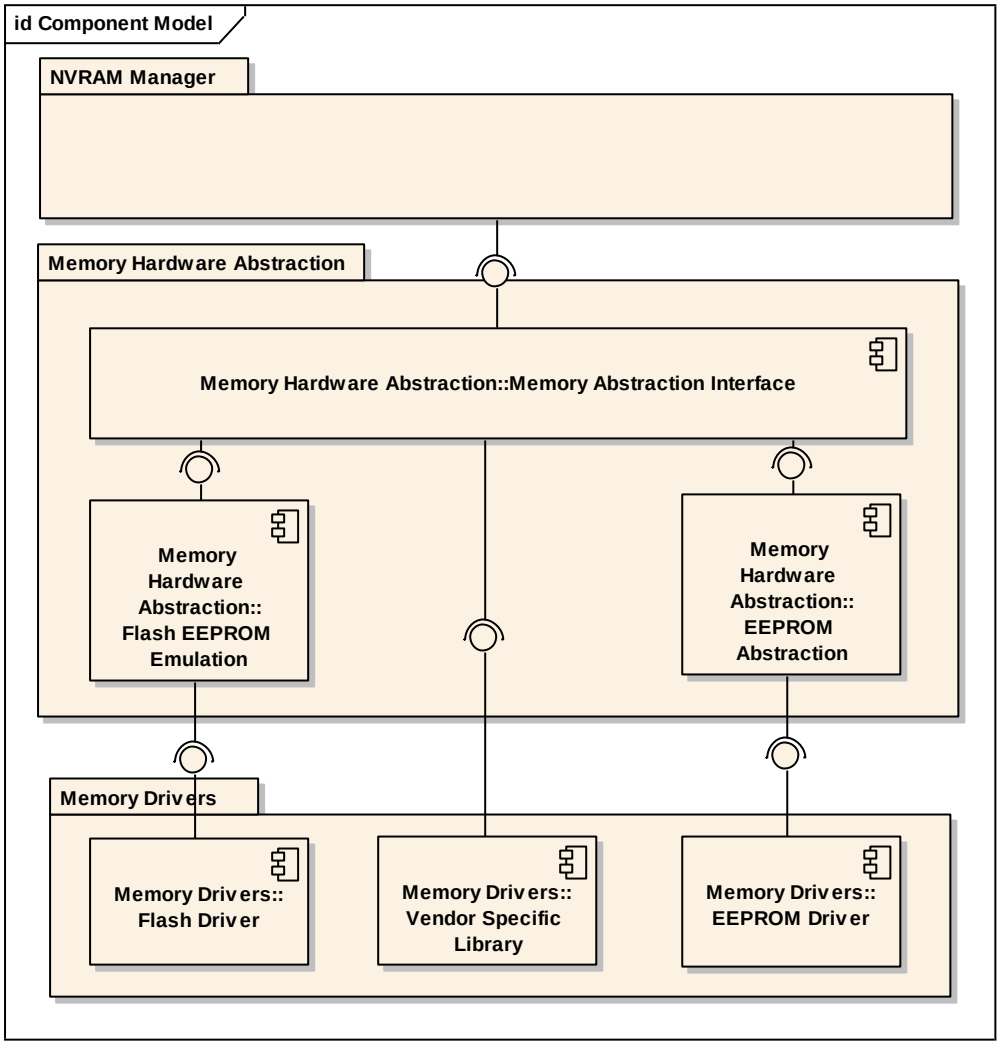

Figure 1: Module overview of memory hardware abstraction layer

MemIf001: 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.*

3 Related documentation

3.1 Input documents

[1] List of Basic Software Modules
AUTOSAR_BasicSoftwareModules.pdf

[2] Layered Software Architecture
AUTOSAR_LayeredSoftwareArchitecture.pdf

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

[4] General Requirements on SPAL
AUTOSAR_SRS_SPAL_General.pdf

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

[6] Specification of Development Error Tracer
AUTOSAR_SWS_DET.pdf

3.2 Related standards and norms

[7] AUTOSAR Specification of NVRAM Manager
AUTOSAR_SWS_NVRAM_Manager.doc

[8] Specification of Flash EEPROM Emulation
AUTOSAR_SWS_Flash_EEPROM_Emulation.pdf

[9] Specification of EEPROM Abstraction
AUTOSAR_SWS_EEPROM_Abstraction.pdf

[10] AUTOSAR Basic Software Module Description Template,
AUTOSAR_BSW_Module_Description.pdf

4 Constraints and assumptions

4.1 Limitations

No limitations.

4.2 Applicability to car domains

No restrictions.

5 Dependencies to other modules

5.1 File structure

5.1.1 Code file structure

MemIf033: The code file structure shall not be defined within this specification.

5.2 Header file structure

MemIf002: The file include structure shall be as follows:

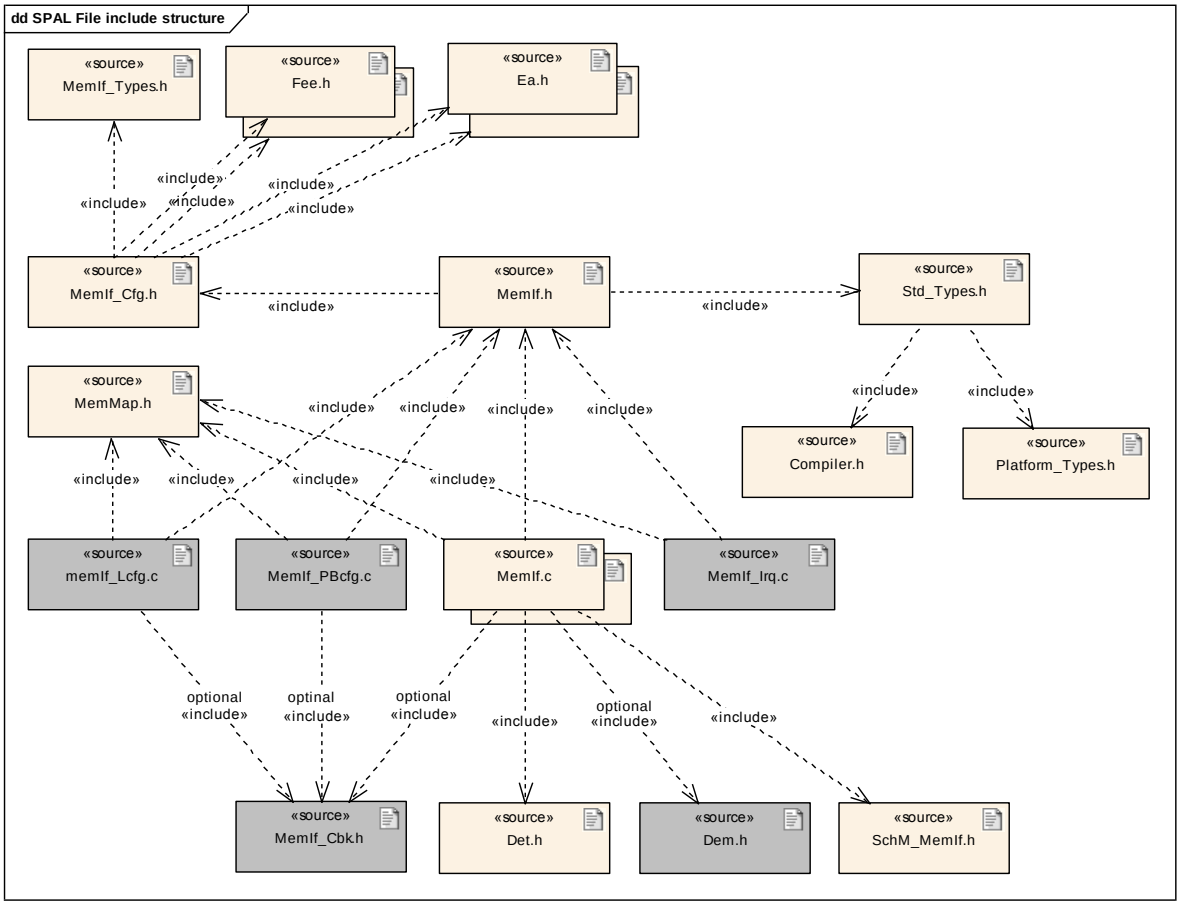

Figure 2: Memory Abstraction Layer File Include Structure

- `MemIf_Cfg.h` shall include MemIf_Types.h and the header files of all underlying memory abstraction modules (FEE and EA modules)
- `MemIf.h` shall include Std_Types.h and `MemIf_Cfg.h`
- Only `MemIf.h` shall be included by the upper layer modules
- `MemIf.c` (if implemented) shall include `MemIf.h, MemMap.h and other standard header files (if needed by the implementation)..`

- `Fee_x.h` shall include `MemIf_Types.h` and the header file of the underlying flash driver
- `Ea_y.h` shall include `MemIf_Types.h` and the header file of the underlying EEPROM driver

MemIf034: The 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

Document: General Requirements on Basic Software Modules

Requirement	*Satisfied by*
[BSW00344] Reference to link-time configuration	Not applicable (this module does not provide link-time configuration)
[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	MemIf025
[BSW159] Tool-based configuration	Not applicable (requirement on configuration, not for the specification)
[BSW167] Static configuration checking	MemIf005, MemIf025
[BSW171] Configurability of optional functionality	MemIf032
[BSW170] Data for reconfiguration of AUTOSAR SW-Components	Not applicable (requirement for SW-C)
[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	MemIf002
[BSW00412] Separate H-File for configuration parameters [approved]	Not applicable (no link-time or post build time configuration parameters)
[BSW00383] List dependencies of configuration files	MemIf002
[BSW00384] List dependencies to other modules	Chapter 5
[BSW00387] Specify the configuration class of callback function	Chapter 8.6
[BSW00388] Introduce containers	Chapter 10.1
[BSW00389] Containers shall have names	Chapter 10.1
[BSW00390] Parameter content shall be unique within the module	Chapter 10.2
[BSW00391] Parameter shall have unique names	Chapter 10.2
[BSW00392] Parameters shall have a type	Chapter 10.2
[BSW00393] Parameters shall have a range	Chapter 10.2
[BSW00394] Specify the scope of the parameters	Chapter 10.2
[BSW00395] List the required parameters (per parameter)	Chapter 10.2
[BSW00396] Configuration classes	Chapter 10.2
[BSW00397] Pre-compile-time parameters	Chapter 10.2
[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 10.2
[BSW00375] Notification of wake-up reason	Not applicable (this module does not provide wakeup capabilities)

[BSW101] Initialization interface	Not applicable (this module does not need an initialization)
[BSW00416] Sequence of Initialization	Not applicable (requirement on system design, not a single module)
[BSW00406] Check module initialization	Not applicable (this module does not need an initialization)
[BSW168] Diagnostic Interface of SW components	Not applicable (this module does not provide special diagnostic features)
[BSW00407] Function to read out published parameters	Chapter 8.3.8, MemIf026
[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 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 implements task bodies	Not applicable (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 schedule module – this is not it)
[BSW00336] Shutdown interface	Not applicable (this module does not need to be shut down)
[BSW00337] Classification of errors	MemIf006
[BSW00338] Detection and Reporting of development errors	MemIf007, MemIf028
[BSW00369] Do not return development error codes via API	MemIf028
[BSW00339] Reporting of production relevant error status	Not applicable (this module does not know any production relevant errors)
[BSW00421] Reporting of production relevant error events	Not applicable (no production relevant errors defined for this module)
[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	MemIf022
[BSW004] Version check	MemIf005
[BSW00409] Header files for production code error IDs	MemIf029
[BSW00385] List possible error notificatons	Chapter 8.6
[BSW00386] Configuration for detecting an error	MemIf006, MemIf007, MemIf023, MemIf028
[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 (architecture decision)
[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 implement any ISRs)
[BSW00325] Runtime of interrupt service routines	Not applicable (this module does not implement any ISRs or callback routines)
[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 time	Not applicable (this module does not provide any timing configuration)
[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	Chapter 8.2
[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	Not applicable (this module does not provide a scheduled function)
[BSW00327] Error values naming convention	MemIf006, MemIf008
[BSW00335] Status values naming convention	Chapter 8.2.1

[BSW00350] Development error detection keyword	MemIf007, MemIf028, MemIf025
[BSW00408] Configuration parameter naming convention	Chapter 10.1
[BSW00410] Compiler switches shall have defined values	Chapter 10.1
[BSW00411] Get version info keyword	Chapter 10.2
[BSW00346] Basic set of module files	MemIf002
[BSW158] Separation of configuration from implementation	MemIf002
[BSW00314] Separation of interrupt frames and service routines	Not applicable (this module does not implement any ISRs)
[BSW00370] Separation of callback interface from API	Not applicable (this module does not implement any callback routines)
[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	MemIf002
[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 8.3.2, Chapter 8.3.3. Chapter 8.3.7, Chapter 8.3.9
[BSW00377] Module specific API return types	Chapter 8.3.5, Chapter 8.3.6
[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 pecifi 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	Not applicable (this module does not provide an initialization function)
[BSW00414] Parameter of init function	Not applicable (this module does not provide an initialization function)
[BSW00376] Return type and parameters of main processing functions	Not applicable (this module does not provide a scheduled function)
[BSW00359] Return type of callback functions	Not applicable (this module does not provide any callback routines)
[BSW00360] Parameters of callback functions	Not applicable (this module does not provide any callback routines)
[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	MemIf028
[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 pecification)
[BSW00374] Module vendor identification	MemIf026
[BSW00379] Module identification	MemIf026
[BSW003] Version identification	MemIf026
[BSW00318] Format of module version numbers	MemIf026
[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 post-compile time parameters)
[BSW12056] Configuration of notification mechanisms	Not applicable (this module does not support any notification mechanisms)
[BSW12267] Configuration of wake-up sources	Not applicable (this module does not provide any wakeup capabilities)
[BSW12057] Driver module initialization	Not applicable (this module does not provide an initialization routine)
[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 an de-initialization routine)
[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 [approved]	Not applicable (this module has no direct hardware access)
[BSW12462] Provide settings for register initialization [approved]	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 provide an initialization routine)
[BSW12068] MCAL initialization sequence	Not applicable (this module does not provide an initialization routine)
[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 support any notification mechanisms)
[BSW12155] Prototypes of callback functions	Not applicable (this module does not provide any callback routines)
[BSW12169] Control of operation mode	Chapter 8.3.1
[BSW12063] Raw value mode	Not applicable (this module does not handle any data)
[BSW12075] Use of application buffers	Not applicable (this module does not handle any data)
[BSW12129] Resetting of interrupt flags	Not applicable (this module does not implement any ISRs)
[BSW12064] Change of operation mode during running operation	Not applicable (this module is only an interface for underlying modules)
[BSW12448] Behavior after development error detection	MemIf023

[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 provide any schedulable routines)
[BSW12078] Runtime and memory efficiency	MemIf019, MemIf020MAI018
[BSW12092] Access to drivers	Not applicable (requirement on system architecture not for one module)
[BSW12265] Configuration data shall be kept constant	Not applicable (this module does not have post-compile time configuration data)
[BSW12264] Specification of configuration items	MemIf025, MemIf026
[BSW12081] Use HIS requirements as input	Not applicable (no corresponding HIS requirements available)

Document: Requirements on Memory Abstraction Interface

Requirement	***Satisfied by***
BSW14019 Provide uniform access to underlying memory abstraction modules	MemIf001, MemIf017
BSW14020 Selection of underlying memory abstraction modules	MemIf018
BSW14021 Number of underlying memory abstraction modules	MemIf018, MemIf019, MemIf020, MemIf022, MemIf025
BSW14022 Preserving of functionality	MemIf017
BSW14023 Parameter checking	MemIf005, MemIf022
BSW14024 Preserving of timing behavior	Not applicable (requirement removed, see RfC 14746)MemIf004
BSW14025 Efficient implementation	MemIf019, MemIf020

7 Functional specification

7.1 General behavior

MemIf005: All pre-compile time configuration parameters shall be checked statically (at least during compile time) for correctness. The version information in the module headers and source files shall be validated and consistent (e.g. by comparing the version information in the module headers and source files with a pre-processor macro).

7.2 Error classification

MemIf029: 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.

MemIf030: Development error values are of type uint8.

MemIf006: The following errors and exceptions shall be detectable by the Memory Abstraction Interface depending on its configuration (development/production):

Type or error	*Relevance*	*Related error code*	*Value [hex]*
API service called with wrong device index parameter	Development	`MEMIF_E_PARAM_DEVICE`	`0x01`

7.3 Error detection

MemIf007: The detection of development errors shall be configurable (on/off) at pre-compile time. The switch `MemIfDevErrorDetect` (see chapter 10) shall activate or deactivate the detection of all development errors.

MemIf008: A detection of errors not listed in the table above [MemIf006] shall not be implemented.

7.4 Error notification

MemIf028: Development errors shall be reported to the Development Error Tracer (DET) if the preprocessor switch `MemIfDevErrorDetect` is set. The error codes shall not be used as return values for the called function.

8 API specification

8.1 Imported types

8.1.1 Standard types

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

MemIf037:

Module	*Imported Type*
Std_Types	Std_ReturnType
	Std_VersionInfoType

8.2 Type definitions

MemIf009: The types specified in this chapter shall be located in the file MemIf_Types.h.

MemIf010: The types specified in this chapter shall not be changed or extended for a specific memory abstraction module or hardware platform.

MemIf011: 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..MemIfNumberOfDevices-1`.

MemIf036: The symbolic name `MEMIF_BROADCAST_ID` shall be used to identify all underlying devices within one call. This special “broadcast” device ID shall only be allowed in the call to MemIf_GetStatus to determine the status of all underlying abstraction modules and device drivers[1].

8.2.1 MemIf_StatusType

Name:	`MemIf_StatusType`	
Type:	`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.

[1] I.e. used to query whether all devices are idle in order to shut down the ECU.

Description:	Denotes the current status of the underlying abstraction module and device drive.

MemIf015: The type MemIf_StatusType denotes the current status of the underlying abstraction module and device driver. It shall be used as the return value of the corresponding driver's "GetStatus" function.

8.2.2 MemIf_JobResultType

Name:	MemIf_JobResultType	
Type:	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_CANCELLED	The job has been cancelled.
	MEMIF_BLOCK_INCONSISTENT	The requested block is inconsistent, it may contain corrupted data.
	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.	

MemIf016: The type MemIf_JobResultType denotes the result of the last job.

8.2.3 MemIf_ModeType

Name:	MemIf_ModeType	
Type:	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.	

MemIf021: The type MemIf_ModeType denotes the operation mode of the underlying abstraction modules and device drivers.

8.3 Function definitions

MemIf017: 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.

MemIf018: 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.

MemIf019: 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)
```

MemIf020: 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. 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.

MemIf022: If more than one memory abstraction module is configured and development error detection is enabled for this module, the parameter `DeviceIndex` shall be checked for being an existing device or the broadcast identifier within the module's services.

MemIf023: Detected errors shall be reported to the Development Error Tracer (DET) with the error code `MEMIF_E_PARAM_DEVICE` and the called service shall not be executed.

MemIf024: If the called function 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`.

MemIf035: If the function MemIf_GetStatus is called with the device index denoting a broadcast (`MEMIF_BROADCAST_ID`) to all configured devices (see MemIf036), this 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`

8.3.1 MemIf_SetMode

MemIf038:

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:	map function calls of MemIf_SetMode to service: Fee_SetMode respectively Ea_SetMode

MemIf049: The function MemIf_SetMode shall be mapped to service: Fee_SetMode respectively Ea_SetMode.

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

MemIf039:

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):	DeviceIndex	--
	BlockNumber	--
	BlockOffset	--
	Length	--
Parameters (inout):	None	
Parameters (out):	DataBufferPtr	--
Return value:	Std_ReturnType	--
Description:	map function calls of MemIf_Read to service: Fee_Read respectively Ea_Read	

MemIf050: The function MemIf_Read shall be mapped to service: Fee_Read respectively Ea_Read.

8.3.3 MemIf_Write

MemIf040:

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):	DeviceIndex	--
	BlockNumber	--
	DataBufferPtr	--
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	--
Description:	map function calls of MemIf_Write to service: Fee_Write respectively Ea_Write	

MemIf051: The function MemIf_Write shall be mapped to service: Fee_Write respectively Ea_Write.

8.3.4 MemIf_Cancel

MemIf041:

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:	map function calls of MemIf_Cancel to service: Fee_Cancel respectively Ea_Cancel	

MemIf052: The function MemIf_Cancel shall be mapped to service: Fee_Cancel respectively Ea_Cancel.

8.3.5 MemIf_GetStatus

MemIf042:

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:	map function calls of MemIf_GetStatus to service: Fee_GetStatus respectively Ea_GetStatus	

MemIf056: The function MemIf_GetStatus shall be mapped to service: Fee_GetStatus respectively Ea_GetStatus

Note: In case the parameter given as device ID is MEMIF_BROADCAST_ID, the memory abstraction interface shall iterate over all underlying devices and return their combined statusn according to MemIf035.

8.3.6 MemIf_GetJobResult

MemIf043:

Service name:	MemIf_GetJobResult	

Syntax:	`MemIf_JobResultType MemIf_GetJobResult(`
	`  uint8 DeviceIndex`
	`)`	

Service ID[hex]:	0x06	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	DeviceIndex	--
Parameters (inout):	None	
Parameters (out):	None	
Return value:	MemIf_JobResultType	--
Description:	map function calls of MemIf_GetJobResult to service: Fee_GetJobResult respectively Ea_GetJobResult	

MemIf053: The function MemIf_GetJobResult shall be mapped to service: Fee_GetJobResult respectively Ea_GetJobResult.

8.3.7 MemIf_InvalidateBlock

MemIf044:

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):	DeviceIndex	--
	BlockNumber	--

Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	--
Description:	map function calls of MemIf_InvalidateBlock to service: Fee_InvalidateBlock respectively Ea_InvalidateBlock	

MemIf054: The function MemIf_InvalidateBlock shall be mapped to service: Fee_InvalidateBlock respectively Ea_InvalidateBlock.

8.3.8 MemIf_GetVersionInfo

MemIf045:

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.	

MemIf031: The function MemIf_GetVersionInfo shallreturn synchronously the version information on the Memory Abstraction Interface module in the structure provided by the caller.

MemIf057: Configurations: The function MemIf_GetVersionInfo is only available if enabled by the pre-processor switch `MemIfVersionInfoApi`.

8.3.9 MemIf_EraseImmediateBlock

MemIf046:

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):	DeviceIndex	--
	BlockNumber	--
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	--
Description:	map function calls of MemIf_EraseImmediateBlock to service:	

	Fee_EraseImmediateBlock respectively Ea_EraseImmediateBlock

MemIf055: The function MemIf_EraseImmediateBlock shall be mapped to service: Fee_EraseImmediateBlock respectively Ea_EraseImmediateBlock.

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.

MemIf047:

API function	*Description*
Ea_Cancel	Cancels the ongoing asynchronous operation.
Ea_EraseImmediateBlock	Erases the block BlockNumber.
Ea_GetJobResult	Service to return the JobResult.
Ea_GetStatus	Service to return the Status.
Ea_InvalidateBlock	Invalidates the block BlockNumber.
Ea_Read	Reads Length bytes of block Blocknumber at offset BlockOffset into the buffer DataBufferPtr.
Ea_SetMode	Sets the mode.
Ea_Write	Writes the contents of the DataBufferPtr to the block BlockNumber.
Fee_Cancel	Service to call the cancel function of the underlying flash driver.
Fee_EraseImmediateBlock	Service to erase a logical block.
Fee_GetJobResult	Service to call the GetJobResult function of the underlying flash driver.
Fee_GetStatus	Service to call the GetStatus function of the underlying flash driver.
Fee_InvalidateBlock	Service to invalidate a logical block.
Fee_Read	Service to initiate a read job.
Fee_SetMode	Service to call the Fls_SetMode function of the underlying flash driver.
Fee_Write	Service to initiate a write job.

8.6.2 Optional Interfaces

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

MemIf048:

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 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 How to read this chapter

In addition to this section, it is highly recommended to read the documents:

- AUTOSAR Layered Software Architecture
- AUTOSAR ECU Configuration Specification
  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 0.

10.2.1 Variants

No variants specified.

10.2.2 MemIf

SWS Item	**ECUC_MemIf_00025 :**
Module Name	*MemIf*
Module Description	Configuration of the MemIf (Memory Abstraction Interface) module.

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

10.2.3 MemIfGeneral

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

SWS Item	**ECUC_MemIf_00035 :**

Name	MemIfDevErrorDetect {MEMIF_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	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**ECUC_MemIf_00033 :**		
Name	MemIfNumberOfDevices {MEMIF_NUMBER_OF_DEVICES}		
Description	Concrete number of underlying memory abstraction modules.		
Multiplicity	1		
Type	DerivedIntegerParamDef		
Range	1 .. 255		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**ECUC_MemIf_00032 :**		
Name	MemIfVersionInfoApi {MEMIF_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	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

No Included Containers

10.3 Published Information

Published information contains data defined by the implementer of the SW module that does not change when the module is adapted (i.e. configured) to the actual HW/SW environment. It thus contains version and manufacturer information.

The standard common published information like

vendorId (<Module>_VENDOR_ID),
moduleId (<Module>_MODULE_ID),

arMajorVersion (<Module>_AR_MAJOR_VERSION),
arMinorVersion (<Module>_ AR_MINOR_VERSION),
arPatchVersion (<Module>_ AR_PATCH_VERSION),
swMajorVersion (<Module>_SW_MAJOR_VERSION),
swMinorVersion (<Module>_ SW_MINOR_VERSION),
swPatchVersion (<Module>_ SW_PATCH_VERSION),
vendorApiInfix (<Module>_VENDOR_API_INFIX)

is provided in the BSW Module Description Template (see [10] Figure 4.1 and Figure 7.1).

Additional published parameters are listed below if applicable for this module.