MICROSAR MemIf

Technical Reference

Version 2.02.00

Authors	Tobias Schmid, Manfred Duschinger, Michael Goß
Status	Released

1 Document Information

1.1 History

Author	Date	Version	Remarks
Tobias Schmid	2008-04-14	1.0	Creation of document
Manfred Duschinger	2013-02-20	1.01.00	Ch. 4.1. Update files according to new generator Ch. 6 Update Configuration
Michael Goß	2014-11-21	2.01.01	Typos were corrected and content was modified a little
Michael Goß	2015-04-23	2.02.00	Content was updated regarding SafeBSW MemIf

Table 1-1 History of the document

1.2 Reference Documents

No.	Title	Version
[1]	AUTOSAR_SWS_Mem_AbstractionInterface.pdf	V1.4.0
[2]	AUTOSAR_SWS_DET.pdf	V2.2.0
[3]	AUTOSAR_BasicSoftwareModules.pdf	V1.0.0
[4]	AUTOSAR_SWS_EEPROM_Abstraction.pdf	V2.0.0
[5]	AUTOSAR_SWS_Flash_EEPROM_Emulation.pdf	V2.0.0

Table 1-2 Reference documents

1.3 Scope of the Document

This technical reference describes the general use of module MemIf (AUTOSAR Memory Abstraction Interface).

Please note
We have configured the programs in accordance with your specifications in the questionnaire. Whereas the programs do support other configurations than the one specified in your questionnaire, Vector´s release of the programs delivered to your company is expressly restricted to the configuration you have specified in the questionnaire.

Contents

Illustrations

Tables

2 Introduction

This document describes the functionality, API and configuration of the AUTOSAR BSW module MemIf as specified in [1].

Supported AUTOSAR Release*:	4	
Supported Configuration Variants:	PRE-COMPILE	
Vendor ID:	MEMIF_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	MEMIF_MODULE_ID	22 decimal (according to ref. [3])

* For the precise AUTOSAR Release 4.x please see the release specific documentation.

MemIf (Memory Abstraction Interface) provides the interface that is used by the NvM to access NV memory devices. Two different types of NV memory are intended for use: Flash memory and EEPROM. To abstract the hardware dependencies of the memory devices, low level drivers with a commonly defined API are used: Fls and Eep (internal or external). These modules are abstracted by the modules Fee (Flash EEPROM Emulation) and Ea (EEPROM Abstraction). Both modules may exist at the same time.

MemIf offers a common interface for accessing Fee or Ea instances. In order to distinguish those different instances MemIf provides a set of device handles, which may be used for configuration of NvM.

2.1 Architecture Overview

The following figure shows where the MemIf is located in the AUTOSAR architecture.

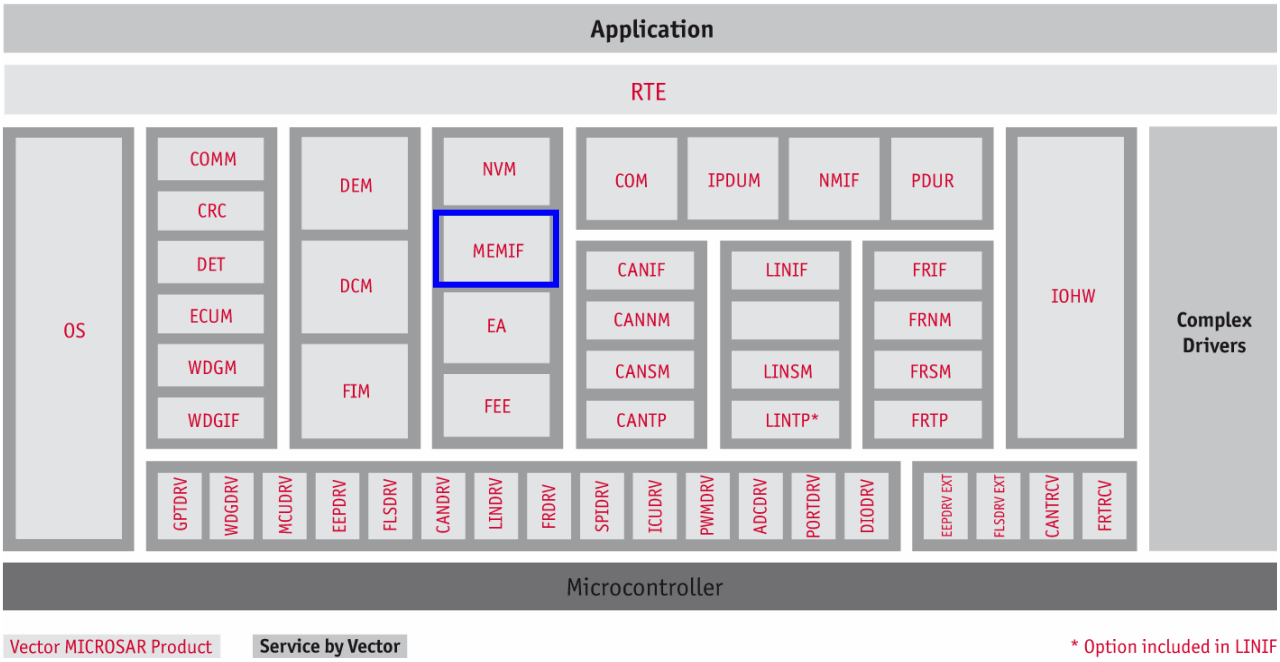

Figure 2-1 AUTOSAR architecture

The next figure shows the interfaces to adjacent modules of the MemIf. These interfaces are described in chapter 5.

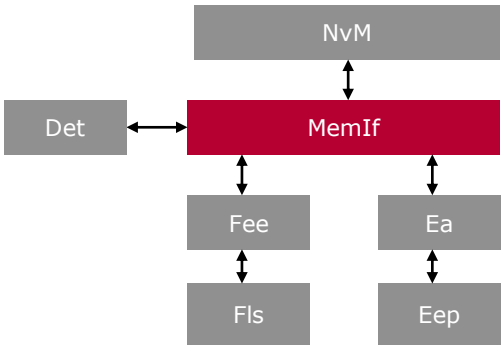

Figure 2-2 Interfaces to adjacent modules of the MemIf

3 Functional Description

3.1 Features

The features listed in this chapter cover the complete functionality specified in [1].

The "supported" and "not supported" features are presented in the following two tables. For further information of not supported features also see chapter 7.

The following features described in [1] are supported:

Feature
MemIf allows the NVRAM manager to access multiple instances of memory hardware abstraction modules (Fee or Ea), regardless of different APIs and implementations.

Table 3-1 Supported SWS features

3.2 Initialization

MemIf does not need any initialization. Nevertheless it is necessary to initialize all memory hardware abstraction module instances that are interfaced by MemIf.

Caution
MemIf does **not** provide any services for initialization of underlying memory hardware abstraction modules. Initialization of these modules is not done by MemIf!

3.3 Main Functions

MemIf does not implement main-functions that would need recurring execution. Job requests are mapped to the appropriate underlying memory hardware abstraction module, which implements the main-function for processing the job.

Caution
MemIf is **not** responsible for calling main-functions of the underlying hardware abstraction modules. Calling main-functions cyclically has to be implemented in the BSW Scheduler (or something similar).

3.4 Error Handling

3.4.1 Development Error Reporting

Development errors are reported by default to DET using the service `Det_ReportError()`, (specified in [2]), if the pre-compile configuration parameter for “Development Mode” and “Development Error Reporting” are enabled.

The reported service IDs identify the services which are described in 5.3. The following table presents the service IDs and the related services:

Service ID	Service
1	MemIf_SetMode
2	MemIf_Read
3	MemIf_Write
4	MemIf_Cancel
5	MemIf_GetStatus
6	MemIf_GetJobResult
7	MemIf_InvalidateBlock
8	MemIf_GetVersionInfo
9	MemIf_EraseImmediateBlock

Table 3-2 Mapping of service IDs to services

The errors reported to DET are described in the following table:

Error Code		Description
0x01	MEMIF_E_PARAM_DEVICE	The parameter denoting the device index passed to the API service is out of range.
0x02	MEMIF_E_PARAM_POINTER	The parameter passed to MemIf_GetVersionInfo references no valid address (NULL-Pointer).

Table 3-3 Errors reported to DET

3.4.1.1 Parameter Checking

The following table shows which parameter checks are performed on which services:

Check / Service	Check device index passed to API services	Check parameter for referencing a valid address
MemIf_GetVersionInfo		■
MemIf_SetMode		
MemIf_Read	■	
MemIf_Write	■	
MemIf_Cancel	■	
MemIf_GetStatus	■	
MemIf_GetJobResult	■	
MemIf_InvalidateBlock	■	
MemIf_EraseImmediateBlock	■	

Table 3-4 Development Error Reporting: Assignment of checks to services

AUTOSAR requires that API functions check the validity of their parameters. The checks in Table 3-4 are internal parameter checks of the API functions.

4 Integration

This chapter gives necessary information for the integration of the MICROSAR MemIf into an application environment of an ECU.

4.1 Scope of Delivery

The delivery of the MemIf contains the files which are described in the chapters 4.1.1 and 4.1.2:

4.1.1 Static Files

File Name	Description
MemIf.h	API of module MemIf, only this file needs to be included by upper layer software (e.g. NvM)
MemIf_Types.h	Defines all standard types, needed by upper layer modules as well as the modules Fee and Ea. This file needs to be included by all memory hardware abstraction modules according to AUTOSAR.
MemIf.c	Implementation of the functionalities of the module MemIf

Table 4-1 Static files

4.1.2 Dynamic Files

The dynamic files are generated by the configuration tool.

File Name	Description
MemIf_Cfg.h	Configuration header file
MemIf_Cfg.c	Configuration source file

Table 4-2 Generated files

4.2 Include Structure

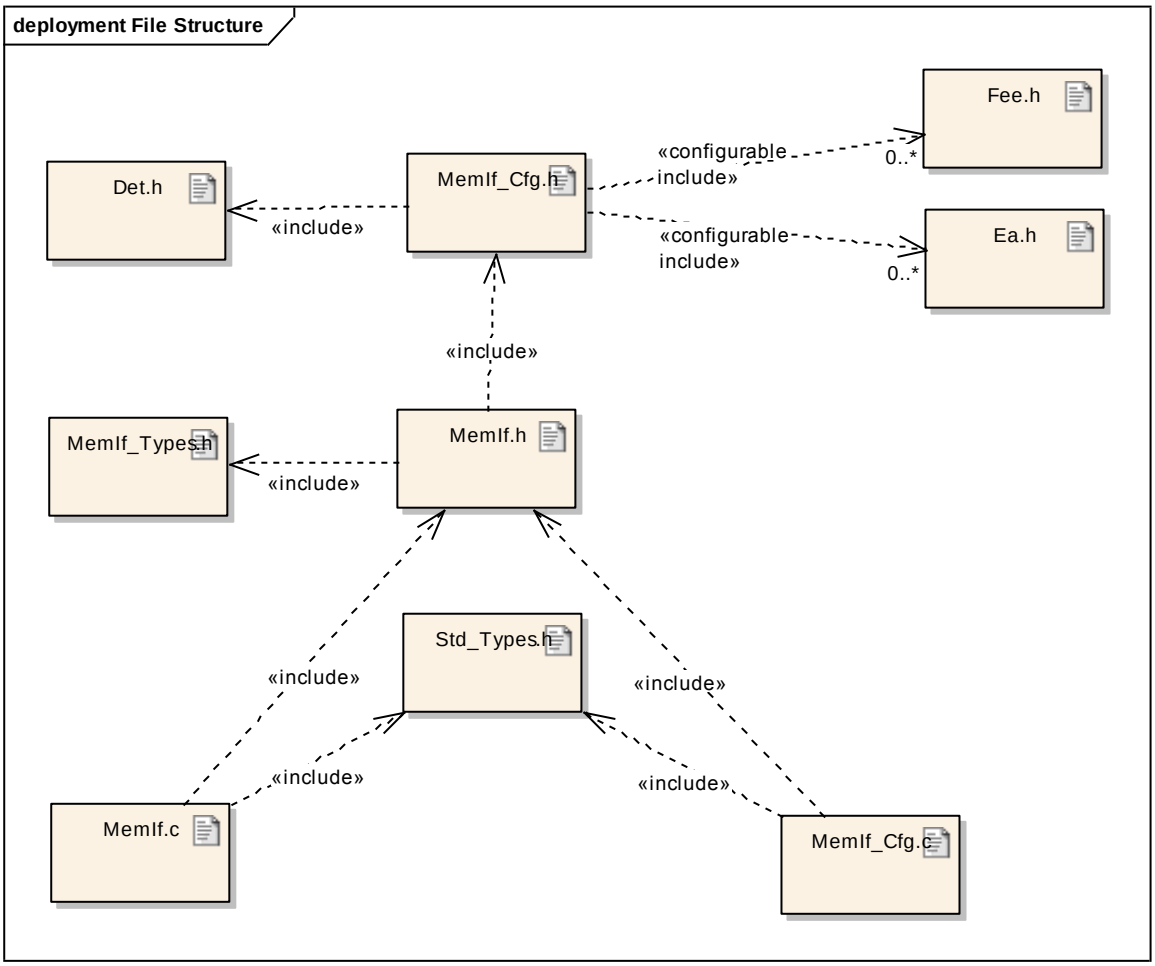

Figure 4-1 Include structure

4.3 Compiler Abstraction and Memory Mapping

The objects (e.g. variables, functions, constants) are declared by compiler independent definitions – the compiler abstraction definitions. Each compiler abstraction definition is assigned to a memory section.

The following table contains the memory section names and the compiler abstraction definitions, which are defined for the MemIf, and illustrates their assignment among each other.

Memory Mapping Sections \ Compiler Abstraction Definitions	MEMIF_CONST	MEMIF_CODE	MEMIF_PRIVATE_CODE	MEMIF_APPL_DATA
MEMIF_START_SEC_CONST_8BIT MEMIF_STOP_SEC_CONST_8BIT	■			

MEMIF_START_SEC_CONST_32BIT MEMIF_STOP_SEC_CONST_32BIT	■			
MEMIF_START_SEC_CODE MEMIF_STOP _SEC_CODE		■	■	
Memory sections in which underlying memory hardware abstraction modules' code resides		■	■	
Memory sections of data buffers, which are passed to the API services for read or write jobs				■
Memory sections of buffers passed to MemIf_GetVersionInfo				■

Table 4-3 Compiler abstraction and memory mapping

5 API Description

5.1 Interfaces Overview

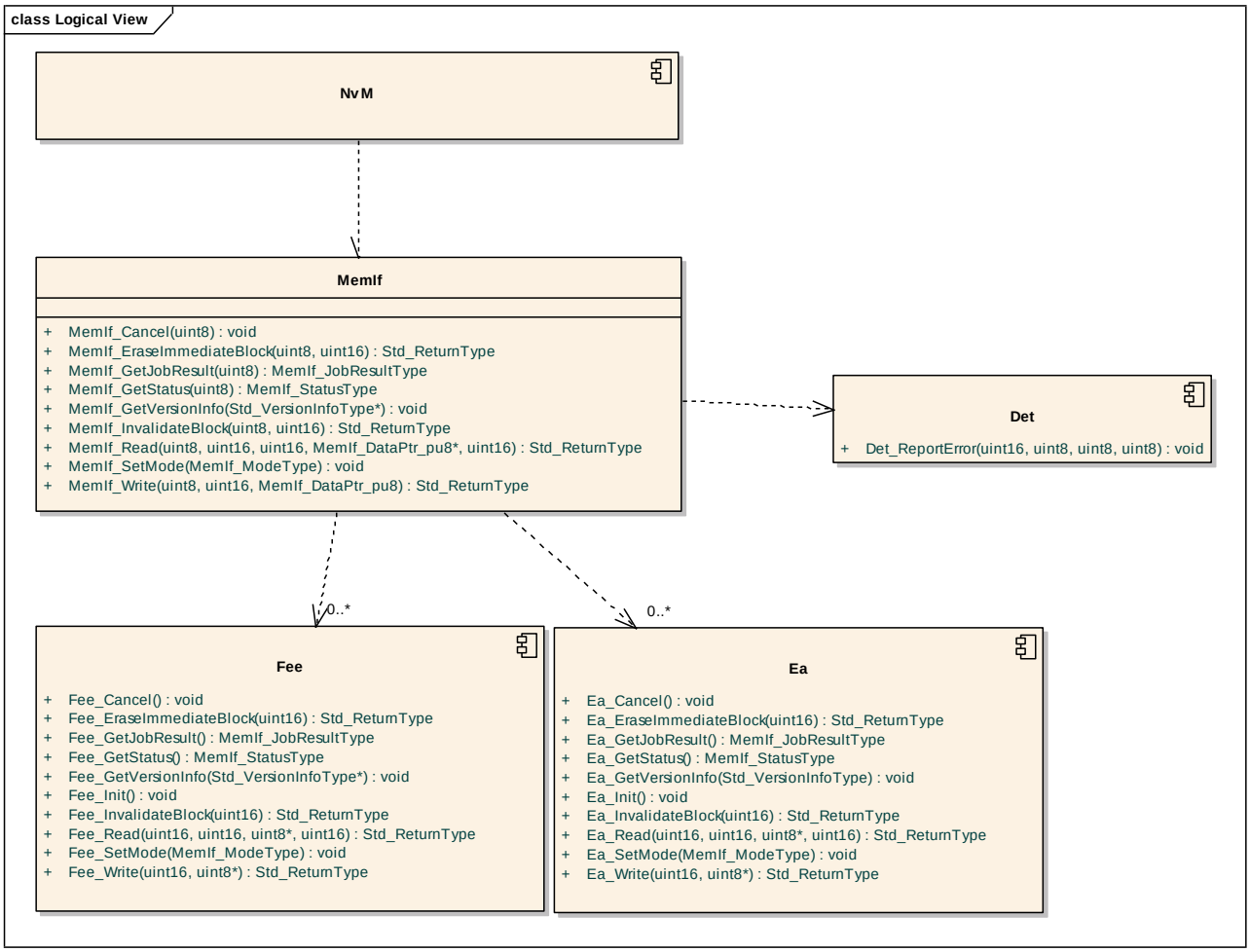

Figure 5-1 MemIf interactions with other BSW

5.2 Type Definitions

Type Name	C-Type	Description	Value Range
MemIf_StatusType	enum	Denotes the states of BSW modules in the memory stack	MEMIF_UNINIT Module is not initialized
			MEMIF_IDLE There are no pending jobs that need processing
			MEMIF_BUSY Module is processing jobs, no further job requests are accepted
			MEMIF_BUSY_INTERNAL No job requests are being processed, but the module is busy executing internal operations

Type Name	C-Type	Description	Value Range
MemIf_JobResultType	enum	Denotes the result of a job request after processing of this job	MEMIF_JOB_OK Job processing finished successfully
			MEMIF_JOB_FAILED Job processing finished with an error
			MEMIF_JOB_PENDING Job is currently being processed
			MEMIF_JOB_CANCELLED Job has been cancelled by the user
			MEMIF_BLOCK_INCONSISTENT Job finished successfully, but data is inconsistent
			MEMIF_BLOCK_INVALID Job finished successfully but data has been invalidated
MemIf_ModeType	enum	Denotes the processing mode for a module in the memory stack	MEMIF_MODE_SLOW Jobs are processed with the configured properties for slow mode
			MEMIF_MODE_FAST Jobs are processed with the configured properties for fast mode

Table 5-1 Type definitions

5.3 Services provided by MemIf

The MemIf API consists of services, which are realized by function calls.

5.3.1 MemIf_GetVersionInfo

Prototype	
`void MemIf_GetVersionInfo ( Std_VersionInfoType * VersionInfoPtr )`	
Parameter	
VersionInfoPtr	Reference to a version information structure in RAM
Return code	
-	-
Functional Description	
This service writes the version information of MemIf to the referenced structure.	
Particularities and Limitations	
■ In case the input parameter references an invalid address (NULL-pointer) the error MEMIF_E_PARAM_POINTER is reported to Det and execution of the service is aborted. ■ This service is only available if "Version Info Api" is configured to STD_ON	

Expected Caller Context
■ This service has no restriction to the allowed or expected caller context.

Table 5-2 MemIf_GetVersionInfo

5.3.2 MemIf_SetMode

Prototype	
`void MemIf_SetMode(MemIf_ModeType Mode )`	
Parameter	
Mode	Mode to switch modules into
Return code	
-	-
Functional Description	
This service switches all underlying memory hardware abstraction modules to the requested mode of operation, by calling [Ea|Fee]_SetMode (See description of respective module's function).	
Particularities and Limitations	
■ All memory abstraction modules have to be in state `MEMIF_IDLE` when this service is executed.	
Call context	
> This service has no restriction to the allowed or expected caller context.	

Table 5-3 MemIf_SetMode

5.3.3 MemIf_Read

Prototype	
`Std_ReturnType MemIf_Read ( uint8 DeviceIndex, uint16 BlockNumber, uint16 BlockOffset, uint8* DataBufferPtr, uint16 Length )`	
Parameter	
DeviceIndex	Index of the memory hardware abstraction module to which the read operation shall be delegated.
BlockNumber	Identifies the block to read in non-volatile memory.
BlockOffset	Offset in the block identified by BlockNumber from which on reading is performed
DataBufferPtr	Reference to the data buffer to which the data in non-volatile memory is read to.
Length	Number of bytes to read

Return code	
E_OK	Read job request is accepted by the addressed memory hardware abstraction module.
E_NOT_OK	Read job request is rejected by the addressed memory hardware abstraction module.
Functional Description	
Delegates the job request to the appropriate memory hardware abstraction module by calling the service [Ea|Fee]_Read of the addressed module instance (See description of respective module's function)	
Particularities and Limitations	
> If the parameter `DeviceIndex` is out of range (greater than the number of configured devices), the error code `MEMIF_E_PARAM_DEVICE` is reported to Det and execution of the service is aborted.	
Call context	
> NvM / Application	

Table 5-4 MemIf_Read

5.3.4 MemIf_Write

Prototype	
`Std_ReturnType MemIf_Write ( uint8 DeviceIndex, uint16 BlockNumber, uint8* DataBufferPtr )`	
Parameter	
DeviceIndex	Index of the memory hardware abstraction module to which the write operation shall be delegated.
BlockNumber	Identifies the block to write to non-volatile memory.
DataBufferPtr	Reference to the data buffer whose content is written to non-volatile memory
Return code	
E_OK	Write job request is accepted by the addressed memory hardware abstraction module.
E_NOT_OK	Write job request is rejected by the addressed memory hardware abstraction module.
Functional Description	
Delegates the job request to the appropriate memory hardware abstraction module by calling the service [Ea|Fee]_Write of the addressed module instance (See description of respective module's function)	
Particularities and Limitations	
> If the parameter `DeviceIndex` is out of range (greater than the number of configured devices), the error code `MEMIF_E_PARAM_DEVICE` is reported to Det and execution of the service is aborted.	
Call context	

> NvM / Application

Table 5-5 MemIf_Write

5.3.5 MemIf_Cancel

Prototype	
`void MemIf_Cancel ( uint8 DeviceIndex )`	
Parameter	
DeviceIndex	Index of the memory hardware abstraction module whose job processing shall be cancelled.
Return code	
-	-
Functional Description	
Delegates the cancel request to the appropriate memory hardware abstraction module by calling the service [Ea|Fee]_Cancel of the addressed module instance (See description of respective module's function)	
Particularities and Limitations	
> If the parameter `DeviceIndex` is out of range (greater than the number of configured devices), the error code `MEMIF_E_PARAM_DEVICE` is reported to Det and execution of the service is aborted.	
Call context	
> NvM / Application	

Table 5-6 MemIf_Cancel

5.3.6 MemIf_GetStatus

Prototype	
`MemIf_StatusType MemIf_GetStatus ( uint8 DeviceIndex )`	
Parameter	
DeviceIndex	Index of the memory hardware abstraction module whose job processing shall be cancelled.
Return code	
MEMIF_IDLE	Addressed module is ready to accept job requests
MEMIF_UNINIT	Addressed module is not initialized
MEMIF_BUSY	Addressed module is processing a job and is not able to accept new job requests
MEMIF_BUSY_INTERNAL	Addressed module is not processing any job requests, but the module is busy executing internal operations
Functional Description	
Delegates the call to this service to the appropriate memory hardware abstraction module by calling the service [Ea|Fee]_GetStatus (See description of respective module's function).	

Particularities and Limitations

> If the parameter `DeviceIndex` is out of range (greater than the number of configured devices and unequal to `MEMIF_BROADCAST_ID`), the error code `MEMIF_E_PARAM_DEVICE` is reported to Det and execution of the service is aborted.
> In case the `MEMIF_BROADCAST_ID` is used as device index parameter, the overall status of all underlying memory abstraction modules is returned. This overall status is computed as follows:
  > `MEMIF_IDLE` – all underlying devices are in this state
  > `MEMIF_UNINIT` – at least one device returned this state
  > `MEMIF_BUSY` – at least one device returned this state and no other returned `MEMIF_UNINIT`
  > `MEMIF_BUSY_INTERNAL` – at least one device returned this state and no other returned `MEMIF_BUSY` or `MEMIF_UNINIT`

Call context
> NvM / Application

Table 5-7 MemIf_GetStatus

5.3.7 MemIf_GetJobResult

Prototype	
`MemIf_JobResultType` **`MemIf_GetJobResult`** `( uint8 DeviceIndex )`	
Parameter	
DeviceIndex	Index of the memory hardware abstraction module whose job processing shall be cancelled.
Return code	
MEMIF_JOB_OK	Job processing finished successfully
MEMIF_JOB_FAILED	Job processing finished with an error
MEMIF_JOB_PENDING	Job is currently being processed
MEMIF_JOB_CANCELLED	Job has been cancelled by the user
MEMIF_BLOCK_INCONSISTENT	Job finished successfully, but data is inconsistent
MEMIF_BLOCK_INVALID	Job finished successfully but data has been invalidated
Functional Description	
Delegates the call to this service to the appropriate memory hardware abstraction module by calling the service [Ea|Fee]_GetJobResult (See description of respective module's function).	
Particularities and Limitations	
> If the parameter `DeviceIndex` is out of range (greater than the number of configured devices), the error code `MEMIF_E_PARAM_DEVICE` is reported to Det and execution of the service is aborted.	
Call context	
> NvM / Application	

Table 5-8 MemIf_GetJobResult

5.3.8 MemIf_EraseImmediateBlock

Prototype	
`Std_ReturnType MemIf_EraseImmediateBlock ( uint8 DeviceIndex, uint16 BlockNumber )`	
Parameter	
DeviceIndex	Index of the memory hardware abstraction module to which the operation shall be delegated.
BlockNumber	Identifies the block to erase in non-volatile memory.
Return code	
E_OK	Erase job request is accepted by the addressed memory hardware abstraction module.
E_NOT_OK	Erase job request is rejected by the addressed memory hardware abstraction module.
Functional Description	
Delegates the job request to the appropriate memory hardware abstraction module by calling the service [Ea|Fee]_EraseImmediateBlock of the addressed module instance (See description of respective module's function)	
Particularities and Limitations	
> If the parameter `DeviceIndex` is out of range (greater than the number of configured devices), the error code `MEMIF_E_PARAM_DEVICE` is reported to Det and execution of the service is aborted.	
Call context	
> NvM / Application	

Table 5-9 MemIf_EraseImmediateBlock

5.3.9 MemIf_InvalidateBlock

Prototype	
`Std_ReturnType MemIf_InvalidateBlock ( uint8 DeviceIndex, uint16 BlockNumber )`	
Parameter	
DeviceIndex	Index of the memory hardware abstraction module to which the operation shall be delegated.
BlockNumber	Identifies the block to invalidate in non-volatile memory.
Return code	
E_OK	Erase job request is accepted by the addressed memory hardware abstraction module.
E_NOT_OK	Erase job request is rejected by the addressed memory hardware abstraction module.

Functional Description
Delegates the job request to the appropriate memory hardware abstraction module by calling the service [Ea|Fee]_InvalidateBlock of the addressed module instance (See description of respective module's function)
Particularities and Limitations
> If the parameter `DeviceIndex` is out of range (greater than the number of configured devices), the error code `MEMIF_E_PARAM_DEVICE` is reported to Det and execution of the service is aborted.
Call context
> NvM / Application

Table 5-10 MemIf_InvalidateBlock

5.4 Services used by MemIf

In the following table services provided by other components, which are used by the MemIf are listed. For details about prototype and functionality refer to the documentation of the providing component.

Component	API
DET (see [2])	`Det_ReportError`
EA (see [4])	`Ea_SetMode`
	`Ea_Read`
	`Ea_Write`
	`Ea_Cancel`
	`Ea_GetStatus`
	`Ea_GetJobResult`
	`Ea_InvalidateBlock`
	`Ea_GetVersionInfo`
	`Ea_EraseImmediateBlock`
FEE (see [5])	`Fee_SetMode`
	`Fee_Read`
	`Fee_Write`
	`Fee_Cancel`
	`Fee_GetStatus`
	`Fee_GetJobResult`
	`Fee_InvalidateBlock`
	`Fee_GetVersionInfo`
	`Fee_EraseImmediateBlock`

Table 5-11 Services used by the MemIf

6 Configuration

The MEMIF attributes can be configured using the DaVinci Configurator. The outputs of the configuration and generation process are the configuration source files.

The description of each used parameter is set in the MemIf bswmd file.

7 AUTOSAR Standard Compliance

7.1 Deviations

7.1.1 Extension of Error Codes

In contradiction to AUTOSAR standard MemIf019, no set of macros is implemented, which maps the Memory Abstraction Interface API to the API of the corresponding memory abstraction module.

7.2 Additions/ Extensions

No Extensions to AUTOSAR standard

8 Glossary and Abbreviations

8.1 Glossary

Term	Description
DaVinci Configurator	Generation tool for MICROSAR components

Table 8-1 Glossary

8.2 Abbreviations

Abbreviation	Description
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSW	Basis Software
DET	Development Error Tracer
ECU	Electronic Control Unit
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
SRS	Software Requirement Specification
SWS	Software Specification
NVRAM	Non Volatile RAM Manager
NvM	NVRAM Manager
Fee	Flash EEPROM Emulation
Ea	EEPROM Abstraction
Fls	Flash Driver
Eep	EEPROM Driver
RAM	Random Access Memory

Table 8-2 Abbreviations

9 Contact

Visit our website for more information on

- News
- Products
- Demo software
- Support
- Training data
- Addresses

www.vector.com