MICROSAR MemIf

Technical Reference

Version 1.0

Authors	Tobias Schmid
Status	Released

1 Document Information

1.1 History

Author	Date	Version	Remarks
Tobias Schmid	2008-04-14	1.0	Creation of document

Table 1-1 History of the document

1.2 Reference Documents

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

Table 1-2 Reference documents

1.1 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

1 Document Information 2
1.1 History 2
1.2 Reference Documents 2
1.1 Scope of the Document 2

2 Introduction 6
2.1 Architecture Overview 7

3 Functional Description 8
3.1 Features 8
3.2 Initialization 8
3.3 Main Functions 8
3.4 Error Handling 8
3.4.1 Development Error Reporting 8
3.4.1.1 Parameter Checking 10

4 Integration 11
4.1 Scope of Delivery 11
4.1.1 Static Files 11
4.1.2 Dynamic Files 11
4.2 Include Structure 12
4.3 Compiler Abstraction and Memory Mapping 12

5 API Description 14
5.1 Interfaces Overview 14
5.2 Type Definitions 14
5.3 Services provided by MemIf 15
5.3.1 MemIf_GetVersionInfo 16
5.3.2 MemIf_SetMode 16
5.3.3 MemIf_Read 17
5.3.4 MemIf_Write 18
5.3.5 MemIf_Cancel 19
5.3.6 MemIf_GetStatus 20
5.3.7 MemIf_GetJobResult 21
5.3.8 MemIf_EraseImmediateBlock 22
5.3.9 MemIf_InvalidateBlock 23
5.4 Services used by MemIf 23

6 Configuration 25
6.1 Configuration with EAD 25
6.1.1 Architecture View 25
6.1.2 Memory Interface Settings 26
6.1.3 Module API 27

7 AUTOSAR Standard Compliance 30
7.1 Deviations 30
7.1.1 Extension of Error Codes 30
7.1.2 Mapping 1 Device only 30
7.2 Additions/ Extensions 30
7.2.1 Additional Error Codes 30

8 Glossary and Abbreviations 31
8.1 Glossary 31
8.2 Abbreviations 31

9 Contact 32

Illustrations

Figure 2-1 AUTOSAR architecture 7
Figure 2-2 Interfaces to adjacent modules of the MemIf 7
Figure 4-1 Include structure 12
Figure 5-1 MemIf interactions with other BSW 14
Figure 6-1 EAD Architecture View 25
Figure 6-2 Memory Interface Settings Configuration 26
Figure 6-3 Module API 28

Tables

Table 1-1 History of the document 2
Table 1-2 Reference documents 2
Table 3-1 Supported SWS features 8
Table 3-2 Mapping of service IDs to services 9
Table 3-3 Errors reported to DET 9
Table 3-4 Development Error Reporting: Assignment of checks to services 10
Table 4-1 Static files 11
Table 4-2 Generated files 11
Table 4-3 Compiler abstraction and memory mapping 13
Table 5-1 Type definitions 15
Table 5-2 MemIf_GetVersionInfo 16
Table 5-3 MemIf_SetMode 16
Table 5-4 MemIf_Read 17
Table 5-5 MemIf_Write 18
Table 5-6 MemIf_Cancel 19
Table 5-7 MemIf_GetStatus 20
Table 5-8 MemIf_GetJobResult 21
Table 5-9 MemIf_EraseImmediateBlock 22
Table 5-10 MemIf_InvalidateBlock 23
Table 5-11 Services used by the MemIf 24
Table 6-1 Memory Interface Settings 27
Table 8-1 Glossary 31
Table 8-2 Abbreviations 31

2 Introduction

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

Supported AUTOSAR Release:	3.0	
Supported Configuration Variants:	MemIf supports a combination of pre-compile and link time configurable configuration parameters.	
Vendor ID:	MEMIF_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	MEMIF_MODULE_ID	22 decimal (according to ref. [3])

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). Multiple instances of these modules may exist.

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.

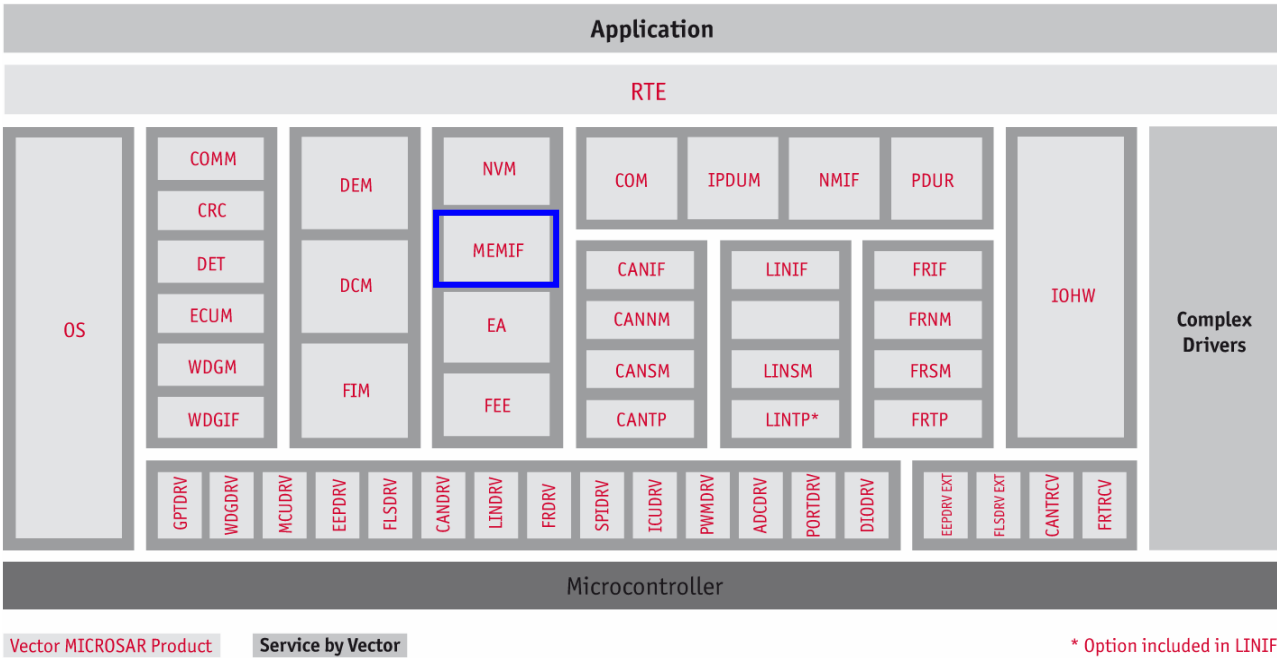

Figure 2-1 AUTOSAR architecture

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

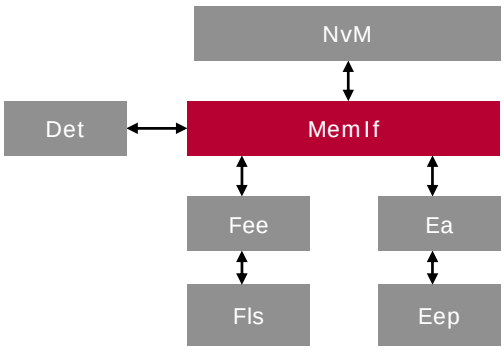

Figure 2-2 Interfaces to adjacent modules of the MemIf

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 “Debug Reporting” are enabled (see chapter 6.1.2).

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_VINFO	The parameter passed to MemIf_GetVersionInfo references no valid address (NULL-Pointer). This error code is an extension to the error codes defined in [1].

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:

Service \ Check	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. These checks are for development error reporting and can be en-/disabled separately (see chapter 3.4.1 for configuration of these paraneters). En-/disabling of single checks is an addition to the AUTOSAR standard which is only available if the global error detection switch “Development Mode” is configured to enabled.

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 (typically 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/.o	Implementation of the functionalities of the module MemIf
T_MemIf_Cfg.h	Template from which the configuration header file is generated from
T_MemIf_Cfg.c	Template from which the configuration source file is generated from
MemIf_bswmd.arxml	AUTOSAR Release 3.0 BSW Module Description
MemIf.xml	Necessary files for MICROSAR EAD support
Generation.xml	
Validation.xml	
Identifier.xml	

Table 4-1 Static files

4.1.2 Dynamic Files

The dynamic files are generated by the configuration tool MICROSAR EAD.

File Name	Description
MemIf_Cfg.h	Configuration header file (generated from T_MemIf_Cfg.h)
MemIf_Cfg.c	Configuration source file (generated from T_MemIf_Cfg.c)

Table 4-2 Generated files

4.2 Include Structure

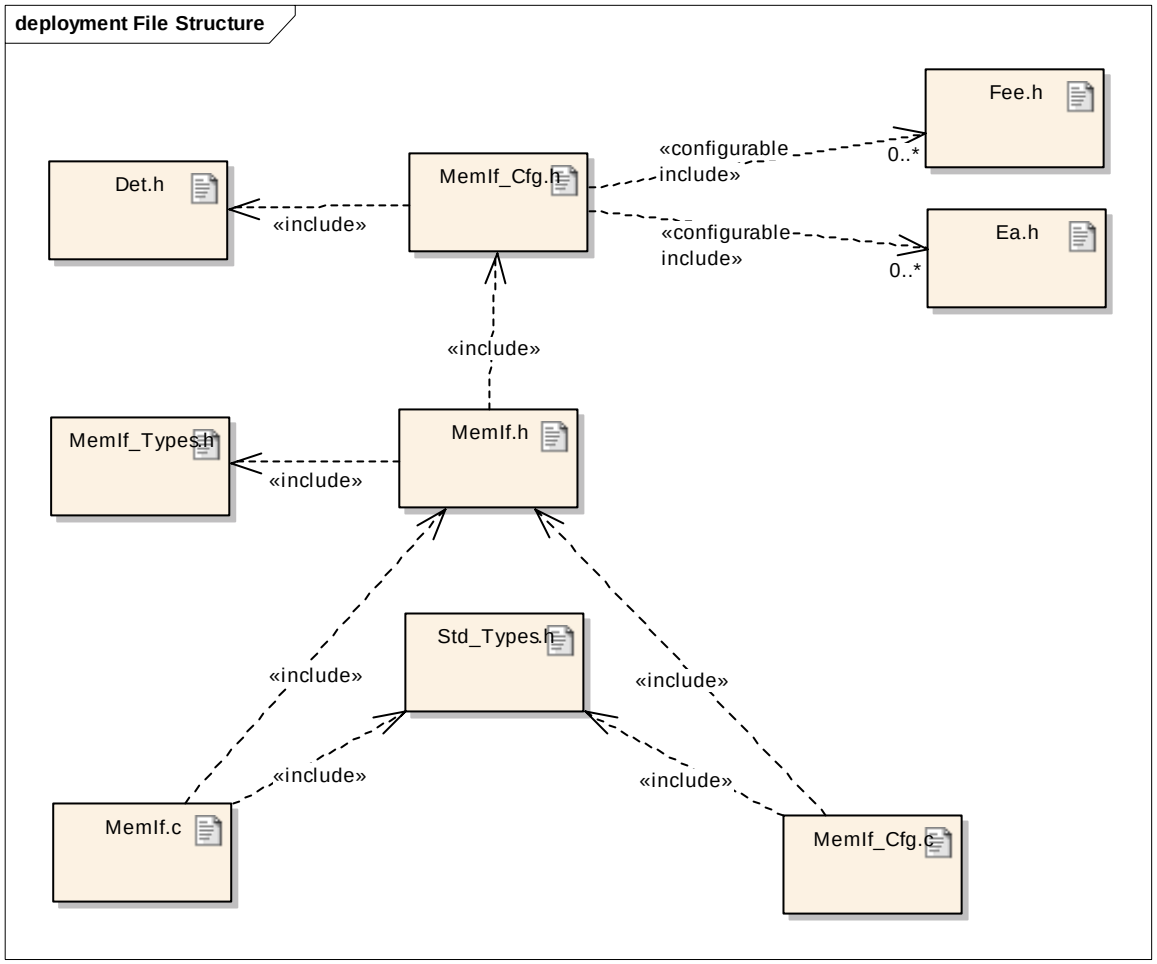

Figure 4-1 Include structure

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 defined for the MemIf and illustrates their assignment among each other.

Compiler Abstraction Definitions / Memory Mapping Sections	MEMIF_CONST	MEMIF_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

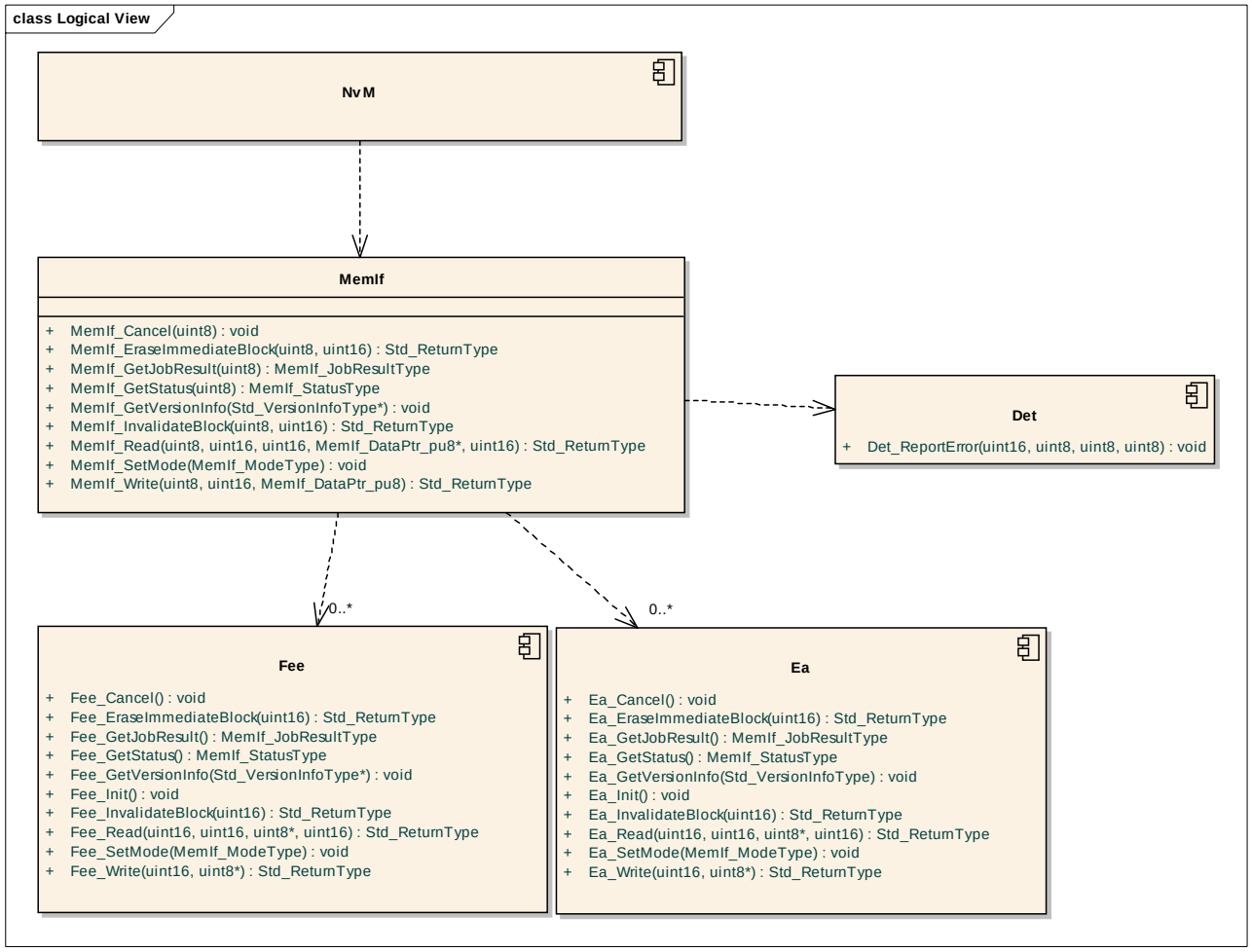

Figure 5-1 MemIf interactions with other BSW

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

Type Name	C-Type	Description	Value Range
			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
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 passed parameter references an invalid address (NULL-pointer) the error `MEMIF_E_PARAM_VINFO` is reported to Det and execution of the service is aborted. ■ This service is only available if `MEMIF_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

In the MemIf the attributes can be configured with the following methods:

- Configuration in EAD for a detailed description see 6.1

6.1 Configuration with EAD

6.1.1 Architecture View

After EAD has been started, create a new project (or open an existing). The architecture view will then be the first screen that appears:

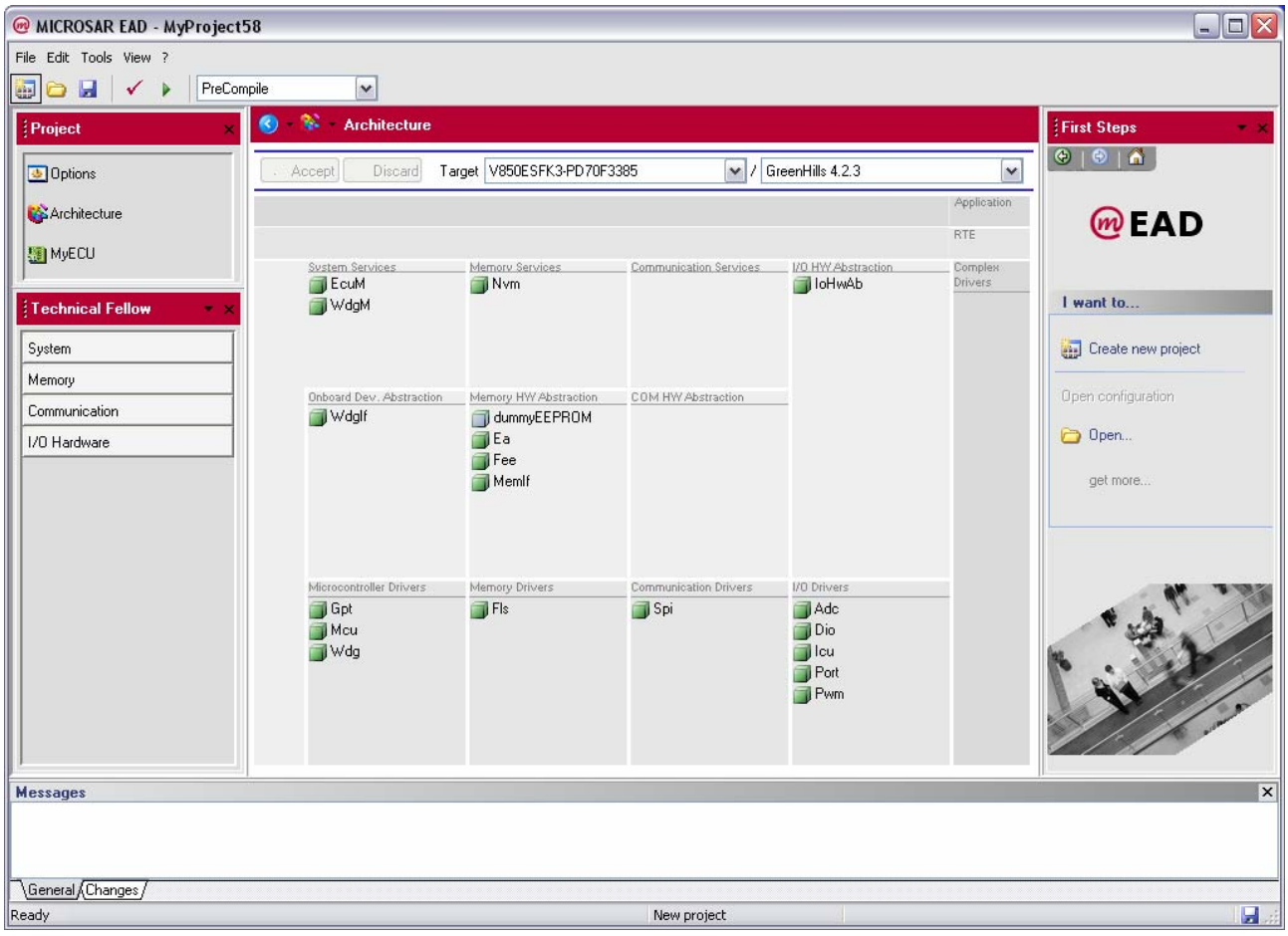

Figure 6-1 EAD Architecture View

Enable MemIf
Select MemIf in the “Memory HW Abstraction”-group and activate it (context menu → Activate)

Enable appropriate memory hardware abstraction modules
Select the appropriate memory hardware abstraction modules that shall be used in the system for non volatile memory access and activate them. The active modules are

	automatically registered to MemIf. **Accept architecture configuration** To commit the changes to the architecture, press button “Accept”

6.1.2 Memory Interface Settings

	Navigate to MemIf’s configuration view Select MemIf in the Architecture View and select “Properties” from the context menu. Another way to navigate to the configuration view is to select Memory → MemIf in the Technical Fellow (located on the left side of the main screen).

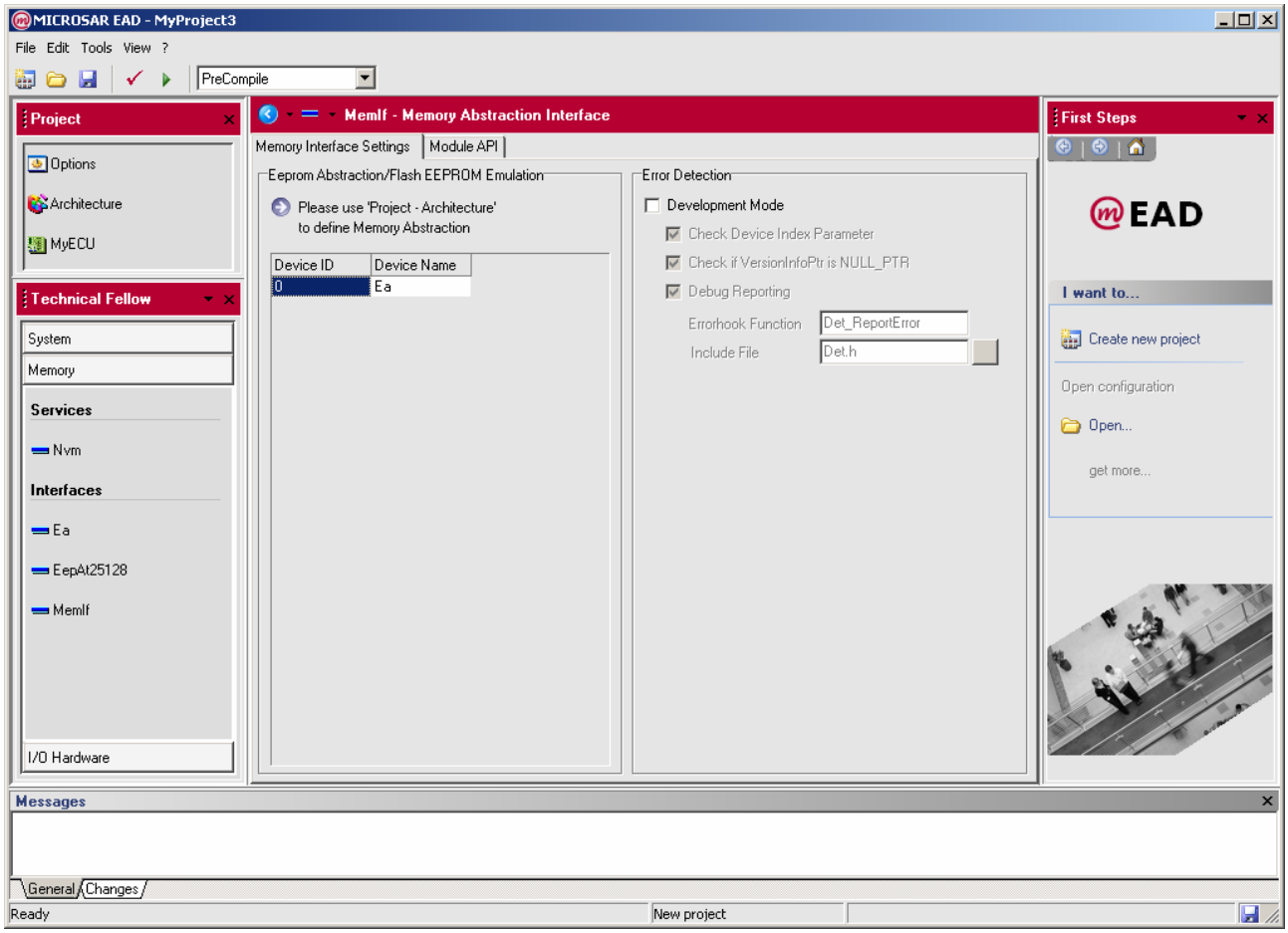

Figure 6-2 Memory Interface Settings Configuration

EAD now presents the configuration view of module MemIf. On the first tab of the register “Memory Interface Settings” the standard configuration parameters of the module are displayed. The left side of this page lists the active memory hardware abstraction modules and the related device index. This table can not be edited in this view, but only by activating or deactivating modules in EAD’s Architecture View:

The right side of this page displays the configuration parameters for detecting development errors.

The following table provides information about the different configuration parameters:

Attribute Name	Configuration Variant	Value Type	Values The default value is written in bold	Description
Eeprom Abstraction/Flash EEPROM Emulation				
-	The list of devices is link time configurable	-	-	The list only shows information on which memory hardware abstraction modules are active. (De-)activating has to be done in the Architecture View
Error Detection				
Development Mode	Pre-Compile	boolean	ON/**OFF**	Enables or disables all checks in the module generally.
Check Device Index Parameter	Pre-Compile	boolean	**ON**/OFF	Enables or disables checking of the device index parameter in API services separately (if Development Mode is enabled)
Check if VersionInfoPtr is NULL_PTR	Pre-Compile	boolean	**ON**/OFF	Enables or disables checking of the parameter `VersionInfoPtr` in `MemIf_GetVersionInfo` (if Development Mode is enabled)
Debug Reporting	Pre-Compile	boolean	**ON**/OFF	Enables or disables reporting of development errors to Det (if Development Mode is enabled)
Errorhook Function	Pre-Compile	Function Name	e.g. **Det_ReportError**	Configure a function name that is used for reporting development errors. The function must have the same prototype as `Det_ReportError`: `void Det_ReportError ( uint16 ModuleId, uint8 InstanceId, uint8 ApiId, uint8 ErrorId )`
Include File	Pre-Compile	C Header File Name	e.g. **Det.h**	Configure the header file in which the prototype for the error reporting service from the configuration parameter “Errorhook Function” is located.

Table 6-1 Memory Interface Settings

6.1.3 Module API

Navigate to the register’s tab “Module API”
Simply click on the register tab “Module API”

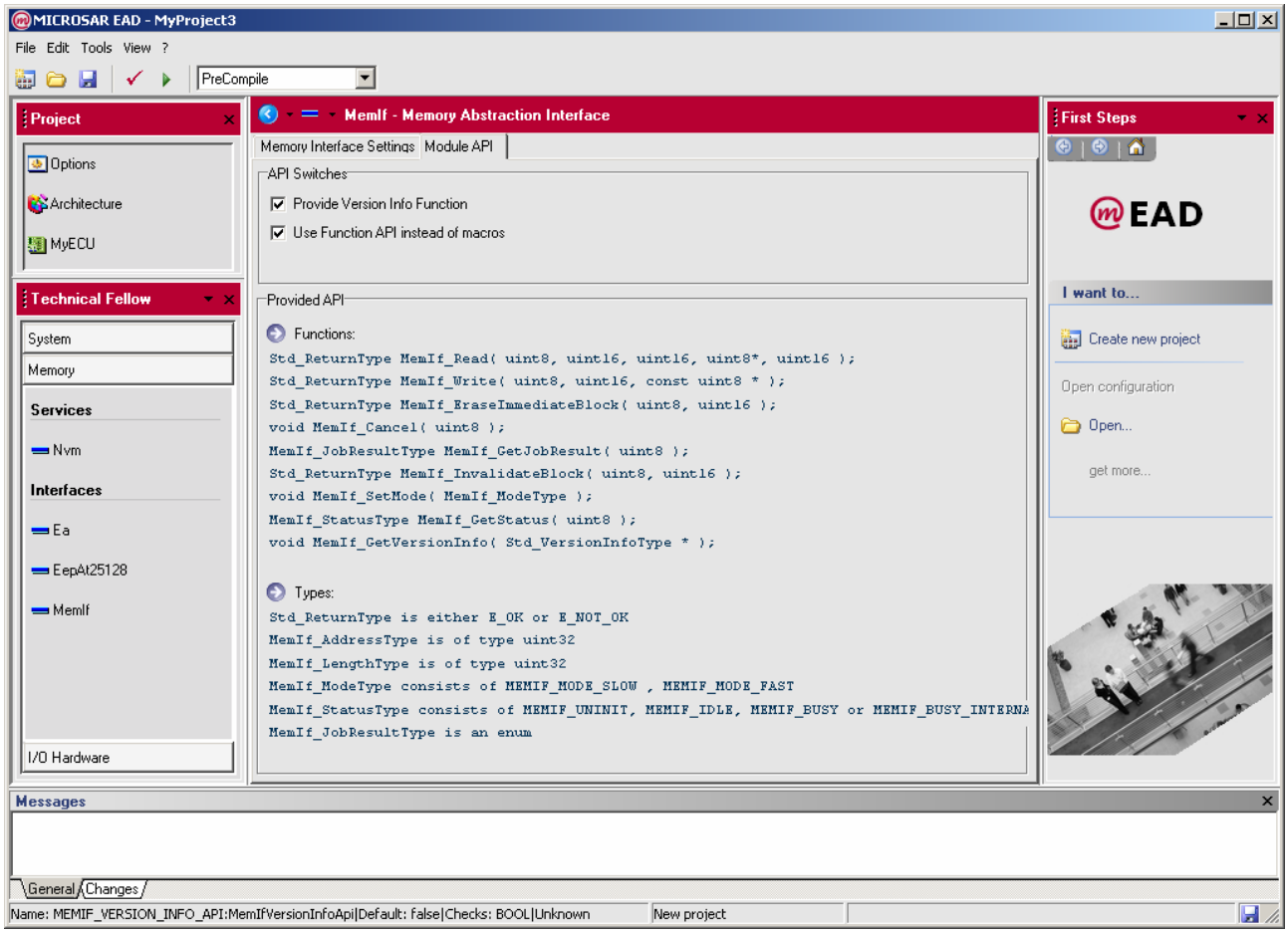

Figure 6-3 Module API

The following table provides information about the different configuration parameters:

Attribute Name	Configuration Variant	Value Type	Values The default value is written in bold	Description
API Switches				
Provide Version Info Function	Pre-Compile	boolean	ON/**OFF**	Enables or disables the API service `MemIf_GetVersionInfo`. This can be used to reduce code size of the module, in case this service is not in use.
Use Function API instead of macros	Pre-Compile	boolean	**ON**/OFF	Decides whether MemIf's API is realized as macros or as real functions. If "Development Mode" is enabled, this switch has no effect. It is not possible to use the macro API in this case.

Attribute Name	Configuration Variant	Value Type	Values The default value is written in bold	Description
Provided API				
-	-	-	**-**	Displays the available API in module MemIf. If parts of the API are disabled by configuration, those parts will be displayed in grey.

7 AUTOSAR Standard Compliance

7.1 Deviations

7.1.1 Extension of Error Codes

In contradiction to AUTOSAR standard MemIf008, an additional error code has been defined: MEMIF_E_PARAM_VINFO (see chapter 3.4.1 for details on this error code).

7.1.2 Mapping 1 Device only

In contradiction to AUTOSAR standard MemIf019, in case there is only one memory hardware abstraction module, its API is not mapped by simple macros. In order to provide link time configuration capability, mapping of the APIs is always realized by function pointer tables.

7.2 Additions/ Extensions

7.2.1 Additional Error Codes

The following error codes are reported additionally to the errors defined in [1] (see chapter 3.4.1 for details):

- MEMIF_E_PARAM_VINFO

8 Glossary and Abbreviations

8.1 Glossary

Term	Description
EAD	Embedded Architecture Designer; generation tool for MICROSAR components
GENy	Generation tool for CANbedded and 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
EAD	Embedded Architecture Designer
ECU	Electronic Control Unit
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
SRS	Software Requirement Specification
SWS	Software Specification
NVRAM	Non volatile memory
NvM	NVRAM Manager
Fee	Flash EEPROM Emulation
Ea	EEPROM Abstraction
Fls	Flash Driver
Eep	EEPROM Driver

Table 8-2 Abbreviations

9 Contact

Visit our website for more information on

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

www.vector-informatik.com