MICROSAR EEP

Technical Reference

MCAL Emulation in VTT

Version 1.2.0

Authors	Christian Leder, Bethina Mausz
Status	Released

Document Information

History

Author	Date	Version	Remarks
Christian Leder	2014-03-05	1.00.00	Initial version
Christian Leder	2015-02-05	1.01.00	> Global renaming of Vip to Vtt > Usage of template 5.11.0 for the Technical reference
Bethina Mausz	18.06.2016	1.02.00	> FEAT-1842, support of external driver. Restriction: Only one driver per system.

Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	AUTOSAR_SWS_EEPROMDriver.pdf	V3.2.0
[2]	AUTOSAR	AUTOSAR_SWS_DevelopmentErrorTracer.pdf	V3.2.0
[3]	AUTOSAR	AUTOSAR_SWS_DiagnosticEventManager.pdf	V4.2.0
[4]	AUTOSAR	AUTOSAR_TR_BSWModuleList.pdf	V1.6.0

Caution
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 Component History 6

2 Introduction 7
2.1 Architecture Overview 7

3 Functional Description 9
3.1 Features 9
3.1.1 Deviations 9
3.1.2 Additions/ Extensions 9
3.1.3 Limitations 10
3.1.3.1 Diagnostic Event Manager 10
3.2 Emulation 10
3.3 Initialization 10
3.4 States 10
3.4.1 Module States 10
3.4.2 Job States 11
3.5 Main Functions 11
3.6 Error Handling 11
3.6.1 Development Error Reporting 11
3.6.1.1 Parameter Checking 12
3.6.2 Production Code Error Reporting 13

4 Integration 14
4.1 Scope of Delivery 14
4.1.1 Static Files 14
4.1.2 Dynamic Files 14
4.2 Include Structure 15
4.3 Dependencies on SW Modules 15
4.3.1 AUTOSAR OS (Optional) 15
4.3.2 DET (Optional) 15
4.3.3 SchM (Optional) 15
4.3.4 EcuM (Optional) 15

5 API Description 16
5.1 Type Definitions 16
5.1.1 EEP Types 16
5.1.2 Imported Types 16
5.2 Services provided by EEP 17
5.2.1 Eep_InitMemory 17

5.2.2 Eep_Init ... 17
5.2.3 Eep_SetMode ... 18
5.2.4 Eep_Read ... 18
5.2.5 Eep_Write ... 19
5.2.6 Eep_Erase ... 20
5.2.7 Eep_Compare ... 21
5.2.8 Eep_Cancel ... 22
5.2.9 Eep_GetStatus ... 22
5.2.10 Eep_GetJobResult ... 23
5.2.11 Eep_GetVersionInfo ... 24
5.2.12 Eep_MainFunction ... 24
5.2.13 Eep_SimulateError ... 25
5.3 Services used by EEP ... 25
5.4 Configurable Interfaces ... 25
5.4.1 Notifications ... 25
5.4.1.1 Job End Notification ... 26
5.4.1.2 Job Error Notification ... 26

6 Configuration ... 27
6.1 Configuration Variants ... 27
6.2 Configuration with DaVinci Configurator 5 ... 27

7 Glossary and Abbreviations ... 28
7.1 Glossary ... 28
7.2 Abbreviations ... 28

8 Contact ... 29

Illustrations

Figure 2-1 AUTOSAR 4.x Architecture Overview 7
Figure 2-2 Interfaces to adjacent modules of the EEP 8
Figure 4-1 Include Structure 15

Tables

Table 1-1 Component history 6
Table 3-1 Supported AUTOSAR standard conform features 9
Table 3-2 Not supported AUTOSAR standard conform features 9
Table 3-3 Features provided beyond the AUTOSAR standard 9
Table 3-4 Service IDs 11
Table 3-5 Errors reported to DET 12
Table 3-6 Development Error Reporting: Assignment of checks to services 13
Table 4-1 Static files 14
Table 4-2 Generated files 14
Table 5-1 Type definitions 16
Table 5-2 Imported Types 16
Table 5-3 [Service name] 17
Table 5-4 Eep_Init() 17
Table 5-5 Eep_SetMode() 18
Table 5-6 Eep_Read() 19
Table 5-7 Eep_Write() 20
Table 5-8 Eep_Erase() 20
Table 5-9 Eep_Compare() 21
Table 5-10 Eep_Cancel() 22
Table 5-11 Eep_GetStatus() 22
Table 5-12 Eep_GetJobResult() 23
Table 5-13 Eep_GetVersionInfo() 24
Table 5-14 Eep_MainFunction() 24
Table 5-15 Eep_SimulateError() 25
Table 5-16 Services used by the EEP 25
Table 5-17 Job End Notification 26
Table 5-18 Job Error Notification 26
Table 7-1 Glossary 28
Table 7-2 Abbreviations 28

1 Component History

The component history gives an overview over the important milestones that are supported in the different versions of the component.

Component Version	New Features
1.0.x	Initial version of the Vip EEP driver
2.0.x	Global renaming of Vip to Vtt

Table 1-1 Component history

2 Introduction

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

Supported AUTOSAR Release*:	4	
Supported Configuration Variants:	pre-compile	
Vendor ID:	EEP_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	EEP_MODULE_ID	090 decimal (according to ref. [4])

* For the detailed functional specification please also refer to the corresponding AUTOSAR SWS.

The EEP Driver provides services to access an emulated non-volatile memory. In this emulated driver, the content of the EEPROM memory is written to and read from a text file on the PC (File extension nvram).

2.1 Architecture Overview

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

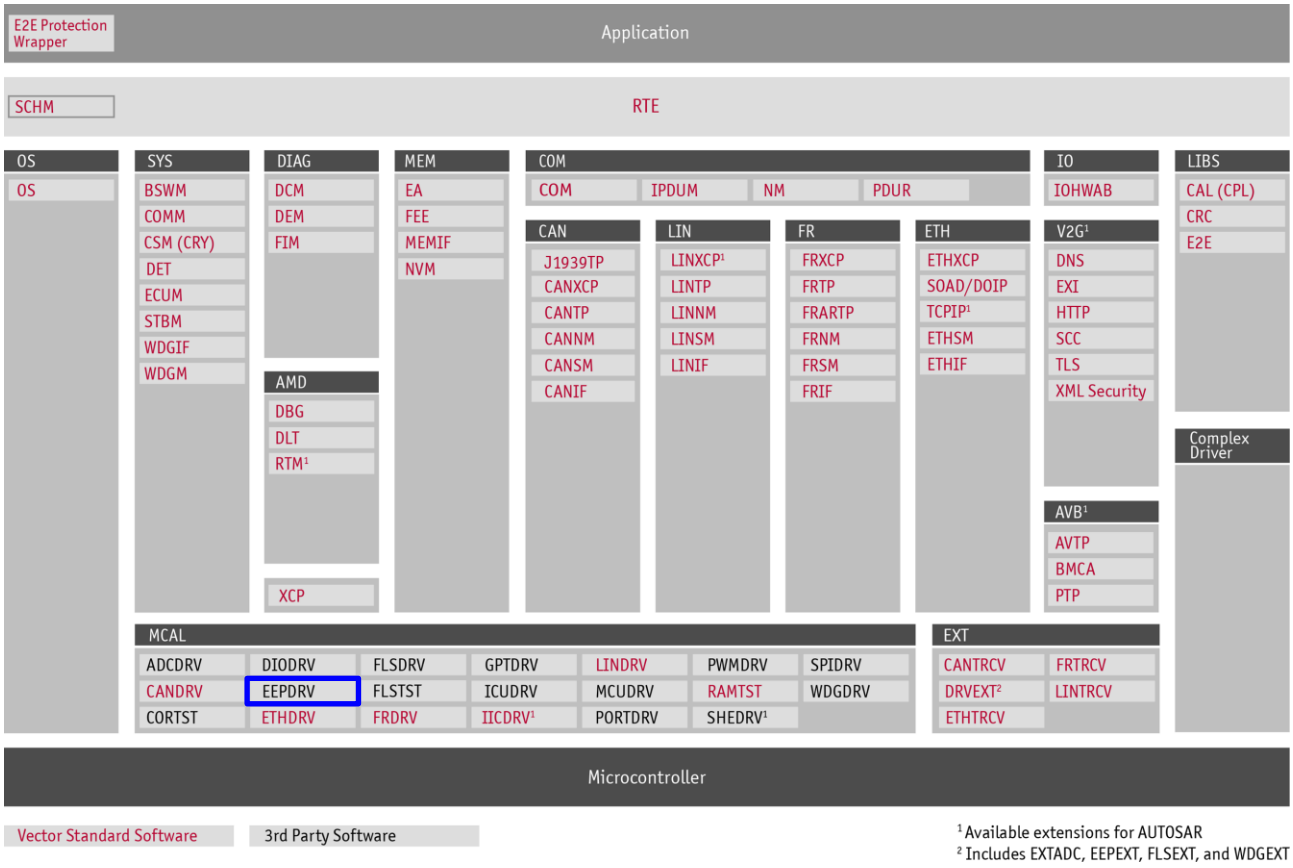

Figure 2-1 AUTOSAR 4.x Architecture Overview

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

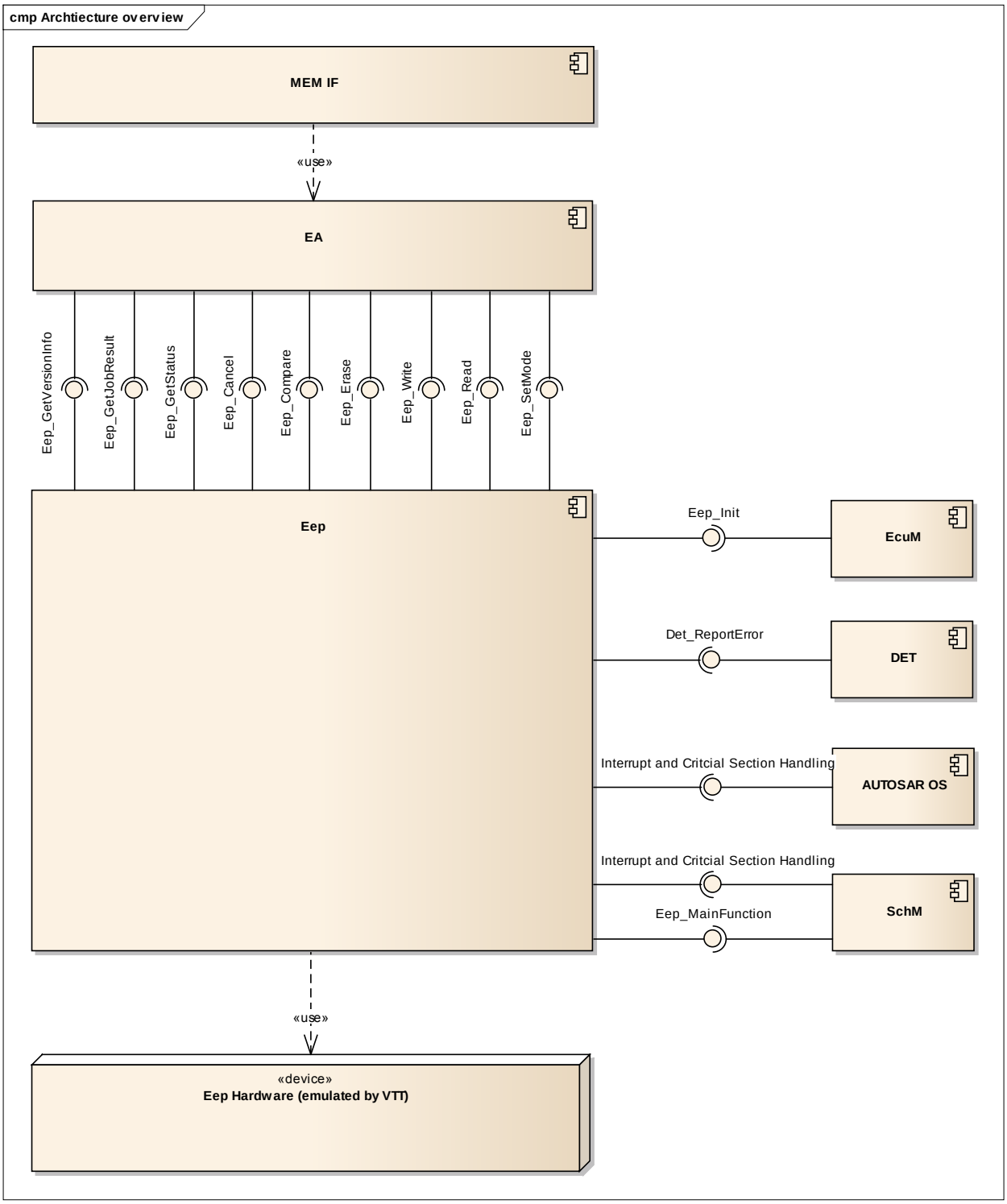

Figure 2-2 Interfaces to adjacent modules of the EEP

3 Functional Description

3.1 Features

The features listed in the following tables cover the complete functionality specified for the EEP.

The AUTOSAR standard functionality is specified in [1], the corresponding features are listed in the tables

> Table 3-1 Supported AUTOSAR standard conform features

> Table 3-2 Not supported AUTOSAR standard conform features

Vector Informatik provides further EEP functionality beyond the AUTOSAR standard. The corresponding features are listed in the table

> Table 3-3 Features provided beyond the AUTOSAR standard

The following features specified in [1] are supported:

Supported AUTOSAR Standard Conform Features
Asynchronous service for reading data from EEPROM
Asynchronous service for writing data to EEPROM
Asynchronous service for erasing data from EEPROM
Asynchronous service for comparing EEPROM data with data from memory (e.g. RAM)

Table 3-1 Supported AUTOSAR standard conform features

3.1.1 Deviations

The following features specified in [1] are not supported:

Not Supported AUTOSAR Standard Conform Features
Number of bytes that is read/written in slow/fast mode (in one mainfunction call cycle) is not configurable.
No erase/write protection implemented
Deadline monitoring for internal functionality is not implemented
Currently, there is only one configuration set (runtime) supported.

Table 3-2 Not supported AUTOSAR standard conform features

3.1.2 Additions/ Extensions

The following features are provided beyond the AUTOSAR standard:

Features Provided Beyond The AUTOSAR Standard
In addition to the existing checks required by the AUTOSAR standard, the parameter versioninfo passed to the service `Eep_GetVersionInfo()` is checked for not referencing `NULL_PTR`. If it does, the error `EEP_E_PARAM_VINFO` is reported to DET instead of `EEP_E_PARAM_POINTER`

Table 3-3 Features provided beyond the AUTOSAR standard

3.1.3 Limitations

3.1.3.1 Diagnostic Event Manager

Due to the fact that the EEP is emulated, reporting of hardware errors to the DEM is not supported. Because of compatibility reasons, the DEM has to be configured in DaVinci Configurator.

3.2 Emulation

This driver is an emulation of an EEP module.

Caution

Be careful using while loops in order to poll any status.

The user has to ensure, that the application does not block the emulation. So, within every while loop the following function call has to be called:

```
while(ANY_STATUS == temp_status)
{
  Schedule();
}
```

Use the function call Schedule() which is available once the header file of the module EEP is included.

3.3 Initialization

The EEP module is being initialized by calling `Eep_Init(&EepInitConfiguration)`. All global variables are initialized by calling `Eep_InitMemory()`. So, `Eep_InitMemory()` has to be called prior to `Eep_Init()`.

3.4 States

3.4.1 Module States

The module EEP provides the following global states:

- `MEMIF_UNINIT`: EEP is not initialized
- `MEMIF_IDLE`: Currently no active read-, write-, erase- or compare-job
- `MEMIF_BUSY`: Read-, write-, erase- or compare-job is ongoing

3.4.2 Job States

The EEP provides the following job states:

- `MEMIF_JOB_OK`: Job finished successfully
- `MEMIF_JOB_CANCELLED`: Eep_Cancel() has been called
- `MEMIF_JOB_FAILED`: Eep_SimulateError() has been called
- `MEMIF_BLOCK_INCONSISTENT`: Compare job detected inconsistencies.

3.5 Main Functions

`Eep_MainFunction` has to be called cyclically for processing read-, write-, compare- or erase-jobs.

3.6 Error Handling

3.6.1 Development Error Reporting

By default, development errors are reported to the DET using the service `Det_ReportError()` as specified in [2], if development error reporting is enabled (i.e. pre-compile parameter `EEP_DEV_ERROR_DETECT==STD_ON`).

If another module is used for development error reporting, the function prototype for reporting the error can be configured by the integrator, but must have the same signature as the service `Det_ReportError()`.

The reported EEP ID is 090.

The reported service IDs identify the services which are described in 5.1.2. The following table presents the service IDs and the related services:

Service ID	Service
0x00	Eep_Init
0x01	Eep_SetMode
0x02	Eep_Read
0x03	Eep_Write
0x04	Eep_Erase
0x05	Eep_Compare
0x06	Eep_Cancel
0x07	Eep_GetStatus
0x08	Eep_GetJobResult
0x09	Eep_MainFunction
0x0A	Eep_GetVersionInfo

Table 3-4 Service IDs

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

Error Code		Description
0x10	EEP_E_PARAM_CONFIG	Eep_Init is called with parameter ConfigPtr referencing NULL_PTR
0X11	EEP_E_PARAM_ADDRESS	API service is called with an invalid address parameter, e.g. out of memory range
0x12	EEP_E_PARAM_DATA	API service is called with a data buffer pointer referencing NULL_PTR
0x13	EEP_E_PARAM_LENGTH	API service is called with an invalid data length parameter, e.g. resulting address is out of memory range
0x14	EEP_E_PARAM_POINTER	API service called with a NULL_PTR
0x15	EEP_E_PARAM_VINFO	Eep_GetVersionInfo is called with paramter versioninfo referencing NULL_PTR
0x20	EEP_E_UNINIT	API service called without initialization
0x21	EEP_E_BUSY	API service called while driver is still busy

Table 3-5 Errors reported to DET

3.6.1.1 Parameter Checking

AUTOSAR requires that API functions check the validity of their parameters. The checks in Table 3-6 are internal parameter checks of the API functions. These checks are for development error reporting and can be en-/disabled.

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

Service \ Check	EEP_E_PARAM_CONFIG	EEP_E_PARAM_ADDRESS	EEP_E_PARAM_DATA	EEP_E_PARAM_LENGTH	EEP_E_PARAM_POINTER	EEP_E_UNINIT	EEP_E_BUSY
Eep_Init	■						■
Eep_SetMode			■			■	■
Eep_Read		■	■	■		■	■
Eep_Write		■	■	■		■	■
Eep_Erase		■		■		■	■
Eep_Compare		■	■	■		■	■
Eep_Cancel							
Eep_GetStatus							
Eep_GetJobResult							

Service \ Check	EEP_E_PARAM_CONFIG	EEP_E_PARAM_ADDRESS	EEP_E_PARAM_DATA	EEP_E_PARAM_LENGTH	EEP_E_PARAM_POINTER	EEP_E_UNINIT	EEP_E_BUSY
Eep_MainFunction						■	
Eep_GetVersionInfo					■		

Table 3-6 Development Error Reporting: Assignment of checks to services

3.6.2 Production Code Error Reporting

Info
Production errors are not supported in this emulation.

4 Integration

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

4.1 Scope of Delivery

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

4.1.1 Static Files

File Name	Description
Eep.h	The module header defines the interface of the EEP. This file must be included by upper layer software components
Eep.c	This C-source contains the implementation of the module's functionalities
DrvEep_VttCanoe01Asr.jar	This jar-file contains the generator and the validator for the DaVinci Configurator
VTTEep_bswmd.arxml	Basic Software Module Description according to AUTOSAR for VTT Emulation
Eep_bswmd.arxml	Optional Basic Software Module Description. Placeholder for real target (semiconductor manufacturer) in VTT only use case

Table 4-1 Static files

4.1.2 Dynamic Files

The dynamic files are generated by the configuration tool DaVinci Configurator.

File Name	Description
Eep_Cfg.h	The configuration-header contains the static configuration part of this module
Eep_PBcfg.c	The configuration-source contains the object independent part of the runtime configuration
Eep_VendorId_ApiInfix.c	The source contains the wrapper APIs which maps the vendor/infix specific APIs to VTTEep APIs. This file is generated in case an API-Infix is configured.
Eep_VendorId_ApiInfix.h	The header contains the vendor/infix specific API declarations. It also contains the extern declaration of `Eep_MainFunction`. This file is generated in case an API-Infix is configured.

Table 4-2 Generated files

4.2 Include Structure

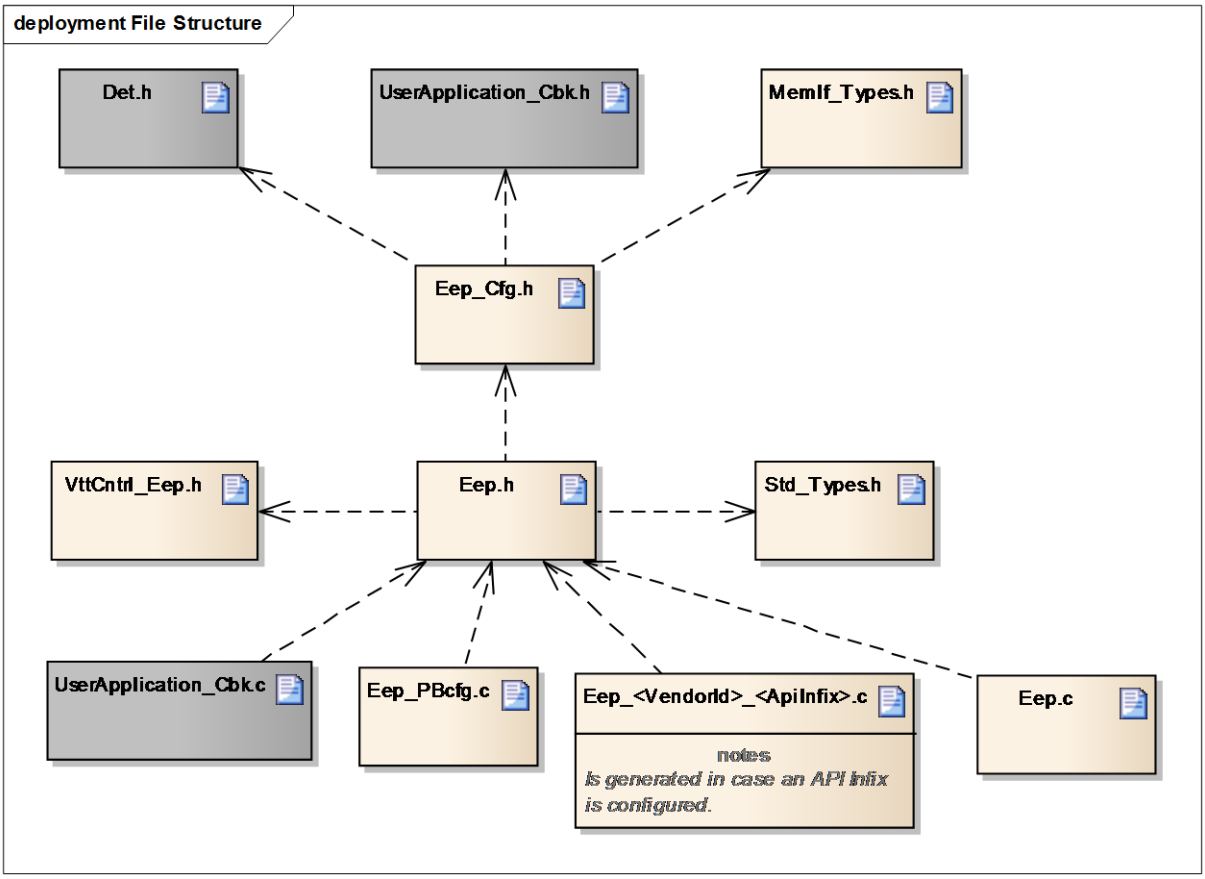

Figure 4-1 Include Structure

4.3 Dependencies on SW Modules

4.3.1 AUTOSAR OS (Optional)

An operating system can be used for task scheduling, interrupt handling, global suspend and restore of interrupts and creating of the Interrupt Vector Table.

4.3.2 DET (Optional)

The EEP module depends on the DET (by default) in order to report development errors. Detection and reporting of development errors can be enabled or disabled by the switch "Enable Development Error Detection".

4.3.3 SchM (Optional)

Beside the AUTOSAR OS the Schedule Manager provides functions that module EEP calls at begin and end of critical sections. Besides, the Schedule Manager is responsible for calling the main functions.

4.3.4 EcuM (Optional)

The EcuM cares for the initialization of the module EEP.

5 API Description

For an interfaces overview please see Figure 2-2.

5.1 Type Definitions

5.1.1 EEP Types

The types defined by the EEP are described in this chapter.

Type Name	C-Type	Description	Value Range
`Eep_AddressType`	`uint32`	Used as address offset from the configured EEPROM base address to access a certain EEPROM memory area.	0 … 10485760
`Eep_LengthType`	`uint32`	This type specifies the number of bytes to read/write/erase/compare .	0 … 10485760

Table 5-1 Type definitions

5.1.2 Imported Types

The following types are imported from the module MemIf.

Type Name	Reference
MemIf_StatusType	Defined in MemIf_Types.h
MemIf_JobResultType	Defined in MemIf_Types.h
MemIf_ModeType	Defined in MemIf_Types.h

Table 5-2 Imported Types

5.2 Services provided by EEP

5.2.1 Eep_InitMemory

Prototype	
`void Eep_InitMemory (void)`	
Parameter	
-	-
Return code	
-	-
Functional Description	
This service initializes the global variables in case the startup code does not work	
Particularities and Limitations	
> This function is synchronous. > This function is non-reentrant. > Module must not be initialized	
Expected Caller Context	
> Called during startup	

Table 5-3 [Service name]

5.2.2 Eep_Init

Prototype	
`void Eep_Init (P2CONST(Eep_ConfigType, AUTOMATIC, EEP_APPL_CONST) ConfigPtr)`	
Parameter	
`ConfigPtr`	Pointer to the configuration struct of the EEP
Return code	
-	-
Functional Description	
This service sets up the text-file used for flash emulation and initializes the module EEP. The EEPROM status is set to `MEMIF_IDLE` and the job status is set to `MEMIF_JOB_OK`.	
Particularities and Limitations	
> This function is synchronous. > This function is non re-entrant. > Module must not be initialized.	
Expected Caller Context	
> ECU State Manager or comparable software module, responsible for driver initialization after startup.	

Table 5-4 Eep_Init()

5.2.3 Eep_SetMode

Prototype	
`void` **`Eep_SetMode`** `(MemIf_ModeType Mode)`	
Parameter	
`Mode`	Switch module EEP to this job processing mode. Valid values are `MEMIF_MODE_FAST` and `MEMIF_MODE_SLOW`.
Return code	
-	-
Functional Description	
This service switches to the job processing mode passed in parameter Mode. This mode determines the amount of bytes processed in the execution of Eep_MainFunction().	
Particularities and Limitations	
> This function is synchronous. > This function is non re-entrant. > This service may only be called if the module has been initialized before. > This service may only be called while the module is in state `MEMIF_IDLE`.	
Expected Caller Context	
> Task context	

Table 5-5 Eep_SetMode()

5.2.4 Eep_Read

Prototype	

`Std_ReturnType` **`Eep_Read`**
`(`
`  MemIf_AddressType EepromAddress,`
`  uint8* DataBufferPtr,`
`  MemIf_LengthType Length`
`)`	

Parameter	
`EepromAddress`	Address in EEPROM memory, from which data should be read Min: 0 Max: `EEP_TOTAL_SIZE` - 1
`DataBufferPtr`	Reference to the buffer to which the read data shall be copied
`Length`	Amount of data in bytes to read Min: 1 Max: `EEP_TOTAL_SIZE` - `EepromAddress`
Return code	
`Std_ReturnType`	`E_OK`, success. `E_NOT_OK`, fail or request not accepted.

Functional Description
This service requests a read job, that is, the job's data (passed as parameters) is stored internally and the service returns. The job itself is processed asynchronously by executing `Eep_MainFunction()` cyclically.
Particularities and Limitations
> This function is asynchronous. > This function is non reentrant. > This service may only be called if the module has been initialized before. > This service may only be called while the module is in state `MEMIF_IDLE`.
Expected Caller Context
> Task context

Table 5-6 Eep_Read()

5.2.5 Eep_Write

Prototype

```
Std_ReturnType Eep_Write
(
  MemIf_AddressType EepromAddress,
  const uint8* DataBufferPtr,
  MemIf_LengthType Length
)
```

Parameter	
`EepromAddress`	Address in EEPROM memory, to which data should be written Min: 0 Max: `EEP_TOTAL_SIZE - 1`
`DataBufferPtr`	Reference to the buffer whose data shall be written
`Length`	Amount of data in bytes to write Min: 1 Max: `EEP_TOTAL_SIZE - EepromAddress`

Return code	
`Std_ReturnType`	`E_OK`, success. `E_NOT_OK`, fail or request not accepted.

Functional Description
This service requests a write job, that is, the job's data (passed as parameters) is stored internally and the service returns. The job itself is processed asynchronously by executing `Eep_MainFunction()` cyclically.
Particularities and Limitations
> This function is asynchronous. > This function is non reentrant. > This service may only be called if the module has been initialized before. > This service may only be called while the module is in state `MEMIF_IDLE`.
Expected Caller Context

> Task context

Table 5-7 Eep_Write()

5.2.6 Eep_Erase

Prototype	
`Std_ReturnType Eep_Erase ( MemIf_AddressType EepromAddress, MemIf_LengthType Length )`	
Parameter	
`EepromAddress`	Address in EEPROM memory, from which should be erased Min: 0 Max: `EEP_TOTAL_SIZE - 1`
`Length`	Amount of bytes to erase Min: 1 Max: `EEP_TOTAL_SIZE - EepromAddress`
Return code	
`Std_ReturnType`	`E_OK`, success. `E_NOT_OK`, fail or request not accepted.
Functional Description	
This service requests an erase job, that is, the job's data (passed as parameters) is stored internally and the service returns. The job itself is processed asynchronously by executing `Eep_MainFunction()` cyclically.	
Particularities and Limitations	
> This function is asynchronous. > This function is non reentrant. > This service may only be called if the module has been initialized before. > This service may only be called while the module is in state `MEMIF_IDLE`.	
Expected Caller Context	
> Task context	

Table 5-8 Eep_Erase()

5.2.7 Eep_Compare

Prototype	
`Std_ReturnType Eep_Compare ( MemIf_AddressType EepromAddress, const uint8* DataBufferPtr, MemIf_LengthType Length )`	
Parameter	
`EepromAddress`	Address in EEPROM memory, whose data should be compared Min: 0 Max: `EEP_TOTAL_SIZE - 1`
`DataBufferPtr`	Reference to the buffer whose data shall be compared to EEPROM
`Length`	Amount of data in bytes to compare Min: 1 Max: `EEP_TOTAL_SIZE - EepromAddress`
Return code	
`Std_ReturnType`	`E_OK`, success. `E_NOT_OK`, fail or request not accepted.
Functional Description	
This service requests a compare job, that is, the job's data (passed as parameters) is stored internally and the service returns. The job itself is processed asynchronously by executing `Eep_MainFunction()` cyclically.	
Particularities and Limitations	
> This function is asynchronous. > This function is non reentrant. > This service may only be called if the module has been initialized before. > This service may only be called while the module is in state `MEMIF_IDLE`.	
Expected Caller Context	
> Task context	

Table 5-9 Eep_Compare()

5.2.8 Eep_Cancel

Prototype	
`void Eep_Cancel ( void )`	
Parameter	
-	-
Return code	
-	-
Functional Description	
This service allows cancelling a currently processed job synchronously. New jobs can be requested right after this service has returned. In case no job is pending, the service is left without further action. A pending job is cancelled and the error notification is called (synchronously), if configured. Data sets may be incomplete, if a job is aborted.	
Particularities and Limitations	
> This function is synchronous. > This function is non reentrant. > This service may only be called if the module has been initialized before.	
Expected Caller Context	
> Task context	

Table 5-10 Eep_Cancel()

5.2.9 Eep_GetStatus

Prototype	
`MemIf_StatusType Eep_GetStatus (void)`	
Parameter	
-	-
Return code	
`MemIf_StatusType`	`MEMIF_UNINIT`, module has not been initialized before `MEMIF_IDLE`, no job is processed currently `MEMIF_BUSY`, module is busy processing a job
Functional Description	
This service returns the current status of module EEP.	
Particularities and Limitations	
> This function is synchronous. > This function is reentrant.	
Expected Caller Context	
> Task context	

Table 5-11 Eep_GetStatus()

5.2.10 Eep_GetJobResult

Prototype	
`MemIf_JobResultType` **`Eep_GetJobResult`** `(void)`	
Parameter	
-	-
Return code	
`MemIf_JobResultType`	`MEMIF_JOB_OK`, last processed job has finished successfully `MEMIF_JOB_FAILED`, last processed job has finished with errors. `MEMIF_JOB_PENDING`, job is being processed currently `MEMIF_JOB_CANCELED`, last processed job has been cancelled by `Eep_Cancel()` `MEMIF_BLOCK_INCONSISTENT`, last processed (compare) job has finished successfully, but data did not match
Functional Description	
This service returns information about the result of the latest or currently processed job.	
Particularities and Limitations	
> This function is synchronous. > This function is reentrant. > This service may only be called if the module has been initialized before. > In case module EEP has not been initialized before, this service will return MEMIF_JOB_OK.	
Expected Caller Context	
> Task context	

Table 5-12 Eep_GetJobResult()

5.2.11 Eep_GetVersionInfo

Prototype	
`void Eep_GetVersionInfo ( P2VAR(Std_VersionInfoType, AUTOMATIC, EEP_APPL_DATA) versioninfo )`	
Parameter	
`versioninfo`	Reference to the structure to which the information should be written
Return code	
-	-
Functional Description	
This function returns the version information of the module. The version information includes: > Module Id > Vendor Id > Software version numbers	
Particularities and Limitations	
> This function is synchronous. > This function is reentrant. > This function is configurable.	
Expected Caller Context	
> Task context	

Table 5-13 Eep_GetVersionInfo()

5.2.12 Eep_MainFunction

Prototype	
`void Eep_MainFunction (void)`	
Parameter	
-	-
Return code	
-	-
Functional Description	
This service has to be executed periodically for asynchronous job processing. The amount of data being processed within one call to this service is limited by the configured read- and write-sizes depending on the current mode (`MEMIF_MODE_SLOW` or `MEMIF_MODE_FAST`). After all data of a job has been processed completely, job end notification is called, or, if errors occurred, job error notification is performed.	
Particularities and Limitations	
> Module has to be initialized before	
Expected Caller Context	
> This function should be executed periodically e.g. by the Basic Software Scheduler (SchM)	

Table 5-14 Eep_MainFunction()

5.2.13 Eep_SimulateError

Prototype	
`Std_ReturnType` **`Eep_SimulateError`** `(void)`	
Parameter	
-	-
Return code	
`Std_ReturnType`	`E_OK`: Error has been successfully simulated `E_NOT_OK`: Error has not been simulated
Functional Description	
This service is used for simulating errors. If it is called and a job is currently pending, then the job is aborted and the job result is set to `MEMIF_JOB_FAILED`. If no job is pending, function returns immediately (return code `E_NOT_OK`).	
Particularities and Limitations	
> Module has to be initialized before	
Expected Caller Context	
> Task Context	

Table 5-15 Eep_SimulateError()

5.3 Services used by EEP

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

Component	API
DET	Det_ReportError

Table 5-16 Services used by the EEP

5.4 Configurable Interfaces

5.4.1 Notifications

At its configurable interfaces the EEP defines notifications that can be mapped to callback functions provided by other modules. The mapping is not statically defined by the EEP but can be performed at configuration time. The function prototypes that can be used for the configuration have to match the appropriate function prototype signatures, which are described in the following sub-chapters.

5.4.1.1 Job End Notification

Prototype	
`void <JobEndNotificationName> (void)`	
Parameter	
-	-
Return code	
-	-
Functional Description	
Job end notification functions have to adhere to this function prototype. The job end notification is called by `Eep_MainFunction()` when a job is finished successfully.	
Particularities and Limitations	
-	
Call Context	
> Task context	

Table 5-17 Job End Notification

5.4.1.2 Job Error Notification

Prototype	
`void <JobErrorNotificationName> (void)`	
Parameter	
-	-
Return code	
-	-
Functional Description	
Job error notification functions have to adhere to this function prototype. The job error notification is called by `Eep_MainFunction()` when > a job can't be finished because of errors (i.e. a call of `Eep_SimulateError()` occurred) > a (compare) job finished successfully, but data did not match > a job is cancelled by > `Eep_Cancel()`	
Particularities and Limitations	
-	
Call Context	
> Task context	

Table 5-18 Job Error Notification

6 Configuration

6.1 Configuration Variants

The EEP supports the configuration variants

> VARIANT-PRE-COMPILE

The configuration classes of the EEP parameters depend on the supported configuration variants. For their definitions please see the VTTEep_bswmd.arxml file.

6.2 Configuration with DaVinci Configurator 5

The EEP module is configured with the help of the configuration tool DaVinci Configurator 5 (CFG5). The definition of each parameter is given in the corresponding BSWMD file.

7 Glossary and Abbreviations

7.1 Glossary

Term	Description
CANoe	Tool for simulation and testing of networks and electronic control units.
DaVinci Configurator	Configuration and generation tool for MICROSAR components

Table 7-1 Glossary

7.2 Abbreviations

Abbreviation	Description
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSW	Basis Software
DEM	Diagnostic Event Manager
DET	Development Error Tracer
EA	Eeprom Abstraction
ECU	Electronic Control Unit
EcuM	ECU State Manager
EEP	Eeprom driver
MemIf	Memory Inteface
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
SchM	Basic Software Scheduler
VTT	vVIRTUALtarget

Table 7-2 Abbreviations

8 Contact

Visit our website for more information on

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

www.vector.com