Document Title	Specification of Memory Driver
Document Category	Software specification (SWS)
Document UID	1018
Document Status	published
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	R26-11

Document Change History			
Date	**Release**	**Changed by**	**Description**
2026-11-26	R26-11	AUTOSAR Release Management	• Added support for external memory using I2C • Added configuration parameter to indicate if the driver is used in the HSM or in the host • Removed API Mem_GetVersionInfo and the extended production errors • Editorial changes
2025-11-27	R25-11	AUTOSAR Release Management	• Clarification of the usage of the terms "direct" and "indirect"
2024-11-27	R24-11	AUTOSAR Release Management	• Removed all draft markings • Added MemUseFuncPtrTable, MemNamePrefix and MemInvocation • Minor corrections and bugfixes
2023-11-23	R23-11	AUTOSAR Release Management	• Fixed inconsistencies
2022-11-24	R22-11	AUTOSAR Release Management	• Renamed MEM_JOB_PENDING into MEM_E_JOB_PENDING for DET errors • Additional DET checks added • Minor corrections and bugfixes • Editorial changes
2021-11-25	R21-11	AUTOSAR Release Management	• Initial release

Disclaimer

This work (specification and/or software implementation) and the material contained in it, as released by AUTOSAR, is for the purpose of information only. AUTOSAR and the companies that have contributed to it shall not be liable for any use of the work.

The material contained in this work is protected by copyright and other types of intellectual property rights. The commercial exploitation of the material contained in this work requires a license to such intellectual property rights.

This work may be utilized or reproduced without any modification, in any form or by any means, for informational purposes only. For any other purpose, no part of the work may be utilized or reproduced, in any form or by any means, without permission in writing from the publisher.

The work has been developed for automotive applications only. It has neither been developed, nor tested for non-automotive applications.

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

Table of Contents

1 Introduction and Functional Overview 6
2 Definition of acronyms and abbreviations 7
2.1 Acronyms and abbreviations . . . 7
2.2 Definition of terms . . . 7
2.3 Physical Segmentation . . . 9
3 Related Documentation 10
3.1 Input Documents & Related Standards and Norms . . . 10
3.2 Related Specification . . . 10
4 Constraints and Assumptions 11
4.1 Limitations . . . 11
4.1.1 General Limitations . . . 11
4.1.2 Implementation Limitations . . . 11
4.1.3 Memory Test Capabilities . . . 11
4.2 Applicability to Car Domains . . . 11
5 Dependencies to Other Modules 12
5.1 SPI and I2C Drivers . . . 12
6 Requirements Tracing 13
7 Functional Specification 15
7.1 Overview . . . 15
7.1.1 Key Aspects . . . 15
7.2 Functional Elements . . . 15
7.2.1 Job Management . . . 15
7.2.1.1 Job Status/Job Results . . . 16
7.2.1.2 Job Suspend/Resume . . . 17
7.2.2 Hardware Specific Services . . . 18
7.2.3 Multi Memory Device Instance Support . . . 18
7.2.4 Dynamic Driver Activation . . . 18
7.2.5 Service Invocation . . . 19
7.2.5.1 Direct Invocation . . . 19
7.2.5.2 Indirect Invocation . . . 19
7.2.6 Binary Image Format . . . 19
7.2.6.1 Header . . . 20
7.2.6.2 Service Function Pointer Table . . . 23
7.2.6.3 Delimiter . . . 23
7.2.7 Optional Services . . . 24
7.3 Module Handling . . . 24
7.3.1 Initialization . . . 24
7.3.2 Scheduling . . . 24

7.3.3 ECC Handling . . . 24
7.4 General Design Rules . . . 25
7.4.1 Address Alignment . . . 25
7.4.2 64-Bit Support . . . 25
7.5 Error Classification . . . 26
7.5.1 Development Errors . . . 26
7.5.2 Runtime Errors . . . 26
7.5.3 Production Errors . . . 26
7.6 Security Events . . . 27
8 API Specification 28
8.1 Imported Types . . . 28
8.2 Type Definitions . . . 28
8.2.1 Mem_AddressType . . . 28
8.2.2 Mem_ConfigType . . . 29
8.2.3 Mem_DataType . . . 29
8.2.4 Mem_InstanceIdType . . . 29
8.2.5 Mem_LengthType . . . 30
8.2.6 Mem_HwServiceIdType . . . 30
8.3 Function Definitions . . . 30
8.3.1 Synchronous Functions . . . 31
8.3.1.1 Mem_Init . . . 31
8.3.1.2 Mem_DeInit . . . 32
8.3.1.3 Mem_GetJobResult . . . 32
8.3.1.4 Mem_Suspend . . . 33
8.3.1.5 Mem_Resume . . . 34
8.3.1.6 Mem_PropagateError . . . 35
8.3.2 Asynchronous Functions . . . 36
8.3.2.1 Mem_Read . . . 36
8.3.2.2 Mem_Write . . . 37
8.3.2.3 Mem_Erase . . . 39
8.3.2.4 Mem_BlankCheck . . . 40
8.3.2.5 Mem_HwSpecificService . . . 41
8.4 Callback Notifications . . . 42
8.5 Scheduled Functions . . . 42
8.5.1 Mem_MainFunction . . . 43
8.6 Expected Interfaces . . . 43
8.6.1 Mandatory Interfaces . . . 43
8.6.2 Optional Interfaces . . . 43
8.6.3 Configurable Interfaces . . . 44
9 Sequence Diagrams 45
9.1 Hardware Specific Error Handling . . . 45
9.2 ECC Handling Example Sequence . . . 46

10 Configuration Specification 47
10.1 How to Read this Chapter . . . 47
10.2 Containers and Configuration Parameters . . . 47
10.2.1 Mem . . . 48
10.2.2 MemInstance . . . 53
10.3 Published Information . . . 59
A Change history of AUTOSAR traceable items 61
A.1 Traceable item history of this document according to AUTOSAR Release R26-11 . . . 61
A.1.1 Added Specification Items in R26-11 . . . 61
A.1.2 Changed Specification Items in R26-11 . . . 61
A.1.3 Deleted Specification Items in R26-11 . . . 61
A.2 Traceable item history of this document according to AUTOSAR Release R25-11 . . . 61
A.2.1 Added Specification Items in R25-11 . . . 61
A.2.2 Changed Specification Items in R25-11 . . . 62
A.2.3 Deleted Specification Items in R25-11 . . . 62
A.3 Traceable item history of this document according to AUTOSAR Release R24-11 . . . 62
A.3.1 Added Specification Items in R24-11 . . . 62
A.3.2 Changed Specification Items in R24-11 . . . 62
A.3.3 Deleted Specification Items in R24-11 . . . 63
A.4 Traceable item history of this document according to AUTOSAR Release R23-11 . . . 63
A.4.1 Added Specification Items in R23-11 . . . 63
A.4.2 Changed Specification Items in R23-11 . . . 63
A.4.3 Deleted Specification Items in R23-11 . . . 63
B Not applicable requirements 64

1 Introduction and Functional Overview

This specification describes the functionality, API and the configuration for the AUTOSAR Basic Software module Memory Driver (Mem).

The Memory Driver provides the basic services for accessing different kinds of memory devices like reading, writing, erasing and blank checking.

Although flash memory is still the most common non-volatile memory technology, the Memory Driver specification considers all relevant memory device technologies like EEPROM, phase change memory (PCM) and ferro electric RAM.
To harmonize the memory access for the upper layers, the Memory Driver specification also covers access of RAM. Aside from microcontroller internal memory devices, the Memory Device specification can also be applied to external memory devices attached, e.g. via a serial peripheral interface.

In contrast to the Flash and EEPROM Driver specification, the Memory Driver specification explicitly covers also code memory access to support new use cases like background OTA software update which require code memory access.

Figure 1.1 shows an example architectural overview with different Memory Drivers and upper layers:

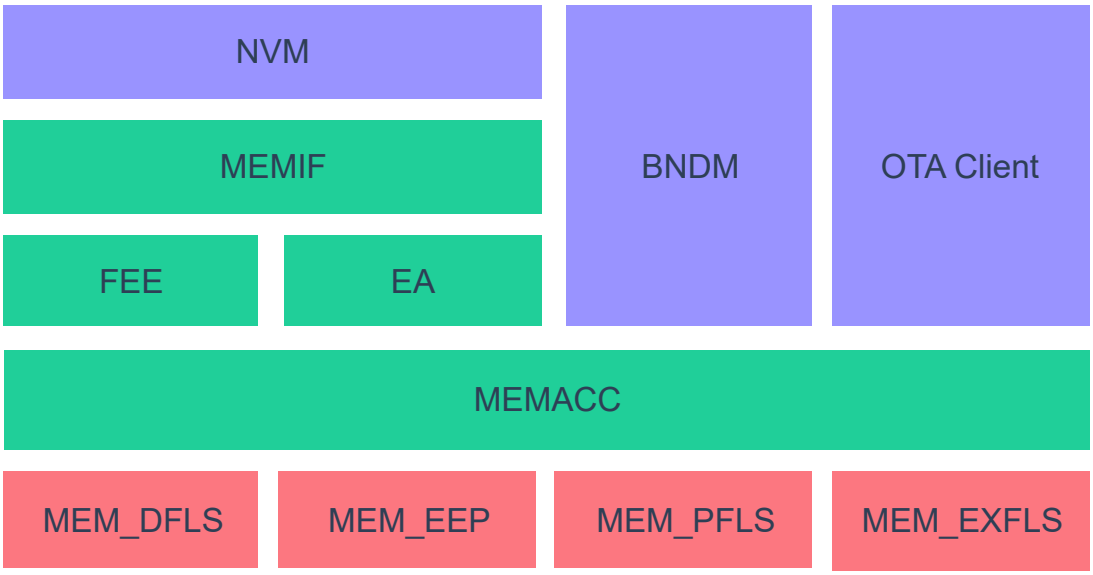


Figure 1.1: MemAcc Architecture Example

2 Definition of acronyms and abbreviations

2.1 Acronyms and abbreviations

The table below contains all acronyms, abbreviations or initialisms that are component-specific and not available in the AUTOSAR glossary [1].

Acronym/ Abbreviation	Description
BndM	BulkNon-VolatileDataManager
ECC	Error Correction Code
HSM	Hardware Security Module - dedicated security MCU core
OTA	Over The Air - general term for wireless communication between OEM backend and vehicle
RWW	Read While Write - capability of a memory device to perform a read operation in one memory bank while at the same time a write/erase operation takes place in another bank
SOTA	Software Over The Air - remote software update using wireless communication

Table 2.1: Acronyms and Abbreviations

The table below contains all document-independant acronyms or abbreviations that are defined in the AUTOSAR glossary [1].

Acronym/ Abbreviation	Description
ABI	Application Binary Interface
I2C	Inter-Integrated Circuit
SPI	Serial Peripheral Interface

Table 2.2: Acronyms and Abbreviations used from AUTOSAR Glossary

2.2 Definition of terms

The table below contains all definitions of terms that are used in this document.

Terms	Description
Address Area	Contiguous memory area in the logical address space. Typically multiple physical memory sectors are combined to one logical address area.
Bank	Group of sector batches. In case a memory technology is segmented in sectors, a bank is an instance of a sector batch group in which no read-while-write operation is permitted. In case of a flash memory device, this typically maps to an individual flash controller.
Job Request	Memory access request by an upper layer module for an address area.
Memory Device	Group of banks.
Page Burst	Aggregated access of memory pages for improved performance. In case a memory device technology has a physical segmentation, some memory devices provide an optimized access method to read or write multiple pages at a time. Page burst denotes the aggregation of memory pages used for the access optimization.
Sector	Smallest erasable memory unit (in bytes). Some memory device technologies require an explicit physical erase operation before the memory can be written. A sector defines the minimum size of such an erase unit. Depending on the memory device, sectors can be either uniform- or variable-sized.
Sector Batch	Aggregation of sectors with uniform size. Logical aggregation of contiguous sectors with the same size.
Sector Burst	Aggregation of sectors for improved erase performance. In case a memory technology needs a physical erase operation, some devices provide an erase performance optimization by erasing an aggregation of sectors in one step.
Sub Address Area	Contiguous memory area in the logical address space mapped to a sector batch of one memory device.
Write Page	Smallest writeable unit of a memory device (in bytes). Some memory device technologies must be accessed considering a physical segmentation. Hence a byte-wise access is not possible for all memory device technologies. This term defines the minimum size that needs to be written in one access.

Table 2.3: Definition of terms in the scope of this Document

2.3 Physical Segmentation

Figure 2.1 gives an overview of the physical segmentation and the according technical terms:

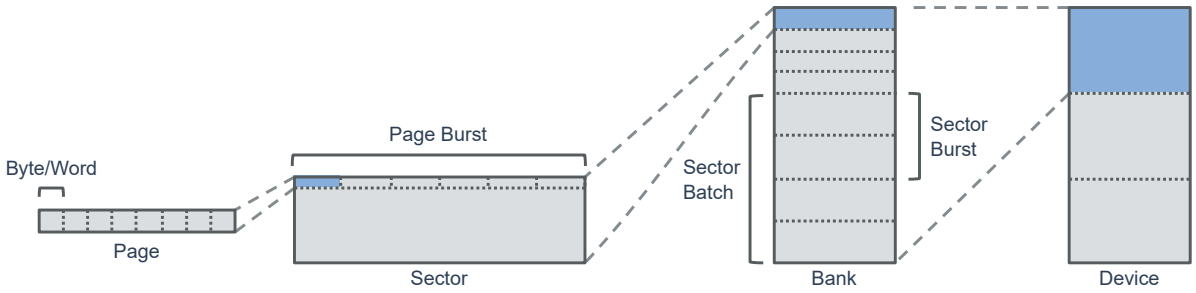


Figure 2.1: Overview of Physical Segmentation

3 Related Documentation

3.1 Input Documents & Related Standards and Norms

[1] Glossary
AUTOSAR_FO_TR_Glossary

[2] General Specification of Adaptive Platform
AUTOSAR_AP_SWS_General

[3] Requirements on Memory Hardware Abstraction Layer
AUTOSAR_CP_RS_MemoryHWAbstractionLayer

[4] General Requirements on Basic Software Modules
AUTOSAR_CP_RS_BSWGeneral

[5] General Specification of Basic Software Modules
AUTOSAR_CP_SWS_BSWGeneral

3.2 Related Specification

AUTOSAR provides a general specification [2] which is also applicable for this functional cluster. It shall be considered an additional and required specification for implementing this functional cluster.

4 Constraints and Assumptions

The following constraints apply for the Mem driver:

- The Mem driver only works on physical segments, i.e. pages/page bursts and sectors/sector bursts for flash memory.
- The Mem driver expects any requests to be aligned to the physical segmentation, i.e. pages and sectors for flash memory.

Due to this constraints, Mem drivers can only be used in combination with the Memory Access Module.

4.1 Limitations

4.1.1 General Limitations

Block based memory devices like NAND flash devices are out of scope of this specification.

4.1.2 Implementation Limitations

The following implementation limitations apply for the Mem driver:

- The Mem driver does not provide any strategy for write accesses smaller than the physical segmentation, i.e. pages and sectors for flash memory since it does not use any internal buffers.
- The Mem driver does not provide mechanisms for providing data integrity (e.g. checksums, redundant storage, etc.).
- The size of Mem driver service requests is limited to 32-Bits to avoid a resource overhead for 32-Bit microcontroller.

4.1.3 Memory Test Capabilities

The Mem driver does not provide any general APIs for performing background memory tests since the memory test capabilities are very hardware dependent. To implement memory tests, the Mem driver's hardware specific request service API can be used.

4.2 Applicability to Car Domains

The Mem driver can be used in any domain application that needs memory access to either store data or perform a software update.

5 Dependencies to Other Modules

5.1 SPI and I2C Drivers

Typically, external memory devices are connected via a serial bus like SPI or I2C. To access such devices, a driver for the SPI or I2C peripheral in the microcontroller is needed to access the external memory device. Depending on the implementation of the Mem driver, the SPI or I2C driver might be part of the Mem driver or the Mem driver uses an existing MCAL SPI or I2C handler/driver.

6 Requirements Tracing

The following tables reference the requirements specified in [3], [4] and [5] and links to the fulfillment of these. Please note that if column “Satisfied by” is empty for a specific requirement this means that this requirement is not fulfilled by this document.

Requirement	Description	Satisfied by
[SRS_BSW_00004]	All Basic SW Modules shall perform a pre-processor check of the versions of all imported include files	[SWS_Mem_00033]
[SRS_BSW_00167]	All AUTOSAR Basic Software Modules shall provide configuration rules and constraints to enable plausibility checks	[SWS_Mem_00033]
[SRS_BSW_00172]	The scheduling strategy that is built inside the Basic Software Modules shall be compatible with the strategy used in the system	[SWS_Mem_00066]
[SRS_BSW_00323]	All AUTOSAR Basic Software Modules shall check passed API parameters for validity	[SWS_Mem_00004] [SWS_Mem_00005] [SWS_Mem_00006] [SWS_Mem_00007] [SWS_Mem_00009] [SWS_Mem_00010] [SWS_Mem_00011] [SWS_Mem_00012] [SWS_Mem_00013] [SWS_Mem_00015] [SWS_Mem_00016] [SWS_Mem_00017] [SWS_Mem_00018] [SWS_Mem_00020] [SWS_Mem_00022] [SWS_Mem_00023] [SWS_Mem_00024] [SWS_Mem_00025] [SWS_Mem_00026] [SWS_Mem_00027] [SWS_Mem_00033] [SWS_Mem_00072] [SWS_Mem_00090] [SWS_Mem_00091] [SWS_Mem_00092]
[SRS_BSW_00347]	A Naming seperation of different instances of BSW drivers shall be in place	[SWS_Mem_00074]
[SRS_BSW_00385]	List possible error notifications	[SWS_Mem_00052]
[SRS_BSW_00386]	The BSW shall specify the configuration and conditions for detecting an error	[SWS_Mem_00004] [SWS_Mem_00005] [SWS_Mem_00006] [SWS_Mem_00007] [SWS_Mem_00009] [SWS_Mem_00010] [SWS_Mem_00011] [SWS_Mem_00012] [SWS_Mem_00013] [SWS_Mem_00020] [SWS_Mem_00072] [SWS_Mem_00087] [SWS_Mem_00090] [SWS_Mem_00091] [SWS_Mem_00092]
[SRS_BSW_00414]	Init functions shall have a pointer to a configuration structure as single parameter	[SWS_Mem_00087] [SWS_Mem_10000]
[SRS_BSW_00415]	Interfaces which are provided exclusively for one module shall be separated into a dedicated header file	[SWS_Mem_10020] [SWS_Mem_10021] [SWS_Mem_10022]
[SRS_MemHwAb_-14031]	The FEE and EA modules shall provide a service that allows canceling an ongoing asynchronous operation	[SWS_Mem_00079] [SWS_Mem_00080] [SWS_Mem_00081] [SWS_Mem_00082] [SWS_Mem_00083] [SWS_Mem_00084]
[SRS_MemHwAb_-14034]	MemAcc module shall allow the configuration of the priority for different logical address areas	[SWS_Mem_10024] [SWS_Mem_10025]



△

Requirement	Description	Satisfied by
[SRS_MemHwAb_-14036]	Mem driver shall be statically configurable	[SWS_Mem_00035]
[SRS_MemHwAb_-14037]	MemAcc module and Mem driver shall provide an interface for initialization	[SWS_Mem_00001] [SWS_Mem_10008] [SWS_Mem_10018]
[SRS_MemHwAb_-14038]	MemAcc module and Mem driver shall provide asynchronous memory access functions	[SWS_Mem_10010] [SWS_Mem_10012] [SWS_Mem_10013] [SWS_Mem_10014] [SWS_Mem_10016] [SWS_Mem_10017]
[SRS_MemHwAb_-14039]	MemAcc module and Mem driver shall support optional services	[SWS_Mem_00070] [SWS_Mem_10012] [SWS_Mem_10013] [SWS_Mem_10014] [SWS_Mem_10016] [SWS_Mem_10017]
[SRS_MemHwAb_-14040]	MemAcc module and Mem driver shall provide a synchronous status function	[SWS_Mem_00029] [SWS_Mem_00030] [SWS_Mem_00031] [SWS_Mem_00063] [SWS_Mem_00067] [SWS_Mem_00076] [SWS_Mem_00077] [SWS_Mem_00078] [SWS_Mem_10011]
[SRS_MemHwAb_-14042]	MemAcc module shall support multiple Mem drivers for different types of memory	[SWS_Mem_10026]
[SRS_MemHwAb_-14043]	Mem driver and shall support multiple instances of the same memory device	[SWS_Mem_00060] [SWS_Mem_10004]
[SRS_MemHwAb_-14045]	MemAcc module and Mem driver shall provide measures for dynamic driver activation	[SWS_Mem_00038] [SWS_Mem_00039] [SWS_Mem_00040] [SWS_Mem_00041] [SWS_Mem_00042] [SWS_Mem_00043] [SWS_Mem_00044] [SWS_Mem_00045] [SWS_Mem_00046] [SWS_Mem_00048] [SWS_Mem_00051] [SWS_Mem_00073] [SWS_Mem_00093] [SWS_Mem_00094] [SWS_Mem_00095] [SWS_Mem_00096]
[SRS_MemHwAb_-14046]	MemAcc module and Mem driver shall provide support for 64-Bit address range	[SWS_Mem_00036] [SWS_Mem_00037] [SWS_Mem_10002] [SWS_Mem_10007]
[SRS_MemHwAb_-14047]	MemAcc module shall provide optional support for the initialization and main function triggering of memory drivers	[SWS_Mem_00001]
[SRS_MemHwAb_-14049]	Mem driver shall use a standard binary format for dynamic driver activation	[SWS_Mem_00038] [SWS_Mem_00041] [SWS_Mem_00042] [SWS_Mem_00043] [SWS_Mem_00044] [SWS_Mem_00045] [SWS_Mem_00046] [SWS_Mem_00048] [SWS_Mem_00051] [SWS_Mem_00073] [SWS_Mem_00093] [SWS_Mem_00094] [SWS_Mem_00095] [SWS_Mem_00096]
[SRS_MemHwAb_-14050]	Mem driver shall handle only one job at one time	[SWS_Mem_00057] [SWS_Mem_00059]
[SRS_MemHwAb_-14051]	Mem driver shall not buffer data	[SWS_Mem_10003]
[SRS_MemHwAb_-14053]	Mem driver shall provide a function to a system ECC handle to propagate ECC errors	[SWS_Mem_00061] [SWS_Mem_10015]
[SRS_MemHwAb_-14056]	MemAcc module and Mem driver shall provide a generic function to access the hardware specific functionalities	[SWS_Mem_00053] [SWS_Mem_10017]

Table 6.1: Requirements Tracing

7 Functional Specification

This chapter defines the behavior of the Mem driver.
The API of the module is defined in chapter 8, while the configuration is defined in 10.

7.1 Overview

The Mem driver's task is the low level memory access based on the physical segmentation of the underlying memory device technology.

The API of the Mem driver is memory device technology and context independent and thus provides a memory device agnostic interface to the Memory Access Module upper layer.

All high level functionality like cross-segment operations are handled by the Memory Access Module to keep the complexity and the footprint of the Mem drivers as small as possible.

7.1.1 Key Aspects

- Harmonized, memory device agnostic upper layer interface
- Support of code memories
- Multi instance support
- Dynamic activation of Mem drivers
- 64-Bit device support
- Support of optional memory services
- Generic interface for providing memory device specific services

7.2 Functional Elements

7.2.1 Job Management

[SWS_Mem_00057]

Upstream requirements: SRS_MemHwAb_14050

⌈A Mem driver instance shall allow only one job request at a time.⌋

Note: Since the MemAcc module takes care about the job management of the upper layer, there is no need to implement a job queue in the Mem driver.

[SWS_Mem_00059]

Upstream requirements: SRS_MemHwAb_14050

⌈If the Mem driver is not able to process a job request, e.g. due to a pending request or due to an invalid parameter, the job request shall be rejected by an `E_NOT_OK` return code.⌋

[SWS_Mem_00066]

Upstream requirements: SRS_BSW_00172

⌈All job requests triggered by asynchronous Mem driver services shall be executed within the `Mem_MainFunction`.⌋

[SWS_Mem_00063]

Upstream requirements: SRS_MemHwAb_14040

⌈If the memory hardware provides ECC information, the Mem driver shall check for uncorrectable ECC errors and set the job result code to `MEM_ECC_UNCORRECTED` and abort the current job processing.⌋

7.2.1.1 Job Status/Job Results

The Mem driver provides the `Mem_GetJobResult` service to retrieve the current job processing status as well as the result of the last processed job. Once a new job is processed, the result of the last processed job gets overwritten by the current status/result.

[SWS_Mem_00029]

Upstream requirements: SRS_MemHwAb_14040

⌈The Mem driver shall keep track of the job processing and the result of the last processed job.⌋

[SWS_Mem_00030]

Upstream requirements: SRS_MemHwAb_14040

⌈Once a job request is accepted, the job processing status shall be set to `MEM_JOB_PENDING`.⌋

[SWS_Mem_00067]

Upstream requirements: SRS_MemHwAb_14040

⌈After a job was successfully processed, the job processing status shall be set to `MEM_JOB_OK`.⌋

[SWS_Mem_00031]

Upstream requirements: SRS_MemHwAb_14040

⌈In case a pending job was not able to complete, the job processing status shall be set to `MEM_JOB_FAILED`.⌋

[SWS_Mem_00076]

Upstream requirements: SRS_MemHwAb_14040

⌈In case the job processing was completed but the results of the last Mem job didn't meet the expected result, e.g. a blank check operation was applied on a non-blank memory area, the job result shall be set to `MEM_INCONSISTENT`.⌋

[SWS_Mem_00077]

Upstream requirements: SRS_MemHwAb_14040

⌈If the memory hardware supplies ECC information, the Mem driver shall check for correctable ECC errors and, in the case of correctable ECC errors, continue with the job processing. After completion of the job, the job result shall be set to `MEM_ECC_CORRECTED`.⌋

[SWS_Mem_00078]

Upstream requirements: SRS_MemHwAb_14040

⌈In case the last memory operation didn't complete due to an uncorrectable ECC error, the job result shall be set to `MEM_ECC_UNCORRECTED`.⌋

7.2.1.2 Job Suspend/Resume

To reduce delays for prioritizing memory accesses, the Mem driver provides services for suspending/resuming memory operations.

[SWS_Mem_00082]

Upstream requirements: SRS_MemHwAb_14031

⌈In case a memory device provides a hardware-based suspend/resume mechanism, the Mem driver shall utilize the hardware capabilities to implement the `Mem_Suspend` and `Mem_Resume` services. If no hardware suspend/resume support is available, the `Mem_Suspend` and `Mem_Resume` services shall return `E_MEM_SERVICE_NOT_AVAIL`.⌋

Note: For the hardware-based suspend/resume operation, it is expected, that the suspend/resume operation is completely handled by the memory hardware (no persistance of any state information needed in software, etc.) and there is no restriction regarding the intermediate operation that is issued while the device is in suspend state. If the hardware does not provide such features, prioritization is only possible based on the physical memory segmentation.

7.2.2 Hardware Specific Services

To support memory device specific services, Mem driver implementations can provide hardware specific services.
Figure 9.1 shows a typical use case for using a hardware specific service to implement a hardware specific error handling.

[SWS_Mem_00053]

Upstream requirements: SRS_MemHwAb_14056

⌈All hardware specific routines shall be accessed by the `Mem_HwSpecificService()` service.⌋

7.2.3 Multi Memory Device Instance Support

[SWS_Mem_00060]

Upstream requirements: SRS_MemHwAb_14043

⌈The Mem driver implementation shall support multiple instances of the same memory device.⌋

Note: For the OTA software update use case, multiple memory devices of the same type are used to expand the memory resources.

7.2.4 Dynamic Driver Activation

To support use cases where the Mem driver shall not be permanently available, i.e. to prevent accidental overwriting of memory areas, the Mem driver shall provide measures that allow dynamic activation or download of Mem drivers.

There are two options to support this use case:

- Download of a Mem driver binary image into RAM
- Copy/decrypt a Mem driver ROM binary image into RAM

Both options require that the Mem driver is compiled as a separate binary. In case the Mem driver binary is part of the application image, the Mem driver binary image should to be encrypted, e.g. by a XOR operation and only be decrypted in RAM if the Mem driver shall be activated. The download or decryption operation of the Mem driver must be implemented as a CDD and is not covered by this specification.

To provide interoperability of different Mem driver implementations, the Mem drivers need to build according to standardized binary image format. The Mem driver binary image format is defined in chapter 7.2.6.

[SWS_Mem_00039]

Upstream requirements: SRS_MemHwAb_14045

⌈If built as a separate image, the Mem driver shall be completely self contained, i.e. it must not call any library or any other external functions.⌋

Note: Since the Mem driver is running in a different context than it was built, it must not depend on any external functions or libraries. Therefore, the development error detection mechanisms is not available in case the Mem driver is built as a separate binary.

7.2.5 Service Invocation

[SWS_Mem_00038]

Upstream requirements: SRS_MemHwAb_14045, SRS_MemHwAb_14049

⌈The Mem driver shall provide two ways for the service function invocation:

- Direct service invocation
- Indirect service invocation by a function pointer table

⌋

Note: Some use cases like OTA background software update might require dynamic Mem driver activation using the indirect service invocation while simple use cases like NVM data storage want to use direct service invocation.

7.2.5.1 Direct Invocation

In case the Mem driver service functions are directly invoked, the Mem driver is linked with the application software and the Mem driver service functions are directly called by the MemAcc module.

7.2.5.2 Indirect Invocation

The indirect service invocation is needed for the dynamic driver activation feature described in chapter 7.2.4.

7.2.6 Binary Image Format

This chapter specifies the Mem driver binary image format. The binary image format is split into five parts:

- Header, containing management information

- Service function pointer table
- Service function implementation
- Delimiter

Figure 7.1 shows an overview of the Mem driver binary image:

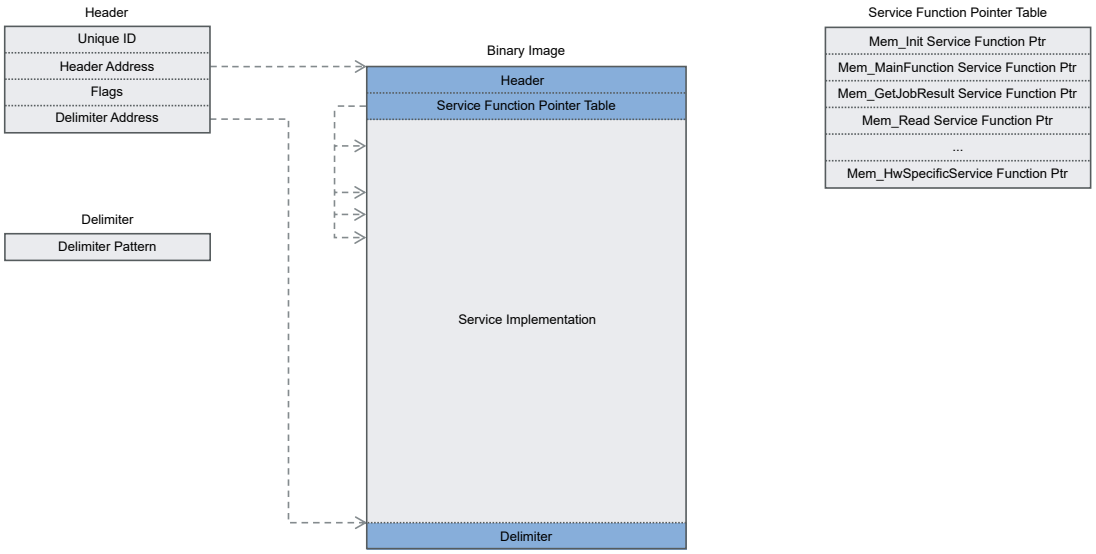


Figure 7.1: Binary Image Format Overview

[SWS_Mem_00040]

Upstream requirements: SRS_MemHwAb_14045

⌈Since Mem drivers are always hardware/CPU specific, the byte order of data fields and address information within the Mem driver binary shall follow the standard CPU byte order.⌋

Note: Avoid unnecessary address/data format conversions.

7.2.6.1 Header

[SWS_Mem_00041]

Upstream requirements: SRS_MemHwAb_14045, SRS_MemHwAb_14049

⌈The header part of the Mem driver binary shall follow the structure defined in [SWS_Mem_00093]. The size of the address information depends on the configured `MemAcc_AddressType` (see [SWS_MemAcc_10000] - 32 or 64 bit).⌋

[SWS_Mem_00093]

Upstream requirements: SRS_MemHwAb_14045, SRS_MemHwAb_14049

⌈

Offset [bytes]	*Size [bytes]*	*Name*	*Description*
0	8	Unique ID	Mem driver unique identifier - used to validate Mem driver version information, etc.
8	8	Flags	Flags used for additional development error detection.
16	4/8	Header address	Start address of Mem driver image header - used to verify consistency of Mem driver RAM buffer/ROM image location.
20/24	4/8	Delimiter address	Address of Mem driver binary image delimiter pattern - used to validate if the binary is complete.

Header Binary Format

⌋

7.2.6.1.1 Unique ID

[SWS_Mem_00042]

Upstream requirements: SRS_MemHwAb_14045, SRS_MemHwAb_14049

⌈The unique identifier is used for unique identification of Mem drivers. The format of the unique identifier shall follow the structure defined in [SWS_Mem_00094].⌋

[SWS_Mem_00094]

Upstream requirements: SRS_MemHwAb_14045, SRS_MemHwAb_14049

⌈

Offset [bytes]	*Size [bytes]*	*Name*	*Description*
0	2	ABI version	BCD-encoded Mem driver binary interface version. Any change of the interface version will also require an update of the MemAcc module.
2	2	Vendor ID	Standard AUTOSAR vendor identification.
4	4	Driver ID	Vendor specific driver identification.

Unique ID Format

⌋

[SWS_Mem_00043]

Upstream requirements: SRS_MemHwAb_14045, SRS_MemHwAb_14049

⌈The ABI version of a Mem driver following this specification shall be 0001.⌋

7.2.6.1.2 Header Address

[SWS_Mem_00044]

Upstream requirements: SRS_MemHwAb_14045, SRS_MemHwAb_14049

⌈The header address is used for development error checks to verify the consistency of the linked Mem driver binary image with the location of the RAM buffer which is used for execution of the Mem driver.
In case of a relocatable/position independent Mem driver binary, the header address shall be set to zero, otherwise, the header address shall hold the physical start address of the Mem driver binary.⌋

7.2.6.1.3 Flags

[SWS_Mem_00045]

Upstream requirements: SRS_MemHwAb_14045, SRS_MemHwAb_14049

⌈The flag part of the Mem driver header is a bit-field which holds additional information for develop error checks.

The format of the flag bit-field shall follow the structure defined in [SWS_Mem_00095].⌋

[SWS_Mem_00095]

Upstream requirements: SRS_MemHwAb_14045, SRS_MemHwAb_14049

⌈

Offset [bits]	*Size [bits]*	*Name*	*Description*
0	1	Relocatable binary	If this bit is set, the Mem driver binary is relocatable and no address consistency checks can be done.
1	31	Reserved	Reserved by this specification - shall be 0.
32	32	Vendor specific	Vendor specific flags.

Flag Format

⌋

7.2.6.1.4 Delimiter Address

[SWS_Mem_00046]

Upstream requirements: SRS_MemHwAb_14045, SRS_MemHwAb_14049

⌈The delimiter address part of the Mem driver header is used for development error checks to verify that the Mem driver binary is complete by checking the delimiter pattern linked to the end of the Mem driver binary.
In case of a relocatable/position independent Mem driver binary, the delimiter address shall be set to zero, otherwise, the delimiter address shall hold the physical address of the delimiter pattern.⌋

7.2.6.2 Service Function Pointer Table

[SWS_Mem_00073]

Upstream requirements: SRS_MemHwAb_14045, SRS_MemHwAb_14049

⌈The function pointer table is a standardized structure used to reference the Mem driver service functions.

The format of function pointer table shall follow the structure defined in [SWS_Mem_00096].⌋

[SWS_Mem_00096]

Upstream requirements: SRS_MemHwAb_14045, SRS_MemHwAb_14049

⌈

Entry	*Name*	*Description*
1	Init service pointer	Function pointer to Mem driver Init service.
2	DeInit service pointer	Function pointer to Mem driver DeInit service.
3	MainFunction service pointer	Function pointer to Mem driver MainFunction service.
4	GetJobResult service pointer	Function pointer to Mem driver GetJobResult service.
5	Read service pointer	Function pointer to Mem driver Read service.
6	Write service pointer	Function pointer to Mem driver Write service.
7	Erase service pointer	Function pointer to Mem driver Erase service.
8	PropagateError service pointer	Function pointer to Mem driver PropagateError service.
9	BlankCheck service pointer	Function pointer to Mem driver BlankCheck service.
10	Suspend service pointer	Function pointer to Mem driver Suspend service.
11	Resume service pointer	Function pointer to Mem driver Resume service.
12	HwSpecificService service pointer	Function pointer to Mem driver HwSpecificService service.

Function Pointer Table

⌋

[SWS_Mem_00048]

Upstream requirements: SRS_MemHwAb_14045, SRS_MemHwAb_14049

⌈The size of the Mem driver function pointers shall be machine/CPU specific.⌋

Note: The system integrator has to ensure to use the same memory model for the Mem driver and the MemAcc module.

7.2.6.3 Delimiter

The delimiter field marks the end of the Mem driver binary image. The format of the delimiter field shall follow the unique identifier defined in paragraph 7.2.6.1.1.

[SWS_Mem_00051]

Upstream requirements: SRS_MemHwAb_14045, SRS_MemHwAb_14049

⌈The value of the delimiter field shall be the ones' complement of the unique identifier value.⌋

7.2.7 Optional Services

Since not all Mem driver services are relevant for all memory device technologies, the Mem driver provides mechanisms to indicate to the MemAcc module which services are available.

Independent, if a service is relevant, the Mem driver shall provide all standard services as defined in chapter 8.3.

[SWS_Mem_00070]

Upstream requirements: SRS_MemHwAb_14039

⌈In case a service is not relevant for a specific memory device technology, the service shall always return `E_MEM_SERVICE_NOT_AVAIL`.⌋

7.3 Module Handling

7.3.1 Initialization

The Mem driver is initialized via `Mem_Init`. Except for `Mem_MainFunction` and `Mem_Init`, the API functions of the Mem driver may only be called after the module has been properly initialized.

7.3.2 Scheduling

Since most of the Mem driver services are asynchronous services, the `Mem_MainFunction` needs to be cyclically triggered. Since Mem driver don't need to measure times or do any timing supervision, there is no need to call the `Mem_MainFunction` with a fixed cycle.

7.3.3 ECC Handling

Handling ECC errors is very hardware specific but typically a non-maskable interrupt is triggered by the ECC circuit. Dealing with ECC errors has to be done on a system level as the error reaction needs to be handled on system level as well.

To propagate memory ECC errors to the upper layer, the Mem driver provides the `Mem_PropagateError` API which can be called by the system ECC error handler to forward ECC errors.

Figure 9.2 shows an example ECC handling sequence to illustrate the usage of the `Mem_PropagateError` API.

7.4 General Design Rules

[SWS_Mem_00033]

Upstream requirements: SRS_BSW_00323, SRS_BSW_00167, SRS_BSW_00004

⌈The Mem driver shall check static configuration parameters statically (at the latest during compile time) for correctness.⌋

[SWS_Mem_00088] ⌈The Mem driver shall not verify the result of a write or erase operation.⌋

Note: Consistency checks shall be implemented by the upper layers, e.g., by reading back the data or performing an explicit blank check.

7.4.1 Address Alignment

[SWS_Mem_00035]

Upstream requirements: SRS_MemHwAb_14036

⌈The Mem driver shall not perform any sort of address or length alignment in case physical segmentation needs to be considered, e.g. for flash memory.⌋

Note: The MemAcc module takes care about alignment/splitting Mem service requests according to the configure physical segmentation.

7.4.2 64-Bit Support

[SWS_Mem_00036]

Upstream requirements: SRS_MemHwAb_14046

⌈The Mem driver shall inherit the size of `Mem_AddressType` from `MemAcc_AddressType` (see [SWS_MemAcc_10000]) to support memories larger than 4GBytes.⌋

Note: This requirement applies for all Mem drivers, independent if the physical memory size is less than 4GBytes.

[SWS_Mem_00037]

Upstream requirements: SRS_MemHwAb_14046

⌈If the maximum physical memory size of a device does not exceed 4GBytes, the Mem driver shall perform an explicit cast and work internally with a 32-Bit address type.⌋

Note: Avoid resource overhead on 32-Bit microcontroller.

7.5 Error Classification

Chapter [5] 7.2 Error Handling describes the error handling of the Basic Software in detail. Above all, it constitutes a classification scheme consisting of following error types which may occur in BSW modules. Based on this foundation, the following section specifies particular errors arranged in the respective subsections below.

7.5.1 Development Errors

[SWS_Mem_00052] Definition of development errors in module Mem

Upstream requirements: SRS_BSW_00385

⌈

Type of error	*Related error code*	*Error value*
API service called without module initialization	MEM_E_UNINIT	0x01
API service called with NULL pointer	MEM_E_PARAM_POINTER	0x02
API service called with an invalid address	MEM_E_PARAM_ADDRESS	0x03
API service called with an invalid length	MEM_E_PARAM_LENGTH	0x04
API service called with an invalid driver instance ID	MEM_E_PARAM_INSTANCE_ID	0x05
API service called while a job request is still in progress	MEM_E_JOB_PENDING	0x06

⌋

7.5.2 Runtime Errors

There are no runtime errors.

Note: Simplify system design by just having one fault handling mechanism.

7.5.3 Production Errors

There are no production errors.

7.6 Security Events

The module does not report security events.

8 API Specification

[SWS_Mem_00074]

Upstream requirements: SRS_BSW_00347

⌈The Mem driver API names, type definitions and file naming scheme shall follow the standard AUTOSAR BSW Module implementation prefix with `vendorId` and `vendorApiInfix`.⌋

8.1 Imported Types

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

[SWS_Mem_10020] Definition of imported datatypes of module Mem

Upstream requirements: SRS_BSW_00415

⌈

Module	*Header File*	*Imported Type*
MemAcc	MemAcc_GeneralTypes.h	MemAcc_AddressType
	MemAcc_GeneralTypes.h	MemAcc_MemJobResultType
Std	Std_Types.h	Std_ReturnType

⌋

8.2 Type Definitions

This chapter contains all APIs provided by the module.

8.2.1 Mem_AddressType

[SWS_Mem_10002] Definition of datatype Mem_AddressType

Upstream requirements: SRS_MemHwAb_14046

⌈

Name	Mem_AddressType
Kind	Type
Derived from	MemAcc_AddressType
Description	Physical memory device address type
Available via	Mem.h

⌋

8.2.2 Mem_ConfigType

[SWS_Mem_10000] Definition of datatype Mem_ConfigType

Upstream requirements: SRS_BSW_00414

⌈

Name	Mem_ConfigType
Kind	Structure
Description	Postbuild configuration structure type
Available via	Mem.h

⌋

8.2.3 Mem_DataType

[SWS_Mem_10003] Definition of datatype Mem_DataType

Upstream requirements: SRS_MemHwAb_14051

⌈

Name	Mem_DataType
Kind	Type
Derived from	uint8
Description	Read data user buffer type
Available via	Mem.h

⌋

8.2.4 Mem_InstanceIdType

[SWS_Mem_10004] Definition of datatype Mem_InstanceIdType

Upstream requirements: SRS_MemHwAb_14043

⌈

Name	Mem_InstanceIdType
Kind	Type
Derived from	uint32
Description	Memory driver instance ID type
Available via	Mem.h

⌋

8.2.5 Mem_LengthType

[SWS_Mem_10007] Definition of datatype Mem_LengthType

Upstream requirements: SRS_MemHwAb_14046

⌈

Name	Mem_LengthType
Kind	Type
Derived from	uint32
Description	Physical memory device length type
Available via	Mem.h

⌋

8.2.6 Mem_HwServiceIdType

[SWS_Mem_10026] Definition of datatype Mem_HwServiceIdType

Upstream requirements: SRS_MemHwAb_14042

⌈

Name	Mem_HwServiceIdType
Kind	Type
Derived from	uint32
Description	Hardware specific service request identifier type
Available via	Mem.h

⌋

8.3 Function Definitions

This chapter contains all APIs provided by the module.

8.3.1 Synchronous Functions

8.3.1.1 Mem_Init

[SWS_Mem_10008] Definition of API function Mem_Init

Upstream requirements: SRS_MemHwAb_14037

⌈

Service Name	Mem_Init	
Syntax	`void Mem_Init ( const Mem_ConfigType* configPtr )`	
Service ID [hex]	0x01	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	configPtr	Pointer to the configuration data structure - since Mem driver is a precompile module this parameter is typically not used.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Initialization function - initializes all variables and sets the module state to initialized.	
Available via	Mem.h	

⌋

[SWS_Mem_00001]

Upstream requirements: SRS_MemHwAb_14037, SRS_MemHwAb_14047

⌈The service `Mem_Init` shall initialize the Mem driver internal states and set the Mem driver job processing state to `MEM_JOB_OK`.⌋

[SWS_Mem_00087]

Upstream requirements: SRS_BSW_00414, SRS_BSW_00386

⌈If development error detection is enabled by `MemDevErrorDetect`, the service `Mem_Init` shall raise the development error `MEM_E_PARAM_POINTER` if the `configPtr` argument is not a NULL pointer.⌋

Note: The configuration pointer `configPtr` is currently not used and shall therefore be set to a NULL pointer value.

8.3.1.2 Mem_DeInit

[SWS_Mem_10018] Definition of API function Mem_DeInit

Upstream requirements: SRS_MemHwAb_14037

⌈

Service Name	Mem_DeInit

Syntax	`void Mem_DeInit (`
	`  void`
	`)`

Service ID [hex]	0x0b
Sync/Async	Synchronous
Reentrancy	Non Reentrant
Parameters (in)	None
Parameters (inout)	None
Parameters (out)	None
Return value	None
Description	De-initialize module. If there is still an access job pending, it is immediately terminated (using hardware cancel operation) and the Mem driver module state is set to unitialized. Therefore, Mem must be re-initialized before it will accept any new job requests after this service is processed.
Available via	Mem.h

⌋

[SWS_Mem_00079]

Upstream requirements: SRS_MemHwAb_14031

⌈The service `Mem_DeInit` shall cancel any ongoing flash operations in the hardware and de-initialize the Mem driver internal states.⌋

8.3.1.3 Mem_GetJobResult

[SWS_Mem_10011] Definition of API function Mem_GetJobResult

Upstream requirements: SRS_MemHwAb_14040

⌈

Service Name	Mem_GetJobResult	

Syntax	`MemAcc_MemJobResultType Mem_GetJobResult (`
	`  Mem_InstanceIdType instanceId`
	`)`	

Service ID [hex]	0x04	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	instanceId	ID of the related memory driver instance.
Parameters (inout)	None	
Parameters (out)	None	

▽

△

Return value	MemAcc_MemJobResult Type	Most recent job result.
Description	Service to return results of the most recent job.	
Available via	Mem.h	

⌋

[SWS_Mem_00090]

Upstream requirements: SRS_BSW_00323, SRS_BSW_00386

⌈If development error detection is enabled by `MemDevErrorDetect`, the service `Mem_GetJobResult` shall check that the provided `instanceId` is consistent with the configuration. If this check fails, `Mem_GetJobResult` shall raise the development error `MEM_E_PARAM_INSTANCE_ID`.⌋

8.3.1.4 Mem_Suspend

[SWS_Mem_10024] Definition of API function Mem_Suspend

Upstream requirements: SRS_MemHwAb_14034

⌈

Service Name	Mem_Suspend	
Syntax	`Std_ReturnType Mem_Suspend ( Mem_InstanceIdType instanceId )`	
Service ID [hex]	0x0c	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	instanceId	ID of the related memory driver instance.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: The requested job has been accepted by the module. `E_NOT_OK`:The requested job has not been accepted by the module. `E_MEM_SERVICE_NOT_AVAIL`: The underlying Mem driver service function is not available.
Description	Suspend active memory operation using hardware mechanism.	
Available via	Mem.h	

⌋

[SWS_Mem_00080]

Upstream requirements: SRS_MemHwAb_14031

⌈The service `Mem_Suspend` shall suspend any ongoing flash operations using an according hardware mechanism.⌋

[SWS_Mem_00083]

Upstream requirements: SRS_MemHwAb_14031

⌈In case a suspend operation is already in pending, `Mem_Suspend` shall reject the request by returning `E_NOT_OK` without further actions.⌋

[SWS_Mem_00091]

Upstream requirements: SRS_BSW_00323, SRS_BSW_00386

⌈If development error detection is enabled by `MemDevErrorDetect`, the service `Mem_Suspend` shall check that the provided `instanceId` is consistent with the configuration. If this check fails, `Mem_Suspend` shall raise the development error `MEM_E_PARAM_INSTANCE_ID`.⌋

8.3.1.5 Mem_Resume

[SWS_Mem_10025] Definition of API function Mem_Resume

Upstream requirements: SRS_MemHwAb_14034

⌈

Service Name	Mem_Resume	
Syntax	`Std_ReturnType Mem_Resume ( Mem_InstanceIdType instanceId )`	
Service ID [hex]	0x0d	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	instanceId	ID of the related memory driver instance.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: The requested job has been accepted by the module. `E_NOT_OK`:The requested job has not been accepted by the module. `E_MEM_SERVICE_NOT_AVAIL`: The underlying Mem driver service function is not available.
Description	Resume suspended memory operation using hardware mechanism.	
Available via	Mem.h	

⌋

[SWS_Mem_00081]

Upstream requirements: SRS_MemHwAb_14031

⌈The service `Mem_Resume` shall resume a flash operations that was suspended by the service `Mem_Suspend`.⌋

[SWS_Mem_00084]

Upstream requirements: SRS_MemHwAb_14031

⌈In case no suspend operation is pending, `Mem_Resume` shall reject the request by returning `E_NOT_OK` without further actions.⌋

[SWS_Mem_00092]

Upstream requirements: SRS_BSW_00323, SRS_BSW_00386

⌈If development error detection is enabled by `MemDevErrorDetect`, the service `Mem_Resume` shall check that the provided `instanceId` is consistent with the configuration. If this check fails, `Mem_Resume` shall raise the development error `MEM_E_PARAM_INSTANCE_ID`.⌋

8.3.1.6 Mem_PropagateError

[SWS_Mem_10015] Definition of API function Mem_PropagateError

Upstream requirements: SRS_MemHwAb_14053

⌈

Service Name	Mem_PropagateError	
Syntax	`void Mem_PropagateError ( Mem_InstanceIdType instanceId )`	
Service ID [hex]	0x08	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	instanceId	ID of the related memory driver instance.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	This service can be used to report an access error in case the Mem driver cannot provide the access error information - typically for ECC faults. It is called by the system ECC handler to propagate an ECC error to the memory upper layers..	
Available via	Mem.h	

⌋

[SWS_Mem_00020]

Upstream requirements: SRS_BSW_00323, SRS_BSW_00386

⌈If development error detection is enabled by `MemDevErrorDetect`, the service `Mem_PropagateError` shall check that the provided `instanceId` is consistent with the configuration. If this check fails, `Mem_PropagateError` shall raise the development error `MEM_E_PARAM_INSTANCE_ID`.⌋

[SWS_Mem_00061]

Upstream requirements: SRS_MemHwAb_14053

⌈If the `Mem_PropagateError` service is called, the Mem driver shall set the job result code to `MEM_ECC_UNCORRECTED` and cancel the current job processing.⌋

8.3.2 Asynchronous Functions

8.3.2.1 Mem_Read

[SWS_Mem_10012] Definition of API function Mem_Read

Upstream requirements: SRS_MemHwAb_14038, SRS_MemHwAb_14039

⌈

Service Name	Mem_Read	
Syntax	`Std_ReturnType Mem_Read ( Mem_InstanceIdType instanceId, Mem_AddressType sourceAddress, Mem_DataType* destinationDataPtr, Mem_LengthType length )`	
Service ID [hex]	0x05	
Sync/Async	Asynchronous	
Reentrancy	Non Reentrant	
Parameters (in)	instanceId	ID of the related memory driver instance.
	sourceAddress	Physical address to read data from.
	length	Read length in bytes.
Parameters (inout)	None	
Parameters (out)	destinationDataPtr	Destination memory pointer to store the read data.
Return value	Std_ReturnType	`E_OK`: The requested job has been accepted by the module. `E_NOT_OK`: The requested job has not been accepted by the module. `E_MEM_SERVICE_NOT_AVAIL`: The service function is not implemented.
Description	Triggers a read job to copy the from the source address into the referenced destination data buffer. The result of this service can be retrieved using the Mem_GetJobResult API. If the read operation was successful, the result of the job is MEM_JOB_OK. If the read operation failed, the result of the job is either MEM_JOB_FAILED in case of a general error or MEM_ECC_CORRECTED/MEM_ECC_UNCORRECTED in case of a correctable/uncorrectable ECC error.	
Available via	Mem.h	

⌋

[SWS_Mem_00004]

Upstream requirements: SRS_BSW_00323, SRS_BSW_00386

⌈If development error detection is enabled by `MemDevErrorDetect`, the service `Mem_Read` shall check that the provided `instanceId` is consistent with the configuration. If this check fails, `Mem_Read` shall raise the development error `MEM_E_PARAM_INSTANCE_ID`.⌋

[SWS_Mem_00005]

Upstream requirements: SRS_BSW_00323, SRS_BSW_00386

⌈If development error detection is enabled by `MemDevErrorDetect`, the service `Mem_Read` shall raise the development error `MEM_E_PARAM_POINTER` if the `destinationDataPtr` argument is a NULL pointer.⌋

[SWS_Mem_00006]

Upstream requirements: SRS_BSW_00323, SRS_BSW_00386

⌈If development error detection is enabled by MemDevErrorDetect, the service Mem_Read shall raise the development error MEM_E_PARAM_ADDRESS if the address defined by sourceAddress is invalid.⌋

[SWS_Mem_00072]

Upstream requirements: SRS_BSW_00323, SRS_BSW_00386

⌈If development error detection is enabled by MemDevErrorDetect, the service Mem_Read shall raise the development error MEM_E_PARAM_LENGTH if the read length defined by length is invalid.⌋

[SWS_Mem_00007]

Upstream requirements: SRS_BSW_00323, SRS_BSW_00386

⌈If development error detection is enabled by MemDevErrorDetect, the service Mem_Read shall raise the development error MEM_E_JOB_PENDING if a previous Mem job is still being processed.⌋

8.3.2.2 Mem_Write

[SWS_Mem_10013] Definition of API function Mem_Write

Upstream requirements: SRS_MemHwAb_14038, SRS_MemHwAb_14039

⌈

Service Name	Mem_Write	

Syntax	`Std_ReturnType Mem_Write (`
	`  Mem_InstanceIdType instanceId,`
	`  Mem_AddressType targetAddress,`
	`  const Mem_DataType* sourceDataPtr,`
	`  Mem_LengthType length`
	`)`	

Service ID [hex]	0x06	
Sync/Async	Asynchronous	
Reentrancy	Non Reentrant	
Parameters (in)	instanceId	ID of the related memory driver instance.
	targetAddress	Physical write address (aligned to page size).
	sourceDataPtr	Source data pointer (aligned to page size).
	length	Write length in bytes (aligned to page size).
Parameters (inout)	None	
Parameters (out)	None	

▽

△

Return value	Std_ReturnType	E_OK: The requested job has been accepted by the module. E_NOT_OK: The requested job has not been accepted by the module. E_MEM_SERVICE_NOT_AVAIL: The service function is not implemented.
Description	Triggers a write job to store the passed data to the provided address area with given address and length. The result of this service can be retrieved using the Mem_GetJobResult API. If the write operation was successful, the job result is MEM_JOB_OK. If there was an issue writing the data, the result is MEM_FAILED.	
Available via	Mem.h	

⌋

[SWS_Mem_00009]

Upstream requirements: SRS_BSW_00323, SRS_BSW_00386

⌈If development error detection is enabled by MemDevErrorDetect, the service Mem_Write shall check that the provided instanceId is consistent with the configuration. If this check fails, Mem_Write shall raise the development error MEM_E_PARAM_INSTANCE_ID.⌋

[SWS_Mem_00010]

Upstream requirements: SRS_BSW_00323, SRS_BSW_00386

⌈If development error detection is enabled by MemDevErrorDetect, the service Mem_Write shall raise the development error MEM_E_PARAM_POINTER if the sourceDataPtr argument is a NULL pointer.⌋

[SWS_Mem_00011]

Upstream requirements: SRS_BSW_00323, SRS_BSW_00386

⌈If development error detection is enabled by MemDevErrorDetect, the service Mem_Write shall raise the development error MEM_E_PARAM_ADDRESS if the address defined by targetAddress is invalid.⌋

[SWS_Mem_00012]

Upstream requirements: SRS_BSW_00323, SRS_BSW_00386

⌈If development error detection is enabled by MemDevErrorDetect, the service Mem_Write shall raise the development error MEM_E_PARAM_LENGTH if the write length defined by length is invalid.⌋

[SWS_Mem_00013]

Upstream requirements: SRS_BSW_00323, SRS_BSW_00386

⌈If development error detection is enabled by MemDevErrorDetect, the service Mem_Write shall raise the development error MEM_E_JOB_PENDING if a previous Mem job is still being processed.⌋

8.3.2.3 Mem_Erase

[SWS_Mem_10014] Definition of API function Mem_Erase

Upstream requirements: SRS_MemHwAb_14038, SRS_MemHwAb_14039

⌈

Service Name	Mem_Erase	

Syntax	`Std_ReturnType Mem_Erase (`
	`  Mem_InstanceIdType instanceId,`
	`  Mem_AddressType targetAddress,`
	`  Mem_LengthType length`
	`)`	

Service ID [hex]	0x07	
Sync/Async	Asynchronous	
Reentrancy	Non Reentrant	
Parameters (in)	instanceId	ID of the related memory driver instance.
	targetAddress	Physical erase address (aligned to sector size).
	length	Erase length in bytes (aligned to sector size).
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: The requested job has been accepted by the module. `E_NOT_OK`: The requested job has not been accepted by the module. `E_MEM_SERVICE_NOT_AVAIL`: The service function is not implemented.
Description	Triggers an erase job of the given sector/sector batch defined by targetAddress and length. The result of this service can be retrieved using the Mem_GetJobResult API. If the erase operation was successful, the result of the job is MEM_JOB_OK. If the erase operation failed, e.g. due to a hardware issue, the result of the job is MEM_JOB_FAILED.	
Available via	Mem.h	

⌋

[SWS_Mem_00015]

Upstream requirements: SRS_BSW_00323

⌈If development error detection is enabled by `MemDevErrorDetect`, the service `Mem_Erase` shall check that the provided `instanceId` is consistent with the configuration. If this check fails, `Mem_Erase` shall raise the development error `MEM_E_PARAM_INSTANCE_ID`.⌋

[SWS_Mem_00016]

Upstream requirements: SRS_BSW_00323

⌈If development error detection is enabled by `MemDevErrorDetect`, the service `Mem_Erase` shall raise the development error `MEM_E_PARAM_ADDRESS` if the address defined by `targetAddress` is invalid.⌋

[SWS_Mem_00017]

Upstream requirements: SRS_BSW_00323

⌈If development error detection is enabled by `MemDevErrorDetect`, the service `Mem_Erase` shall raise the development error `MEM_E_PARAM_LENGTH` if the read length defined by `length` is invalid.⌋

[SWS_Mem_00018]

Upstream requirements: SRS_BSW_00323

⌈If development error detection is enabled by `MemDevErrorDetect`, the service `Mem_Erase` shall raise the development error `MEM_E_JOB_PENDING` if a previous Mem job is still being processed.⌋

8.3.2.4 Mem_BlankCheck

[SWS_Mem_10016] Definition of API function Mem_BlankCheck

Upstream requirements: SRS_MemHwAb_14038, SRS_MemHwAb_14039

⌈

Service Name	Mem_BlankCheck	
Syntax	`Std_ReturnType Mem_BlankCheck ( Mem_InstanceIdType instanceId, Mem_AddressType targetAddress, Mem_LengthType length )`	
Service ID [hex]	0x09	
Sync/Async	Asynchronous	
Reentrancy	Non Reentrant	
Parameters (in)	instanceId	ID of the related memory driver instance.
	targetAddress	Physical blank check address.
	length	Blank check length.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: The requested job has been accepted by the module. `E_NOT_OK`: The requested job has not been accepted by the module. `E_MEM_SERVICE_NOT_AVAIL`: The service function is not implemented.
Description	Triggers a job to check the erased state of the page which is referenced by targetAddress. The result of this service can be retrieved using the Mem_GetJobResult API. If the checked page is blank, the result of the job is MEM_JOB_OK. Otherwise, if the page is not blank, the result is MEM_INCONSISTENT.	
Available via	Mem.h	

⌋

[SWS_Mem_00022]

Upstream requirements: SRS_BSW_00323

⌈If development error detection is enabled by `MemDevErrorDetect`, the service `Mem_BlankCheck` shall check that the provided `instanceId` is consistent with the configuration. If this check fails, `Mem_BlankCheck` shall raise the development error `MEM_E_PARAM_INSTANCE_ID`.⌋

[SWS_Mem_00023]

Upstream requirements: SRS_BSW_00323

⌈If development error detection is enabled by MemDevErrorDetect, the service Mem_BlankCheck shall raise the development error MEM_E_PARAM_ADDRESS if the address defined by targetAddress is invalid.⌋

[SWS_Mem_00024]

Upstream requirements: SRS_BSW_00323

⌈If development error detection is enabled by MemDevErrorDetect, the service Mem_BlankCheck shall raise the development error MEM_E_PARAM_LENGTH if the read length defined by length is invalid.⌋

[SWS_Mem_00025]

Upstream requirements: SRS_BSW_00323

⌈If development error detection is enabled by MemDevErrorDetect, the service Mem_BlankCheck shall raise the development error MEM_E_JOB_PENDING if a previous Mem job is still being processed.⌋

8.3.2.5 Mem_HwSpecificService

[SWS_Mem_10017] Definition of API function Mem_HwSpecificService

Upstream requirements: SRS_MemHwAb_14038, SRS_MemHwAb_14039, SRS_MemHwAb_14056

⌈

Service Name	Mem_HwSpecificService	
Syntax	`Std_ReturnType Mem_HwSpecificService ( Mem_InstanceIdType instanceId, Mem_HwServiceIdType hwServiceId, Mem_DataType* dataPtr, Mem_LengthType* lengthPtr )`	
Service ID [hex]	0x0a	
Sync/Async	Asynchronous	
Reentrancy	Non Reentrant	
Parameters (in)	instanceId	ID of the related memory driver instance.
	hwServiceId	Hardware specific service request identifier for dispatching the request.
	dataPtr	Request specific data pointer.
	lengthPtr	Size pointer of the data passed by dataPtr.
Parameters (inout)	None	
Parameters (out)	None	

▽

△

Return value	Std_ReturnType	E_OK: The requested job has been accepted by the module. E_NOT_OK: The requested job has not been accepted by the module. E_MEM_SERVICE_NOT_AVAIL: The service function is not implemented.
Description	Triggers a hardware specific memory driver job. dataPtr can be used to pass and return data to/ from this service. This service is just a dispatcher to the hardware specific service implementation referenced by hwServiceId. The result of this service can be retrieved using the Mem_GetJobResult API. If the hardware specific operation was successful, the result of the job is MEM_JOB_OK. If the hardware specific operation failed, the result of the job is MEM_JOB_ FAILED.	
Available via	Mem.h	

⌋

[SWS_Mem_00026]

Upstream requirements: SRS_BSW_00323

⌈If development error detection is enabled by MemDevErrorDetect, the service Mem_ HwSpecificService shall check that the provided instanceId is consistent with the configuration. If this check fails, Mem_HwSpecificService shall raise the development error MEM_E_PARAM_INSTANCE_ID.⌋

[SWS_Mem_00027]

Upstream requirements: SRS_BSW_00323

⌈If development error detection is enabled by MemDevErrorDetect, the service Mem_ HwSpecificService shall raise the development error MEM_E_PARAM_POINTER if the dataPtr or lengthPtr argument is a NULL pointer.⌋

8.4 Callback Notifications

There are no callback functions to lower layer modules provided by the Mem driver since this module is at the lowest (software) layer.

8.5 Scheduled Functions

These functions are directly called by Basic Software Scheduler. The following func tions shall have no return value and no parameter. All functions shall be non reentrant.

8.5.1 Mem_MainFunction

[SWS_Mem_10010] Definition of scheduled function Mem_MainFunction

Upstream requirements: SRS_MemHwAb_14038

⌈

Service Name	Mem_MainFunction
Syntax	`void Mem_MainFunction ( void )`
Service ID [hex]	0x03
Description	Service to handle the requested jobs and the internal management operations.
Available via	SchM_Mem.h

⌋

8.6 Expected Interfaces

In this chapter all interfaces required from other modules are listed.

8.6.1 Mandatory Interfaces

This section defines all interfaces, which are required to fulfill the core functionality of the module.

[SWS_Mem_10022] Definition of mandatory interfaces required by module Mem

Upstream requirements: SRS_BSW_00415

⌈

API Function	***Header File***	***Description***
There are no mandatory interfaces.		

⌋

8.6.2 Optional Interfaces

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

[SWS_Mem_10021] Definition of optional interfaces requested by module Mem

Upstream requirements: SRS_BSW_00415

⌈

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

⌋

8.6.3 Configurable Interfaces

The Mem driver does not have any configurable interfaces.

9 Sequence Diagrams

9.1 Hardware Specific Error Handling

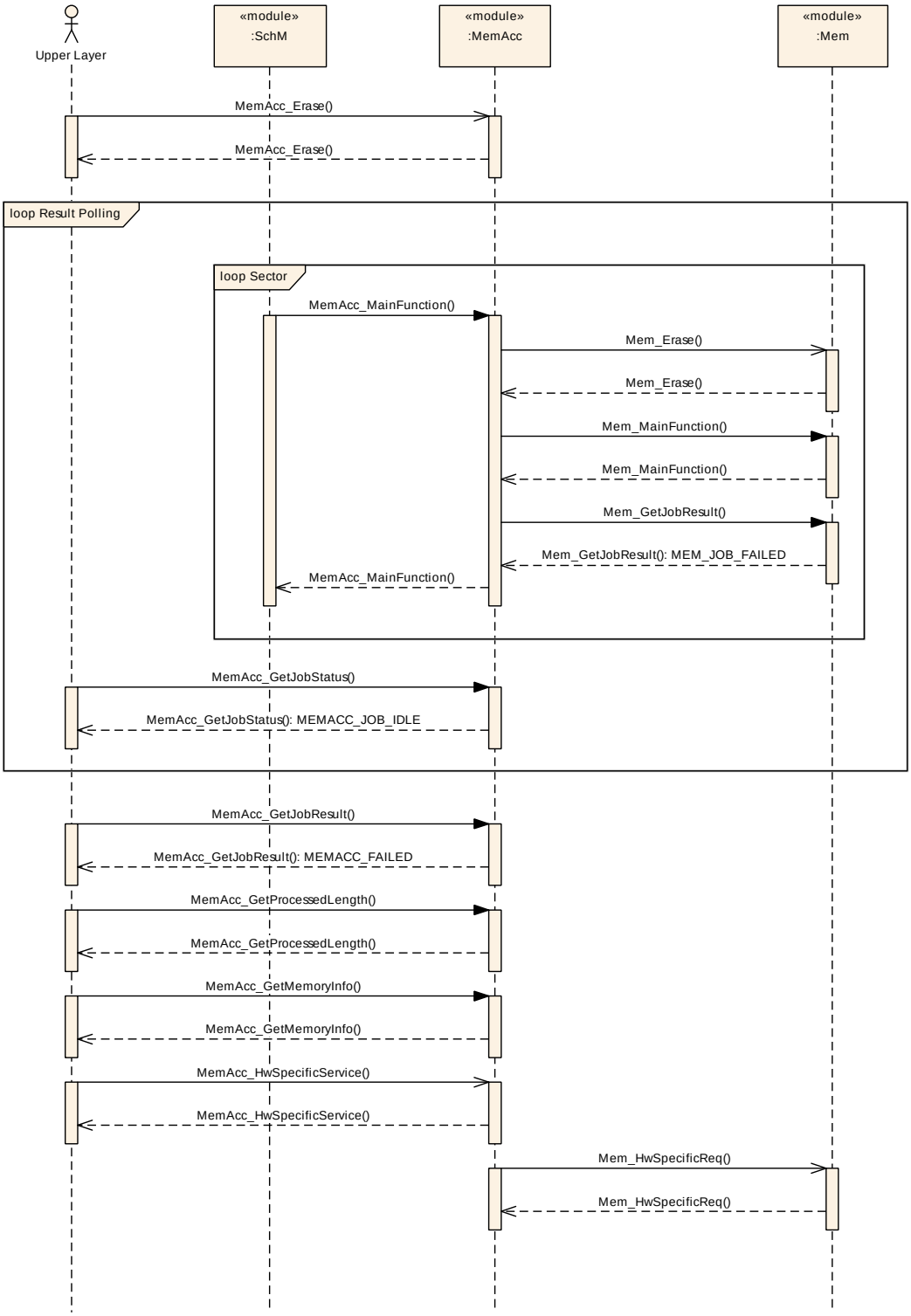


Figure 9.1: Hardware Specific Error Handling

9.2 ECC Handling Example Sequence

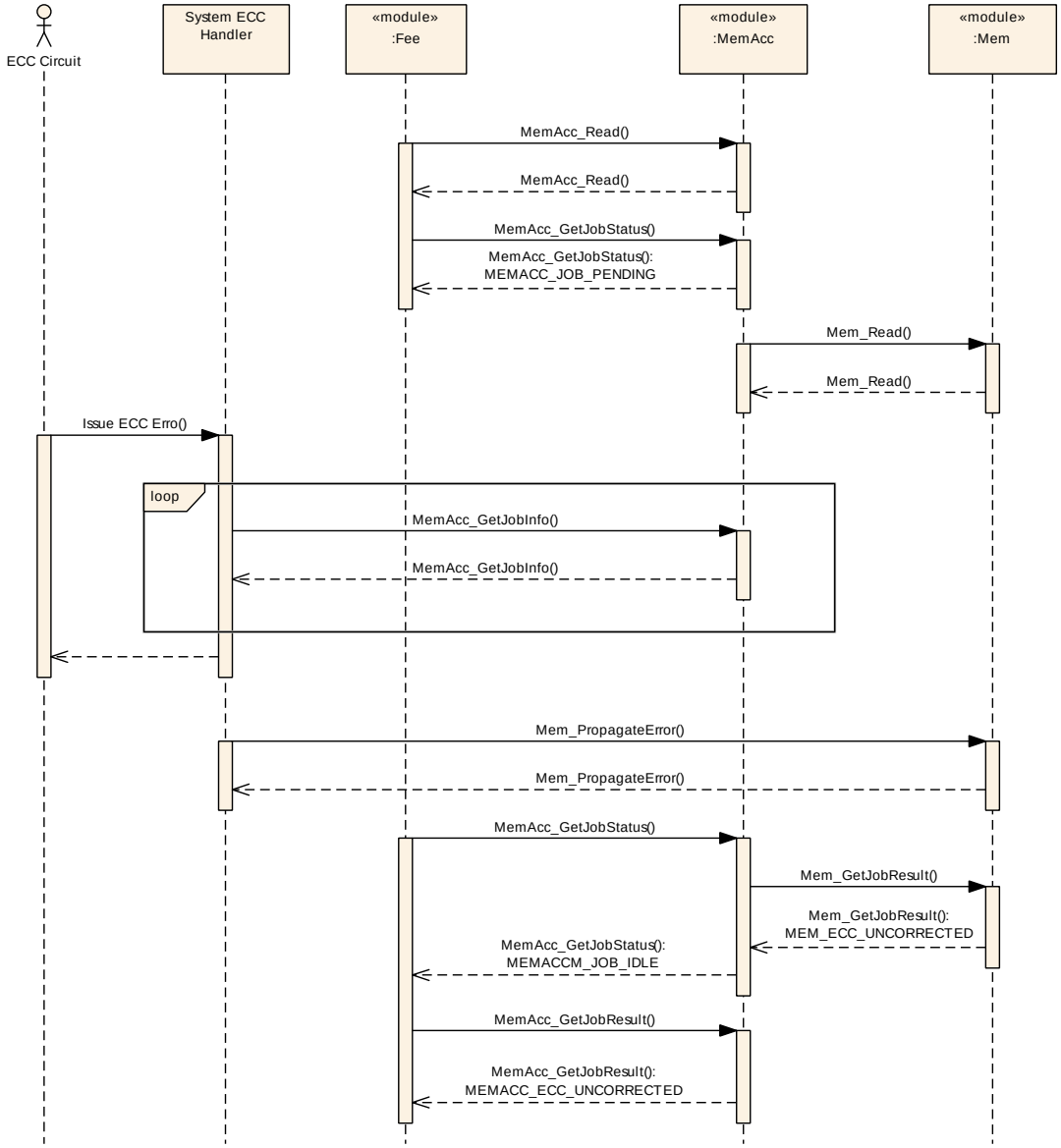


Figure 9.2: ECC Handling Example Sequence

10 Configuration Specification

In general, this chapter defines configuration parameters and their clustering into containers. In order to support the specification chapter 10.1 describes fundamentals. It also specifies a template (table) you shall use for the parameter specification. We intend to leave chapter 10.1 in the specification to guarantee comprehension.

Chapter 10.2 specifies the structure (containers) and the parameters of the module MEM.

Chapter 10.3 specifies published information of the module MEM.

10.1 How to Read this Chapter

For details refer to [5] Chapter 10.1 *"Introduction to configuration specification"*.

10.2 Containers and Configuration Parameters

The following chapters summarize all configuration parameters. The detailed meanings of the parameters describe chapter 7 and chapter 8.

10.2.1 Mem

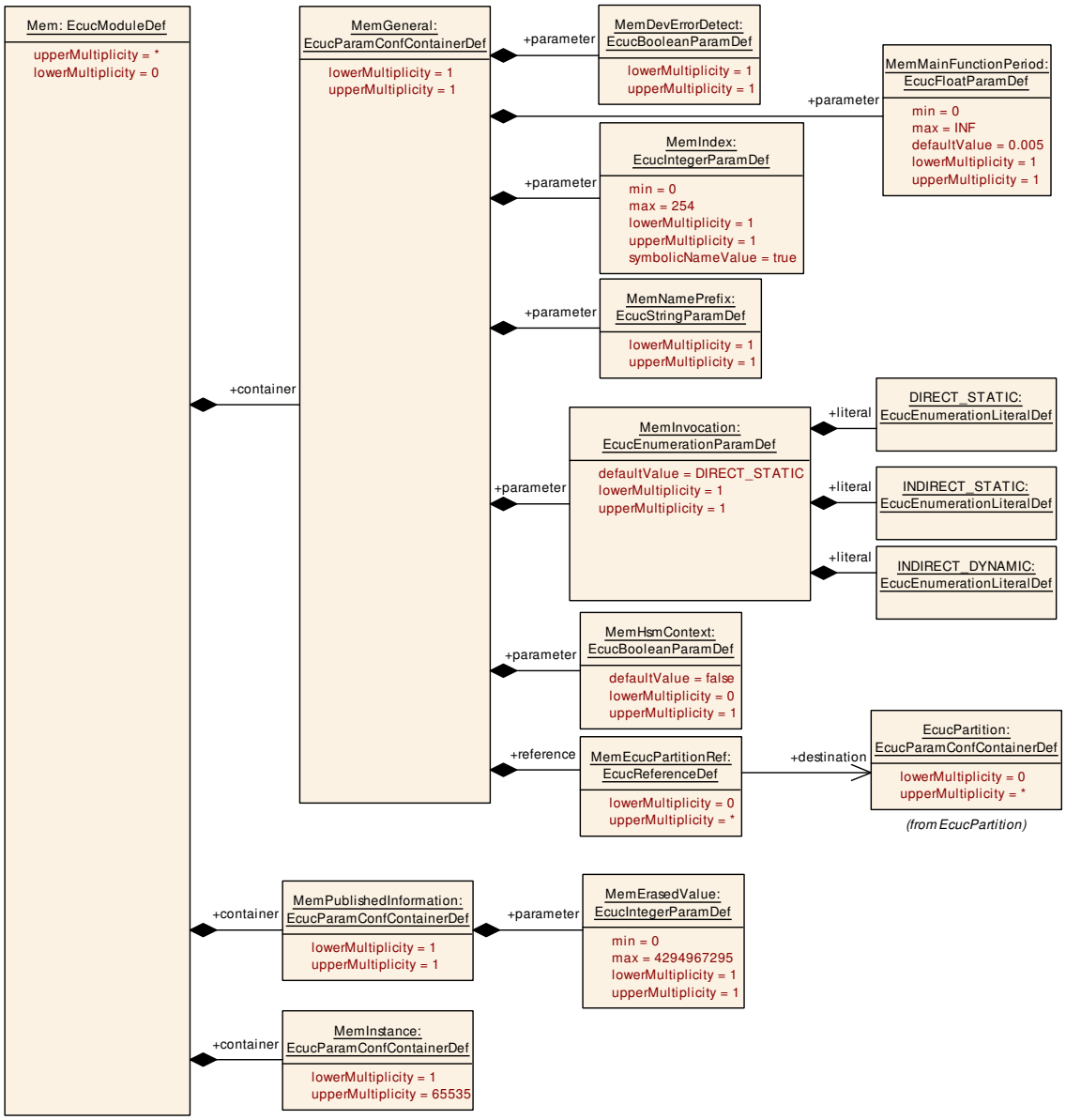


Figure 10.1: Mem

[ECUC_Mem_00001] Definition of EcucModuleDef Mem ⌈

Module Name	Mem
Description	Configuration of the Mem driver (internal or external memory driver) module. Its multiplicity describes the actual number of Mem drivers. There will be one container for each Mem driver.
Post-Build Selectable Support	false
Post-Build Loadable Support	false

Included Containers		
Container Name	**Multiplicity**	**Description**
MemGeneral	1	Container for general configuration parameters of the Mem driver. These parameters are always pre-compile.
MemInstance	1..65535	This container includes the Mem driver instance specific configuration parameters. Its multiplicity describes the number of Mem driver instances of this Mem driver. There will be one container for each Mem driver instance.
MemPublishedInformation	1	Additional published parameters not covered by Common PublishedInformation container. Note that these parameters do not have any configuration class setting, since they are published information.

⌋

[ECUC_Mem_00002] Definition of EcucParamConfContainerDef MemGeneral ⌈

Container Name	MemGeneral
Parent Container	Mem
Upstream mapping defined	–
Description	Container for general configuration parameters of the Mem driver. These parameters are always pre-compile.
Multiplicity	1
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
MemDevErrorDetect	1	[ECUC_Mem_00004]
MemHsmContext	0..1	[ECUC_Mem_00031]
MemIndex	1	[ECUC_Mem_00023]
MemInvocation	1	[ECUC_Mem_00025]
MemMainFunctionPeriod	1	[ECUC_Mem_00029]
MemNamePrefix	1	[ECUC_Mem_00024]
MemEcucPartitionRef	0..*	[ECUC_Mem_00027]

No Included Containers

⌋

[ECUC_Mem_00004] Definition of EcucBooleanParamDef MemDevErrorDetect ⌈

Parameter Name	MemDevErrorDetect
Parent Container	MemGeneral
Upstream mapping defined	–
Description	Switches the development error detection and notification on or off.
Multiplicity	1
Type	EcucBooleanParamDef
Default value	–

▽

△

Pre-Compile only	true

⌋

[ECUC_Mem_00031] Definition of EcucBooleanParamDef MemHsmContext ⌈

Parameter Name	MemHsmContext
Parent Container	MemGeneral
Upstream mapping defined	–
Description	Switches the usage of Mem driver for HOST or HSM context. False - Mem driver is build for HOST context. True-Mem driver is built for HSM context.
Multiplicity	0..1
Type	EcucBooleanParamDef
Default value	false
Pre-Compile only	true

⌋

[ECUC_Mem_00023] Definition of EcucIntegerParamDef MemIndex ⌈

Parameter Name	MemIndex	
Parent Container	MemGeneral	
Upstream mapping defined	–	
Description	Specifies the InstanceId of this module instance. If only one instance is present it shall have the Id 0.	
Multiplicity	1	
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)	
Range	0 .. 254	
Default value	–	
Pre-Compile only	true	

⌋

[ECUC_Mem_00025] Definition of EcucEnumerationParamDef MemInvocation ⌈

Parameter Name	MemInvocation	
Parent Container	MemGeneral	
Upstream mapping defined	–	
Description	Defines how the Mem driver services are accessed and how the Mem driver is scheduled and activated/initialized.	
Multiplicity	1	
Type	EcucEnumerationParamDef	
Range	DIRECT_STATIC	Mem driver is linked with application. Mem_Init is called by EcuM and Mem_MainFunction is triggered by SchM.
	INDIRECT_DYNAMIC	Mem driver is linked as a separate binary and is dynamically activated. Call of Mem_Init and Mem_MainFunction is handled by MemAcc.

▽

△

	INDIRECT_STATIC	Mem driver is linked with application. Call of Mem_Init and Mem_MainFunction is handled by MemAcc. In contrast to INDIRECT_DYNAMIC the Mem driver service functions can be called anytime.
Default value	DIRECT_STATIC	
Pre-Compile only	true	

⌋

[ECUC_Mem_00029] Definition of EcucFloatParamDef MemMainFunctionPeriod ⌈

Parameter Name	MemMainFunctionPeriod	
Parent Container	MemGeneral	
Upstream mapping defined	–	
Description	This value specifies the fixed call cycle for Mem_MainFunction(). In case of indirect invocation the Mem_MainFunction will be triggered directly by Mem Acc. Mem does not depend on a fixed cycle time; it can be triggered at arbitrary rates. Allow to configure the time for the MainFunction (in seconds). Please note: This configuration value shall be equal to the value in the ScheduleManager module.	
Multiplicity	1	
Type	EcucFloatParamDef	
Range	[0 .. INF[	
Default value	0.005	
Pre-Compile only	true	

⌋

[ECUC_Mem_00024] Definition of EcucStringParamDef MemNamePrefix ⌈

Parameter Name	MemNamePrefix
Parent Container	MemGeneral
Upstream mapping defined	–
Description	Depending on the MemInvocation configuration, this prefix is either used to reference the Mem driver header structure or the according Mem API function.
Multiplicity	1
Type	EcucStringParamDef
Default value	–
Regular Expression	–
Pre-Compile only	true

⌋

[ECUC_Mem_00027] Definition of EcucReferenceDef MemEcucPartitionRef ⌈

Parameter Name	MemEcucPartitionRef
Parent Container	MemGeneral
Upstream mapping defined	–

▽

△

Description	Maps the Mem driver to zero or multiple ECUC partition(s) to make the driver API available in these partition(s).
Multiplicity	0..*
Type	Reference to EcucPartition
Pre-Compile only	true

⌋

10.2.2 MemInstance

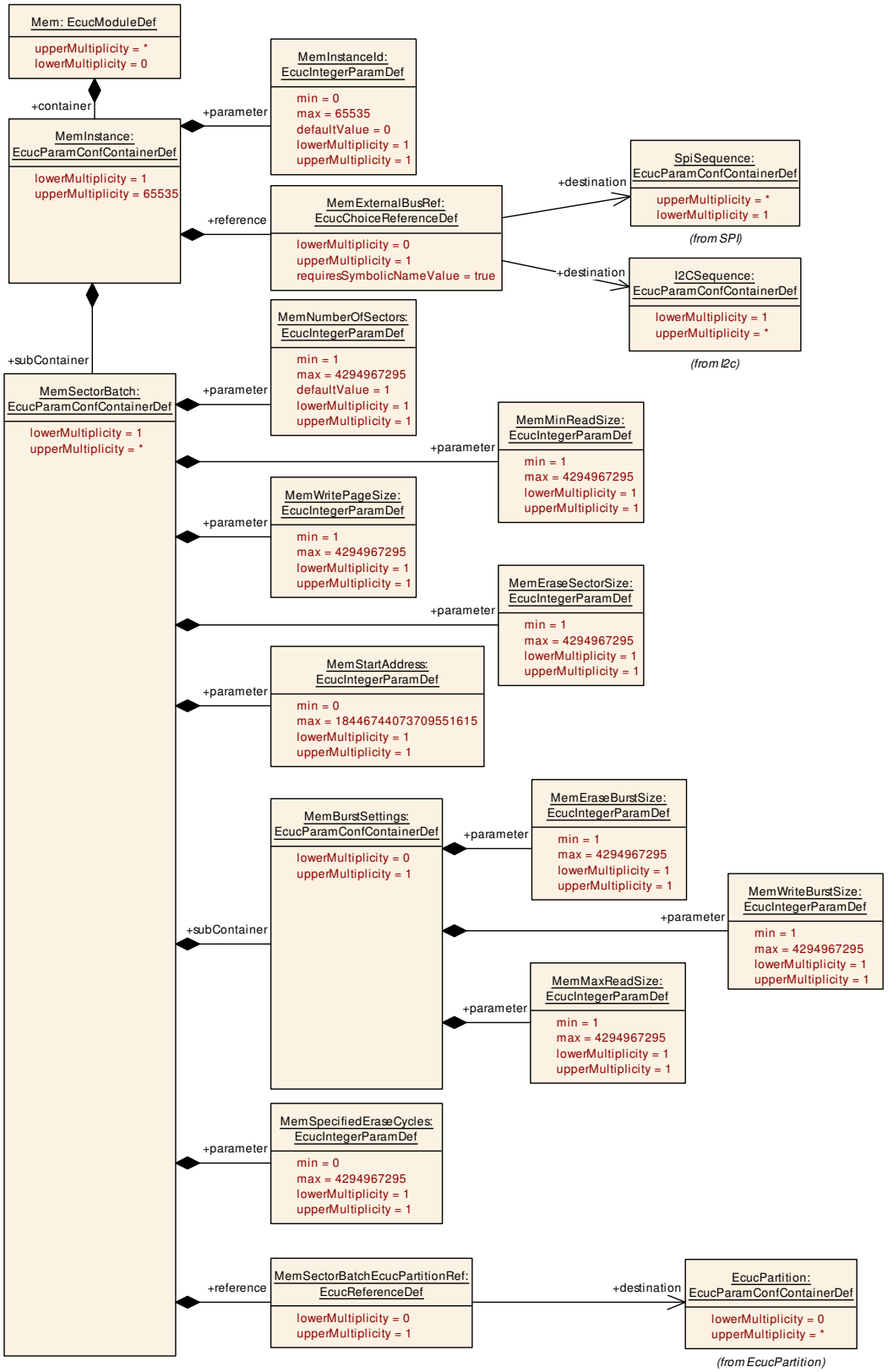


Figure 10.2: MemInstance

[ECUC_Mem_00003] Definition of EcucParamConfContainerDef MemInstance ⌈

Container Name	MemInstance
Parent Container	Mem
Upstream mapping defined	–
Description	This container includes the Mem driver instance specific configuration parameters. Its multiplicity describes the number of Mem driver instances of this Mem driver. There will be one container for each Mem driver instance.
Multiplicity	1..65535
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
MemInstanceId	1	[ECUC_Mem_00007]
MemExternalBusRef	0..1	[ECUC_Mem_00030]

Included Containers		
Container Name	**Multiplicity**	**Description**
MemSectorBatch	1..*	Configuration description of a programmable sector or sector batch. Sector batch means that homogenous and coherent sectors can be configured as one MemSector element. It is recommended to group as many identical sectors as possible together.

⌋

[ECUC_Mem_00007] Definition of EcucIntegerParamDef MemInstanceId ⌈

Parameter Name	MemInstanceId	
Parent Container	MemInstance	
Upstream mapping defined	–	
Description	This value specifies the unique numeric identifier which is used to reference a Mem driver instance in case multiple devices of the same type shall be addressed by one Mem driver. This value will be assigned to the symbolic name derived of the Mem Instance container short name.	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	0 .. 65535	
Default value	0	
Pre-Compile only	true	

⌋

[ECUC_Mem_00030] Definition of EcucChoiceReferenceDef MemExternalBusRef ⌈

Parameter Name	MemExternalBusRef
Parent Container	MemInstance
Upstream mapping defined	–

▽

△

Description	Reference to either: • an SpiSequenceConfiguration container in case the EEP hardware is accessed via SPI • an I2CSequenceConfiguration container in case the EEP hardware is accessed via I2C Dependency: See SPI SWS resp. I2C SWS
Multiplicity	0..1
Type	Choice symbolic name reference to [I2CSequence, SpiSequence]
Pre-Compile only	true

⌋

[ECUC_Mem_00009] Definition of EcucParamConfContainerDef MemSectorBatch ⌈

Container Name	MemSectorBatch
Parent Container	MemInstance
Upstream mapping defined	–
Description	Configuration description of a programmable sector or sector batch. Sector batch means that homogenous and coherent sectors can be configured as one MemSector element. It is recommended to group as many identical sectors as possible together.
Multiplicity	1..*
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
MemEraseSectorSize	1	[ECUC_Mem_00013]
MemMinReadSize	1	[ECUC_Mem_00011]
MemNumberOfSectors	1	[ECUC_Mem_00010]
MemSpecifiedEraseCycles	1	[ECUC_Mem_00022]
MemStartAddress	1	[ECUC_Mem_00014]
MemWritePageSize	1	[ECUC_Mem_00012]
MemSectorBatchEcucPartitionRef	0..1	[ECUC_Mem_00028]

Included Containers		
Container Name	**Multiplicity**	**Description**
MemBurstSettings	0..1	Container for burst setting configuration parameters of the Mem driver. A sector burst can be used for improved performance.

⌋

[ECUC_Mem_00013] Definition of EcucIntegerParamDef MemEraseSectorSize ⌈

Parameter Name	MemEraseSectorSize	
Parent Container	MemSectorBatch	
Upstream mapping defined	–	
Description	Size of this sector in bytes. A sector is the smallest erasable unit.	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	1 .. 4294967295	
Default value	–	
Pre-Compile only	true	

⌋

[ECUC_Mem_00011] Definition of EcucIntegerParamDef MemMinReadSize ⌈

Parameter Name	MemMinReadSize	
Parent Container	MemSectorBatch	
Upstream mapping defined	–	
Description	Smallest readable unit in bytes.	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	1 .. 4294967295	
Default value	–	
Pre-Compile only	true	

⌋

[ECUC_Mem_00010] Definition of EcucIntegerParamDef MemNumberOfSectors ⌈

Parameter Name	MemNumberOfSectors	
Parent Container	MemSectorBatch	
Upstream mapping defined	–	
Description	Number of contiguous sectors with identical values for MemSectorSize and MemPage Size. If this parameter is configured to be greater than 1, the sectors are grouped in a sector batch.	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	1 .. 4294967295	
Default value	1	
Pre-Compile only	true	

⌋

[ECUC_Mem_00022] Definition of EcucIntegerParamDef MemSpecifiedEraseCycles ⌈

Parameter Name	MemSpecifiedEraseCycles	
Parent Container	MemSectorBatch	
Upstream mapping defined	–	
Description	Number of erase cycles specified for the memory device (usually given in the device data sheet).	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	0 .. 4294967295	
Default value	–	
Pre-Compile only	true	

⌋

[ECUC_Mem_00014] Definition of EcucIntegerParamDef MemStartAddress ⌈

Parameter Name	MemStartAddress	
Parent Container	MemSectorBatch	
Upstream mapping defined	–	
Description	Physical start address of the sector (batch). In case of a sector batch, the physical start address is the address of the first sector. The physical start address of the other sectors can be calculated by the MemSector Size parameter.	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	0 .. 18446744073709551615	
Default value	–	
Pre-Compile only	true	

⌋

[ECUC_Mem_00012] Definition of EcucIntegerParamDef MemWritePageSize ⌈

Parameter Name	MemWritePageSize	
Parent Container	MemSectorBatch	
Upstream mapping defined	–	
Description	Size of a write page of this sector in bytes. A write page is the smallest writeable unit.	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	1 .. 4294967295	
Default value	–	
Pre-Compile only	true	

⌋

[ECUC_Mem_00028] Definition of EcucReferenceDef MemSectorBatchEcucPartitionRef ⌈

Parameter Name	MemSectorBatchEcucPartitionRef
Parent Container	MemSectorBatch
Upstream mapping defined	–
Description	Maps the memory area defined by the sector batch to zero or one ECUC partition.
Multiplicity	0..1
Type	Reference to EcucPartition
Pre-Compile only	true

⌋

[ECUC_Mem_00015] Definition of EcucParamConfContainerDef MemBurstSettings ⌈

Container Name	MemBurstSettings
Parent Container	MemSectorBatch
Upstream mapping defined	–
Description	Container for burst setting configuration parameters of the Mem driver. A sector burst can be used for improved performance.
Multiplicity	0..1
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
MemEraseBurstSize	1	[ECUC_Mem_00016]
MemMaxReadSize	1	[ECUC_Mem_00018]
MemWriteBurstSize	1	[ECUC_Mem_00017]

No Included Containers

⌋

[ECUC_Mem_00016] Definition of EcucIntegerParamDef MemEraseBurstSize ⌈

Parameter Name	MemEraseBurstSize	
Parent Container	MemBurstSettings	
Upstream mapping defined	–	
Description	Size of sector erase burst in bytes. A sector burst can be used for improved performance and is typically (a subset of) a sector batch. To make use of the sector erase burst feature, the physical start address of the sector batch must be aligned to the sector erase burst size.	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	1 .. 4294967295	
Default value	–	
Pre-Compile only	true	

⌋

[ECUC_Mem_00018] Definition of EcucIntegerParamDef MemMaxReadSize ⌈

Parameter Name	MemMaxReadSize	
Parent Container	MemBurstSettings	
Upstream mapping defined	–	
Description	This value specifies the maximum number of bytes the MemAcc module requests within one Mem read request.	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	1 .. 4294967295	
Default value	–	
Pre-Compile only	true	

⌋

[ECUC_Mem_00017] Definition of EcucIntegerParamDef MemWriteBurstSize ⌈

Parameter Name	MemWriteBurstSize	
Parent Container	MemBurstSettings	
Upstream mapping defined	–	
Description	Size of page write/program burst in bytes. A sector burst can be used for improved performance and is typically (a subset of) a sector batch. To make use of the write burst feature, the physical start address must be aligned to the write burst size.	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	1 .. 4294967295	
Default value	–	
Pre-Compile only	true	

⌋

10.3 Published Information

[ECUC_Mem_00020] Definition of EcucParamConfContainerDef MemPublished Information ⌈

Container Name	MemPublishedInformation
Parent Container	Mem
Upstream mapping defined	–
Description	Additional published parameters not covered by CommonPublishedInformation container. Note that these parameters do not have any configuration class setting, since they are published information.
Multiplicity	1
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
MemErasedValue	1	[ECUC_Mem_00021]

No Included Containers

⌋

[ECUC_Mem_00021] Definition of EcucIntegerParamDef MemErasedValue ⌈

Parameter Name	MemErasedValue	
Parent Container	MemPublishedInformation	
Upstream mapping defined	–	
Description	The contents of an erased memory cell.	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	0 .. 4294967295	
Default value	–	
Pre-Compile only	true	

⌋

A Change history of AUTOSAR traceable items

A.1 Traceable item history of this document according to AUTOSAR Release R26-11

A.1.1 Added Specification Items in R26-11

Number	Heading
[ECUC_Mem_00030]	Definition of EcucChoiceReferenceDef MemExternalBusRef
[ECUC_Mem_00031]	Definition of EcucBooleanParamDef MemHsmContext

Table A.1: Added Specification Items in R26-11

A.1.2 Changed Specification Items in R26-11

Number	Heading
[ECUC_Mem_00002]	Definition of EcucParamConfContainerDef MemGeneral
[ECUC_Mem_00003]	Definition of EcucParamConfContainerDef MemInstance
[SWS_Mem_10020]	Definition of imported datatypes of module Mem

Table A.2: Changed Specification Items in R26-11

A.1.3 Deleted Specification Items in R26-11

Number	Heading
[SWS_Mem_00002]	
[SWS_Mem_10009]	Definition of API function Mem_GetVersionInfo

Table A.3: Deleted Specification Items in R26-11

A.2 Traceable item history of this document according to AUTOSAR Release R25-11

A.2.1 Added Specification Items in R25-11

none

A.2.2 Changed Specification Items in R25-11

Number	Heading
[ECUC_Mem_00024]	Definition of EcucStringParamDef MemNamePrefix

Table A.4: Changed Specification Items in R25-11

A.2.3 Deleted Specification Items in R25-11

Number	Heading
[ECUC_Mem_00026]	Definition of EcucBooleanParamDef MemUseFuncPtrTable

Table A.5: Deleted Specification Items in R25-11

A.3 Traceable item history of this document according to AUTOSAR Release R24-11

A.3.1 Added Specification Items in R24-11

Number	Heading
[ECUC_Mem_00024]	Definition of EcucStringParamDef MemNamePrefix
[ECUC_Mem_00025]	Definition of EcucEnumerationParamDef MemInvocation
[ECUC_Mem_00026]	Definition of EcucBooleanParamDef MemUseFuncPtrTable
[ECUC_Mem_00027]	Definition of EcucReferenceDef MemEcucPartitionRef
[ECUC_Mem_00028]	Definition of EcucReferenceDef MemSectorBatchEcucPartitionRef
[ECUC_Mem_00029]	Definition of EcucFloatParamDef MemMainFunctionPeriod

Table A.6: Added Specification Items in R24-11

A.3.2 Changed Specification Items in R24-11

Number	Heading
[SWS_Mem_10010]	Definition of scheduled function Mem_MainFunction
[SWS_Mem_10017]	Definition of API function Mem_HwSpecificService
[SWS_Mem_10024]	Definition of API function Mem_Suspend
[SWS_Mem_10025]	Definition of API function Mem_Resume

Table A.7: Changed Specification Items in R24-11

A.3.3 Deleted Specification Items in R24-11

Number	Heading
[SWS_Mem_00065]	

Table A.8: Deleted Specification Items in R24-11

A.4 Traceable item history of this document according to AUTOSAR Release R23-11

A.4.1 Added Specification Items in R23-11

[SWS_Mem_00093] [SWS_Mem_00094] [SWS_Mem_00095] [SWS_Mem_00096]

A.4.2 Changed Specification Items in R23-11

[SWS_Mem_10004] [SWS_Mem_10011] [SWS_Mem_10020]

A.4.3 Deleted Specification Items in R23-11

[SWS_Mem_00003] [SWS_Mem_00008] [SWS_Mem_00014] [SWS_Mem_00019] [SWS_Mem_00021] [SWS_Mem_00068] [SWS_Mem_00085] [SWS_Mem_00086] [SWS_Mem_10019]

B Not applicable requirements

No content.