Document Title	Specification of Flash EEPROM Emulation
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	286

Document Status	published
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	R24-11

Document Change History

Date	Release	Changed by	Description
2024-11-27	R24-11	AUTOSAR Release Management	• Removed draft status of MemAcc_Compare in [SWS_Fee_00105]
2023-11-23	R23-11	AUTOSAR Release Management	• Fixed incorrect description of return value in `Fee_InvalidateBlock` and `Fee_EraseImmediateBlock`
2022-11-24	R22-11	AUTOSAR Release Management	• Removed obsolete items • Changed [SWS_Fee_00999] to [SWS_Fee_NA_00999] • Set items to valid: – [SWS_Fee_00194] – [SWS_Fee_00195] – [SWS_Fee_00196]
2021-11-25	R21-11	AUTOSAR Release Management	• Updated for new memory stack • Removed return codes for Det errors • Removed definitions of NVM functions
2020-11-30	R20-11	AUTOSAR Release Management	• Fixed inconsistency in the example of [SWS_Fee_00100] • Removed `FEE_E_INIT_FAILED`

▽

△

2019-11-28	R19-11	AUTOSAR Release Management	• Added diagrams in chapter 10 • Added limitation about parallel access to Flash Driver • Changed Document Status from Final to published
2018-10-31	4.4.0	AUTOSAR Release Management	• Fixed typo in sequence diagram
2017-12-08	4.3.1	AUTOSAR Release Management	• Introduction of runtime errors • Adjusted references
2016-11-30	4.3.0	AUTOSAR Release Management	• Updated tracing information • Behaviour during MEMIF_BUSY_INTERNAL reworked • Range of main function adapted
2015-07-31	4.2.2	AUTOSAR Release Management	• Behaviour during FEE_BUSY_INTERNAL reworked • Error classification reworked • Debugging support marked as obsolete • Job result clarified if requested block can’t be found
2014-10-31	4.2.1	AUTOSAR Release Management	• Requirement for blank checking added • Requirements linked to features, general and module specific requirements
2014-03-31	4.1.3	AUTOSAR Release Management	• Editorial changes
2013-10-31	4.1.2	AUTOSAR Release Management	• Timing requirement removed from module’s main function • "const" qualifier added to prototype of function Fee_Write • New configuration parameter FeeMainFunctionPeriod • Editorial changes • Removed chapter(s) on change documentation

▽

△

2013-03-15	4.1.1	AUTOSAR Administration	• Reworked according to the new SWS_BSWGeneral • Scope attribute in tables in chapter 10 added • Published parameter FeeMaximumBlockingTime deprecated • Configuration parameter FeeIndex deprecated
2011-12-22	4.0.3	AUTOSAR Administration	• DET errors added / removed • Handling of internal management operations detailed • Module short name changed • Consistency checking reformulated
2010-09-30	3.1.5	AUTOSAR Administration	• Inter-module checks clarified [SWS_Fee_00013] • Sequence diagram for Fee_Cancel replaced for generated one • Naming in [ECUC_Fee_00150] corrected to NVM_DATASET_SELECTION_BITS • Sequence diagram for Fee_Init extended • Handling of internal management operations refined ([SWS_Fee_00022], [SWS_Fee_00025], [SWS_Fee_00173], [SWS_Fee_00174], [SWS_Fee_00183]) • Inter module checks detailed ([SWS_Fee_00013]) • NvM_Cbk.h added to file include structure ([SWS_Fee_00002]) • Ranges for FeeBlockNumber ([ECUC_Fee_00150]) and FeeBlockSize ([ECUC_Fee_00148]) adjusted ▽

▽

△

			△ • Initialization might not be finished within `Fee_Init`, state machine adapted accordingly ([SWS_Fee_00120], [SWS_Fee_00168], [SWS_Fee_00169]) • Handling of internal management operations refined ([SWS_Fee_00170] .. [SWS_Fee_00182] e.a.)
2010-02-02	3.1.4	AUTOSAR Administration	• Configuration variants clarified • Job result handling re-formulated • Range of configuration parameters restricted • Legal disclaimer revised
2008-08-13	3.1.1	AUTOSAR Administration	• Legal disclaimer revised
2007-12-21	3.0.1	AUTOSAR Administration	• Small reformulations resulting from table generation • Tables in chapters 8 and 10 generated from UML model • Document meta information extended • Small layout adaptations made
2007-01-24	2.1.15	AUTOSAR Administration	• File include structure updated • API of initialization function adapted • Range of FEE block numbers adapted • Various API descriptions enhanced • Legal disclaimer revised • Release Notes added • "Advice for users" revised • "Revision Information" added
2006-05-16	2.0	AUTOSAR Administration	• Initial release

Disclaimer

Contents

1 Introduction and functional overview 8

2 Acronyms and Abbreviations 10

3 Related documentation 11

3.1 Input documents & related standards and norms 11
3.2 Related specification 11

4 Constraints and assumptions 12

4.1 Limitations 12
4.2 Applicability to car domains 12

5 Dependencies to other modules 13

6 Requirements Tracing 14

7 Functional specification 16

7.1 General behavior 16
7.1.1 Addressing scheme and segmentation 16
7.1.2 Address calculation 18
7.1.3 Limitation of erase cycles 19
7.1.4 Handling of "immediate" data 20
7.1.5 Managing block correctness information 21
7.1.6 Buffer Alignment 21
7.2 Error Classification 21
7.2.1 Development Errors 22
7.2.2 Runtime Errors 22
7.2.3 Production Errors 22
7.2.4 Extended Production Errors 22

8 API specification 23

8.1 Imported types 23
8.2 Type definitions 23
8.3 Function definitions 24
8.3.1 Fee_Init 24
8.3.2 Fee_Read 25
8.3.3 Fee_Write 27
8.3.4 Fee_Cancel 29
8.3.5 Fee_GetStatus 31
8.3.6 Fee_GetJobResult 32
8.3.7 Fee_InvalidateBlock 34
8.3.8 Fee_GetVersionInfo 36
8.3.9 Fee_EraseImmediateBlock 36
8.4 Callback notifications 38
8.4.1 Fee_JobEndNotification 39

8.5 Scheduled functions . . . 40
8.5.1 Fee_MainFunction . . . 40
8.6 Expected interfaces . . . 41
8.6.1 Mandatory Interfaces . . . 41
8.6.2 Optional Interfaces . . . 42
8.6.3 Configurable interfaces . . . 43

9 Sequence diagrams 45

9.1 Fee_Init . . . 45
9.2 Fee_Write . . . 46
9.3 Fee_Cancel . . . 46

10 Configuration specification 49

10.1 Containers and configuration parameters . . . 49
10.1.1 Fee . . . 49
10.1.2 FeeGeneral . . . 50
10.1.3 FeeBlockConfiguration . . . 55
10.2 Published Information . . . 58
10.2.1 FeePublishedInformation . . . 58

A Not applicable requirements 60

B Change history of AUTOSAR traceable items 61

B.1 Traceable item history of this document according to AUTOSAR Release R24-11 . . . 61
B.1.1 Added Specification Items in R24-11 . . . 61
B.1.2 Changed Specification Items in R24-11 . . . 61
B.1.3 Deleted Specification Items in R24-11 . . . 61

1 Introduction and functional overview

This specification describes the functionality, API and configuration of the Flash EEPROM Emulation Module.

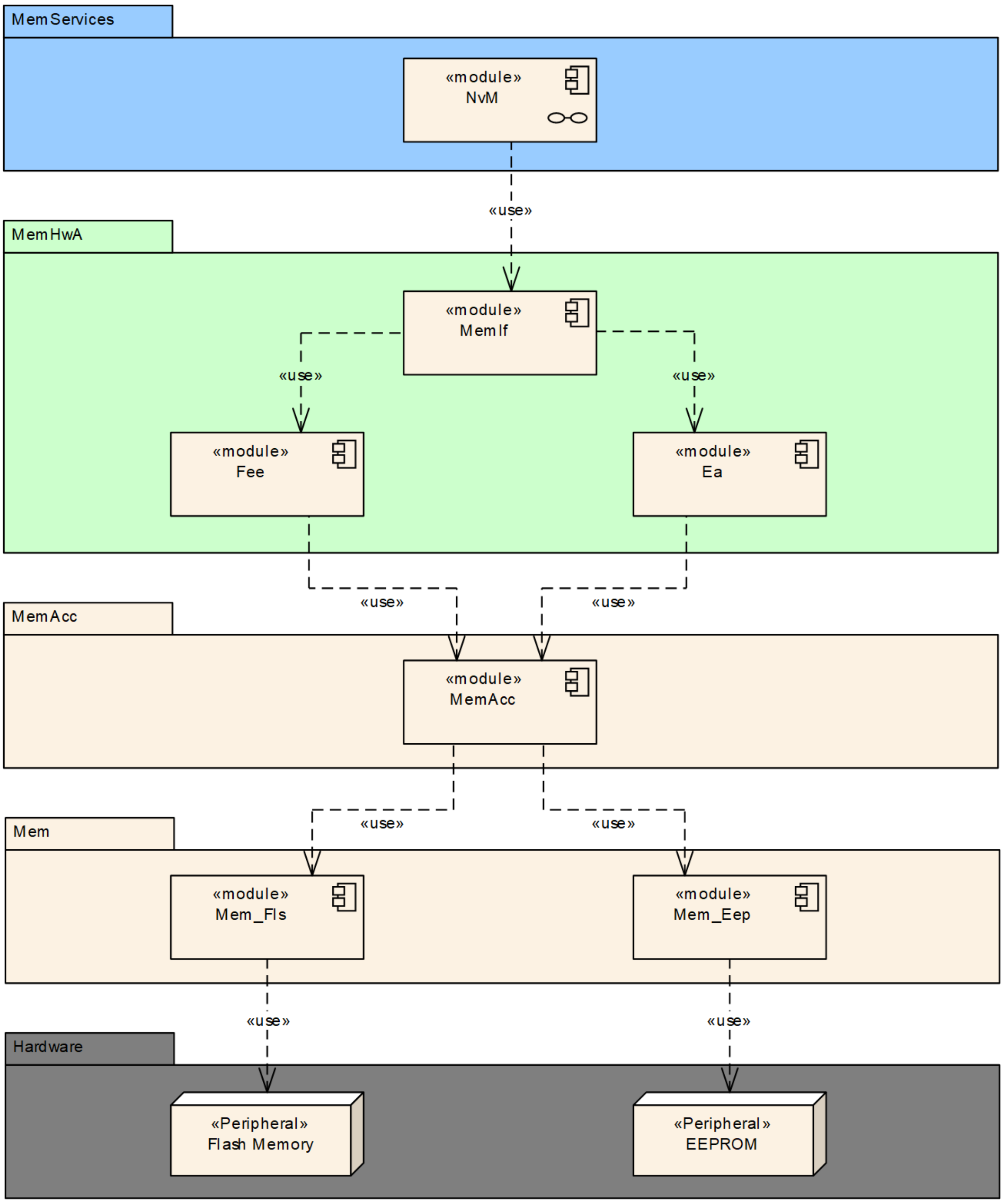

Figure 1.1: Module overview of memory stack

The Flash EEPROM Emulation (FEE) shall abstract from the device specific addressing scheme and segmentation and provide the upper layers with a virtual addressing scheme and segmentation as well as a "virtually" unlimited number of erase cycles.

2 Acronyms and Abbreviations

The glossary below includes acronyms and abbreviations relevant to the Fee module that are not included in the [1, AUTOSAR glossary].

Abbreviation / Acronym:	Description:
EA	EEPROM Abstraction
Address Area	Contiguous memory area in the logical address space typically multiple physical memory sectors are combined to one logical address area.
EEPROM	Electrically Erasable and Programmable ROM (Read Only Memory)
FEE	Flash EEPROM Emulation
LSB	Least significant bit / byte (depending on context). Here, "bit" is meant.
Mem	Memory Driver
MemAcc	Memory Access
MemIf	Memory Abstraction Interface
MSB	Most significant bit / byte (depending on context). Here, "bit" is meant.
NvM	NVRAM Manager
NVRAM	Non-volatile RAM (Random Access Memory)
NVRAM block	Management unit as seen by the NVRAM Manager
(Logical) block	Smallest writable / erasable unit as seen by the modules user. Consists of one or more virtual pages.
Virtual page	May consist of one or several physical pages to ease handling of logical blocks and address calculation.
Internal residue	Unused space at the end of the last virtual page if the configured block size isn't an integer multiple of the virtual page size (see Figure 7.2).
Virtual address	Consisting of 16 bit block number and 16 bit offset inside the logical block.
Physical address	Address information in device specific format (depending on the underlying EEPROM driver and device) that is used to access a logical block.
Dataset	Concept of the NVRAM manager: A user addressable array of blocks of the same size. E.g. could be used to provide different configuration settings for the CAN driver (CAN IDs, filter settings, ...) to an ECU which has otherwise identical application software (e.g. door module).
Redundant copy	Concept of the NVRAM manager: Storing the same information twice to enhance reliability of data storage.

Table 2.1: Acronyms and abbreviations used in the scope of this Document

3 Related documentation

3.1 Input documents & related standards and norms

[1] Glossary
AUTOSAR_FO_TR_Glossary

[2] General Specification of Basic Software Modules
AUTOSAR_CP_SWS_BSWGeneral

[3] Specification of ECU State Manager
AUTOSAR_CP_SWS_ECUStateManager

[4] Specification of Memory Abstraction Interface
AUTOSAR_CP_SWS_MemoryAbstractionInterface

3.2 Related specification

AUTOSAR provides a General Specification on Basic Software modules [2, SWS BSW General], which is also valid for Flash EEPROM Emulation.

Thus, the specification SWS BSW General shall be considered as additional and required specification for Flash EEPROM Emulation.

4 Constraints and assumptions

4.1 Limitations

No limitations.

4.2 Applicability to car domains

No restrictions.

5 Dependencies to other modules

This module depends on the capabilities of the underlying flash driver as well as the configuration of the NVRAM manager.

6 Requirements Tracing

Requirement	**Description**	**Satisfied by**
[RS_BRF_01048]	AUTOSAR module design shall support modules to cooperate in a multitasking environment	[SWS_Fee_00026] [SWS_Fee_00035] [SWS_Fee_00057] [SWS_Fee_00073] [SWS_Fee_00074] [SWS_Fee_00075] [SWS_Fee_00091] [SWS_Fee_00097] [SWS_Fee_00128] [SWS_Fee_00129] [SWS_Fee_00133] [SWS_Fee_00144] [SWS_Fee_00145] [SWS_Fee_00146] [SWS_Fee_00155] [SWS_Fee_00156] [SWS_Fee_00158] [SWS_Fee_00162] [SWS_Fee_00163] [SWS_Fee_00164] [SWS_Fee_00172] [SWS_Fee_00174]
[RS_BRF_01064]	AUTOSAR BSW shall provide callback functions in order to access upper layer modules	[SWS_Fee_00052] [SWS_Fee_00055] [SWS_Fee_00056] [SWS_Fee_00095] [SWS_Fee_00142] [SWS_Fee_00194]
[RS_BRF_01076]	AUTOSAR basic software shall perform module local error recovery to the extent possible	[SWS_Fee_00187]
[RS_BRF_01812]	AUTOSAR non-volatile memory functionality shall support the prioritization and asynchronous execution of jobs	[SWS_Fee_00193]
[SRS_BSW_00101]	The Basic Software Module shall be able to initialize variables and hardware in a separate initialization function	[SWS_Fee_00085] [SWS_Fee_00168] [SWS_Fee_00169]
[SRS_BSW_00323]	All AUTOSAR Basic Software Modules shall check passed API parameters for validity	[SWS_Fee_00068] [SWS_Fee_00134] [SWS_Fee_00135] [SWS_Fee_00136] [SWS_Fee_00137] [SWS_Fee_00138] [SWS_Fee_00139] [SWS_Fee_00140] [SWS_Fee_00141] [SWS_Fee_00147]
[SRS_BSW_00327]	Error values naming convention	[SWS_Fee_00010]
[SRS_BSW_00331]	All Basic Software Modules shall strictly separate error and status information	[SWS_Fee_00010]
[SRS_BSW_00337]	Classification of development errors	[SWS_Fee_00010]
[SRS_BSW_00384]	The Basic Software Module specifications shall specify at least in the description which other modules they require	[SWS_Fee_00104] [SWS_Fee_00105]
[SRS_BSW_00386]	The BSW shall specify the configuration and conditions for detecting an error	[SWS_Fee_00010]
[SRS_BSW_00392]	Parameters shall have a type	[SWS_Fee_00016] [SWS_Fee_00084]
[SRS_BSW_00406]	API handling in uninitialized state	[SWS_Fee_00010] [SWS_Fee_00034] [SWS_Fee_00090] [SWS_Fee_00122] [SWS_Fee_00123] [SWS_Fee_00124] [SWS_Fee_00125] [SWS_Fee_00126] [SWS_Fee_00127]
[SRS_BSW_00407]	Each BSW module shall provide a function to read out the version information of a dedicated module implementation	[SWS_Fee_00093]
[SRS_BSW_00414]	Init functions shall have a pointer to a configuration structure as single parameter	[SWS_Fee_00188]

▽

△

Requirement	Description	Satisfied by
[SRS_MemHwAb_-14001]	The FEE and EA modules shall allow the configuration of the alignment of the start and end addresses of logical blocks	[SWS_Fee_00005] [SWS_Fee_00071] [SWS_Fee_00076]
[SRS_MemHwAb_-14002]	The FEE and EA modules shall allow the configuration of a required number of write cycles for each logical block	[SWS_Fee_00102] [SWS_Fee_00103]
[SRS_MemHwAb_-14005]	The FEE and EA modules shall provide upper layer modules with a virtual 32bit address space	[SWS_Fee_00076]
[SRS_MemHwAb_-14006]	The start address for a block erase or write operation shall always be aligned to the virtual 64K boundary	[SWS_Fee_00024]
[SRS_MemHwAb_-14007]	The start address and length for reading a block shall not be limited to a certain alignment	[SWS_Fee_00021]
[SRS_MemHwAb_-14009]	The FEE and EA modules shall provide a conversion between the logical linear addresses and the physical memory addresses	[SWS_Fee_00007] [SWS_Fee_00036] [SWS_Fee_00066] [SWS_Fee_00100]
[SRS_MemHwAb_-14010]	The FEE and EA modules shall provide a write service that operates only on complete configured logical blocks	[SWS_Fee_00025] [SWS_Fee_00026] [SWS_Fee_00088]
[SRS_MemHwAb_-14012]	Spreading of write access	[SWS_Fee_00102] [SWS_Fee_00103]
[SRS_MemHwAb_-14013]	Writing of immediate data shall not be delayed by internal management operations nor by erasing the memory area to be written to	[SWS_Fee_00009] [SWS_Fee_00067]
[SRS_MemHwAb_-14014]	The FEE and EA modules shall detect possible data inconsistencies due to aborted / interrupted write operations	[SWS_Fee_00023] [SWS_Fee_00049] [SWS_Fee_00153] [SWS_Fee_00154] [SWS_Fee_00159]
[SRS_MemHwAb_-14015]	The FEE and EA modules shall report possible data inconsistencies	[SWS_Fee_00023]
[SRS_MemHwAb_-14016]	The FEE and EA modules shall not return inconsistent data to the caller	[SWS_Fee_00023]
[SRS_MemHwAb_-14026]	The block numbers 0x0000 and 0x FFFF shall not be used	[SWS_Fee_00006]
[SRS_MemHwAb_-14028]	The FEE and EA modules shall provide a service to invalidate a logical block	[SWS_Fee_00037] [SWS_Fee_00075] [SWS_Fee_00092] [SWS_Fee_00160] [SWS_Fee_00165] [SWS_Fee_00192]
[SRS_MemHwAb_-14029]	The FEE and EA modules shall provide a read service that allows reading all or part of a logical block	[SWS_Fee_00022] [SWS_Fee_00087]
[SRS_MemHwAb_-14031]	The FEE and EA modules shall provide a service that allows canceling an ongoing asynchronous operation	[SWS_Fee_00080] [SWS_Fee_00081] [SWS_Fee_00089] [SWS_Fee_00157] [SWS_Fee_00184]
[SRS_MemHwAb_-14032]	The FEE and EA modules shall provide an erase service that operates only on complete logical blocks containing immediate data	[SWS_Fee_00094] [SWS_Fee_00166]

Table 6.1: Requirements Tracing

7 Functional specification

7.1 General behavior

7.1.1 Addressing scheme and segmentation

The Flash EEPROM Emulation (FEE) module provides upper layers with a 32bit virtual linear address space and uniform segmentation scheme. This virtual 32bit addresses shall consist of

- a 16bit block number - allowing a (theoretical) number of 65536 logical blocks
- a 16bit block offset - allowing a (theoretical) block size of 64KByte per block

The 16bit block number represents a configurable (virtual) paging mechanism. The values for this address alignment can be derived from that of the underlying flash driver and device. This virtual paging shall be configurable via the parameter `FeeVirtualPageSize`.

[SWS_Fee_00076]

Upstream requirements: SRS_MemHwAb_14001, SRS_MemHwAb_14005

⌈The configuration of the Fee module shall be such that the virtual page size (defined in `FeeVirtualPageSize`) is an integer multiple of the physical page size, i.e. it is not allowed to configure a smaller virtual page than the actual physical page size.⌋

Note: This specification requirement allows the physical start address of a logical block to be calculated rather than making a lookup table necessary for the address mapping.

Example:

The size of a virtual page is configured to be eight bytes, thus the address alignment is eight bytes. The logical block with block number 1 is placed at physical address x. The logical block with the block number 2 then would be placed at x+8, block number 3 would be placed at x+16.

[SWS_Fee_00005]

Upstream requirements: SRS_MemHwAb_14001

⌈Each configured logical block shall take up an integer multiple of the configured virtual page size (see also Chapter configuration parameter `FeeVirtualPageSize`).⌋

Example:

The address alignment / virtual paging is configured to be eight bytes by setting the parameter `FeeVirtualPageSize` accordingly. The logical block number 1 is configured to have a size of 32 bytes (see Figure 7.1). This logical block would use exactly 4 virtual pages. The next logical block thus would get the block number 5, since block *numbers 2, 3 and 4 are "blocked" by the first logical block. This second block is configured to have a size of 100 bytes, taking up 13 virtual pages and leaving 4 bytes of the last page unused. The next available logical block number thus would be 17.*

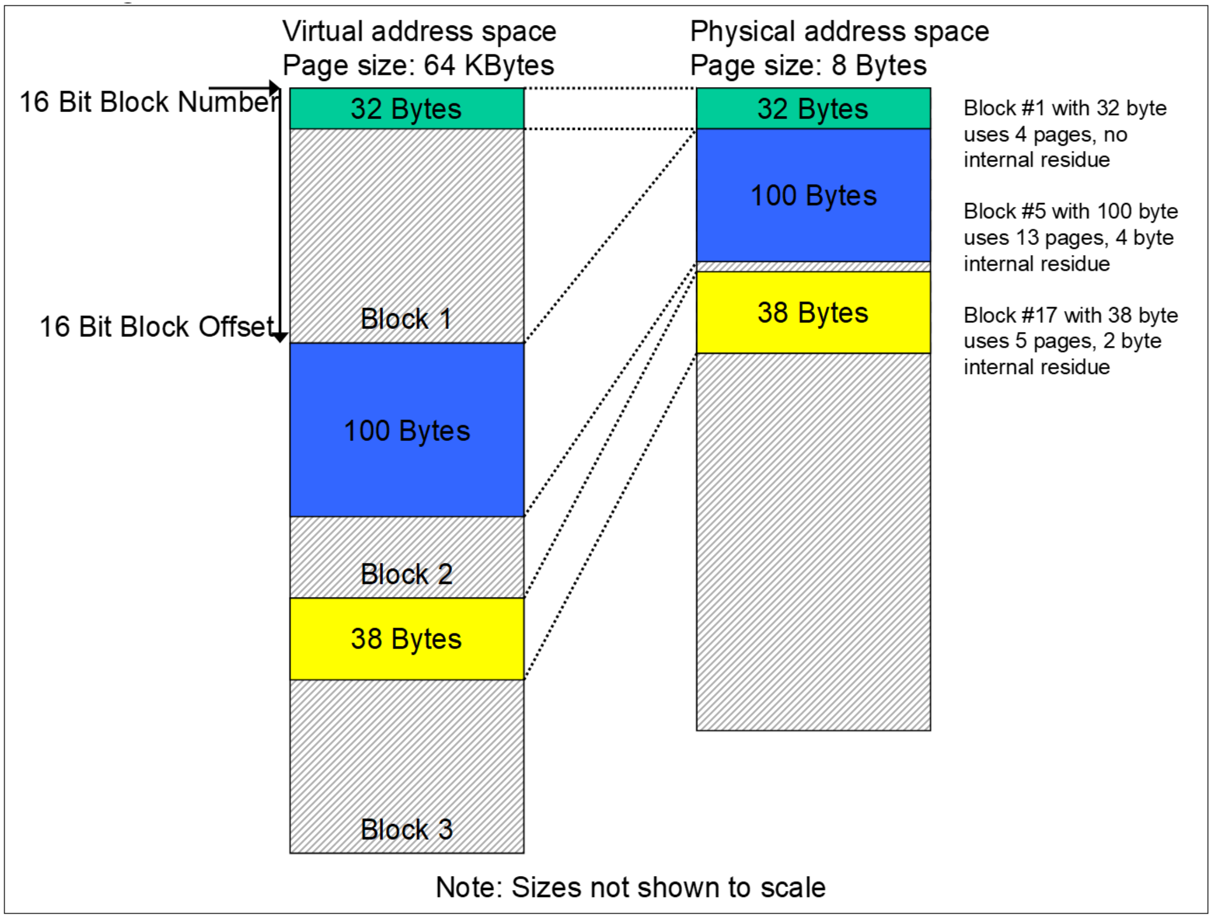
Virtual address space
Page size: 64 KBytes
Physical address space
Page size: 8 Bytes
16 Bit Block Number
16 Bit Block Offset
32 Bytes
Block 1
100 Bytes
Block 2
38 Bytes
Block 3
32 Bytes
100 Bytes
38 Bytes
Block #1 with 32 byte uses 4 pages, no internal residue
Block #5 with 100 byte uses 13 pages, 4 byte internal residue
Block #17 with 38 byte uses 5 pages, 2 byte internal residue
Note: Sizes not shown to scale

Figure 7.1: Virtual vs. physical memory layout

[SWS_Fee_00071]

Upstream requirements: SRS_MemHwAb_14001

⌈Logical blocks must not overlap each other and must not be contained within one another.⌋

[SWS_Fee_00006]

Upstream requirements: SRS_MemHwAb_14026

⌈The block numbers 0x0000 and 0xFFFF shall not be configurable for a logical block.⌋

7.1.2 Address calculation

[SWS_Fee_00007]

Upstream requirements: SRS_MemHwAb_14009

⌈Depending on the implementation of the FEE module and the exact address format used, the functions of the FEE module shall combine the 16bit block number and 16bit address offset to derive the physical flash address needed for the underlying flash driver.⌋

Note: The exact address format needed by the underlying flash driver and therefore the mechanism how to derive the physical flash address from the given 16bit block number and 16bit address offset depends on the flash device and the implementation of this module and shall therefore not be standardized.

[SWS_Fee_00100]

Upstream requirements: SRS_MemHwAb_14009

⌈Only those bits of the 16bit block number, that do not denote a specific dataset or redundant copy shall be used for address calculation.⌋

Note: Since this information is needed by the NVRAM manager, the number of bits to encode this can be configured for the NVRAM manager with the parameter `NVM_DATASET_SELECTION_BITS`.

Example:

Dataset information is configured to be encoded in the four LSB's of the 16bit block number (allowing for a maximum of 16 datasets per NVRAM block and a total of 4094 NVRAM blocks). An implementer decides to store all datasets of a NVRAM block directly adjacent and using the length of the block and a pointer to access each dataset. To calculate the start address of the block (the address of the first dataset) she/he uses only the 12 MSB's, to access a specific dataset she/he adds the size of the block multiplied by the dataset index (the four LSB's) to this start address (Figure 7.2).

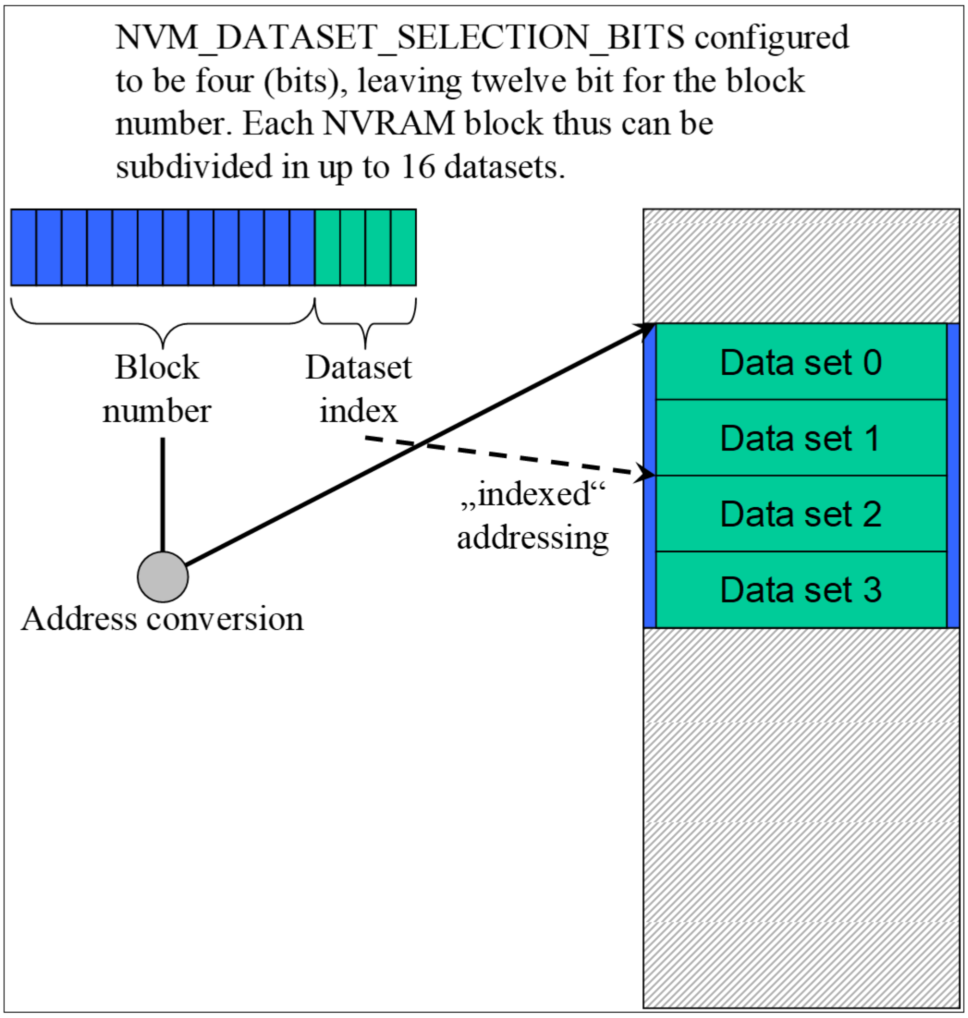
NVM_DATASET_SELECTION_BITS configured to be four (bits), leaving twelve bit for the block number. Each NVRAM block thus can be subdivided in up to 16 datasets.
Block number
Dataset index
„indexed“ addressing
Address conversion
Data set 0
Data set 1
Data set 2
Data set 3

Figure 7.2: Block number and dataset index

7.1.3 Limitation of erase cycles

[SWS_Fee_00102]

Upstream requirements: SRS_MemHwAb_14002, SRS_MemHwAb_14012

⌈The configuration of the FEE module shall define the expected number of erase/write cycles for each logical block in the configuration parameter `FeeNumberOfWriteCycles`.⌋

[SWS_Fee_00103]

Upstream requirements: SRS_MemHwAb_14002, SRS_MemHwAb_14012

⌈If the underlying flash device or device driver does not provide at least the configured number of erase/write cycles per physical memory cell, the FEE module shall provide mechanisms to spread the write access such that the physical device is not overstressed. This shall also apply to all management data used internally by the FEE module.⌋

Example:

The logical block number 1 is configured for an expected 500.000 write cycles, the underlying flash device and device driver are only specified for 100.000 erase cycles. In this case, the FEE module has to provide (at least) five separate memory areas and alternate the access between those areas internally so that each physical memory location is only erased for a maximum of the specified 100.000 cycles.

7.1.4 Handling of "immediate" data

[SWS_Fee_00009]

Upstream requirements: SRS_MemHwAb_14013

⌈Blocks containing immediate data have to be written instantaneously, i.e. the FEE module has to ensure that it can write such blocks without the need to erase the corresponding memory area (e.g. by using pre-erased memory) and that the write request is not delayed by currently running module internal management operations.⌋

Note: An ongoing lower priority read / erase / write or compare job shall be canceled by the NVRAM manager before immediate data is written. The FEE module has only to ensure that this write request can be performed immediately.

Note: A running operation on the hardware (e.g. writing one page or erasing one sector) can usually not be aborted once it has been started. The maximum time of the longest hardware operation thus has to be accepted as delay even for immediate data.

Example:

Three blocks with 10 bytes each have been configured for immediate data. The FEE module / configuration tool reserves these 30 bytes (plus the implementation specific overhead per block / page if needed) for use by this immediate data only. That is, this memory area shall not be used for storage of other data blocks.

Now, the NVRAM manager has requested the FEE module to write a data block of 100 bytes. While this block is being written, a situation occurs that one (or several) of the immediate data blocks need to be written. Therefore the NVRAM manager cancels the ongoing write request and subsequently issues the write request for the (first) block containing immediate data. The cancelation of the ongoing write request is performed synchronously by the FEE module and the underlying flash driver (i.e. the write request for the immediate data) can be started without any further delay. However, before the first bytes of immediate data can be written, the FEE module or rather the underlying flash driver have to wait for the end of an ongoing hardware access from the previous write request (e.g. writing of a page, erasing of a sector, transfer via SPI, ...).

7.1.5 Managing block correctness information

[SWS_Fee_00049]

Upstream requirements: SRS_MemHwAb_14014

⌈The FEE module shall manage for each block the information, whether this block is correct (i.e. "not corrupted") from the point of view of the FEE module or not. This information shall only concern the internal handling of the block, not the block's contents.⌋

[SWS_Fee_00153]

Upstream requirements: SRS_MemHwAb_14014

⌈When a block write operation is started, the FEE module shall mark the corresponding block as "corrupted"[1].⌋

[SWS_Fee_00154]

Upstream requirements: SRS_MemHwAb_14014

⌈Upon the successful end of the block write operation, the block shall be marked as "not corrupted" (again).⌋

Note: This internal management information should not be mixed up with the validity information of a block which can be manipulated by using the Fee_InvalidateBlock service, i.e. the FEE shall be able to distinguish between a corrupted block and a block that has been deliberately invalidated by the upper layer.

7.1.6 Buffer Alignment

[SWS_Fee_00195] ⌈The Fee shall align internal buffers to the value of `FeeBufferAlignmentValue`⌋

[SWS_Fee_00196] ⌈The Fee shall align read request to the value of `FeeMinimumReadPageSize`⌋

7.2 Error Classification

Section "Error Handling" of the document [2] "General Specification of Basic Software Modules" describes the error handling of the Basic Software in detail. Above all, it

[1]This does not necessarily mean a write operation on the physical device, if there are other means to detect the consistency of a logical block.

constitutes a classification scheme consisting of five error types which may occur in BSW modules.

Based on this foundation, the following section specifies particular errors arranged in the respective subsections below.

7.2.1 Development Errors

[SWS_Fee_00010] Definiton of development errors in module Fee

Upstream requirements: SRS_BSW_00406, SRS_BSW_00337, SRS_BSW_00386, SRS_BSW_-00327, SRS_BSW_00331

⌈

Type of error	*Related error code*	*Error value*
API service called when module was not initialized	FEE_E_UNINIT	0x01
API service called with invalid block number	FEE_E_INVALID_BLOCK_NO	0x02
API service called with invalid block offset	FEE_E_INVALID_BLOCK_OFS	0x03
API service called with invalid data pointer	FEE_E_PARAM_POINTER	0x04
API service called with invalid length information	FEE_E_INVALID_BLOCK_LEN	0x05

⌋

7.2.2 Runtime Errors

[SWS_Fee_91002] Definiton of runtime errors in module Fee ⌈

Type of error	*Related error code*	*Error value*
API service called while module is busy processing a user request	FEE_E_BUSY	0x06
Fee_Cancel called while no job was pending.	FEE_E_INVALID_CANCEL	0x08

⌋

7.2.3 Production Errors

There are no production errors.

7.2.4 Extended Production Errors

There are no extended production errors.

8 API specification

8.1 Imported types

[SWS_Fee_00084] Definition of imported datatypes of module Fee

Upstream requirements: SRS_BSW_00392

⌈

Module	***Header File***	***Imported Type***
MemAcc	MemAcc_GeneralTypes.h	MemAcc_AddressAreaIdType
	MemAcc_GeneralTypes.h	MemAcc_AddressType
	MemAcc_GeneralTypes.h	MemAcc_DataType
	MemAcc_GeneralTypes.h	MemAcc_JobResultType
	MemAcc_GeneralTypes.h	MemAcc_LengthType
MemIf	MemIf.h	MemIf_JobResultType
	MemIf.h	MemIf_StatusType
Std	Std_Types.h	Std_ReturnType
	Std_Types.h	Std_VersionInfoType

⌋

[SWS_Fee_00016]

Upstream requirements: SRS_BSW_00392

⌈The types mentioned in [SWS_Fee_00084] shall not be changed or extended for a specific FEE module or hardware platform.⌋

8.2 Type definitions

[SWS_Fee_00188] Definition of datatype Fee_ConfigType

Upstream requirements: SRS_BSW_00414

⌈

Name	Fee_ConfigType	
Kind	Structure	
Elements	implementation specific	
	Type	–
	Comment	–
Description	Configuration data structure of the Fee module.	
Available via	Fee.h	

⌋

8.3 Function definitions

8.3.1 Fee_Init

[SWS_Fee_00085] Definition of API function Fee_Init

Upstream requirements: SRS_BSW_00101

⌈

Service Name	Fee_Init	

Syntax	`void Fee_Init (`
	`  const Fee_ConfigType* ConfigPtr`
	`)`	

Service ID [hex]	0x00	
Sync/Async	Asynchronous	
Reentrancy	Non Reentrant	
Parameters (in)	ConfigPtr	Pointer to the selected configuration set.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Service to initialize the FEE module.	
Available via	Fee.h	

⌋

[SWS_Fee_00168]

Upstream requirements: SRS_BSW_00101

⌈If initialization is finished within `Fee_Init`, the function `Fee_Init` shall set the module state from `MEMIF_UNINIT` to `MEMIF_IDLE` once initialization has been successfully finished.⌋

Note: The FEE module's environment shall not call the function `Fee_Init` during a running operation of the FEE module.

8.3.2 Fee_Read

[SWS_Fee_00087] Definition of API function Fee_Read

Upstream requirements: SRS_MemHwAb_14029

⌈

Service Name	Fee_Read	

Syntax	`Std_ReturnType Fee_Read (`
	`  uint16 BlockNumber,`
	`  uint16 BlockOffset,`
	`  uint8* DataBufferPtr,`
	`  uint16 Length`
	`)`	

Service ID [hex]	0x02	
Sync/Async	Asynchronous	
Reentrancy	Non Reentrant	
Parameters (in)	BlockNumber	Number of logical block, also denoting start address of that block in flash memory.
	BlockOffset	Read address offset inside the block
	Length	Number of bytes to read
Parameters (inout)	None	
Parameters (out)	DataBufferPtr	Pointer to data buffer
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.
Description	Service to initiate a read job.	
Available via	Fee.h	

⌋

[SWS_Fee_00021]

Upstream requirements: SRS_MemHwAb_14007

⌈The function Fee_Read shall take the block start address and offset and calculate the corresponding memory read address.⌋

Note: The address offset and length parameter can take any value within the given types range. This allows reading of an arbitrary number of bytes from an arbitrary start address inside a logical block.

[SWS_Fee_00022]

Upstream requirements: SRS_MemHwAb_14029

⌈If the current module status is MEMIF_IDLE or if the current module status is MEMIF_BUSY_INTERNAL, the function Fee_Read shall accept the read request, copy the given / computed parameters to module internal variables, initiate a read job, set the FEE module status to MEMIF_BUSY, set the job result to MEMIF_JOB_PENDING and return with E_OK.⌋

[SWS_Fee_00172]

Upstream requirements: RS_BRF_01048

⌈If the current module status is MEMIF_UNINIT or MEMIF_BUSY, the function Fee_Read shall reject the job request and return with E_NOT_OK.⌋

[SWS_Fee_00073]

Upstream requirements: RS_BRF_01048

⌈The FEE module shall execute the read operation asynchronously within the FEE module's main function.⌋

[SWS_Fee_00122]

Upstream requirements: SRS_BSW_00406

⌈If development error detection is enabled for the module: the function Fee_Read shall check if the module state is MEMIF_UNINIT. If this is the case, the function Fee_Read shall raise the development error FEE_E_UNINIT.⌋

[SWS_Fee_00133]

Upstream requirements: RS_BRF_01048

⌈The function Fee_Read shall check if the module state is MEMIF_BUSY. If this is the case, the function Fee_Read shall reject the read request, raise the runtime error FEE_E_BUSY and return with E_NOT_OK.⌋

[SWS_Fee_00134]

Upstream requirements: SRS_BSW_00323

⌈If development error detection is enabled for the module: the function Fee_Read shall check that the given block number is valid (i.e. it has been configured). If this is not the case, the function Fee_Read shall raise the development error FEE_E_INVALID_BLOCK_NO.⌋

[SWS_Fee_00135]

Upstream requirements: SRS_BSW_00323

⌈If development error detection is enabled for the module: the function Fee_Read shall check that the given block offset is valid (i.e. that it is less than the block length configured for this block). If this is not the case, the function Fee_Read shall raise the development error FEE_E_INVALID_BLOCK_OFS.⌋

[SWS_Fee_00136]

Upstream requirements: SRS_BSW_00323

⌈If development error detection is enabled for the module: the function Fee_Read shall check that the given data pointer is valid (i.e. that it is not NULL). If this is not the case, the function Fee_Read shall raise the development error FEE_E_PARAM_POINTER.⌋

[SWS_Fee_00137]

Upstream requirements: SRS_BSW_00323

⌈If development error detection is enabled for the module: the function Fee_Read shall check that the given length information is valid, i.e. that the requested length information plus the block offset do not exceed the block end address (block start address plus configured block length). If this is not the case, the function Fee_Read shall raise the development error FEE_E_INVALID_BLOCK_LEN.⌋

[SWS_Fee_00162]

Upstream requirements: RS_BRF_01048

⌈If a read request is rejected by the function Fee_Read, i.e. requirements [SWS_Fee_00122], [SWS_Fee_00133], [SWS_Fee_00134], [SWS_Fee_00135], [SWS_Fee_00136] or [SWS_Fee_00137] apply, the function Fee_Read shall not change the current module status or job result.⌋

[SWS_Fee_00187]

Upstream requirements: RS_BRF_01076

⌈The function Fee_Read shall call the function MemAcc_BlankCheck to determine in advance whether a given memory area can be read without encountering e.g. ECC errors due to trying to read erased but not programmed flash cells.⌋

8.3.3 Fee_Write

[SWS_Fee_00088] Definition of API function Fee_Write

Upstream requirements: SRS_MemHwAb_14010

⌈

Service Name	Fee_Write

Syntax	`Std_ReturnType Fee_Write (`
	`  uint16 BlockNumber,`
	`  const uint8* DataBufferPtr`
	`)`

Service ID [hex]	0x03

▽

△

Sync/Async	Asynchronous	
Reentrancy	Non Reentrant	
Parameters (in)	BlockNumber	Number of logical block, also denoting start address of that block in EEPROM.
	DataBufferPtr	Pointer to data buffer
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.
Description	Service to initiate a write job.	
Available via	Fee.h	

⌋

[SWS_Fee_00024]

Upstream requirements: SRS_MemHwAb_14006

⌈The function Fee_Write shall take the block start address and calculate the corresponding memory write address. The block address offset shall be fixed to zero.⌋

[SWS_Fee_00025]

Upstream requirements: SRS_MemHwAb_14010

⌈If the current module status is MEMIF_IDLE or if the current module status is MEMIF_BUSY_INTERNAL, the function Fee_Write shall accept the write request, copy the given / computed parameters to module internal variables, initiate a write job, set the FEE module status to MEMIF_BUSY, set the job result to MEMIF_JOB_PENDING and return with E_OK.⌋

[SWS_Fee_00174]

Upstream requirements: RS_BRF_01048

⌈If the current module status is MEMIF_UNINIT or MEMIF_BUSY, the function Fee_Write shall reject the job request and return with E_NOT_OK.⌋

[SWS_Fee_00026]

Upstream requirements: SRS_MemHwAb_14010, RS_BRF_01048

⌈The FEE module shall execute the write operation asynchronously within the FEE module's main function.⌋

[SWS_Fee_00123]

Upstream requirements: SRS_BSW_00406

⌈If development error detection is enabled for the module: the function Fee_Write shall check if the module state is MEMIF_UNINIT. If this is the case, the function Fee_Write shall raise the development error FEE_E_UNINIT.⌋

[SWS_Fee_00144]

Upstream requirements: RS_BRF_01048

⌈The function Fee_Write shall check if the module state is MEMIF_BUSY. If this is the case, the function Fee_Write shall reject the write request, raise the runtime error FEE_E_BUSY and return with E_NOT_OK.⌋

[SWS_Fee_00138]

Upstream requirements: SRS_BSW_00323

⌈If development error detection is enabled for the module: the function Fee_Write shall check that the given block number is valid (i.e. it has been configured). If this is not the case, the function Fee_Write shall raise the development error FEE_E_-INVALID_BLOCK_NO.⌋

[SWS_Fee_00139]

Upstream requirements: SRS_BSW_00323

⌈If development error detection is enabled for the module: the function Fee_Write shall check that the given data pointer is valid (i.e. that it is not NULL). If this is not the case, the function Fee_Write shall raise the development error FEE_E_PARAM_-POINTER.⌋

[SWS_Fee_00163]

Upstream requirements: RS_BRF_01048

⌈If a write request is rejected by the function Fee_Write, i.e. requirements [SWS_Fee_00123], [SWS_Fee_00144], [SWS_Fee_00138] or [SWS_Fee_00139] apply, the function Fee_Write shall not change the current module status or job result.⌋

8.3.4 Fee_Cancel

[SWS_Fee_00089] Definition of API function Fee_Cancel

Upstream requirements: SRS_MemHwAb_14031

⌈

Service Name	Fee_Cancel

Syntax	`void Fee_Cancel (`
	`  void`
	`)`

Service ID [hex]	0x04
Sync/Async	Asynchronous
Reentrancy	Non Reentrant

▽

△

Parameters (in)	None
Parameters (inout)	None
Parameters (out)	None
Return value	None
Description	Service to call the cancel function of the underlying flash driver.
Available via	Fee.h

⌋

[SWS_Fee_00124]

Upstream requirements: SRS_BSW_00406

⌈If development error detection is enabled for the module: the function `Fee_Cancel` shall check if the module state is `MEMIF_UNINIT`. If this is the case the function `Fee_Cancel` shall raise the development error `FEE_E_UNINIT`.⌋

[SWS_Fee_00080]

Upstream requirements: SRS_MemHwAb_14031

⌈If the current module status is `MEMIF_BUSY` (i.e. the request to cancel a pending job is accepted by the function `Fee_Cancel`), the function `Fee_Cancel` shall call the cancel function of the underlying flash driver.⌋

[SWS_Fee_00081]

Upstream requirements: SRS_MemHwAb_14031

⌈If the current module status is `MEMIF_BUSY` (i.e. the request to cancel a pending job is accepted by the function `Fee_Cancel`), the function `Fee_Cancel` shall reset the FEE module's internal variables to make the module ready for a new job request from the upper layer, i.e. it shall set the module status to `MEMIF_IDLE`.⌋

[SWS_Fee_00164]

Upstream requirements: RS_BRF_01048

⌈If the current module status is not `MEMIF_BUSY` (i.e. the request to cancel a pending job is rejected by the function `Fee_Cancel`), the function `Fee_Cancel` shall not change the current module status or job result.⌋

[SWS_Fee_00184]

Upstream requirements: SRS_MemHwAb_14031

⌈If the current module status is not `MEMIF_BUSY` (i.e. there is no job to cancel and therefore the request to cancel a pending job is rejected by the function `Fee_Cancel`), the function `Fee_Cancel` shall raise the runtime error `FEE_E_INVALID_CANCEL`.⌋

8.3.5 Fee_GetStatus

[SWS_Fee_00090] Definition of API function Fee_GetStatus

Upstream requirements: SRS_BSW_00406

⌈

Service Name	Fee_GetStatus	
Syntax	`MemIf_StatusType Fee_GetStatus ( void )`	
Service ID [hex]	0x05	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	None	
Return value	MemIf_StatusType	`MEMIF_UNINIT`: The FEE module has not been initialized. `MEMIF_IDLE`: The FEE module is currently idle. `MEMIF_BUSY`: The FEE module is currently busy. `MEMIF_BUSY_INTERNAL`: The FEE module is busy with internal management operations.
Description	Service to return the status.	
Available via	Fee.h	

⌋

[SWS_Fee_00034]

Upstream requirements: SRS_BSW_00406

⌈The function `Fee_GetStatus` shall return `MEMIF_UNINIT` if the module has not (yet) been initialized.⌋

[SWS_Fee_00128]

Upstream requirements: RS_BRF_01048

⌈The function `Fee_GetStatus` shall return `MEMIF_IDLE` if the module is neither processing a request from the upper layer nor is it doing an internal management operation.⌋

[SWS_Fee_00129]

Upstream requirements: RS_BRF_01048

⌈The function `Fee_GetStatus` shall return `MEMIF_BUSY` if it is currently processing a request from the upper layer.⌋

[SWS_Fee_00074]

Upstream requirements: RS_BRF_01048

⌈The function Fee_GetStatus shall return MEMIF_BUSY_INTERNAL, if an internal management operation is currently ongoing.⌋

Note: Internal management operation may e.g. be a re-organization of the used flash memory (garbage collection). This may imply that the underlying device driver is - at least temporarily - busy.

8.3.6 Fee_GetJobResult

[SWS_Fee_00091] Definition of API function Fee_GetJobResult

Upstream requirements: RS_BRF_01048

⌈

Service Name	Fee_GetJobResult	

Syntax	`MemIf_JobResultType Fee_GetJobResult (`
	`  void`
	`)`	

Service ID [hex]	0x06	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	None	
Return value	MemIf_JobResultType	MEMIF_JOB_OK: The last job has been finished successfully. MEMIF_JOB_PENDING: The last job is waiting for execution or currently being executed. MEMIF_JOB_CANCELED: The last job has been canceled (which means it failed). MEMIF_JOB_FAILED: The last job has not been finished successfully (it failed). MEMIF_BLOCK_INCONSISTENT: The requested block is inconsistent, it may contain corrupted data. MEMIF_BLOCK_INVALID: The requested block has been invalidated, the requested read operation can not be performed.
Description	Service to query the result of the last accepted job issued by the upper layer software.	
Available via	Fee.h	

⌋

[SWS_Fee_00035]

Upstream requirements: RS_BRF_01048

⌈The function Fee_GetJobResult shall return MEMIF_JOB_OK if the last job has been finished successfully.⌋

[SWS_Fee_00156]

Upstream requirements: RS_BRF_01048

⌈The function `Fee_GetJobResult` shall return `MEMIF_JOB_PENDING` if the requested job is still waiting for execution or is currently being executed.⌋

[SWS_Fee_00157]

Upstream requirements: SRS_MemHwAb_14031

⌈The function `Fee_GetJobResult` shall return `MEMIF_JOB_CANCELED` if the last job has been canceled by the upper layer.⌋

[SWS_Fee_00158]

Upstream requirements: RS_BRF_01048

⌈The function `Fee_GetJobResult` shall return `MEMIF_JOB_FAILED` if the last job has failed.⌋

[SWS_Fee_00159]

Upstream requirements: SRS_MemHwAb_14014

⌈The function `Fee_GetJobResult` shall return `MEMIF_BLOCK_INCONSISTENT` if the requested block is found to be inconsistent.⌋

The management of block inconsistency is specified in chapter 7.1.5.

[SWS_Fee_00160]

Upstream requirements: SRS_MemHwAb_14028

⌈The function `Fee_GetJobResult` shall return `MEMIF_BLOCK_INVALID` if the requested block has been invalidated by the upper layer.⌋

[SWS_Fee_00155]

Upstream requirements: RS_BRF_01048

⌈Only those jobs which have been requested directly by the upper layer shall have influence on the job result returned by the function `Fee_GetJobResult`. I.e. jobs which are issued by the FEE module itself in the course of internal management operations shall not alter the job result.⌋

[SWS_Fee_00125]

Upstream requirements: SRS_BSW_00406

⌈If development error detection is enabled for the module: the function `Fee_GetJobResult` shall check if the module state is `MEMIF_UNINIT`. If this is the case, the function `Fee_GetJobResult` shall raise the development error `FEE_E_UNINIT`.⌋

8.3.7 Fee_InvalidateBlock

[SWS_Fee_00092] Definition of API function Fee_InvalidateBlock

Upstream requirements: SRS_MemHwAb_14028

⌈

Service Name	Fee_InvalidateBlock	
Syntax	`Std_ReturnType Fee_InvalidateBlock ( uint16 BlockNumber )`	
Service ID [hex]	0x07	
Sync/Async	Asynchronous	
Reentrancy	Non Reentrant	
Parameters (in)	BlockNumber	Number of logical block, also denoting start address of that block in flash memory.
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.
Description	Service to invalidate a logical block.	
Available via	Fee.h	

⌋

[SWS_Fee_00036]

Upstream requirements: SRS_MemHwAb_14009

⌈The function `Fee_InvalidateBlock` shall take the block number and calculate the corresponding memory block address.⌋

[SWS_Fee_00037]

Upstream requirements: SRS_MemHwAb_14028

⌈The function `Fee_InvalidateBlock` shall invalidate the requested block `BlockNumber` by calling the erase function of the underlying device driver and / or by changing some module internal management information accordingly.⌋

Note: How exactly the requested block is invalidated depends on the module's implementation and will not be further detailed in this specification. The internal management information has to be stored in NV memory since it has to be resistant against resets. What this information is and how it is stored will not be further detailed in this specification.

[SWS_Fee_00126]

Upstream requirements: SRS_BSW_00406

⌈If development error detection is enabled for the module: the function `Fee_InvalidateBlock` shall check if the module status is `MEMIF_UNINIT`. If this is the case, the function `Fee_InvalidateBlock` shall raise the development error `FEE_E_UNINIT`.⌋

[SWS_Fee_00145]

Upstream requirements: RS_BRF_01048

⌈The function `Fee_InvalidateBlock` shall check if the module status is `MEMIF_BUSY`. If this is the case, the function `Fee_InvalidateBlock` shall reject the request, raise the runtime error `FEE_E_BUSY` and return with `E_NOT_OK`.⌋

[SWS_Fee_00192]

Upstream requirements: SRS_MemHwAb_14028

⌈The function `Fee_InvalidateBlock` shall check if the module state is `MEMIF_IDLE` or `MEMIF_BUSY_INTERNAL`. If this is the case the module shall accept the invalidation request and shall return `E_OK` to the caller. The block invalidation shall be executed asynchronously in the module's main function as soon as the module has finished the internal management operation.⌋

[SWS_Fee_00193]

Upstream requirements: RS_BRF_01812

⌈The FEE module shall execute the block invalidation request asynchronously within the FEE module's main function.⌋

[SWS_Fee_00140]

Upstream requirements: SRS_BSW_00323

⌈If development error detection is enabled for the module: the function `Fee_InvalidateBlock` shall check that the given block number is valid (i.e. it has been configured). If this is not the case, the function `Fee_InvalidateBlock` shall raise the development error `FEE_E_INVALID_BLOCK_NO`.⌋

[SWS_Fee_00165]

Upstream requirements: SRS_MemHwAb_14028

⌈If an invalidation request is rejected by the function `Fee_InvalidateBlock`, i.e. requirements [SWS_Fee_00126], [SWS_Fee_00140] or [SWS_Fee_00145] apply, the function `Fee_InvalidateBlock` shall not change the current module status or job result.⌋

8.3.8 Fee_GetVersionInfo

[SWS_Fee_00093] Definition of API function Fee_GetVersionInfo

Upstream requirements: SRS_BSW_00407

⌈

Service Name	Fee_GetVersionInfo	
Syntax	`void Fee_GetVersionInfo ( Std_VersionInfoType* VersionInfoPtr )`	
Service ID [hex]	0x08	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	VersionInfoPtr	Pointer to standard version information structure.
Return value	None	
Description	Service to return the version information of the FEE module.	
Available via	Fee.h	

⌋

[SWS_Fee_00147]

Upstream requirements: SRS_BSW_00323

⌈If development error detection is enabled for the module: the function `Fee_GetVersionInfo` shall check that the given data pointer is valid (i.e. that it is not NULL). If this is not the case, the function `Fee_GetVersionInfo` shall raise the development error `FEE_E_PARAM_POINTER`.⌋

8.3.9 Fee_EraseImmediateBlock

[SWS_Fee_00094] Definition of API function Fee_EraseImmediateBlock

Upstream requirements: SRS_MemHwAb_14032

⌈

Service Name	Fee_EraseImmediateBlock
Syntax	`Std_ReturnType Fee_EraseImmediateBlock ( uint16 BlockNumber )`
Service ID [hex]	0x09
Sync/Async	Asynchronous
Reentrancy	Non Reentrant

▽

△

Parameters (in)	BlockNumber	Number of logical block, also denoting start address of that block in EEPROM.
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.
Description	Service to erase a logical block.	
Available via	Fee.h	

⌋

Note: The function Fee_EraseImmediateBlock shall only be called by e.g. diagnostic or similar system service to pre-erase the area for immediate data if necessary.

[SWS_Fee_00066]

Upstream requirements: SRS_MemHwAb_14009

⌈The function Fee_EraseImmediateBlock shall take the block number and calculate the corresponding memory block address.⌋

[SWS_Fee_00067]

Upstream requirements: SRS_MemHwAb_14013

⌈The function Fee_EraseImmediateBlock shall ensure that the FEE module can write immediate data. Whether this involves physically erasing a memory area and therefore calling the erase function of the underlying driver depends on the implementation of the module.⌋

[SWS_Fee_00127]

Upstream requirements: SRS_BSW_00406

⌈If development error detection is enabled for the module: the function Fee_EraseImmediateBlock shall check if the module state is MEMIF_UNINIT. If this is the case, the function Fee_EraseImmediateBlock shall raise the development error FEE_E_UNINIT.⌋

[SWS_Fee_00146]

Upstream requirements: RS_BRF_01048

⌈The function Fee_EraseImmediateBlock shall check if the module state is MEMIF_BUSY. If this is the case, the function Fee_EraseImmediateBlock shall reject the erase request, raise the runtime error FEE_E_BUSY and return with E_NOT_OK.⌋

[SWS_Fee_00068]

Upstream requirements: SRS_BSW_00323

⌈If development error detection is enabled for the module: the function Fee_EraseImmediateBlock shall check whether the addressed logical block is configured as containing immediate data (FeeImmediateData == TRUE). If not, the function Fee_EraseImmediateBlock shall raise the development error FEE_E_INVALID_BLOCK_NO.⌋

[SWS_Fee_00141]

Upstream requirements: SRS_BSW_00323

⌈If development error detection is enabled for the module: the function Fee_EraseImmediateBlock shall check that the given block number is valid (i.e. it has been configured). If this is not the case, the function Fee_EraseImmediateBlock shall raise the development error FEE_E_INVALID_BLOCK_NO.⌋

[SWS_Fee_00166]

Upstream requirements: SRS_MemHwAb_14032

⌈If a erase request is rejected by the function Fee_EraseImmediateBlock, i.e. requirements [SWS_Fee_00068], [SWS_Fee_00127], [SWS_Fee_00141] or [SWS_Fee_00146] apply, the function Fee_EraseImmediateBlock shall not change the current module status or job result.⌋

8.4 Callback notifications

This chapter lists all functions provided by the Fee module to lower layer modules.

Note: Depending on the implementation of the modules making up the NV memory stack, callback routines provided by the FEE module may be called on interrupt level. The implementation of the FEE module therefore has to make sure that the runtime of those routines is reasonably short, i.e. since callbacks may be propagated upward through several software layers. Whether callback routines are allowable / feasible on interrupt level depends on the project specific needs (reaction time) and limitations (runtime in interrupt context). Therefore, system design has to make sure that the configuration of the involved modules meets those requirements.

8.4.1 Fee_JobEndNotification

[SWS_Fee_00095] Definition of callback function Fee_JobEndNotification

Upstream requirements: RS_BRF_01064

⌈

Service Name	Fee_JobEndNotification
Syntax	`void Fee_JobEndNotification ( void )`
Service ID [hex]	0x10
Sync/Async	Synchronous
Reentrancy	Non Reentrant
Parameters (in)	None
Parameters (inout)	None
Parameters (out)	None
Return value	None
Description	Service to report to this module the successful end of an asynchronous operation.
Available via	Fee.h

⌋

The underlying flash driver shall call the function `Fee_JobEndNotification` to report the successful end of an asynchronous operation.

[SWS_Fee_00052]

Upstream requirements: RS_BRF_01064

⌈The function `Fee_JobEndNotification` shall perform any necessary block management operations and subsequently call the job end notification routine of the upper layer module if configured.⌋

[SWS_Fee_00142]

Upstream requirements: RS_BRF_01064

⌈If the job result is currently `MEMIF_JOB_PENDING`, the function `Fee_JobEndNotification` shall set the job result to `MEMIF_JOB_OK`, else it shall leave the job result untouched.⌋

[SWS_Fee_00194]

Upstream requirements: RS_BRF_01064

⌈The function `Fee_JobEndNotification` shall perform any necessary block management and error handling operations and subsequently call the job error notification routine of the upper layer module if configured.⌋

Note: The function Fee_JobEndNotification shall be callable on interrupt level.

8.5 Scheduled functions

These functions are directly called by the Basic Software Scheduler. The following functions shall have no return value and no parameter. All functions shall be non re-entrant.

8.5.1 Fee_MainFunction

[SWS_Fee_00097] Definition of scheduled function Fee_MainFunction

Upstream requirements: RS_BRF_01048

⌈

Service Name	Fee_MainFunction

Syntax	`void Fee_MainFunction (`
	`  void`
	`)`

Service ID [hex]	0x12
Description	Service to handle the requested read / write / erase jobs and the internal management operations.
Available via	SchM_Fee.h

⌋

[SWS_Fee_00169]

Upstream requirements: SRS_BSW_00101

⌈If the module initialization (started in the function Fee_Init) is completed in the module's main function, the function Fee_MainFunction shall set the module status from MEMIF_UNINIT to MEMIF_IDLE once initialization of the module has been successfully finished.⌋

[SWS_Fee_00057]

Upstream requirements: RS_BRF_01048

⌈The function Fee_MainFunction shall asynchronously handle the read / write / erase / invalidate jobs requested by the upper layer and internal management operations.⌋

[SWS_Fee_00075]

Upstream requirements: RS_BRF_01048, SRS_MemHwAb_14028

⌈The function `Fee_MainFunction` shall check, whether the block requested for reading has been invalidated by the upper layer module. If so, the function `Fee_MainFunction` shall set the job result to `MEMIF_BLOCK_INVALID` and call the error notification routine of the upper layer if configured.⌋

[SWS_Fee_00023]

Upstream requirements: SRS_MemHwAb_14014, SRS_MemHwAb_14015, SRS_MemHwAb_14016

⌈The function `Fee_MainFunction` shall check the consistency of the logical block being read before notifying the caller. If an inconsistency of the read data is detected or if the requested block can't be found, the function `Fee_MainFunction` shall set the job result to `MEMIF_BLOCK_INCONSISTENT` and call the error notification routine of the upper layer if configured.⌋

Note: In this case, the upper layer must not use the contents of the data buffer.

8.6 Expected interfaces

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

8.6.1 Mandatory Interfaces

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

[SWS_Fee_00105] Definition of mandatory interfaces required by module Fee

Upstream requirements: SRS_BSW_00384

⌈

API Function	***Header File***	***Description***
Det_ReportRuntimeError	Det.h	Service to report runtime errors. If a callout has been configured then this callout shall be called.
MemAcc_Cancel	MemAcc.h	Triggers a cancel operation of the pending job for the address area referenced by the addressAreaId. Cancelling affects only jobs in pending state. For any other states, the request will be ignored.

▽

△

API Function	***Header File***	***Description***
MemAcc_Compare	MemAcc.h	Triggers a job to compare the passed data to the memory content of the provided address area. The job terminates, if all bytes matched or a difference was detected. The result of this service can be retrieved using the MemAcc_GetJobResult() API. If the compare operation determined a mismatch, the result code is MEMACC_INCONSISTENT.
MemAcc_Erase	MemAcc.h	Triggers an erase job of the given area. Triggers an erase job of the given area 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.
MemAcc_GetJobResult	MemAcc.h	Returns the consolidated job result of the address area referenced by addressAreaId.
MemAcc_Read	MemAcc.h	Triggers a read job to copy data from the source address into the referenced destination data buffer. The result of this service can be retrieved using the MemAcc_GetJobResult API. If the read operation was successful, the result of the job is MEMACC_OK. If the read operation failed, the result of the job is either MEMACC_FAILED in case of a general error or MEMACC_ECC_CORRECTED/MEMACC_ECC_UNCORRECTED in case of a correctable/uncorrectable ECC error.
MemAcc_Write	MemAcc.h	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 MemAcc_GetJobResult API. If the write operation was successful, the job result is MEMACC_OK. If there was an issue writing the data, the result is MEMACC_FAILED.

⌋

8.6.2 Optional Interfaces

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

[SWS_Fee_00104] Definition of optional interfaces requested by module Fee

Upstream requirements: SRS_BSW_00384

⌈

API Function	***Header File***	***Description***
Det_ReportError	Det.h	Service to report development errors.
MemAcc_BlankCheck	MemAcc.h	Checks if the passed address space is blank, i.e. erased and writeable. The result of this service can be retrieved using the MemAcc_GetJobResult API. If the address area defined by targetAddress and length is blank, the result is MEMACC_OK, otherwise the result is MEMACC_INCONSISTENT.

⌋

8.6.3 Configurable interfaces

In this chapter all interfaces are listed where the target function could be configured. The target function is usually a callback function. The names of this kind of interfaces are not fixed because they are configurable.

Note: Depending on the implementation of the modules making up the NV memory stack, callback routines invoked by the FEE module may be called on interrupt level. The implementor of the module providing these routines therefore has to make sure that their runtime is reasonably short, i.e. since callbacks may be propagated upward through several software layers. Whether callback routines are allowable / feasible on interrupt level depends on the project specific needs (reaction time) and limitations (runtime in interrupt context). Therefore system design has to make sure that the configuration of the involved modules meets those requirements.

[SWS_Fee_00055]

Upstream requirements: RS_BRF_01064

⌈The FEE module shall call the function defined in the configuration parameter `FeeNvmJobEndNotification` upon successful end of an asynchronous operation and after performing all necessary internal management operations:

- Read job finished & OK
- Write job finished & OK & block marked as valid
- Erase job for immediate data finished & OK (see [SWS_Fee_00067])
- Invalidation of memory block finished & OK

⌋

The function defined in the configuration parameter `FeeNvmJobEndNotification` shall be callable on interrupt level.

[SWS_Fee_00056]

Upstream requirements: RS_BRF_01064

⌈The FEE module shall call the function defined in the configuration parameter `FeeNvmJobEndNotification` upon failure of an asynchronous operation and after performing all necessary internal management and error handling operations:

- Read job finished & failed (e.g. block invalid or inconsistent)
- Write job finished & failed & block marked as invalid
- Erase job for immediate data finished & failed (see [SWS_Fee_00067])
- Invalidation of memory block finished & failed

⌋

The function defined in the configuration parameter `FeeNvmJobEndNotification` shall be callable on interrupt level.

9 Sequence diagrams

Note: For a vendor specific library, the following sequence diagrams are valid only insofar as they show the relation to the calling modules (Ecu_StateManager[3] and memory abstraction interface[4]). The calling relations from a memory abstraction module to an underlying driver are not relevant / binding for a vendor specific library.

9.1 Fee_Init

The following figure shows the call sequence for the Fee_Init routine. It is different from that of all other services of this module as it is not called by the NVRAM manager and not called via the memory abstraction interface.

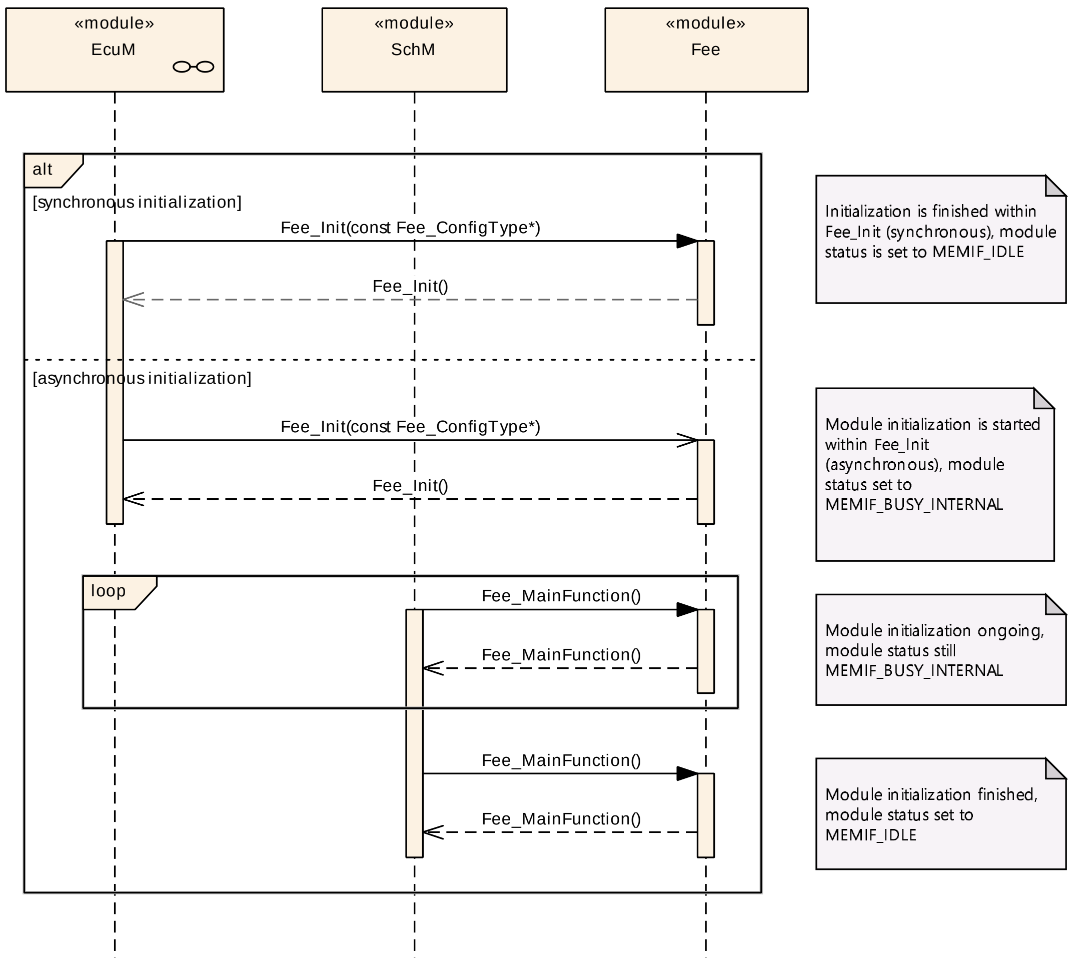
«module» EcuM
«module» SchM
«module» Fee
alt
[synchronous initialization]
Fee_Init(const Fee_ConfigType*)
Fee_Init()
[asynchronous initialization]
Fee_Init(const Fee_ConfigType*)
Fee_Init()
loop
Fee_MainFunction()
Fee_MainFunction()
Fee_MainFunction()
Fee_MainFunction()
Initialization is finished within Fee_Init (synchronous), module status is set to MEMIF_IDLE
Module initialization is started within Fee_Init (asynchronous), module status set to MEMIF_BUSY_INTERNAL
Module initialization ongoing, module status still MEMIF_BUSY_INTERNAL
Module initialization finished, module status set to MEMIF_IDLE

Figure 9.1: Sequence diagram of Fee_Init

9.2 Fee_Write

The following figure shows exemplarily the call sequence for the Fee_Write service. This sequence diagram also applies to the other asynchronous services of this module.

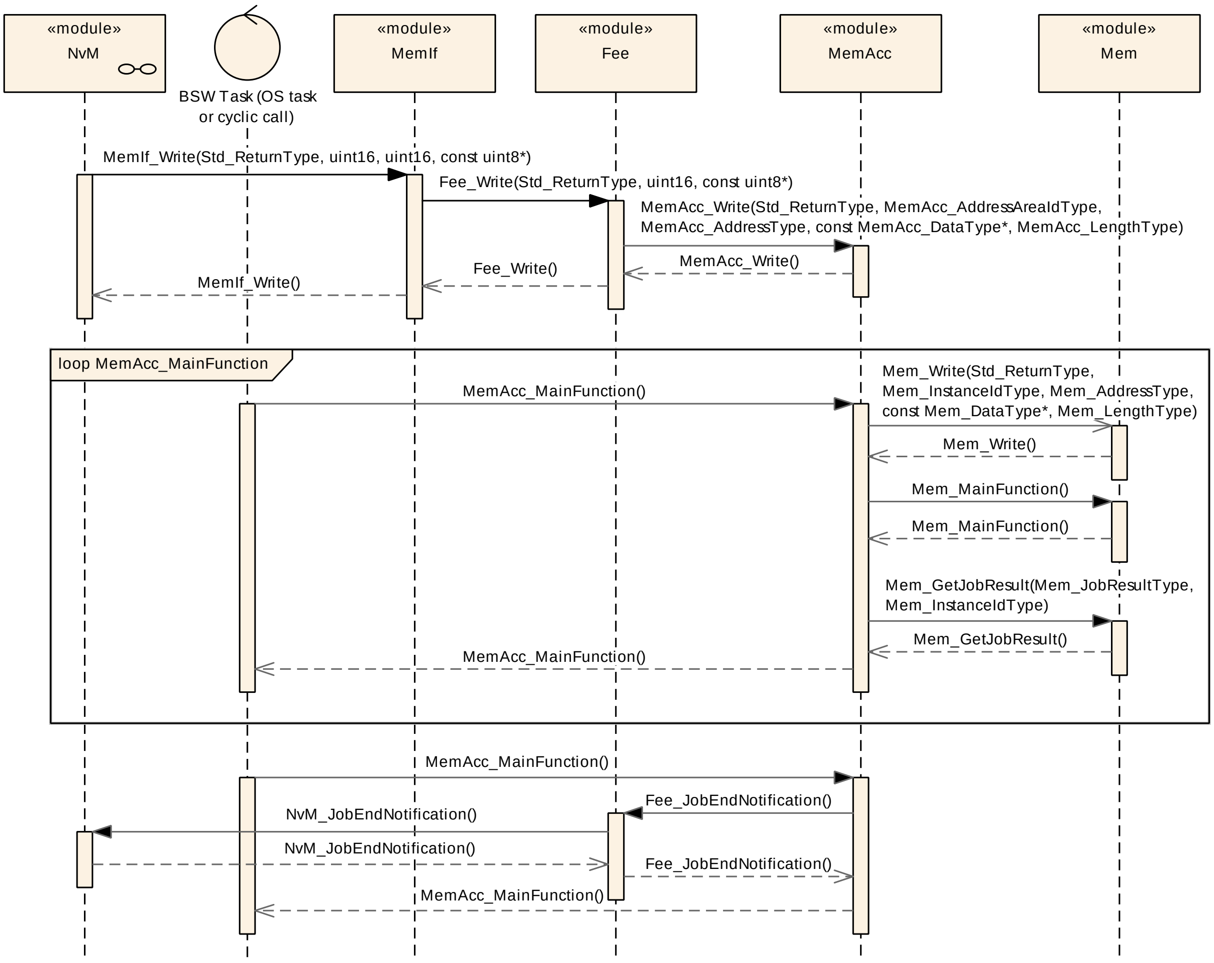
«module» NvM
BSW Task (OS task or cyclic call)
«module» MemIf
«module» Fee
«module» MemAcc
«module» Mem
MemIf_Write(Std_ReturnType, uint16, uint16, const uint8*)
Fee_Write(Std_ReturnType, uint16, const uint8*)
MemAcc_Write(Std_ReturnType, MemAcc_AddressAreaIdType, MemAcc_AddressType, const MemAcc_DataType*, MemAcc_LengthType)
MemAcc_Write()
Fee_Write()
MemIf_Write()
loop MemAcc_MainFunction
MemAcc_MainFunction()
Mem_Write(Std_ReturnType, Mem_InstanceIdType, Mem_AddressType, const Mem_DataType*, Mem_LengthType)
Mem_Write()
Mem_MainFunction()
Mem_MainFunction()
Mem_GetJobResult(Mem_JobResultType, Mem_InstanceIdType)
Mem_GetJobResult()
MemAcc_MainFunction()
MemAcc_MainFunction()
Fee_JobEndNotification()
NvM_JobEndNotification()
NvM_JobEndNotification()
Fee_JobEndNotification()
MemAcc_MainFunction()

Figure 9.2: Sequence diagram of Fee_Write

9.3 Fee_Cancel

The following figure shows as an example the call sequence for a canceled Fee_-Write service and a subsequent new Fee_Write request. This sequence diagram shows that Fee_Cancel is asynchronous w.r.t. the underlying hardware while itself being synchronous.

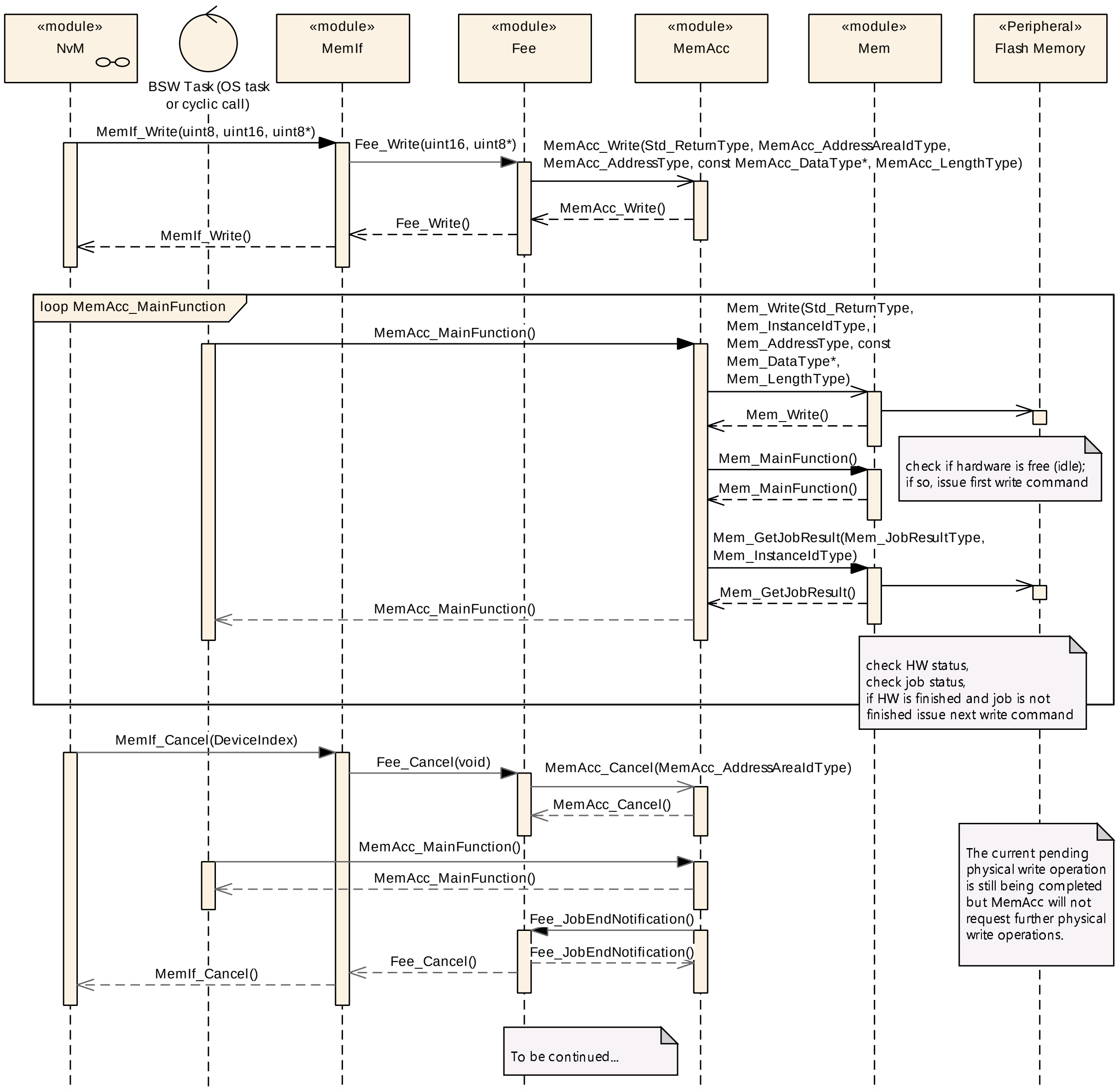
«module» NvM
BSW Task (OS task or cyclic call)
«module» MemIf
«module» Fee
«module» MemAcc
«module» Mem
«Peripheral» Flash Memory
MemIf_Write(uint8, uint16, uint8*)
Fee_Write(uint16, uint8*)
MemAcc_Write(Std_ReturnType, MemAcc_AddressAreaIdType, MemAcc_AddressType, const MemAcc_DataType*, MemAcc_LengthType)
MemAcc_Write()
Fee_Write()
MemIf_Write()
loop MemAcc_MainFunction
MemAcc_MainFunction()
Mem_Write(Std_ReturnType, Mem_InstanceIdType, Mem_AddressType, const Mem_DataType*, Mem_LengthType)
Mem_Write()
Mem_MainFunction()
Mem_MainFunction()
check if hardware is free (idle); if so, issue first write command
Mem_GetJobResult(Mem_JobResultType, Mem_InstanceIdType)
Mem_GetJobResult()
MemAcc_MainFunction()
check HW status, check job status, if HW is finished and job is not finished issue next write command
MemIf_Cancel(DeviceIndex)
Fee_Cancel(void)
MemAcc_Cancel(MemAcc_AddressAreaIdType)
MemAcc_Cancel()
MemAcc_MainFunction()
MemAcc_MainFunction()
The current pending physical write operation is still being completed but MemAcc will not request further physical write operations.
Fee_JobEndNotification()
Fee_JobEndNotification()
Fee_Cancel()
MemIf_Cancel()
To be continued...

Figure 9.3: Part 1 of sequence diagram of Fee_Cancel

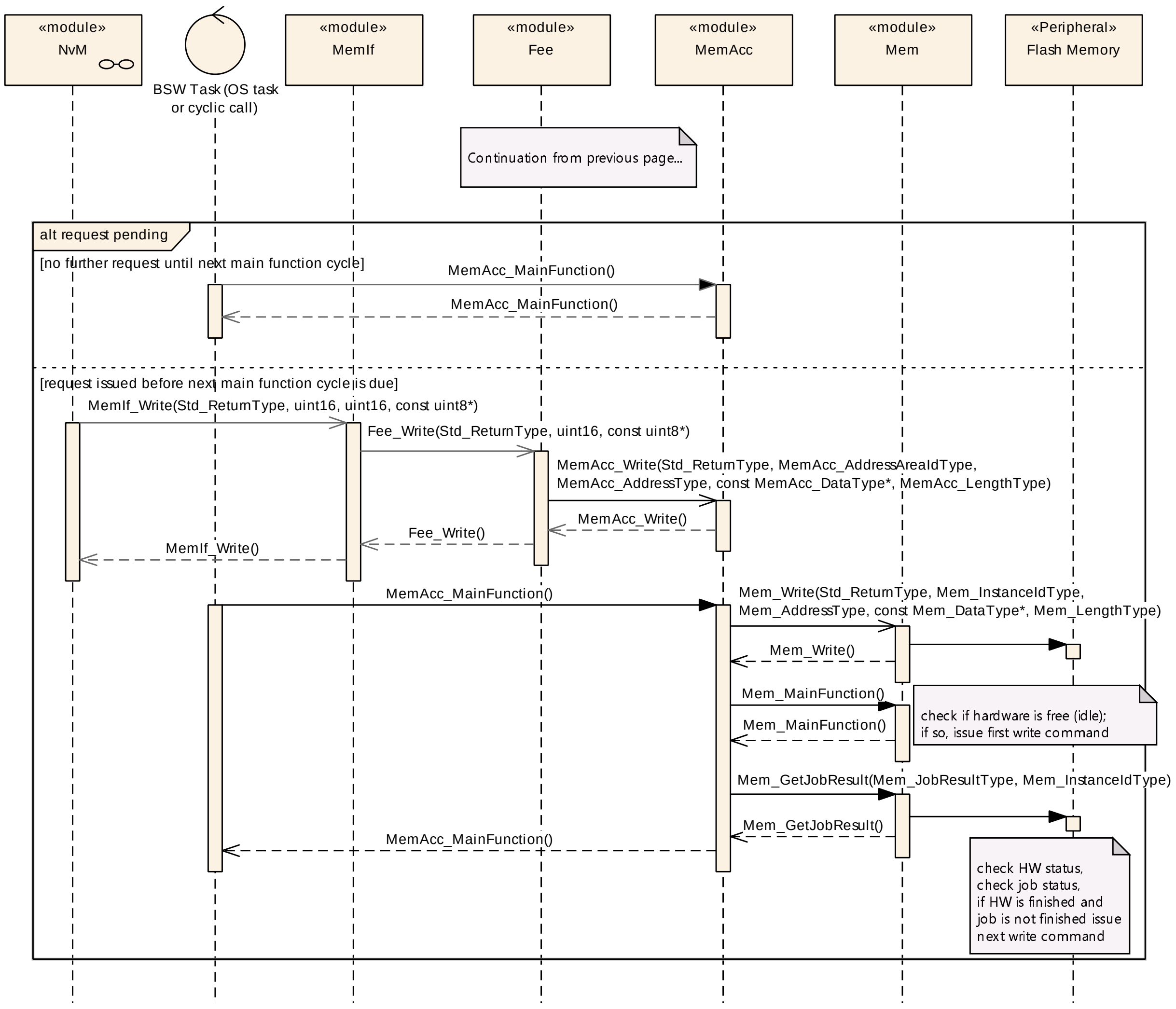
«module»
NvM
BSW Task (OS task or cyclic call)
«module»
MemIf
«module»
Fee
«module»
MemAcc
«module»
Mem
«Peripheral»
Flash Memory
Continuation from previous page...
alt request pending
[no further request until next main function cycle]
MemAcc_MainFunction()
MemAcc_MainFunction()
[request issued before next main function cycle is due]
MemIf_Write(Std_ReturnType, uint16, uint16, const uint8*)
Fee_Write(Std_ReturnType, uint16, const uint8*)
MemAcc_Write(Std_ReturnType, MemAcc_AddressAreaIdType, MemAcc_AddressType, const MemAcc_DataType*, MemAcc_LengthType)
MemAcc_Write()
Fee_Write()
MemIf_Write()
MemAcc_MainFunction()
Mem_Write(Std_ReturnType, Mem_InstanceIdType, Mem_AddressType, const Mem_DataType*, Mem_LengthType)
Mem_Write()
Mem_MainFunction()
Mem_MainFunction()
check if hardware is free (idle); if so, issue first write command
Mem_GetJobResult(Mem_JobResultType, Mem_InstanceIdType)
Mem_GetJobResult()
MemAcc_MainFunction()
check HW status, check job status, if HW is finished and job is not finished issue next write command

Figure 9.4: Part 2 of sequence diagram of Fee_Cancel

10 Configuration specification

10.1 Containers and configuration parameters

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

10.1.1 Fee

[ECUC_Fee_00154] Definition of EcucModuleDef Fee ⌈

Module Name	Fee
Description	Configuration of the Fee (Flash EEPROM Emulation) module.
Post-Build Variant Support	false
Supported Config Variants	VARIANT-PRE-COMPILE

Included Containers		
Container Name	**Multiplicity**	**Scope / Dependency**
FeeBlockConfiguration	1..*	Configuration of block specific parameters for the Flash EEPROM Emulation module.
FeeGeneral	1	Container for general parameters. These parameters are not specific to a block.
FeePublishedInformation	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.

⌋

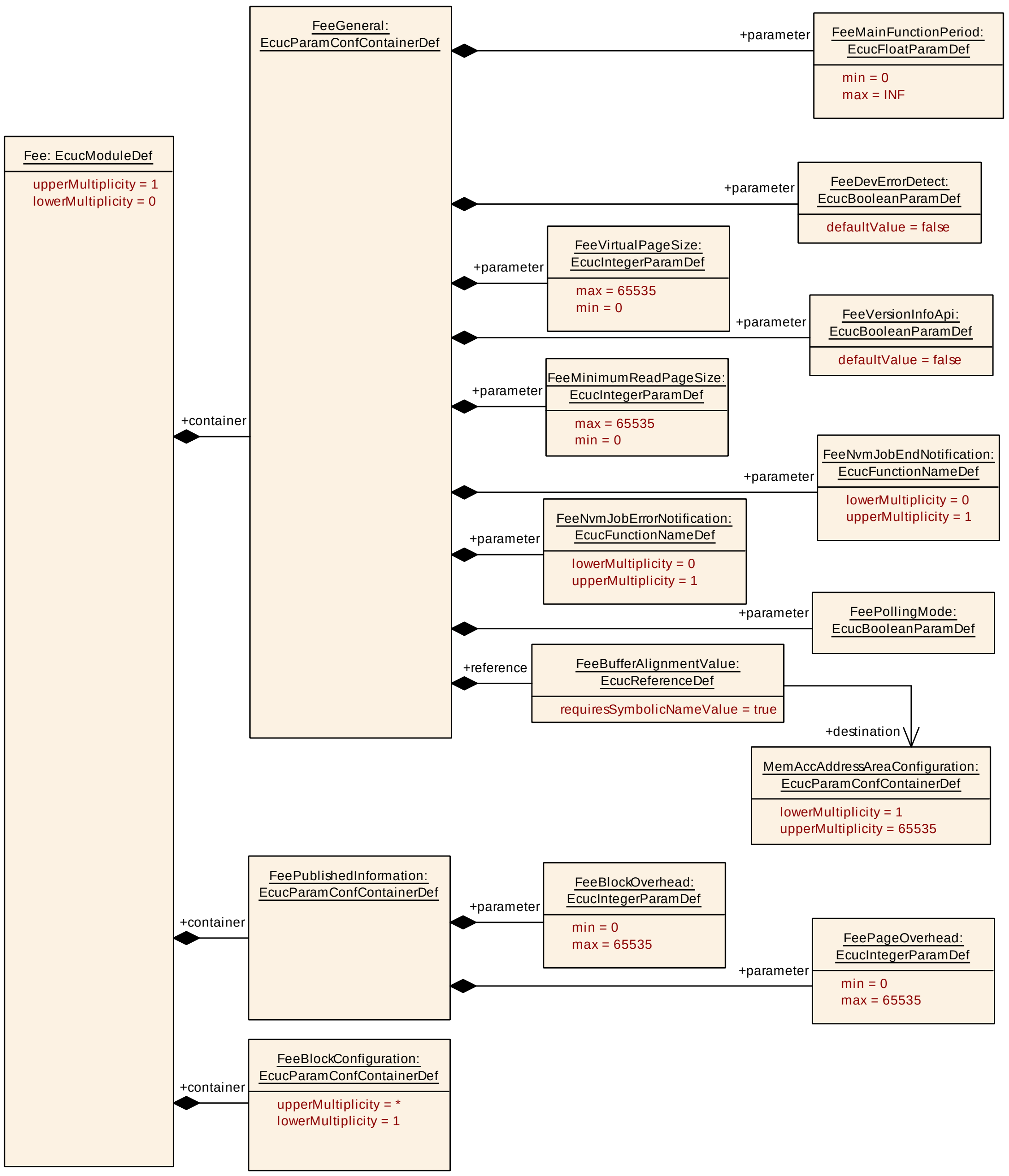
Fee: EcucModuleDef
upperMultiplicity = 1
lowerMultiplicity = 0
+container
FeeGeneral: EcucParamConfContainerDef
+parameter
FeeMainFunctionPeriod: EcucFloatParamDef
min = 0
max = INF
FeeDevErrorDetect: EcucBooleanParamDef
defaultValue = false
FeeVirtualPageSize: EcucIntegerParamDef
max = 65535
min = 0
FeeVersionInfoApi: EcucBooleanParamDef
defaultValue = false
FeeMinimumReadPageSize: EcucIntegerParamDef
max = 65535
min = 0
FeeNvmJobEndNotification: EcucFunctionNameDef
lowerMultiplicity = 0
upperMultiplicity = 1
FeeNvmJobErrorNotification: EcucFunctionNameDef
lowerMultiplicity = 0
upperMultiplicity = 1
FeePollingMode: EcucBooleanParamDef
+reference
FeeBufferAlignmentValue: EcucReferenceDef
requiresSymbolicNameValue = true
+destination
MemAccAddressAreaConfiguration: EcucParamConfContainerDef
lowerMultiplicity = 1
upperMultiplicity = 65535
FeePublishedInformation: EcucParamConfContainerDef
FeeBlockOverhead: EcucIntegerParamDef
min = 0
max = 65535
FeePageOverhead: EcucIntegerParamDef
min = 0
max = 65535
FeeBlockConfiguration: EcucParamConfContainerDef
upperMultiplicity = *
lowerMultiplicity = 1

Figure 10.1: Overview of configuration parameters of Fee

10.1.2 FeeGeneral

[ECUC_Fee_00039] Definition of EcucParamConfContainerDef FeeGeneral ⌈

Container Name	FeeGeneral
Parent Container	Fee
Description	Container for general parameters. These parameters are not specific to a block.
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
FeeDevErrorDetect	1	[ECUC_Fee_00111]
FeeMainFunctionPeriod	1	[ECUC_Fee_00153]
FeeMinimumReadPageSize	1	[ECUC_Fee_00156]
FeeNvmJobEndNotification	0..1	[ECUC_Fee_00112]
FeeNvmJobErrorNotification	0..1	[ECUC_Fee_00113]
FeePollingMode	1	[ECUC_Fee_00114]
FeeVersionInfoApi	1	[ECUC_Fee_00115]
FeeVirtualPageSize	1	[ECUC_Fee_00116]
FeeBufferAlignmentValue	1	[ECUC_Fee_00157]

No Included Containers

⌋

[ECUC_Fee_00111] Definition of EcucBooleanParamDef FeeDevErrorDetect ⌈

Parameter Name	FeeDevErrorDetect		
Parent Container	FeeGeneral		
Description	Switches the development error detection and notification on or off. • true: detection and notification is enabled. • false: detection and notification is disabled.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

⌋

[ECUC_Fee_00153] Definition of EcucFloatParamDef FeeMainFunctionPeriod ⌈

Parameter Name	FeeMainFunctionPeriod		
Parent Container	FeeGeneral		
Description	The period between successive calls to the main function in seconds.		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	]0 .. INF[		
Default value	–		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	

▽

△

Scope / Dependency	scope: ECU		

⌋

[ECUC_Fee_00156] Definition of EcucIntegerParamDef FeeMinimumReadPageSize

Status: DRAFT

⌈

Parameter Name	FeeMinimumReadPageSize		
Parent Container	FeeGeneral		
Description	Minimum Page size will be a multiple of the minimum page size. Fee shall align read requests to this size. **Tags:** atp.Status=draft		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	–		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

⌋

[ECUC_Fee_00112] Definition of EcucFunctionNameDef FeeNvmJobEndNotification ⌈

Parameter Name	FeeNvmJobEndNotification		
Parent Container	FeeGeneral		
Description	Mapped to the job end notification routine provided by the upper layer module (NvM_JobEndNotification).		
Multiplicity	0..1		
Type	EcucFunctionNameDef		
Default value	–		
Regular Expression	–		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	

▽

△

Scope / Dependency	scope: local

⌋

[ECUC_Fee_00113] Definition of EcucFunctionNameDef FeeNvmJobErrorNotification ⌈

Parameter Name	FeeNvmJobErrorNotification		
Parent Container	FeeGeneral		
Description	Mapped to the job error notification routine provided by the upper layer module (NvM_JobErrorNotification).		
Multiplicity	0..1		
Type	EcucFunctionNameDef		
Default value	–		
Regular Expression	–		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

⌋

[ECUC_Fee_00114] Definition of EcucBooleanParamDef FeePollingMode ⌈

Parameter Name	FeePollingMode		
Parent Container	FeeGeneral		
Description	Pre-processor switch to enable and disable the polling mode for this module. true: Polling mode enabled, callback functions (provided to MemAcc module) disabled. false: Polling mode disabled, callback functions (provided to MemAcc module) enabled.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	–		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

⌋

[ECUC_Fee_00115] Definition of EcucBooleanParamDef FeeVersionInfoApi ⌈

Parameter Name	FeeVersionInfoApi		
Parent Container	FeeGeneral		
Description	Pre-processor switch to enable / disable the API to read out the modules version information. true: Version info API enabled. false: Version info API disabled.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

⌋

[ECUC_Fee_00116] Definition of EcucIntegerParamDef FeeVirtualPageSize ⌈

Parameter Name	FeeVirtualPageSize		
Parent Container	FeeGeneral		
Description	The size in bytes to which logical blocks shall be aligned.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	–		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

⌋

[ECUC_Fee_00157] Definition of EcucReferenceDef FeeBufferAlignmentValue

Status: DRAFT

⌈

Parameter Name	FeeBufferAlignmentValue
Parent Container	FeeGeneral
Description	Parameter determines the alignment of the start address that Fee buffers need to have. Value shall be inherited from MemAccBufferAlignmentValue. **Tags:** atp.Status=draft
Multiplicity	1
Type	Symbolic name reference to MemAccAddressAreaConfiguration

▽

△

Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

⌋

10.1.3 FeeBlockConfiguration

[ECUC_Fee_00040] Definition of EcucParamConfContainerDef FeeBlockConfiguration ⌈

Container Name	FeeBlockConfiguration
Parent Container	Fee
Description	Configuration of block specific parameters for the Flash EEPROM Emulation module.
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
FeeBlockNumber	1	[ECUC_Fee_00150]
FeeBlockSize	1	[ECUC_Fee_00148]
FeeImmediateData	1	[ECUC_Fee_00151]
FeeNumberOfWriteCycles	1	[ECUC_Fee_00110]
FeeMemAccAddressArea	0..1	[ECUC_Fee_00155]

No Included Containers

⌋

[ECUC_Fee_00150] Definition of EcucIntegerParamDef FeeBlockNumber ⌈

Parameter Name	FeeBlockNumber	
Parent Container	FeeBlockConfiguration	
Description	Block identifier (handle). 0x0000 and 0xFFFF shall not be used for block numbers (see FEE006). Range: min = 2ˆNVM_DATASET_SELECTION_BITS max = 0xFFFF -2ˆNVM_DATASET_SELECTION_BITS Note: Depending on the number of bits set aside for dataset selection several other block numbers shall also be left out to ease implementation.	
Multiplicity	1	
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)	
Range	1 .. 65534	

▽

△

Default value	–		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: ECU		

⌋

[ECUC_Fee_00148] Definition of EcucIntegerParamDef FeeBlockSize ⌈

Parameter Name	FeeBlockSize		
Parent Container	FeeBlockConfiguration		
Description	Size of a logical block in bytes.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 65535		
Default value	–		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: ECU		

⌋

[ECUC_Fee_00151] Definition of EcucBooleanParamDef FeeImmediateData ⌈

Parameter Name	FeeImmediateData		
Parent Container	FeeBlockConfiguration		
Description	Marker for high priority data. true: Block contains immediate data. false: Block does not contain immediate data.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	–		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: ECU		

⌋

[ECUC_Fee_00110] Definition of EcucIntegerParamDef FeeNumberOfWriteCycles ⌈

Parameter Name	FeeNumberOfWriteCycles		
Parent Container	FeeBlockConfiguration		
Description	Number of write cycles required for this block.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	–		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

⌋

[ECUC_Fee_00155] Definition of EcucReferenceDef FeeMemAccAddressArea

Status: DRAFT

⌈

Parameter Name	FeeMemAccAddressArea		
Parent Container	FeeBlockConfiguration		
Description	Reference to the MemAccAddressAreaConfiguration. **Tags:** atp.Status=draft		
Multiplicity	0..1		
Type	Symbolic name reference to MemAccAddressAreaConfiguration		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

⌋

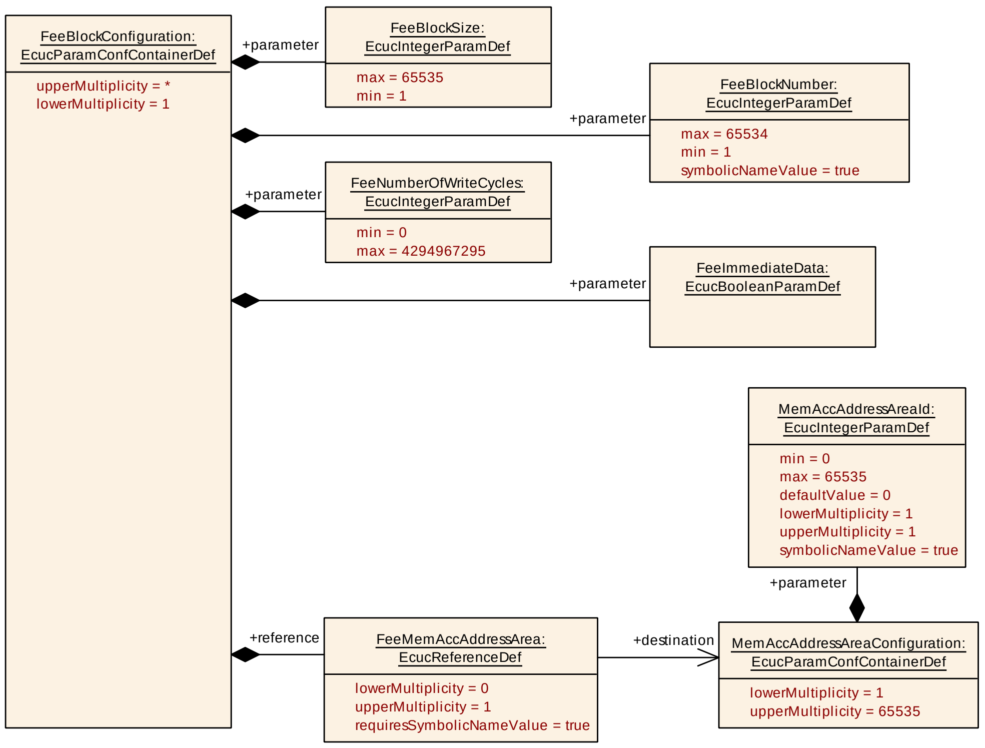
FeeBlockConfiguration: EcucParamConfContainerDef
upperMultiplicity = *
lowerMultiplicity = 1
+parameter
FeeBlockSize: EcucIntegerParamDef
max = 65535
min = 1
+parameter
FeeBlockNumber: EcucIntegerParamDef
max = 65534
min = 1
symbolicNameValue = true
+parameter
FeeNumberOfWriteCycles: EcucIntegerParamDef
min = 0
max = 4294967295
+parameter
FeeImmediateData: EcucBooleanParamDef
MemAccAddressAreaId: EcucIntegerParamDef
min = 0
max = 65535
defaultValue = 0
lowerMultiplicity = 1
upperMultiplicity = 1
symbolicNameValue = true
+parameter
+reference
FeeMemAccAddressArea: EcucReferenceDef
lowerMultiplicity = 0
upperMultiplicity = 1
requiresSymbolicNameValue = true
+destination
MemAccAddressAreaConfiguration: EcucParamConfContainerDef
lowerMultiplicity = 1
upperMultiplicity = 65535

Figure 10.2: Overview of configuration parameters of FeeBlockConfiguration

10.2 Published Information

10.2.1 FeePublishedInformation

[ECUC_Fee_00043] Definition of EcucParamConfContainerDef FeePublishedInformation ⌈

Container Name	FeePublishedInformation
Parent Container	Fee
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.
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
FeeBlockOverhead	1	[ECUC_Fee_00117]
FeePageOverhead	1	[ECUC_Fee_00118]

No Included Containers

⌋

[ECUC_Fee_00117] Definition of EcucIntegerParamDef FeeBlockOverhead ⌈

Parameter Name	FeeBlockOverhead		
Parent Container	FeePublishedInformation		
Description	Management overhead per logical block in bytes. Note: If the management overhead depends on the block size or block location a formula has to be provided that allows the configurator to calculate the management overhead correctly.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	–		
Post-Build Variant Value	false		
Value Configuration Class	**Published Information**	X	All Variants
Scope / Dependency	scope: local		

⌋

[ECUC_Fee_00118] Definition of EcucIntegerParamDef FeePageOverhead ⌈

Parameter Name	FeePageOverhead		
Parent Container	FeePublishedInformation		
Description	Management overhead per page in bytes. Note: If the management overhead depends on the block size or block location a formula has to be provided that allows the configurator to calculate the management overhead correctly.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	–		
Post-Build Variant Value	false		
Value Configuration Class	**Published Information**	X	All Variants
Scope / Dependency	scope: local		

⌋

A Not applicable requirements

[SWS_Fee_NA_00999]

Upstream requirements: SRS_BSW_00344, SRS_BSW_00404, SRS_BSW_00405, SRS_BSW_00171, SRS_BSW_00170, SRS_BSW_00380, SRS_BSW_00398, SRS_BSW_00399, SRS_BSW_00400, SRS_BSW_00375, SRS_BSW_00416, SRS_BSW_00168, SRS_BSW_00423, SRS_BSW_00424, SRS_BSW_00425, SRS_BSW_00426, SRS_BSW_00427, SRS_BSW_00428, SRS_BSW_00429, SRS_BSW_00432, SRS_BSW_00433, SRS_BSW_00336, SRS_BSW_00339, SRS_BSW_00422, SRS_BSW_00417, SRS_BSW_00161, SRS_BSW_00005, SRS_BSW_00415, SRS_BSW_00164, SRS_BSW_00342, SRS_BSW_00160, SRS_BSW_00007, SRS_BSW_00300, SRS_BSW_00347, SRS_BSW_00307, SRS_BSW_00314, SRS_BSW_00348, SRS_BSW_00353, SRS_BSW_00302, SRS_BSW_00328, SRS_BSW_00312, SRS_BSW_00006, SRS_BSW_00304, SRS_BSW_00378, SRS_BSW_00306, SRS_BSW_00308, SRS_BSW_00309, SRS_BSW_00359, SRS_BSW_00360, SRS_BSW_00330, SRS_BSW_00009, SRS_BSW_00401, SRS_BSW_00172, SRS_BSW_00010, SRS_BSW_00333, SRS_BSW_00321, SRS_BSW_00341, SRS_MemHwAb_14017

⌈These requirements are not applicable to this specification.⌋

B Change history of AUTOSAR traceable items

Please note that the lists in this chapter also include traceable items that have been removed from the specification in a later version. These items do not appear as hyperlinks in the document.

B.1 Traceable item history of this document according to AUTOSAR Release R24-11

B.1.1 Added Specification Items in R24-11

none

B.1.2 Changed Specification Items in R24-11

Number	Heading
[SWS_Fee_00105]	Definition of mandatory interfaces required by module Fee

Table B.1: Changed Specification Items in R24-11

B.1.3 Deleted Specification Items in R24-11

none