Document Title	Specification of Flash EEPROM Emulation
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	286
Document Classification	Standard
Document Status	Final
Part of AUTOSAR Release	4.2.2

Document Change History		
Release	**Changed by**	**Change Description**
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
4.2.1	AUTOSAR Release Management	• Requirement for blank checking added • Requirements linked to features, general and module specific requirements
4.1.3	AUTOSAR Release Management	• Editorial changes
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
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
4.0.3	AUTOSAR Administration	• DET errors added / removed • Handling of internal management operations detailed • Module short name changed • Consistency checking reformulated

- AUTOSAR confidential -

Document Change History

Release	Changed by	Change Description
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.)
3.1.4	AUTOSAR Administration	• Configuration variants clarified • Job result handling re-formulated • Range of configuration parameters restricted • Legal disclaimer revised
3.1.1	AUTOSAR Administration	• Legal disclaimer revised
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
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

Document Change History		
Release	**Changed by**	**Change Description**
2.0	AUTOSAR Administration	• Initial release

Disclaimer

This specification 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 specification.

The material contained in this specification is protected by copyright and other types of Intellectual Property Rights. The commercial exploitation of the material contained in this specification requires a license to such Intellectual Property Rights.

This specification 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 specification may be utilized or reproduced, in any form or by any means, without permission in writing from the publisher.

The AUTOSAR specifications have been developed for automotive applications only. They have neither been developed, nor tested for non-automotive applications.

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

Advice for users

AUTOSAR specifications may contain exemplary items (exemplary reference models, "use cases", and/or references to exemplary technical solutions, devices, processes or software).

Any such exemplary items are contained in the specifications for illustration purposes only, and they themselves are not part of the AUTOSAR Standard. Neither their presence in such specifications, nor any later documentation of AUTOSAR conformance of products actually implementing such exemplary items, imply that intellectual property rights covering such exemplary items are licensed under the same rules as applicable to the AUTOSAR Standard.

Table of Contents

1 Introduction and functional overview

This specification describes the functionality, API and configuration of the Flash EEPROM Emulation Module (see **Figure 1**).

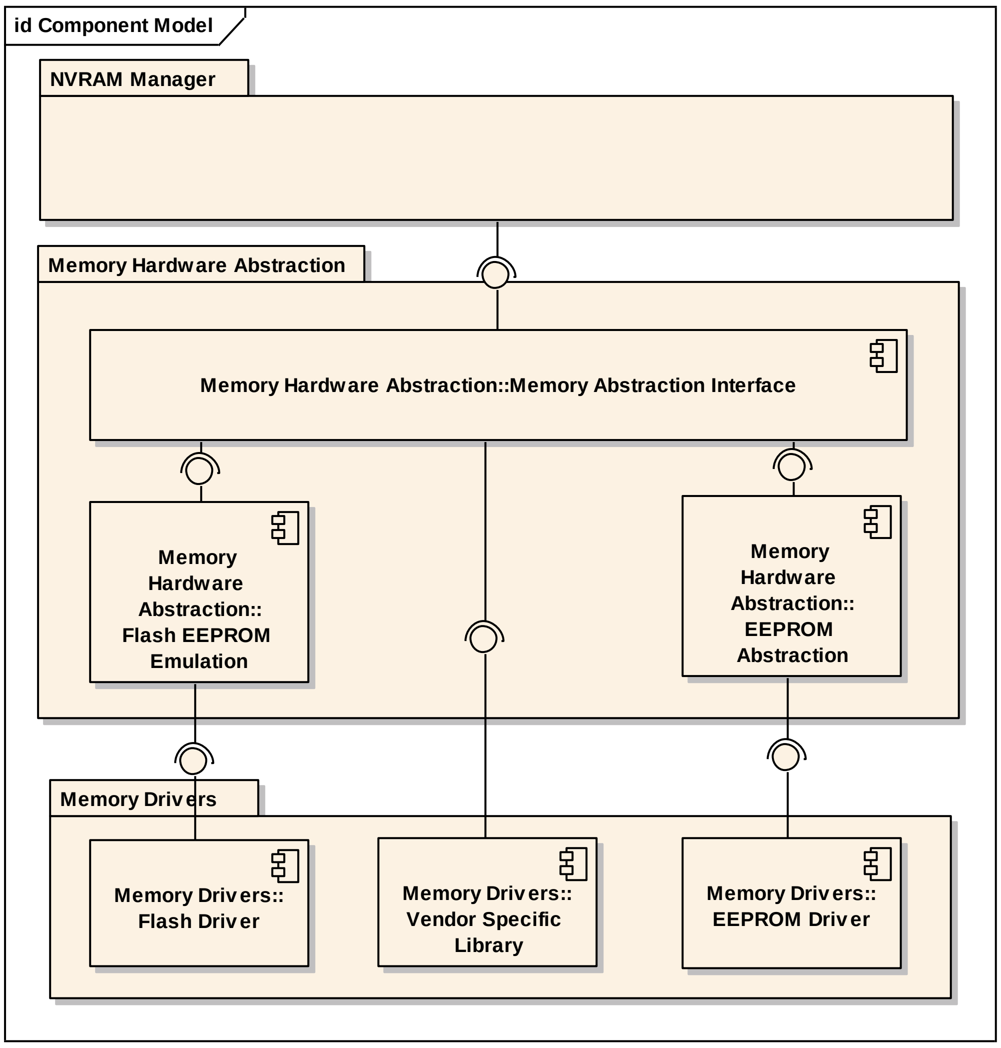
id Component Model
NVRAM Manager
Memory Hardware Abstraction
Memory Hardware Abstraction::Memory Abstraction Interface
Memory Hardware Abstraction:: Flash EEPROM Emulation
Memory Hardware Abstraction:: EEPROM Abstraction
Memory Drivers
Memory Drivers:: Flash Driver
Memory Drivers:: Vendor Specific Library
Memory Drivers:: EEPROM Driver

Figure 1: Module overview of memory hardware abstraction layer

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

Acronyms and abbreviations which have a local scope and therefore are not contained in the AUTOSAR glossary must appear in a local glossary.

Abbreviation / Acronym:	***Description:***
EA	EEPROM Abstraction
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.
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 3)).
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.

3 Related documentation

3.1 Input documents

[1] List of Basic Software Modules
AUTOSAR_TR_BSWModuleList.pdf

[2] Layered Software Architecture
AUTOSAR_EXP_LayeredSoftwareArchitecture..pdf

[3] General Requirements on Basic Software Modules
AUTOSAR_SRS_BSWGeneral.pdf

[4] General Requirements on SPAL
AUTOSAR_SRS_SPALGeneral.pdf

[5] Requirements on Memory Hardware Abstraction Layer
AUTOSAR_SRS_MemoryHWAbstractionLayer.doc

[6] Specification of Default Error Tracer
AUTOSAR_SWS_DefaultErrorTracer.pdf

[7] Specification of ECU Configuration
AUTOSAR_TPS_ECUConfiguration.pdf

[8] Basic Software Module Description Template
AUTOSAR_TPS_BSWModuleDescriptionTemplate.pdf

[9] General Specification of Basic Software Modules
AUTOSAR_SWS_BSWGeneral.pdf

3.2 Related standards and norms

[10] AUTOSAR Specification of NVRAM Manager
AUTOSAR_SWS_NVRAMManager.doc

[11] Specification of Memory Abstraction Interface
AUTOSAR_SWS_MemoryAbstractionInterface.pdf

[12] Specification of EEPROM Abstraction
AUTOSAR_SWS_EEPROMAbstraction.pdf

- AUTOSAR confidential -

3.3 Related specification

AUTOSAR provides a General Specification on Basic Software modules [9] (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.

- AUTOSAR confidential -

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.

5.1 Header file structure

[SWS_Fee_00002] ⌈ The file include structure shall be as follows:

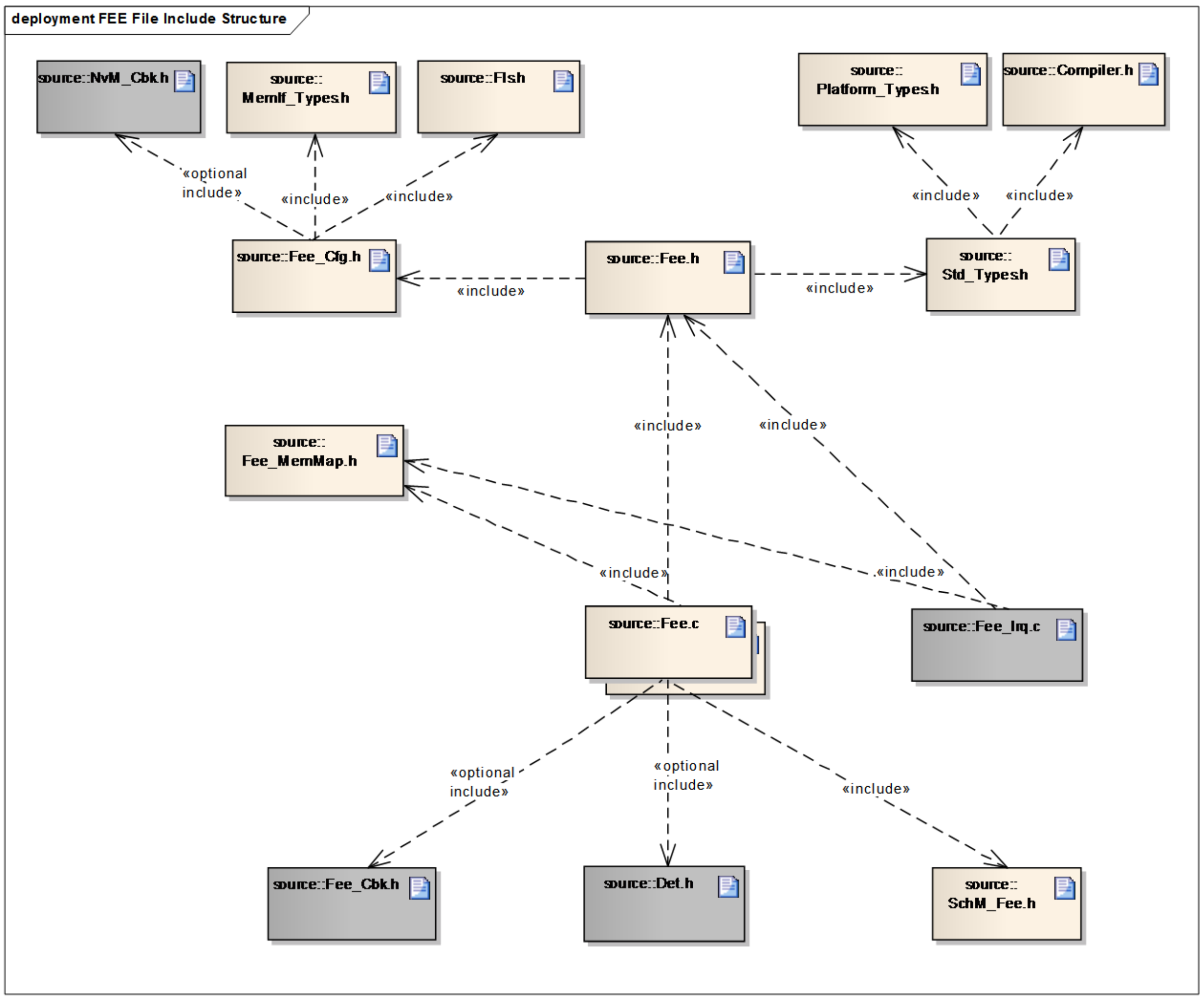

Figure 2: Flash EEPROM Emulation File Include Structure ⌋ (SRS_BSW_00167, SRS_BSW_00383, SRS_BSW_00346, SRS_BSW_00158, SRS_BSW_00301)

Note: Files which are optional (depending on implementation / configuration) are shown in grey.

6 Requirements traceability

Requirement	Description	Satisfied by
-	-	SWS_Fee_00068
BWS00300	-	SWS_Fee_00999
BWS00302	-	SWS_Fee_00999
BWS00304	-	SWS_Fee_00999
BWS00306	-	SWS_Fee_00999
BWS00307	-	SWS_Fee_00999
BWS00308	-	SWS_Fee_00999
BWS00309	-	SWS_Fee_00999
BWS00312	-	SWS_Fee_00999
BWS00314	-	SWS_Fee_00999
BWS00321	-	SWS_Fee_00999
BWS00323	-	SWS_Fee_00999
BWS00324	-	SWS_Fee_00999
BWS00326	-	SWS_Fee_00999
BWS00328	-	SWS_Fee_00999
BWS00330	-	SWS_Fee_00999
BWS00333	-	SWS_Fee_00999
BWS00334	-	SWS_Fee_00999
BWS00336	-	SWS_Fee_00999
BWS00339	-	SWS_Fee_00999
BWS00341	-	SWS_Fee_00999
BWS00342	-	SWS_Fee_00999
BWS00344	-	SWS_Fee_00999
BWS00347	-	SWS_Fee_00999
BWS00348	-	SWS_Fee_00999
BWS00353	-	SWS_Fee_00999
BWS00355	-	SWS_Fee_00999
BWS00359	-	SWS_Fee_00999
BWS00360	-	SWS_Fee_00999
BWS00361	-	SWS_Fee_00999
BWS00371	-	SWS_Fee_00999
BWS00375	-	SWS_Fee_00999
BWS00378	-	SWS_Fee_00999
BWS00380	-	SWS_Fee_00999
BWS00398	-	SWS_Fee_00999

BWS00399	-	SWS_Fee_00999
BWS00400	-	SWS_Fee_00999
BWS00401	-	SWS_Fee_00999
BWS00404	-	SWS_Fee_00999
BWS00405	-	SWS_Fee_00999
BWS00412	-	SWS_Fee_00999
BWS00415	-	SWS_Fee_00999
BWS00416	-	SWS_Fee_00999
BWS00417	-	SWS_Fee_00999
BWS00420	-	SWS_Fee_00999
BWS00421	-	SWS_Fee_00999
BWS00422	-	SWS_Fee_00999
BWS00423	-	SWS_Fee_00999
BWS00424	-	SWS_Fee_00999
BWS00425	-	SWS_Fee_00999
BWS00426	-	SWS_Fee_00999
BWS00427	-	SWS_Fee_00999
BWS00428	-	SWS_Fee_00999
BWS00429	-	SWS_Fee_00999
BWS00431	-	SWS_Fee_00999
BWS00432	-	SWS_Fee_00999
BWS00433	-	SWS_Fee_00999
BWS00434	-	SWS_Fee_00999
BWS005	-	SWS_Fee_00999
BWS006	-	SWS_Fee_00999
BWS007	-	SWS_Fee_00999
BWS009	-	SWS_Fee_00999
BWS010	-	SWS_Fee_00999
BWS12056	-	SWS_Fee_00999
BWS12058	-	SWS_Fee_00999
BWS12059	-	SWS_Fee_00999
BWS12060	-	SWS_Fee_00999
BWS12062	-	SWS_Fee_00999
BWS12063	-	SWS_Fee_00999
BWS12064	-	SWS_Fee_00999
BWS12067	-	SWS_Fee_00999
BWS12068	-	SWS_Fee_00999
BWS12069	-	SWS_Fee_00999
BWS12077	-	SWS_Fee_00999

BWS12078	-	SWS_Fee_00999
BWS12081	-	SWS_Fee_00999
BWS12092	-	SWS_Fee_00999
BWS12125	-	SWS_Fee_00999
BWS12129	-	SWS_Fee_00999
BWS12155	-	SWS_Fee_00999
BWS12163	-	SWS_Fee_00999
BWS12263	-	SWS_Fee_00999
BWS12265	-	SWS_Fee_00999
BWS12267	-	SWS_Fee_00999
BWS12461	-	SWS_Fee_00999
BWS12462	-	SWS_Fee_00999
BWS12463	-	SWS_Fee_00999
BWS14003	-	SWS_Fee_00999
BWS14017	-	SWS_Fee_00999
BWS157	-	SWS_Fee_00999
BWS160	-	SWS_Fee_00999
BWS161	-	SWS_Fee_00999
BWS164	-	SWS_Fee_00999
BWS168	-	SWS_Fee_00999
BWS170	-	SWS_Fee_00999
BWS171	-	SWS_Fee_00999
BWS172	-	SWS_Fee_00999
RS_BRF_00420	-	SWS_Fee_00074, SWS_Fee_00090, SWS_Fee_00128, SWS_Fee_00129, SWS_Fee_00130
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_00075, SWS_Fee_00091, SWS_Fee_00097, 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_00054, SWS_Fee_00055, SWS_Fee_00056, SWS_Fee_00095, SWS_Fee_00096, SWS_Fee_00099
RS_BRF_01076	AUTOSAR basic software shall perform module local error recovery to the extent possible	SWS_Fee_00187
RS_BRF_01448	AUTOSAR services shall support mode and state	SWS_Fee_00086

	management	
RS_BRS_01064	-	SWS_Fee_00098
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_00158	All modules of the AUTOSAR Basic Software shall strictly separate configuration from implementation	SWS_Fee_00002
SRS_BSW_00167	All AUTOSAR Basic Software Modules shall provide configuration rules and constraints to enable plausibility checks	SWS_Fee_00002
SRS_BSW_00301	All AUTOSAR Basic Software Modules shall only import the necessary information	SWS_Fee_00002
SRS_BSW_00323	All AUTOSAR Basic Software Modules shall check passed API parameters for validity	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_00345	BSW Modules shall support pre-compile configuration	SWS_Fee_00167
SRS_BSW_00346	All AUTOSAR Basic Software Modules shall provide at least a basic set of module files	SWS_Fee_00002
SRS_BSW_00383	The Basic Software Module specifications shall specify which other configuration files from other modules they use at least in the description	SWS_Fee_00002
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 for detecting an error	SWS_Fee_00010
SRS_BSW_00387	-	SWS_Fee_00052, SWS_Fee_00054, SWS_Fee_00055, SWS_Fee_00056, SWS_Fee_00142, SWS_Fee_00143
SRS_BSW_00392	Parameters shall have a type	SWS_Fee_00016, SWS_Fee_00084
SRS_BSW_00406	A static status variable denoting if a BSW module is initialized shall be initialized with value 0 before any APIs of the BSW module is called	SWS_Fee_00010, SWS_Fee_00034, SWS_Fee_00120, SWS_Fee_00121, 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, SWS_Fee_00189
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 layers 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

- AUTOSAR confidential -

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_14018	The FEE module shall extend the functional scope of an internal flash driver	SWS_Fee_00020, SWS_Fee_00170
SRS_MemHwAb_14026	The block numbers 0x0000 and 0xFFFF 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_00176
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

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] ⌈ 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. ⌋ (SRS_MemHwAb_14001, SRS_MemHwAb_14005)

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] ⌈ Each configured logical block shall take up an integer multiple of the configured virtual page size (see also Chapter 10.1 configuration parameter `FeeVirtualPageSize`). ⌋ (SRS_MemHwAb_14001)

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 3*). 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.*

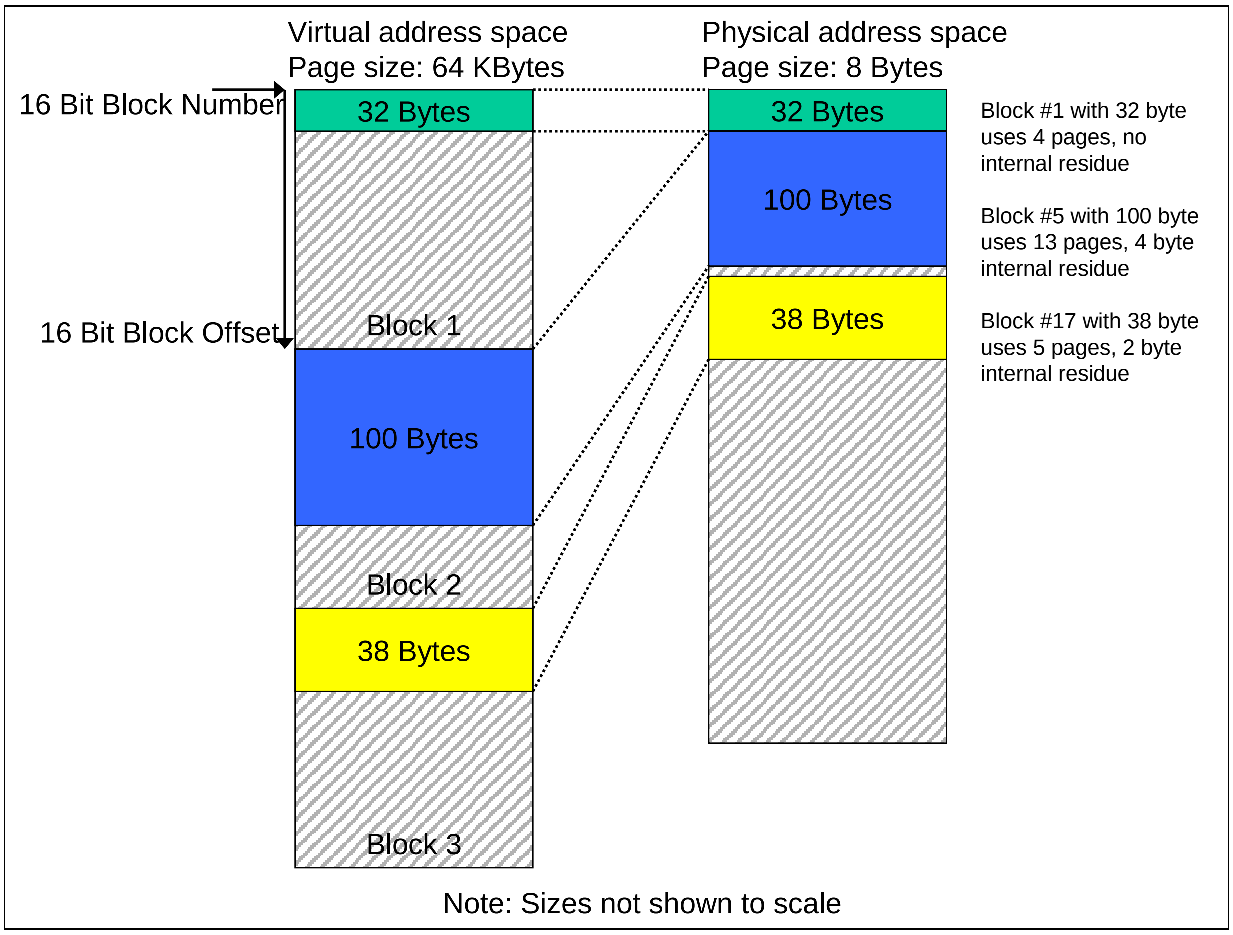
Virtual address space
Page size: 64 KBytes
Physical address space
Page size: 8 Bytes
16 Bit Block Number
16 Bit Block Offset
32 Bytes
Block 1
100 Bytes
Block 2
38 Bytes
Block 3
32 Bytes
100 Bytes
38 Bytes
Block #1 with 32 byte uses 4 pages, no internal residue
Block #5 with 100 byte uses 13 pages, 4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

Figure 3: Virtual vs. physical memory layout

[SWS_Fee_00071] ⌈ Logical blocks must not overlap each other and must not be contained within one another. ⌋ (SRS_MemHwAb_14001)

[SWS_Fee_00006] ⌈ The block numbers 0x0000 and 0xFFFF shall not be configurable for a logical block. ⌋ (SRS_MemHwAb_14026)

7.1.2 Address calculation

[SWS_Fee_00007] ⌈ 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. ⌋ (SRS_MemHwAb_14009)

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] ⌈ Only those bits of the 16bit block number, that do not denote a specific dataset or redundant copy shall be used for address calculation. ⌋ (SRS_MemHwAb_14009)

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 MSB's) to this start address (***Figure 4***).*

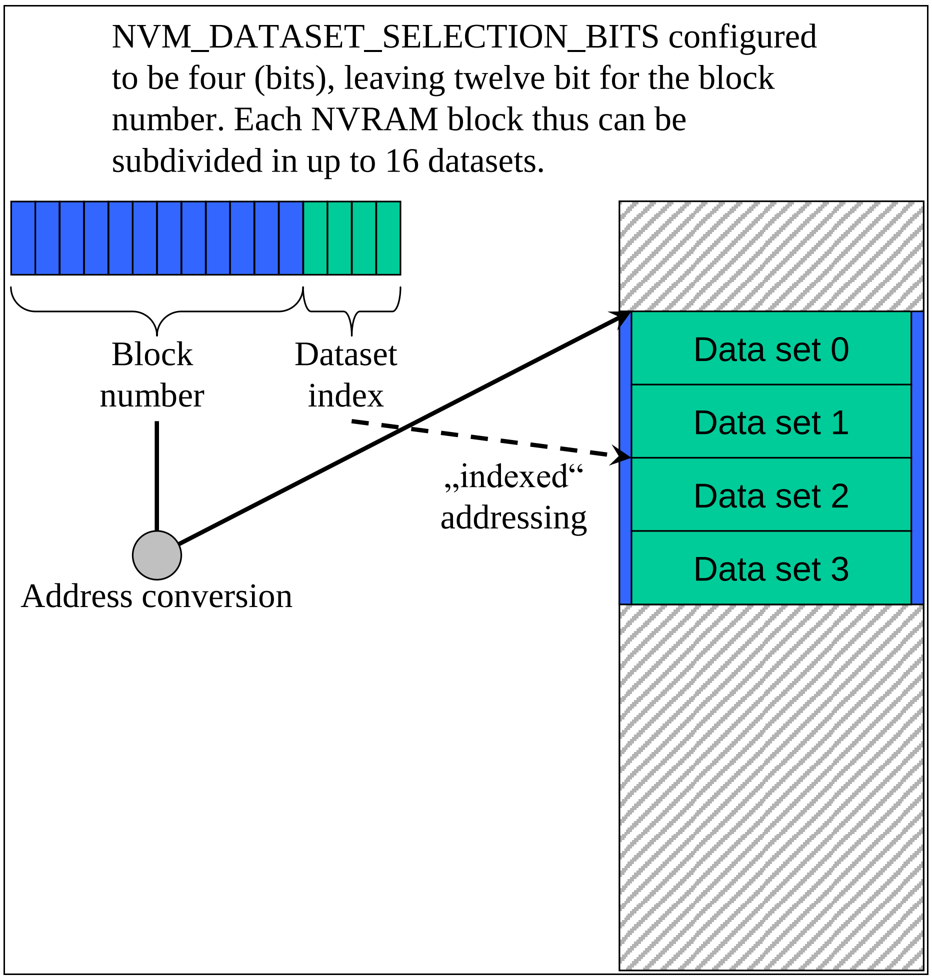
NVM_DATASET_SELECTION_BITS configured to be four (bits), leaving twelve bit for the block number. Each NVRAM block thus can be subdivided in up to 16 datasets.
Block number
Dataset index
„indexed“ addressing
Address conversion
Data set 0
Data set 1
Data set 2
Data set 3

Figure 4: Block number and dataset index

7.1.3 Limitation of erase cycles

[SWS_Fee_00102] ⌈ The configuration of the FEE module shall define the expected number of erase/write cycles for each logical block in the configuration parameter `FeeNumberOfWriteCycles`. ⌋ (SRS_MemHwAb_14002, SRS_MemHwAb_14012)

[SWS_Fee_00103] ⌈ 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. ⌋ (SRS_MemHwAb_14002, SRS_MemHwAb_14012)

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] ⌈ 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. ⌋ (SRS_MemHwAb_14013)

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] ⌈ 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. ⌋ (SRS_MemHwAb_14014)

[SWS_Fee_00153] ⌈ When a block write operation is started, the FEE module shall mark the corresponding block as "corrupted"[1]. ⌋ (SRS_MemHwAb_14014)

[SWS_Fee_00154] ⌈ Upon the successful end of the block write operation, the block shall be marked as "not corrupted" (again). ⌋ (SRS_MemHwAb_14014)

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.

7.2 Error classification

7.2.1 Development Errors

[SWS_Fee_00010] ⌈ The FEE module shall detect the following errors and exceptions depending on its configuration (development/production):

Type or error	*Relevance*	*Related error code*	*Value [hex]*
API service called when module was not initialized	Development	FEE_E_UNINIT	0x01
API service called with invalid block number	Development	FEE_E_INVALID_BLOCK_NO	0x02
API service called with invalid block offset	Development	FEE_E_INVALID_BLOCK_OFS	0x03
API service called with invalid data pointer	Development	FEE_E_PARAM_POINTER	0x04
API service called with invalid length information	Development	FEE_E_INVALID_BLOCK_LEN	0x05
API service called while module is busy processing a user request	Development	FEE_E_BUSY	0x06
Fee_Cancel called while no job was pending.	Development	FEE_E_INVALID_CANCEL	0x08
Fee_Init failed.	Development	FEE_E_INIT_FAILED	0x09

⌋ (SRS_BSW_00406, SRS_BSW_00337, SRS_BSW_00386, SRS_BSW_00327, SRS_BSW_00331)

[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.

Note: The error `FEE_E_BUSY_INTERNAL` *is not caused by a misbehaviour of the software but rather by a wrong (or better unlucky) timing of function calls. Therefore it shall only be a development error, even though this behaviour may also be observed in a production system.*

Note: The error `FEE_BUSY_INTERNAL` *shall only be reported, if the internal management operation cannot be suspended or aborted (see e.g. SWS_Fee_00173). Whether an internal management operation can be suspended or aborted depends first on the underlying hardware (flash technology) and second on the implementation of the FEE (design decision of the software implementor / customer).*

7.2.2 Runtime Errors

There are no runtime errors.

7.2.3 Transient Faults

There are no transient faults.

7.2.4 Production Errors

There are no production errors.

7.2.5 Extended Production Errors

There are no extended production errors.

7.3 Support for Debugging

[SWS_Fee_00130] {Obsolete} ⌈ The modules status, the job result and the block meta information (see SWS_Fee_00049) shall be made available for debugging (reading). ⌋ (RS_BRF_00420)

8 API specification

8.1 Imported Types

[SWS_Fee_00084]
⌈

Module	***Imported Type***
Fls	Fls_AddressType
	Fls_LengthType
MemIf	MemIf_JobResultType
	MemIf_ModeType
	MemIf_StatusType
Std_Types	Std_ReturnType
	Std_VersionInfoType

⌋ (SRS_BSW_00392)

[SWS_Fee_00016] ⌈ The types mentioned in SWS_Fee_00084 shall not be changed or extended for a specific FEE module or hardware platform. ⌋ (SRS_BSW_00392)

8.2 Type definitions

[**SWS_Fee_00188**]⌈

Name:	`Fee_ConfigType`	
Type:	`Structure`	
Range:	`implementation specific`	--
Description:	Configuration data structure of the Fee module.	

⌋ (SRS_BSW_00414)

8.3 Function definitions

8.3.1 Fee_Init

[SWS_Fee_00085]
⌈

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.	

⌋ (SRS_BSW_00101)

[**SWS_Fee_00189**]⌈ The configuration pointer `ConfigPtr` shall always have a `NULL_PTR` value.
⌋ (SRS_BSW_00414)

Note: the Configuration pointer ConfigPtr is currently not used and shall therefore be set `NULL_PTR` value.

[SWS_Fee_00120] ⌈ The function `Fee_Init` shall set the module state from `MEMIF_UNINIT` to `MEMIF_BUSY_INTERNAL` once it starts the module's initialization.
⌋ (SRS_BSW_00406)

[SWS_Fee_00168] ⌈ If initialization is finished within `Fee_Init`, the function Fee_Init shall set the module state from `MEMIF_BUSY_INTERNAL` to `MEMIF_IDLE` once initialization has been successfully finished. ⌋ (SRS_BSW_00101)

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_SetMode

[SWS_Fee_00086]
⌈

Service name:	Fee_SetMode	

Syntax:	`void Fee_SetMode(`
	`    MemIf_ModeType Mode`
	`)`	

Service ID[hex]:	0x01	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	Mode	Desired mode for the underlying flash driver
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Service to call the Fls_SetMode function of the underlying flash driver.	

⌋ (RS_BRF_01448)

[SWS_Fee_00020] ⌈ If the current module status is `MEMIF_IDLE` and if supported by the underlying hardware and device driver, the function `Fee_SetMode` shall call the function `Fls_SetMode` of the underlying flash driver with the given "Mode" parameter. ⌋ (SRS_MemHwAb_14018)

Example: During normal operation of an ECU the FEE module and underlying device driver shall use as few (runtime) resources as possible, therefore the flash driver is switched to "slow" mode. During startup and especially during shutdown it might be desirable to read / write the NV memory blocks as fast as possible, therefore the FEE and the underlying device driver could be switched into "fast" mode.

[SWS_Fee_00121] ⌈ If development error detection is enabled for the module: the function `Fee_SetMode` shall check if the module status is `MEMIF_UNINIT`. If this is the case, the function `Fee_SetMode` shall raise the development error `FEE_E_UNINIT` and return to the caller without executing the mode switch. ⌋ (SRS_BSW_00406)

[SWS_Fee_00170] ⌈ If development error detection is enabled for the module: the function `Fee_SetMode` shall check if the module state is `MEMIF_BUSY`. If this is the case, the function `Fee_SetMode` shall raise the development error `FEE_E_BUSY` and return to the caller without executing the mode switch. ⌋ (SRS_MemHwAb_14018)

8.3.3 Fee_Read

[SWS_Fee_00087]
⌈

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.	

⌋ (SRS_MemHwAb_14029)

[SWS_Fee_00021] ⌈ The function `Fee_Read` shall take the block start address and offset and calculate the corresponding memory read address. ⌋ (SRS_MemHwAb_14007)

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] ⌈ 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. ⌋ (SRS_MemHwAb_14029)

[SWS_Fee_00172] ⌈ 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. ⌋ (RS_BRF_01048)

[SWS_Fee_00073] ⌈ The FEE module shall execute the read operation asynchronously within the FEE module's main function. ⌋ (RS_BRF_01048)

[SWS_Fee_00122] ⌈ 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 reject the read request, raise the development error FEE_E_UNINIT and return with E_NOT_OK. ⌋ (SRS_BSW_00406)

[SWS_Fee_00133] ⌈ If development error detection is enabled for the module: 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 development error FEE_E_BUSY and return with E_NOT_OK. ⌋ (RS_BRF_01048)

[SWS_Fee_00134] ⌈ 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 reject the read request, raise the development error FEE_E_INVALID_BLOCK_NO and return with E_NOT_OK. ⌋ (SRS_BSW_00323)

[SWS_Fee_00135] ⌈ 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 reject the read request, raise the development error FEE_E_INVALID_BLOCK_OFS and return with E_NOT_OK. ⌋ (SRS_BSW_00323)

[SWS_Fee_00136] ⌈ 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 reject the read request, raise the development error FEE_E_PARAM_POINTER and return with E_NOT_OK. ⌋ (SRS_BSW_00323)

[SWS_Fee_00137] ⌈ 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 reject the read request, raise the development error `FEE_E_INVALID_BLOCK_LEN` and return with `E_NOT_OK`. ⌋ (SRS_BSW_00323)

[SWS_Fee_00162] ⌈ 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, SWS_Fee_00137 or SWS_Fee_00173 apply, the function `Fee_Read` shall not change the current module status or job result. ⌋ (RS_BRF_01048)

[SWS_Fee_00187] ⌈ If the function Fls_BlankCheck is configured (in the flash driver), the function Fee_Read shall call the function Fls_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. ⌋ (RS_BRF_01076)

Note: Whether calling Fls_BlankCheck from Fee_Read is necessary or not depends on the underlying hardware and the implementation of the flash driver and shall not be further detailed in this specification. The manual of the flash driver shall contain detailed information, whether Fls_BlankCheck is required for a certain hardware and driver implementation or not.

8.3.4 Fee_Write

[SWS_Fee_00088]
⌈

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.	

⌋ (SRS_MemHwAb_14010)

[SWS_Fee_00024] ⌈ 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. ⌋ (SRS_MemHwAb_14006)

[SWS_Fee_00025] ⌈ 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`. ⌋ (SRS_MemHwAb_14010)

[SWS_Fee_00174] ⌈ 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`. ⌋ (RS_BRF_01048)

[SWS_Fee_00026] ⌈ The FEE module shall execute the write operation asynchronously within the FEE module's main function. ⌋ (SRS_MemHwAb_14010, RS_BRF_01048)

[SWS_Fee_00123] ⌈ 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 reject the write request, raise the development error `FEE_E_UNINIT` and return with `E_NOT_OK`. ⌋ (SRS_BSW_00406)

[SWS_Fee_00144] ⌈ If development error detection is enabled for the module: 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 development error `FEE_E_BUSY` and return with `E_NOT_OK`. ⌋ (RS_BRF_01048)

[SWS_Fee_00138] ⌈ 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 reject the write request, raise the development error `FEE_E_INVALID_BLOCK_NO` and return with `E_NOT_OK`. ⌋ (SRS_BSW_00323)

[SWS_Fee_00139] ⌈ 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 reject the write request, raise the development error `FEE_E_PARAM_POINTER` and return with `E_NOT_OK`. ⌋ (SRS_BSW_00323)

[SWS_Fee_00163] ⌈ If a write request is rejected by the function `Fee_Write`, i.e. requirements SWS_Fee_00123, SWS_Fee_00144, SWS_Fee_00138, SWS_Fee_00139 or SWS_Fee_00175 apply, the function `Fee_Write` shall not change the current module status or job result. ⌋ (RS_BRF_01048)

8.3.5 Fee_Cancel

[SWS_Fee_00089]
⌈

Service name:	Fee_Cancel

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

Service ID[hex]:	0x04
Sync/Async:	Synchronous
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.

⌋ (SRS_MemHwAb_14031)

Note: The function `Fee_Cancel` and the cancel function of the underlying flash driver are – from their behaviour – synchronous functions but they are asynchronous w.r.t. an ongoing read, erase or write job in the flash memory. The cancel functions shall only reset their modules internal variables so that a new job can be accepted by the modules. They do not cancel an ongoing job in the hardware and they do not wait for an ongoing job to be finished by the hardware. This might lead to the situation in which the module's state is reported as `MEMIF_IDLE` while there is still an ongoing job being executed by the hardware. Therefore, the flash driver's main function shall check that the hardware is indeed free before starting a new job (see chapter 9.4 for a detailed sequence diagram).

Note: The function `Fee_Cancel` should only be used by the NvM to abort a read or write request for an NV block if higher priority data (i.e. immediate data) has to be written.

[SWS_Fee_00124] ⌈ 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` and return to the caller without changing any internal variables. ⌋ (SRS_BSW_00406)

[SWS_Fee_00080] ⌈ 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. ⌋ (SRS_MemHwAb_14031)

[SWS_Fee_00081] ⌈ 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`. ⌋ (SRS_MemHwAb_14031)

[SWS_Fee_00164] ⌈ 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. ⌋ (RS_BRF_01048)

[SWS_Fee_00184] ⌈ 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 development error `FEE_E_INVALID_CANCEL`. ⌋ (SRS_MemHwAb_14031)

8.3.6 Fee_GetStatus

[SWS_Fee_00090]
⌈

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.	

⌋ (RS_BRF_00420)

[SWS_Fee_00034] ⌈ The function `Fee_GetStatus` shall return `MEMIF_UNINIT` if the module has not (yet) been initialized. ⌋ (SRS_BSW_00406)

[SWS_Fee_00128] ⌈ 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. ⌋ (RS_BRF_00420)

[SWS_Fee_00129] ⌈ The function `Fee_GetStatus` shall return `MEMIF_BUSY` if it is currently processing a request from the upper layer. ⌋ (RS_BRF_00420)

[SWS_Fee_00074] ⌈ The function `Fee_GetStatus` shall return `MEMIF_BUSY_INTERNAL`, if an internal management operation is currently ongoing. ⌋ (RS_BRF_00420)

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.

8.3.7 Fee_GetJobResult

[SWS_Fee_00091]
⌈

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.	

⌋ (RS_BRF_01048)

[SWS_Fee_00035] ⌈ The function `Fee_GetJobResult` shall return `MEMIF_JOB_OK` if the last job has been finished successfully. ⌋ (RS_BRF_01048)

[SWS_Fee_00156] ⌈ The function `Fee_GetJobResult` shall return `MEMIF_JOB_PENDING` if the requested job is still waiting for execution or is currently being executed. ⌋ (RS_BRF_01048)

[SWS_Fee_00157] ⌈ The function `Fee_GetJobResult` shall return `MEMIF_JOB_CANCELED` if the last job has been canceled by the upper layer. ⌋ (SRS_MemHwAb_14031)

[SWS_Fee_00158] ⌈ The function `Fee_GetJobResult` shall return `MEMIF_JOB_FAILED` if the last job has failed. ⌋ (RS_BRF_01048)

[SWS_Fee_00159] ⌈ The function `Fee_GetJobResult` shall return `MEMIF_BLOCK_INCONSISTENT` if the requested block is found to be inconsistent (see chapter 7.1.5 for details). ⌋ (SRS_MemHwAb_14014)

[SWS_Fee_00160] ⌈ The function `Fee_GetJobResult` shall return `MEMIF_BLOCK_INVALID` if the requested block has been invalidated by the upper layer. ⌋ (SRS_MemHwAb_14028)

[SWS_Fee_00155] ⌈ 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. ⌋ (RS_BRF_01048)

[SWS_Fee_00125] ⌈ 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` and return with `MEMIF_JOB_FAILED`. ⌋ (SRS_BSW_00406)

8.3.8 Fee_InvalidateBlock

[SWS_Fee_00092]
⌈

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 - only if DET is enabled: The requested job has not been accepted by the module.
Description:	Service to invalidate a logical block.	

⌋ (SRS_MemHwAb_14028)

[SWS_Fee_00036] ⌈ The function `Fee_InvalidateBlock` shall take the block number and calculate the corresponding memory block address. ⌋ (SRS_MemHwAb_14009)

[SWS_Fee_00037] ⌈ 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. ⌋ (SRS_MemHwAb_14028)

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_00176] ⌈ If the current module status is not MEMIF_IDLE, the function Fee_InvalidateBlock shall reject the invalidation request and return with E_NOT_OK. ⌋ (SRS_MemHwAb_14028)

[SWS_Fee_00126] ⌈ 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 reject the invalidation request, raise the development error FEE_E_UNINIT and return with E_NOT_OK. ⌋ (SRS_BSW_00406)

[SWS_Fee_00145] ⌈ If development error detection is enabled for the module: 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 development error FEE_E_BUSY and return with E_NOT_OK. ⌋ (RS_BRF_01048)

[SWS_Fee_00140] ⌈ 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 reject the request, raise the development error FEE_E_INVALID_BLOCK_NO and return with E_NOT_OK. ⌋ (SRS_BSW_00323)

[SWS_Fee_00165] ⌈ If an invalidation request is rejected by the function Fee_InvalidateBlock, i.e. requirements SWS_Fee_00126, SWS_Fee_00140, SWS_Fee_00145 or SWS_Fee_00177 apply, the function Fee_InvalidateBlock shall not change the current module status or job result. ⌋ (SRS_MemHwAb_14028)

8.3.9 Fee_GetVersionInfo

[SWS_Fee_00093]
⌈

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.	

⌋ (SRS_BSW_00407)

[SWS_Fee_00147] ⌈ 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. ⌋ (SRS_BSW_00323)

8.3.10 Fee_EraseImmediateBlock

[SWS_Fee_00094]
⌈

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 - only if DET is enabled: The requested job has not been accepted by the module.
Description:	Service to erase a logical block.	

⌋ (SRS_MemHwAb_14032)

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] ⌈ The function Fee_EraseImmediateBlock shall take the block number and calculate the corresponding memory block address. ⌋ (SRS_MemHwAb_14009)

[SWS_Fee_00067] ⌈ 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. ⌋ (SRS_MemHwAb_14013)

[SWS_Fee_00127] ⌈ 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 reject the erase request, raise the development error FEE_E_UNINIT and return with E_NOT_OK. ⌋ (SRS_BSW_00406)

[SWS_Fee_00146] ⌈ If development error detection is enabled for the module: 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 development error FEE_E_BUSY and return with E_NOT_OK. ⌋ (RS_BRF_01048)

[SWS_Fee_00068] ⌈ 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 and return E_NOT_OK without erasing the addressed logical block. ⌋ (-SRS_BSW_00323)

[SWS_Fee_00141] ⌈ 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 reject the erase request, raise the development error FEE_E_INVALID_BLOCK_NO and return with E_NOT_OK. ⌋ (SRS_BSW_00323)

[SWS_Fee_00166] ⌈ If a erase request is rejected by the function Fee_EraseImmediateBlock, i.e. requirements SWS_Fee_00068, SWS_Fee_00127, SWS_Fee_00141, SWS_Fee_00146 or SWS_Fee_00178 apply, the function Fee_EraseImmediateBlock shall not change the current module status or job result. ⌋ (SRS_MemHwAb_14032)

8.4 Call-back 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]
⌈

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.

⌋ (RS_BRF_01064)

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

[SWS_Fee_00052] ⌈ 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. ⌋ (RS_BRF_01064, SRS_BSW_00387)

[SWS_Fee_00142] ⌈ 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. ⌋ (SRS_BSW_00387)

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

8.4.2 Fee_JobErrorNotification

[SWS_Fee_00096]
⌈

Service name:	Fee_JobErrorNotification

Syntax:	`void Fee_JobErrorNotification(`
	`    void`
	`)`

Service ID[hex]:	0x11
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 failure of an asynchronous operation.

⌋ (RS_BRF_01064)

The underlying flash driver shall call the function `Fee_JobErrorNotification` to report the failure of an asynchronous operation.

[SWS_Fee_00054] ⌈ The function `Fee_JobErrorNotification` 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. ⌋ (RS_BRF_01064, SRS_BSW_00387)

[SWS_Fee_00143] ⌈ If the job result is currently MEMIF_JOB_PENDING, the function Fee_JobErrorNotification shall set the job result to MEMIF_JOB_FAILED, else it shall leave the job result untouched. ⌋ (SRS_BSW_00387)

Note: The function Fee_JobErrorNotification 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]
⌈

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.

⌋ (RS_BRF_01048)

[SWS_Fee_00169] ⌈ 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_BUSY_INTERNAL to MEMIF_IDLE once initialization of the module has been successfully finished. ⌋ (SRS_BSW_00101)

[SWS_Fee_00057] ⌈ The function Fee_MainFunction shall asynchronously handle the read / write / erase / invalidate jobs requested by the upper layer and internal management operations. ⌋ (RS_BRF_01048)

[SWS_Fee_00075] ⌈ 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. ⌋ (RS_BRF_01048, SRS_MemHwAb_14028)

[SWS_Fee_00023] ⌈ 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. ⌋ (SRS_MemHwAb_14014, SRS_MemHwAb_14015, SRS_MemHwAb_14016)

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] ⌈

API function	*Description*
Fls_Cancel	Cancels an ongoing job.
Fls_Compare	Compares the contents of an area of flash memory with that of an application data buffer.
Fls_Erase	Erases flash sector(s).
Fls_GetJobResult	Returns the result of the last job.
Fls_GetStatus	Returns the driver state.
Fls_Read	Reads from flash memory.
Fls_SetMode	Sets the flash driver's operation mode.
Fls_Write	Writes one or more complete flash pages.

⌋ (SRS_BSW_00384)

8.6.2 Optional Interfaces

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

[SWS_Fee_00104] ⌈

API function	*Description*
Det_ReportError	Service to report development errors.
Fls_BlankCheck	The function Fls_BlankCheck shall verify, whether a given memory area has been erased but not (yet) programmed. The function shall limit the maximum number of checked flash cells per main function cycle to the configured value FlsMaxReadNormalMode or FlsMaxReadFastMode respectively.

⌋ (SRS_BSW_00384)

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_00098]
⌈

Service name:	NvM_JobEndNotification

Syntax:	`void NvM_JobEndNotification(`
	`    void`
	`)`

Sync/Async:	true
Reentrancy:	Don't care
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	--

⌋ (RS_BRS_01064)

[SWS_Fee_00055] ⌈ 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 ⌋ (RS_BRF_01064, SRS_BSW_00387)

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

[SWS_Fee_00099]
⌈

Service name:	NvM_JobErrorNotification

Syntax:	`void NvM_JobErrorNotification(`
	`    void`
	`)`

Sync/Async:	true
Reentrancy:	Don't care
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	--

⌋ (RS_BRF_01064)

[SWS_Fee_00056] ⌈ The FEE module shall call the function defined in the configuration parameter `FeeNvmJobErrorNotification` 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 ⌋ (RS_BRF_01064, SRS_BSW_00387)

The function defined in the configuration parameter `FeeNvmJobErrorNotification` 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 and memory abstraction interface). 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.

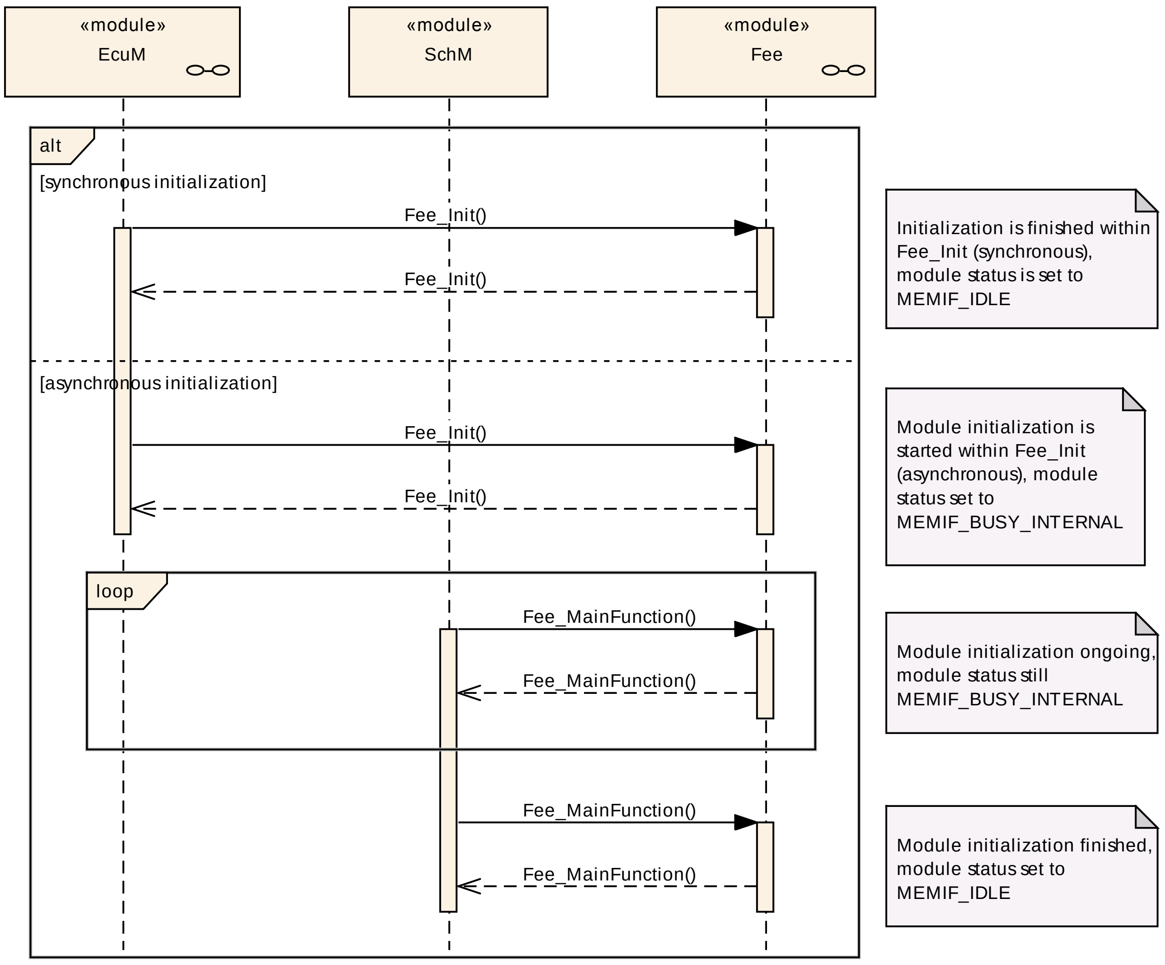
«module»
EcuM
«module»
SchM
«module»
Fee
alt
[synchronous initialization]
Fee_Init()
Fee_Init()
Initialization is finished within Fee_Init (synchronous), module status is set to MEMIF_IDLE
[asynchronous initialization]
Fee_Init()
Fee_Init()
Module initialization is started within Fee_Init (asynchronous),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

Figure 5: Sequence diagram of “Fee_Init” service

9.2 Fee_SetMode

The following figure shows exemplarily the call sequence for the `Fee_SetMode` service. This sequence diagram also applies to the other synchronous services of this module with exception of the `Fee_Init` routine (see above).

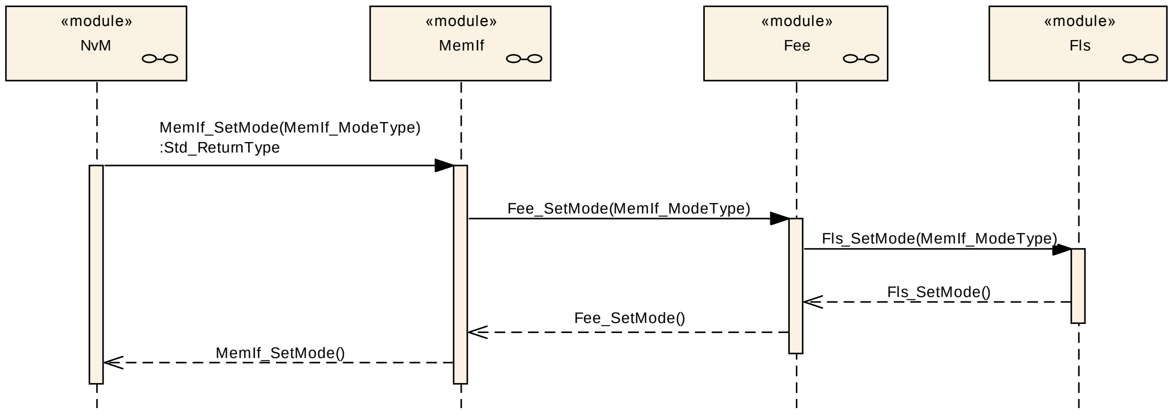
«module» NvM
«module» MemIf
«module» Fee
«module» Fls
MemIf_SetMode(MemIf_ModeType) :Std_ReturnType
Fee_SetMode(MemIf_ModeType)
Fls_SetMode(MemIf_ModeType)
Fls_SetMode()
Fee_SetMode()
MemIf_SetMode()

Figure 6: Sequence diagram of the "Fee_SetMode" service

9.3 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.

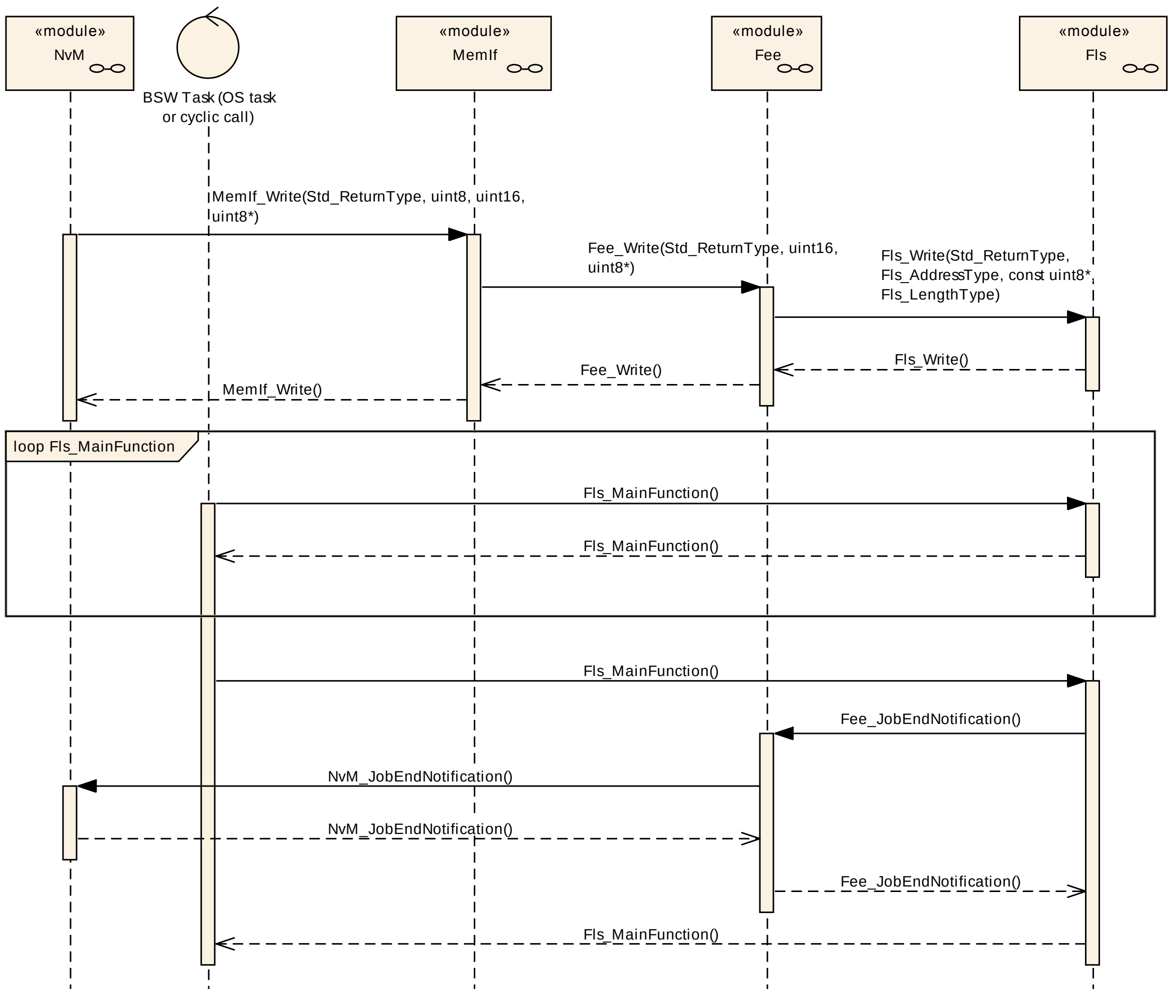

Figure 7: Sequence diagram “Fee_Write”

9.4 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.

- AUTOSAR confidential -

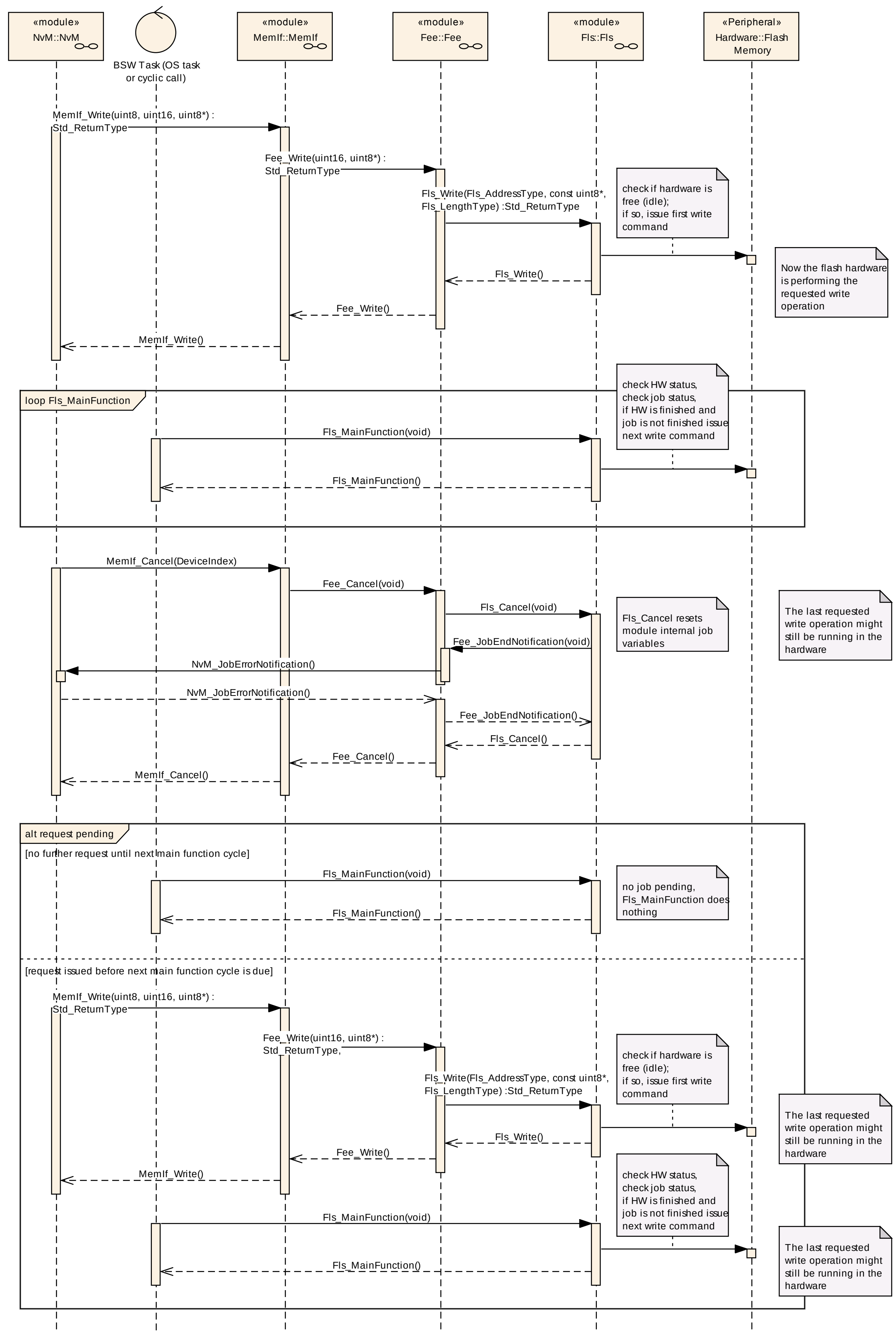
«module» NvM::NvM
BSW Task (OS task or cyclic call)
«module» MemIf::MemIf
«module» Fee::Fee
«module» Fls::Fls
«Peripheral» Hardware::Flash Memory
MemIf_Write(uint8, uint16, uint8*) : Std_ReturnType
Fee_Write(uint16, uint8*) : Std_ReturnType
Fls_Write(Fls_AddressType, const uint8*, Fls_LengthType) :Std_ReturnType
check if hardware is free (idle); if so, issue first write command
Now the flash hardware is performing the requested write operation
Fls_Write()
Fee_Write()
MemIf_Write()
loop Fls_MainFunction
check HW status, check job status, if HW is finished and job is not finished issue next write command
Fls_MainFunction(void)
Fls_MainFunction()
MemIf_Cancel(DeviceIndex)
Fee_Cancel(void)
Fls_Cancel(void)
Fls_Cancel resets module internal job variables
The last requested write operation might still be running in the hardware
Fee_JobEndNotification(void)
NvM_JobErrorNotification()
NvM_JobErrorNotification()
Fee_JobEndNotification()
Fls_Cancel()
Fee_Cancel()
MemIf_Cancel()
alt request pending
[no further request until next main function cycle]
Fls_MainFunction(void)
no job pending, Fls_MainFunction does nothing
Fls_MainFunction()
[request issued before next main function cycle is due]
MemIf_Write(uint8, uint16, uint8*) : Std_ReturnType
Fee_Write(uint16, uint8*) : Std_ReturnType,
Fls_Write(Fls_AddressType, const uint8*, Fls_LengthType) :Std_ReturnType
check if hardware is free (idle); if so, issue first write command
The last requested write operation might still be running in the hardware
Fls_Write()
Fee_Write()
MemIf_Write()
check HW status, check job status, if HW is finished and job is not finished issue next write command
Fls_MainFunction(void)
The last requested write operation might still be running in the hardware
Fls_MainFunction()

Figure 8: Sequence diagram „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 Chapters 7 and Chapter 8.

10.1.1 Variants

[SWS_Fee_00167] ⌈ The FEE module shall support (only) the following configuration variants:

- `VARIANT-PRE-COMPILE`
  Only parameters with “Pre-compile time” configuration are allowed in this variant. ⌋ (SRS_BSW_00345)

10.1.2 Fee

SWS Item	**ECUC_Fee_00154 :**
Module Name	*Fee*
Module Description	Configuration of the Fee (Flash EEPROM Emulation) module.
Post-Build Variant Support	false

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 CommonPublishedInformation container. Note that these parameters do not have any configuration class setting, since they are published information.

10.1.3 FeeGeneral

SWS Item	**ECUC_Fee_00039 :**
Container Name	FeeGeneral
Description	Container for general parameters. These parameters are not specific to a block.
Configuration Parameters	

SWS Item	**ECUC_Fee_00111 :**
Name	FeeDevErrorDetect
Description	Switches the Default Error Tracer (Det) detection and notification ON or

	OFF. • true: enabled (ON). • false: disabled (OFF).		
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		

SWS Item	**ECUC_Fee_00153 :**		
Name	FeeMainFunctionPeriod		
Description	The period between successive calls to the main function in seconds.		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	1E-7 .. 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		

SWS Item	**ECUC_Fee_00112 :**		
Name	FeeNvmJobEndNotification		
Description	Mapped to the job end notification routine provided by the upper layer module (NvM_JobEndNotification).		
Multiplicity	0..1		
Type	EcucFunctionNameDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
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		

SWS Item	**ECUC_Fee_00113 :**
Name	FeeNvmJobErrorNotification
Description	Mapped to the job error notification routine provided by the upper layer module (NvM_JobErrorNotification).
Multiplicity	0..1
Type	EcucFunctionNameDef
Default value	--
maxLength	--

minLength	--		
regularExpression	--		
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		

SWS Item	**ECUC_Fee_00114 :**		
Name	FeePollingMode		
Description	Pre-processor switch to enable and disable the polling mode for this module. true: Polling mode enabled, callback functions (provided to FLS module) disabled. false: Polling mode disabled, callback functions (provided to FLS 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		

SWS Item	**ECUC_Fee_00119 :**		
Name	FeeSetModeSupported		
Description	Compiler switch to enable/disable the 'SetMode' functionality of the FEE module. TRUE: SetMode functionality supported / code present, FALSE: SetMode functionality not supported / code not present. Note: This configuration setting has to be consistent with that of all underlying flash device drivers (configuration parameter FlsSetModeApi).		
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		

SWS Item	**ECUC_Fee_00115 :**
Name	FeeVersionInfoApi
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	--
Post-Build Variant Value	false

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

SWS Item	**ECUC_Fee_00116 :**		
Name	FeeVirtualPageSize		
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		

No Included Containers

10.1.4 FeeBlockConfiguration

SWS Item	**ECUC_Fee_00040 :**
Container Name	FeeBlockConfiguration
Description	Configuration of block specific parameters for the Flash EEPROM Emulation module.
Configuration Parameters	

SWS Item	**ECUC_Fee_00150 :**		
Name	FeeBlockNumber		
Description	Block identifier (handle). 0x0000 and 0xFFFF shall not be used for block numbers (see FEE006). Range: min = 2^NVM_DATASET_SELECTION_BITS max = 0xFFFF - 2^NVM_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		

SWS Item	**ECUC_Fee_00148 :**	
Name	FeeBlockSize	
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		

SWS Item	**ECUC_Fee_00151 :**		
Name	FeeImmediateData		
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		

SWS Item	**ECUC_Fee_00110 :**		
Name	FeeNumberOfWriteCycles		
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		

SWS Item	**ECUC_Fee_00149 :**		
Name	FeeDeviceIndex		
Description	Reference to the device this block is stored in.		
Multiplicity	1		
Type	Symbolic name reference to [FlsGeneral]		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local dependency: This information is needed by the NVRAM manager respectively the Memory Abstraction Interface to address a certain logical block. It is listed in this specification to give a complete overview over all block related configuration parameters.		

No Included Containers

10.2 Published Information

10.2.1 FeePublishedInformation

SWS Item	**ECUC_Fee_00043 :**
Container Name	FeePublishedInformation
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	

SWS Item	**ECUC_Fee_00117 :**		
Name	FeeBlockOverhead		
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		

SWS Item	**ECUC_Fee_00118 :**		
Name	FeePageOverhead		
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		

No Included Containers

- AUTOSAR confidential -

11 Not applicable requirements

[SWS_Fee_00999] ⌈ These requirements are not applicable to this specification. ⌋ (BWS00344, BWS00404, BWS00405, BWS171, BWS170, BWS00380, BWS00412, BWS00398, BWS00399, BWS00400, BWS00375, BWS00416, BWS168, BWS00423, BWS00424, BWS00425, BWS00426, BWS00427, BWS00428, BWS00429, BWS00431, BWS00432, BWS00433, BWS00434, BWS00336, BWS00339, BWS00421, BWS00422, BWS00420, BWS00417, BWS00323, BWS161, BWS00324, BWS005, BWS00415, BWS164, BWS00326, BWS00342, BWS160, BWS007, BWS00300, BWS00347, BWS00307, BWS00314, BWS00348, BWS00353, BWS00361, BWS00302, BWS00328, BWS00312, BWS006, BWS00304, BWS00355, BWS00378, BWS00306, BWS00308, BWS00309, BWS00371, BWS00359, BWS00360, BWS00330, BWS009, BWS00401, BWS172, BWS010, BWS00333, BWS00321, BWS00341, BWS00334, BWS12263, BWS12056, BWS12267, BWS12125, BWS12163, BWS12058, BWS12059, BWS12060, BWS12461, BWS12462, BWS12463, BWS12062, BWS12068, BWS12069, BWS157, BWS12155, BWS12063, BWS12129, BWS12064, BWS12067, BWS12077, BWS12078, BWS12092, BWS12265, BWS12081, BWS14003, BWS14017)