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	R22-11

Document Change History

Date	Release	Changed by	Change Description
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

Document Change History

Date	Release	Changed by	Change Description
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

Document Change History

Date	Release	Changed by	Change Description
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.)

Document Change History

Date	Release	Changed by	Change Description
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

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

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

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

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

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

Table of Contents

List of Figures

1 Introduction and functional overview

This specification describes the functionality, API and configuration of the Flash EEPROM Emulation Module.
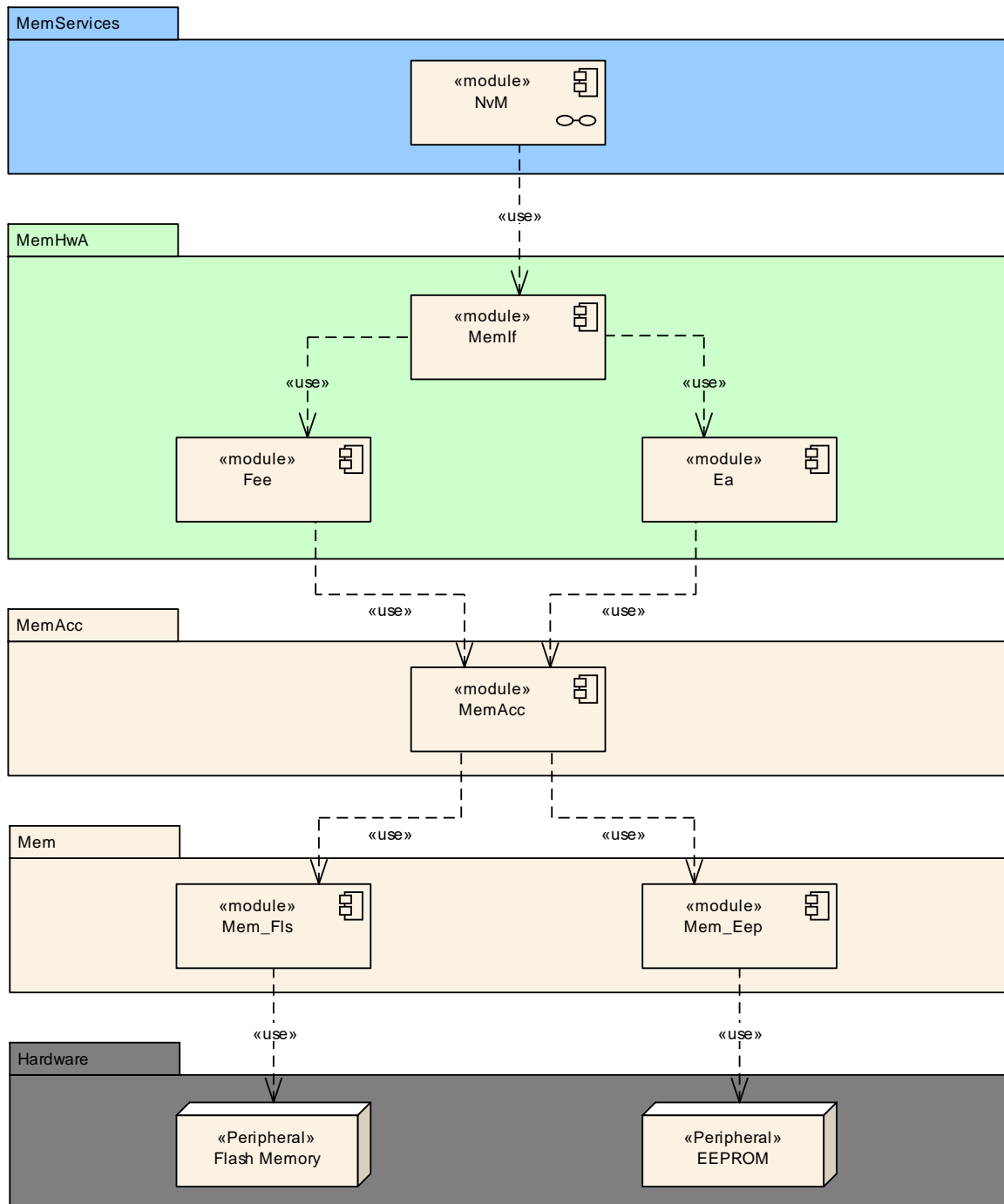
Figure 1: Module overview of memory stack

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

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
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 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

[13]   Specification of Memory Access
AUTOSAR_SWS_MemoryAccess.pdf

[14]   Specification of Memory Driver
AUTOSAR_SWS_MemoryDriver.pdf

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.

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 traceability

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_00005	Modules of the \mu C Abstraction Layer (MCAL) may not have hard coded horizontal interfaces	SWS_Fee_NA_00999
SRS_BSW_00006	The source code of software modules above the \mu C Abstraction Layer (MCAL) shall not be processor and compiler dependent.	SWS_Fee_NA_00999
SRS_BSW_00007	All Basic SW Modules written in C language shall conform to the MISRA C 2012 Standard.	SWS_Fee_NA_00999
SRS_BSW_00009	All Basic SW Modules shall be documented according to a common standard.	SWS_Fee_NA_00999
SRS_BSW_00010	The memory consumption of all Basic SW Modules shall be documented for a defined	SWS_Fee_NA_00999

	configuration for all supported platforms.	
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_00160	Configuration files of AUTOSAR Basic SW module shall be readable for human beings	SWS_Fee_NA_00999
SRS_BSW_00161	The AUTOSAR Basic Software shall provide a microcontroller abstraction layer which provides a standardized interface to higher software layers	SWS_Fee_NA_00999
SRS_BSW_00164	The Implementation of interrupt service routines shall be done by the Operating System, complex drivers or modules	SWS_Fee_NA_00999
SRS_BSW_00168	SW components shall be tested by a function defined in a common API in the Basis-SW	SWS_Fee_NA_00999
SRS_BSW_00170	The AUTOSAR SW Components shall provide information about their dependency from faults, signal qualities, driver demands	SWS_Fee_NA_00999
SRS_BSW_00171	Optional functionality of a Basic-SW component that is not required in the ECU shall be configurable at pre-compile-time	SWS_Fee_NA_00999
SRS_BSW_00172	The scheduling strategy that is built inside the Basic Software Modules shall be compatible with the strategy used in the system	SWS_Fee_NA_00999
SRS_BSW_00300	All AUTOSAR Basic Software Modules shall be identified by an unambiguous name	SWS_Fee_NA_00999
SRS_BSW_00302	All AUTOSAR Basic Software Modules shall only export information needed by other modules	SWS_Fee_NA_00999

SRS_BSW_00304	All AUTOSAR Basic Software Modules shall use only AUTOSAR data types instead of native C data types	SWS_Fee_NA_00999
SRS_BSW_00306	AUTOSAR Basic Software Modules shall be compiler and platform independent	SWS_Fee_NA_00999
SRS_BSW_00307	Global variables naming convention	SWS_Fee_NA_00999
SRS_BSW_00308	AUTOSAR Basic Software Modules shall not define global data in their header files, but in the C file	SWS_Fee_NA_00999
SRS_BSW_00309	All AUTOSAR Basic Software Modules shall indicate all global data with read-only purposes by explicitly assigning the const keyword	SWS_Fee_NA_00999
SRS_BSW_00312	Shared code shall be reentrant	SWS_Fee_NA_00999
SRS_BSW_00314	All internal driver modules shall separate the interrupt frame definition from the service routine	SWS_Fee_NA_00999
SRS_BSW_00321	The version numbers of AUTOSAR Basic Software Modules shall be enumerated according specific rules	SWS_Fee_NA_00999
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, SWS_Fee_NA_00999
SRS_BSW_00327	Error values naming convention	SWS_Fee_00010
SRS_BSW_00328	All AUTOSAR Basic Software Modules shall avoid the duplication of code	SWS_Fee_NA_00999
SRS_BSW_00330	It shall be allowed to use macros instead of functions where source code is used and runtime is critical	SWS_Fee_NA_00999
SRS_BSW_00331	All Basic Software Modules shall strictly	SWS_Fee_00010

	separate error and status information	
SRS_BSW_00333	For each callback function it shall be specified if it is called from interrupt context or not	SWS_Fee_NA_00999
SRS_BSW_00334	All Basic Software Modules shall provide an XML file that contains the meta data	SWS_Fee_NA_00999
SRS_BSW_00336	Basic SW module shall be able to shutdown	SWS_Fee_NA_00999
SRS_BSW_00337	Classification of development errors	SWS_Fee_00010
SRS_BSW_00339	Reporting of production relevant error status	SWS_Fee_NA_00999
SRS_BSW_00341	Module documentation shall contains all needed informations	SWS_Fee_NA_00999
SRS_BSW_00342	It shall be possible to create an AUTOSAR ECU out of modules provided as source code and modules provided as object code, even mixed	SWS_Fee_NA_00999
SRS_BSW_00344	BSW Modules shall support link-time configuration	SWS_Fee_NA_00999
SRS_BSW_00347	A Naming seperation of different instances of BSW drivers shall be in place	SWS_Fee_NA_00999
SRS_BSW_00348	All AUTOSAR standard types and constants shall be placed and organized in a standard type header file	SWS_Fee_NA_00999
SRS_BSW_00353	All integer type definitions of target and compiler specific scope shall be placed and organized in a single type header	SWS_Fee_NA_00999
SRS_BSW_00359	All AUTOSAR Basic Software Modules callback functions shall avoid return types other than void if possible	SWS_Fee_NA_00999
SRS_BSW_00360	AUTOSAR Basic Software Modules callback functions are allowed to have parameters	SWS_Fee_NA_00999

SRS_BSW_00375	Basic Software Modules shall report wake-up reasons	SWS_Fee_NA_00999
SRS_BSW_00378	AUTOSAR shall provide a boolean type	SWS_Fee_NA_00999
SRS_BSW_00380	Configuration parameters being stored in memory shall be placed into separate c-files	SWS_Fee_NA_00999
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_00398	The link-time configuration is achieved on object code basis in the stage after compiling and before linking	SWS_Fee_NA_00999
SRS_BSW_00399	Parameter-sets shall be located in a separate segment and shall be loaded after the code	SWS_Fee_NA_00999
SRS_BSW_00400	Parameter shall be selected from multiple sets of parameters after code has been loaded and started	SWS_Fee_NA_00999
SRS_BSW_00401	Documentation of multiple instances of configuration parameters shall be available	SWS_Fee_NA_00999
SRS_BSW_00404	BSW Modules shall support post-build configuration	SWS_Fee_NA_00999
SRS_BSW_00405	BSW Modules shall support multiple configuration sets	SWS_Fee_NA_00999
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_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	SWS_Fee_00093

	out the version information of a dedicated module implementation	
SRS_BSW_00414	Init functions shall have a pointer to a configuration structure as single parameter	SWS_Fee_00188
SRS_BSW_00415	Interfaces which are provided exclusively for one module shall be separated into a dedicated header file	SWS_Fee_NA_00999
SRS_BSW_00416	The sequence of modules to be initialized shall be configurable	SWS_Fee_NA_00999
SRS_BSW_00417	Software which is not part of the SW-C shall report error events only after the Dem is fully operational.	SWS_Fee_NA_00999
SRS_BSW_00422	Pre-de-bouncing of error status information is done within the Dem	SWS_Fee_NA_00999
SRS_BSW_00423	BSW modules with AUTOSAR interfaces shall be describable with the means of the SW-C Template	SWS_Fee_NA_00999
SRS_BSW_00424	BSW module main processing functions shall not be allowed to enter a wait state	SWS_Fee_NA_00999
SRS_BSW_00425	The BSW module description template shall provide means to model the defined trigger conditions of schedulable objects	SWS_Fee_NA_00999
SRS_BSW_00426	BSW Modules shall ensure data consistency of data which is shared between BSW modules	SWS_Fee_NA_00999
SRS_BSW_00427	ISR functions shall be defined and documented in the BSW module description template	SWS_Fee_NA_00999
SRS_BSW_00428	A BSW module shall state if its main processing function(s) has to be executed in a specific order or sequence	SWS_Fee_NA_00999
SRS_BSW_00429	Access to OS is restricted	SWS_Fee_NA_00999

SRS_BSW_00432	Modules should have separate main processing functions for read/receive and write/transmit data path	SWS_Fee_NA_00999
SRS_BSW_00433	Main processing functions are only allowed to be called from task bodies provided by the BSW Scheduler	SWS_Fee_NA_00999
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_14017	The EA module shall extend the functional scope of an EEPROM driver	SWS_Fee_NA_00999
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_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

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 (seeFigure 2). 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.
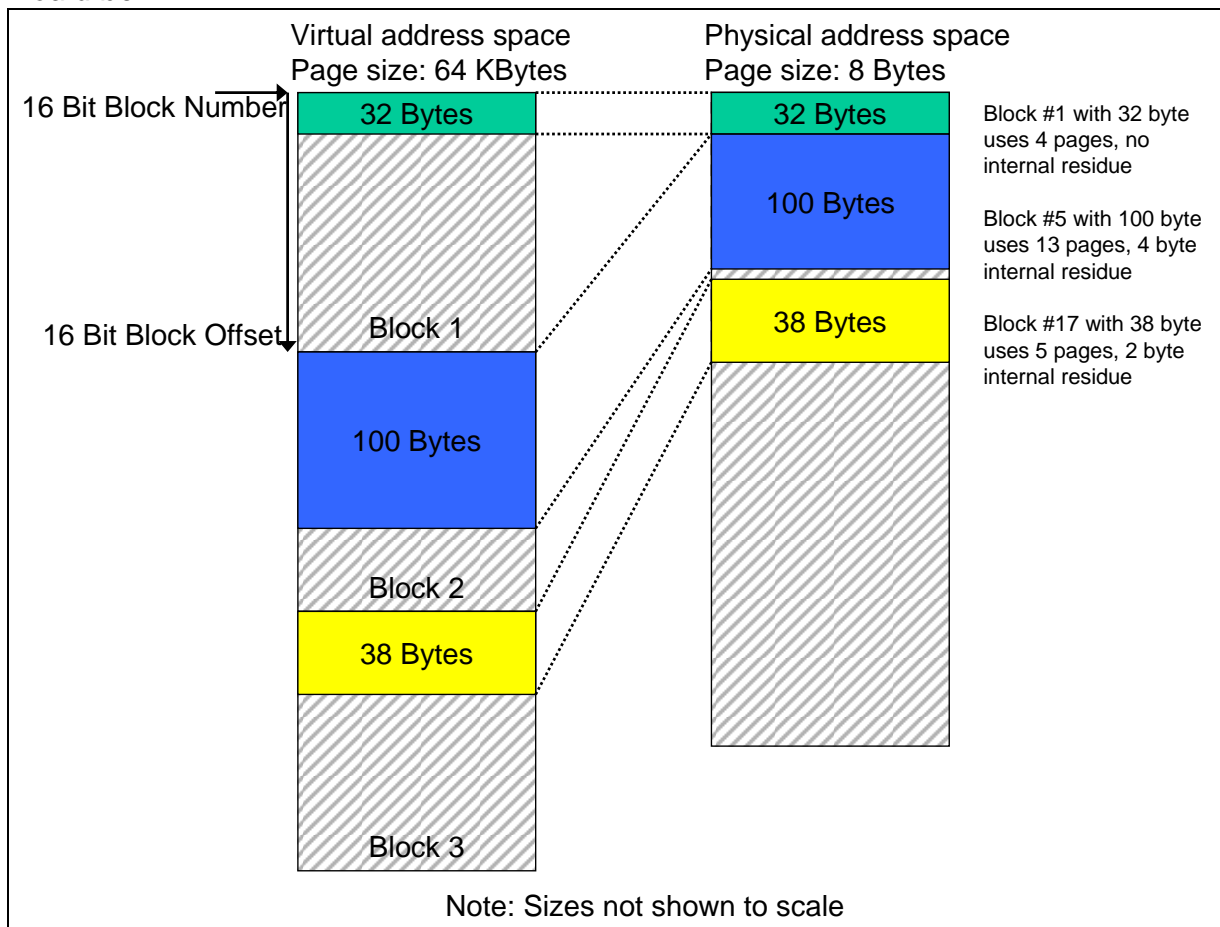
Figure 2: Virtual vs. physical memory layout

[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 LSB's) to this start address (***Figure 3***).*

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.
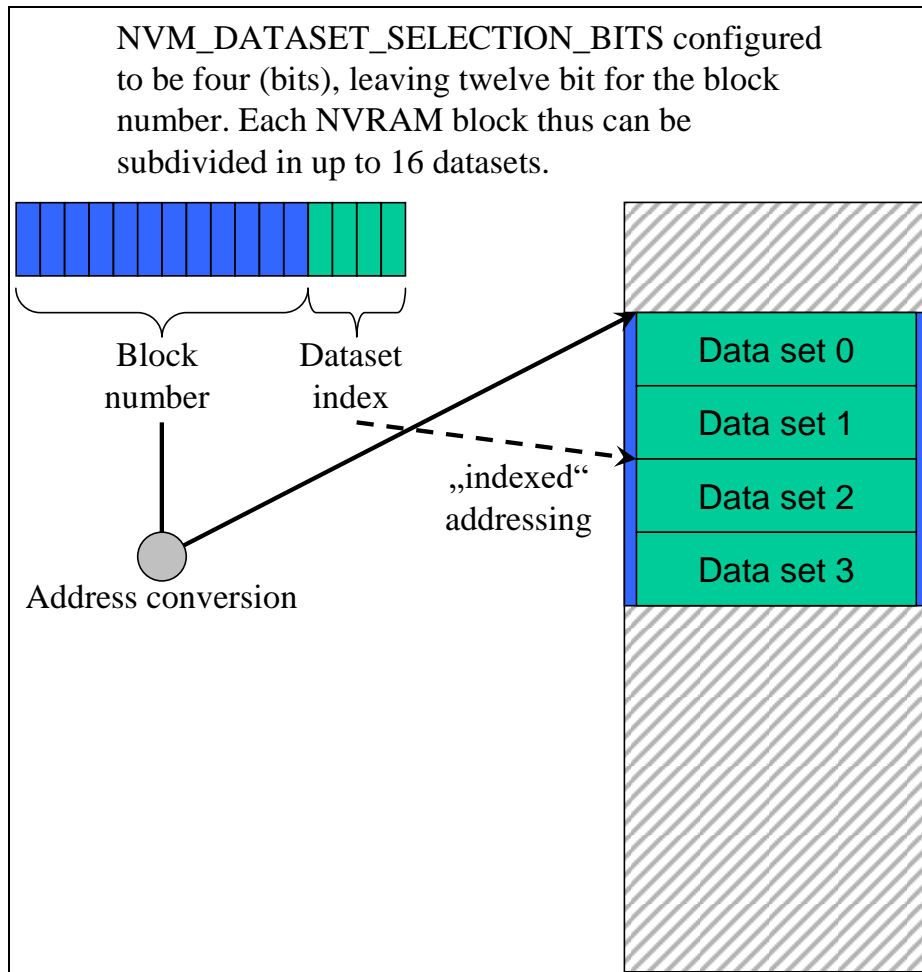
Figure 3: Block number and dataset index

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)

---

[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.

Document ID 286: AUTOSAR_SWS_FlashEEPROMEmulation

[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.

7.1.6 Buffer Alignment

[SWS_Fee_00195] ⌈ The Fee shall align internal buffers to the value of FeeBufferAlignmentValueRef ⌋()

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

7.2 Error classification

Section 7.2 "Error Handling" of the document "General Specification of Basic Software Modules" describes the error handling of the Basic Software in detail. Above all, it 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]⌈

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

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

7.2.2 Runtime Errors

[SWS_Fee_91002]⌈

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 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.

8 API specification

8.1 Imported Types

[SWS_Fee_00084]⌈

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 (obsolete)
	MemIf.h	MemIf_StatusType
Std	Std_Types.h	Std_ReturnType
	Std_Types.h	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	
Kind	Structure	
Elements	implementation specific	
	Type	--
	Comment	--
Description	Configuration data structure of the Fee module.	
Available via	Fee.h	

⌋(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.
Available via	Fee.h

⌋(SRS_BSW_00101)

[SWS_Fee_00168] ⌈ 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. ⌋(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_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_Return-Type	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	

⌋(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 raise the development error `FEE_E_UNINIT`. ⌋(SRS_BSW_00406)

[SWS_Fee_00133] ⌈ 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`. ⌋(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 raise the development error `FEE_E_INVALID_BLOCK_NO`. ⌋(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 raise the development error `FEE_E_INVALID_BLOCK_OFS`. ⌋(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 raise the development error `FEE_E_PARAM_POINTER`. ⌋(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 raise the development error `FEE_E_INVALID_BLOCK_LEN`. ⌋(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 or SWS_Fee_00137 apply, the function `Fee_Read` shall not change the current module status or job result. ⌋(RS_BRF_01048)

[SWS_Fee_00187] ⌈ 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. ⌋(RS_BRF_01076)

8.3.3 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_Return-Type	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	

⌋(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 raise the development error `FEE_E_UNINIT`. ⌋(SRS_BSW_00406)

[SWS_Fee_00144] ⌈ 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`. ⌋(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 raise the development error `FEE_E_INVALID_BLOCK_NO`. ⌋(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 raise the development error `FEE_E_PARAM_POINTER`. ⌋(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](#) or [SWS_Fee_00139](#) apply, the function `Fee_Write` shall not change the current module status or job result. ⌋(RS_BRF_01048)

8.3.4 Fee_Cancel

[SWS_Fee_00089]⌈

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

⌋(SRS_MemHwAb_14031)

[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`. ⌋(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 runtime error `FEE_E_INVALID_CANCEL`. ⌋(SRS_MemHwAb_14031)

8.3.5 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_Status-Type	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

⌋(SRS_BSW_00406)

[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_01048)

[SWS_Fee_00129] ⌈ The function `Fee_GetStatus` shall return `MEMIF_BUSY` if it is currently processing a request from the upper layer. ⌋(RS_BRF_01048)

[SWS_Fee_00074] ⌈ The function `Fee_GetStatus` shall return `MEMIF_BUSY_INTERNAL`, if an internal management operation is currently ongoing. ⌋(RS_BRF_01048)

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]⌈

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_Job-ResultType	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

⌋(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`. ⌋(SRS_BSW_00406)

8.3.7 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

		E_OK: The requested job has been accepted by the module.
Return value	Std_Return-Type	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.	
Available via	Fee.h	

⌟(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_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 raise the development error `FEE_E_UNINIT`. ⌟(SRS_BSW_00406)

[SWS_Fee_00145] ⌈ 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`. ⌟(RS_BRF_01048)

[**SWS_Fee_00192**]⌈ 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. ⌟(SRS_MemHwAb_14028)

[**SWS_Fee_00193**]⌈ The FEE module shall execute the block invalidation request asynchronously within the FEE module's main function.⌟(RS_BRF_01812)

[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 raise the development error `FEE_E_INVALID_BLOCK_NO.` ⌋(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 or SWS_Fee_00145 apply, the function `Fee_InvalidateBlock` shall not change the current module status or job result. ⌋(SRS_MemHwAb_14028)

8.3.8 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.	
Available via	Fee.h	

⌋(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)

8.3.9 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_Return-Type	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.
Available via	Fee.h

⌋(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 raise the development error `FEE_E_UNINIT`. ⌋(SRS_BSW_00406)

[SWS_Fee_00146] ⌜ 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`. ⌟(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`. ⌟(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 raise the development error `FEE_E_INVALID_BLOCK_NO`. ⌟(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 or SWS_Fee_00146 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.
Available via	Fee.h

⌋(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)

[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. ⌋(RS_BRF_01064)

[SWS_Fee_00194] ⌈ 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. ⌋(RS_BRF_01064)

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]⌈

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

⌋(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_UNINIT` 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	Header File	Description
Det_Report-Runtime-Error	Det.h	Service to report runtime errors. If a callout has been configured then this callout shall be called.
MemAcc_-Cancel	Mem Acc.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.
MemAcc_-Compare (draft)	Mem Acc.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_MEM_INCONSISTENT. **Tags:** atp.Status=draft
MemAcc_-Erase	Mem Acc.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_GetJob Result 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_-GetJob-Result	Mem Acc.h	Returns the consolidated job result of the address area referenced by addressAreaId.
MemAcc_-Read	Mem Acc.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_MEM_OK. If the read operation failed, the result of the job is either MEMACC_MEM_FAILED in case of a general error or MEMACC_MEM_ECC_CORRECTED/MEMACC_MEM_ECC_UNCORRECTED in case of a correctable/uncorrectable ECC error.
MemAcc_-Write	Mem Acc.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_MEM_OK. If there was an issue writing the data, the result is MEMACC_MEM_FAILED.

⌋(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	Header File	Description
Det_-ReportError	Det.h	Service to report development errors.
MemAcc_-BlankCheck	Mem Acc.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_MEM_OK, otherwise the result is MEMACC_MEM_INCONSISTENT.

⌋(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_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)

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

[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)

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.
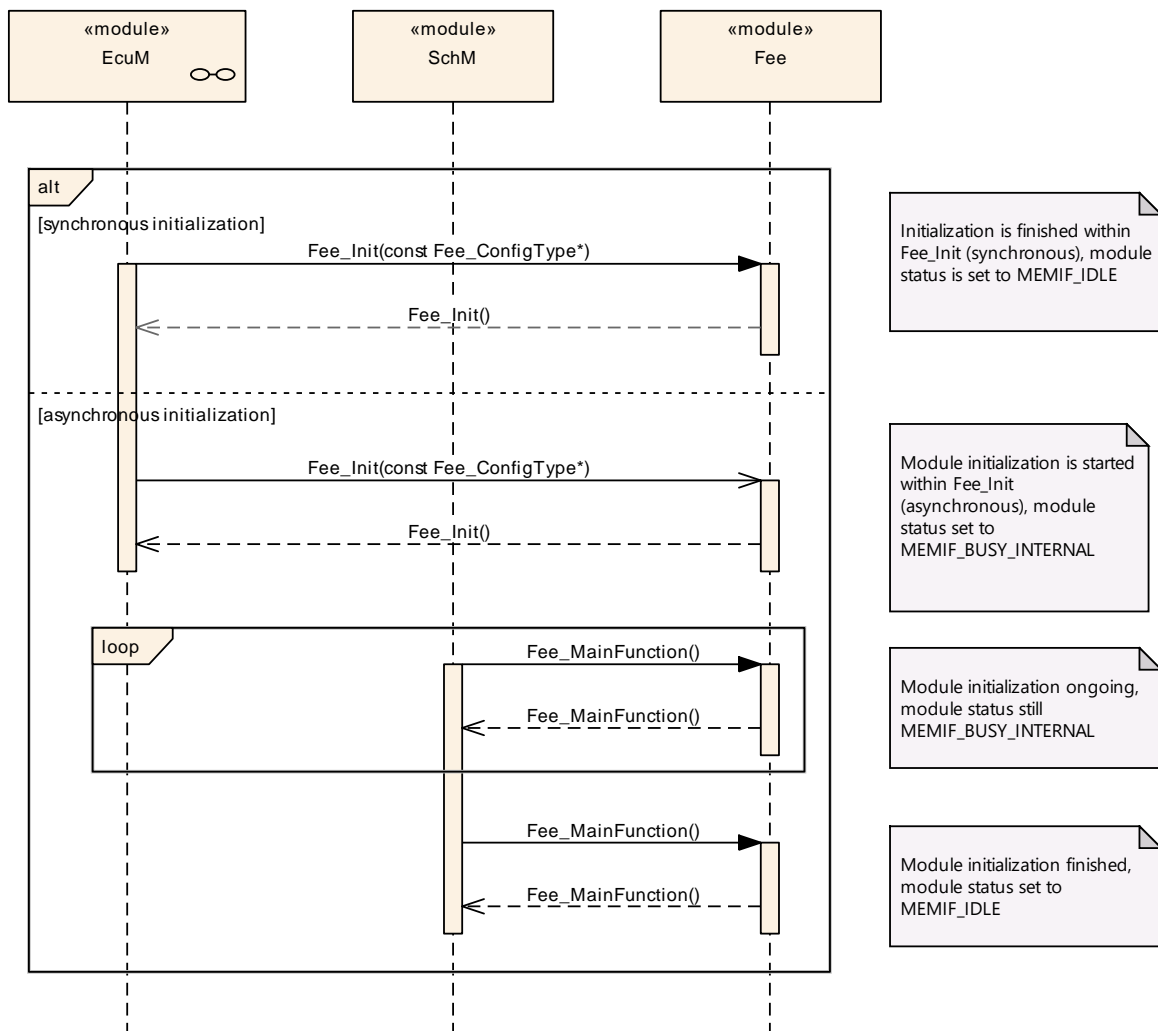
Figure 4: Sequence diagram of Fee_Init

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.
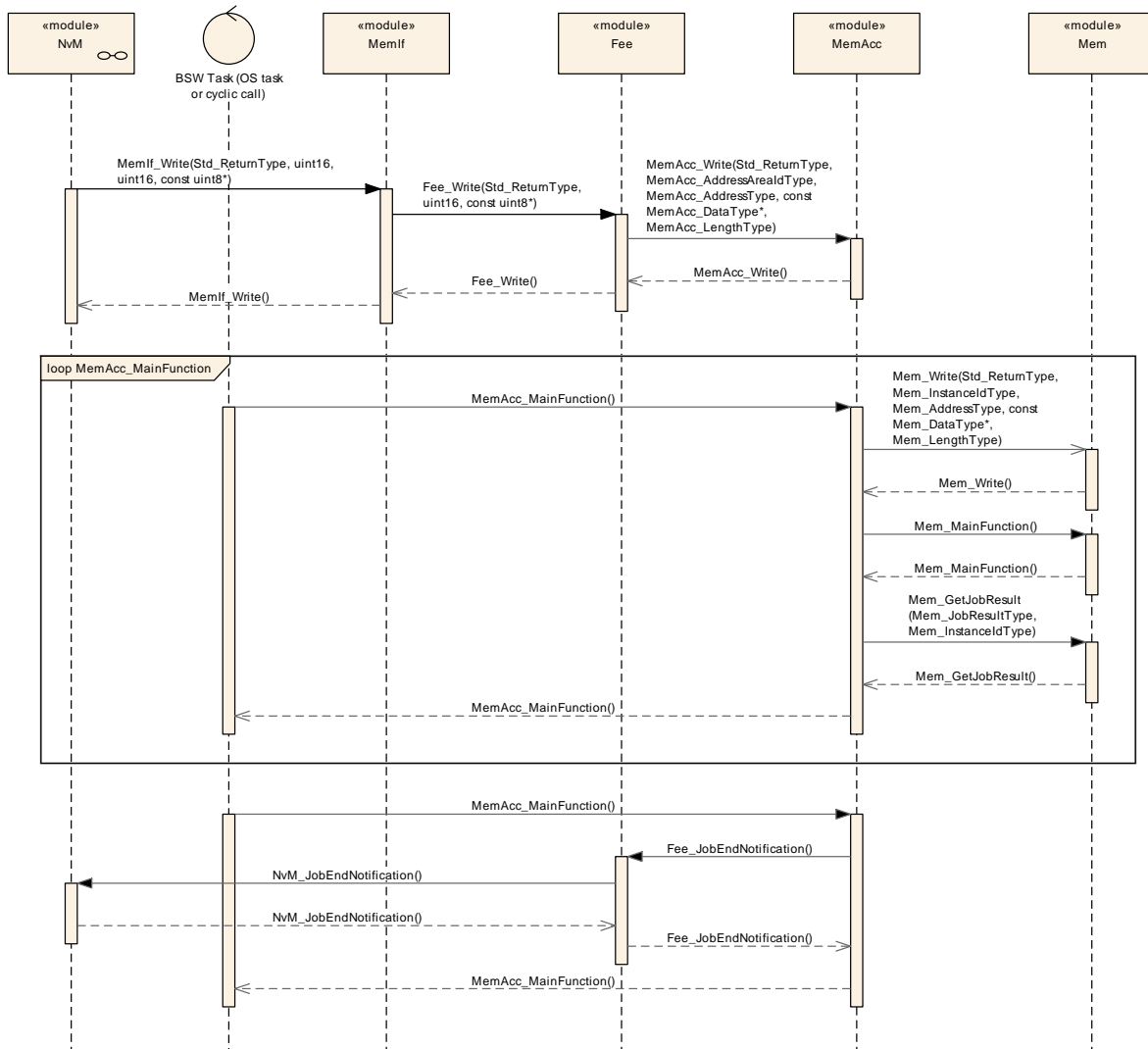
Figure 5: Sequence diagram of Fee_Write

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.
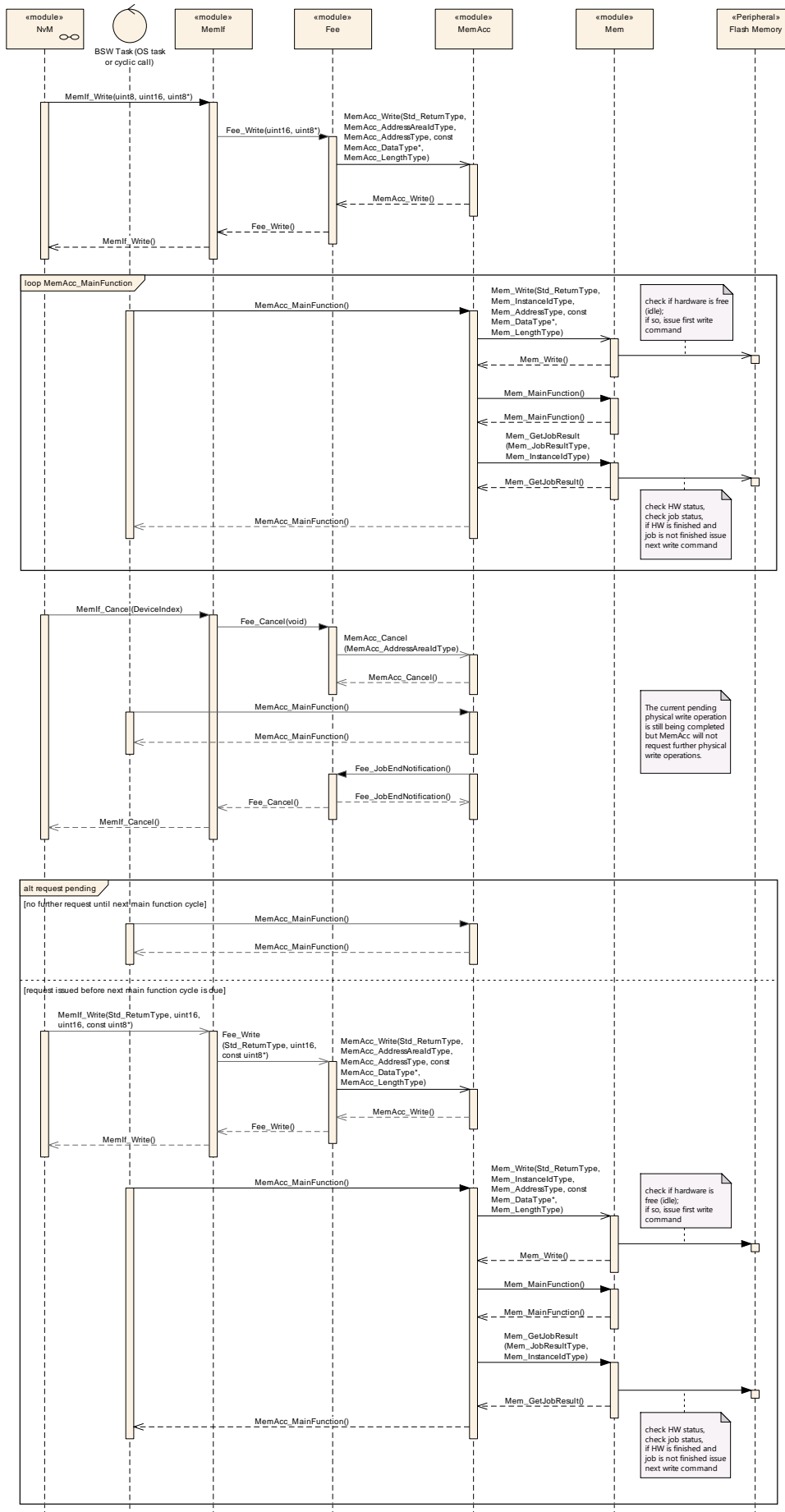
Figure 6: Sequence diagram of Fee_Cancel

10 Configuration specification

10.1 Containers and configuration parameters

The following chapters summarize all configuration parameters. The detailed meaning of the parameters are described in Chapter 7 and Chapter 8.

10.1.1 Fee

SWS Item	[ECUC_Fee_00154]
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**
FeeBlock-Configuration	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.
FeePublished-Information	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.
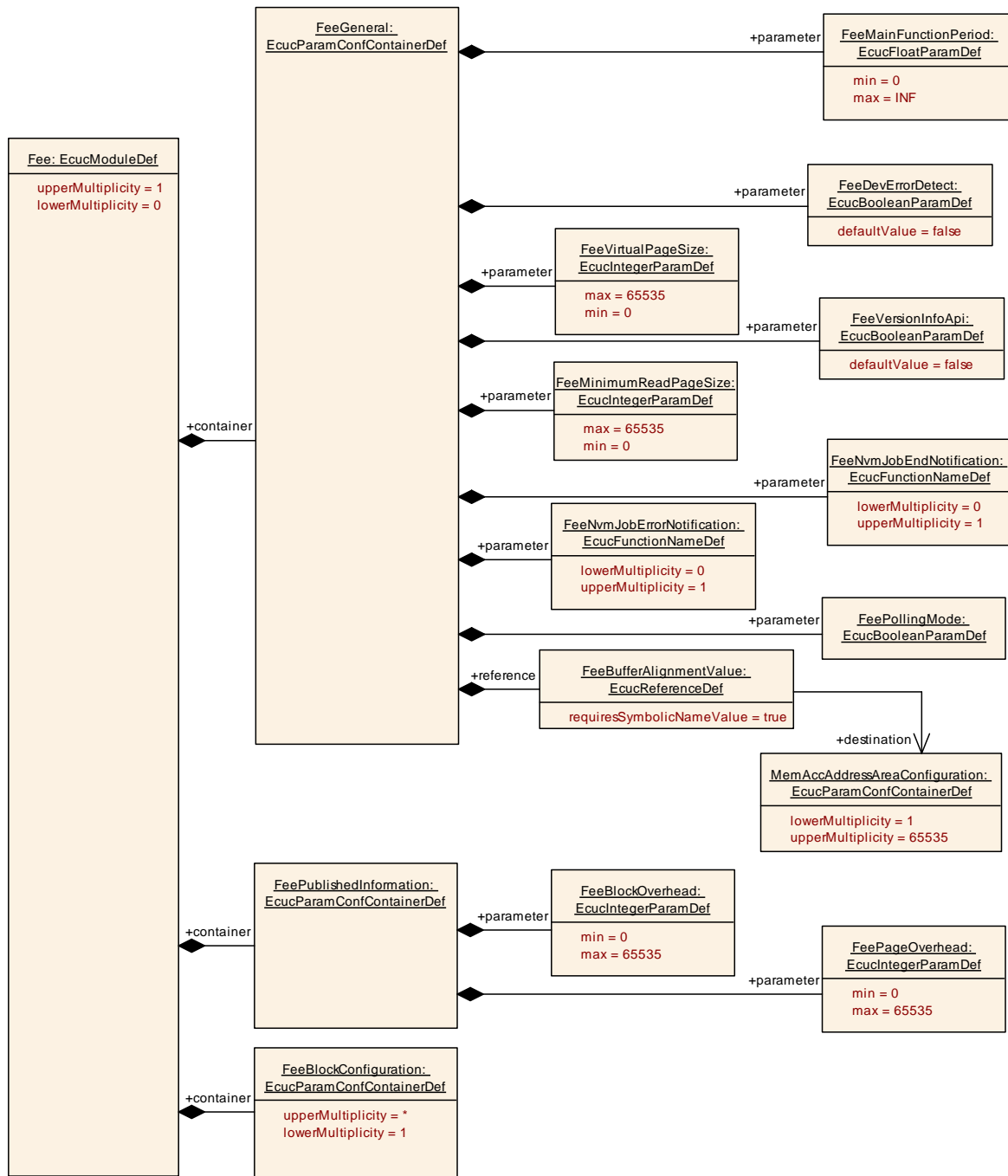
Figure 7: Overview of configuration parameters of Fee

10.1.2 FeeGeneral

SWS Item	[ECUC_Fee_00039]
Container Name	FeeGeneral
Parent Container	Fee
Description	Container for general parameters. These parameters are not specific to a block.

Configuration Parameters

SWS Item	[ECUC_Fee_00111]		
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		

SWS Item	[ECUC_Fee_00153]		
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		

SWS Item	[ECUC_Fee_00156]

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		

SWS Item	[ECUC_Fee_00112]		
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

SWS Item	[ECUC_Fee_00113]
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

SWS Item	[ECUC_Fee_00114]
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		

SWS Item	[ECUC_Fee_00115]		
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		

SWS Item	[ECUC_Fee_00116]		
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		

SWS Item	[ECUC_Fee_00157]		
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		

No Included Containers

10.1.3 FeeBlockConfiguration

SWS Item	[ECUC_Fee_00040]
Container Name	FeeBlockConfiguration
Parent Container	Fee
Description	Configuration of block specific parameters for the Flash EEPROM Emulation module.
Configuration Parameters	

SWS Item	[ECUC_Fee_00150]
Parameter Name	FeeBlockNumber
Parent Container	FeeBlockConfiguration
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]		
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		

SWS Item	[ECUC_Fee_00151]
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		

SWS Item	[ECUC_Fee_00110]		
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		

SWS Item	[ECUC_Fee_00155]
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		

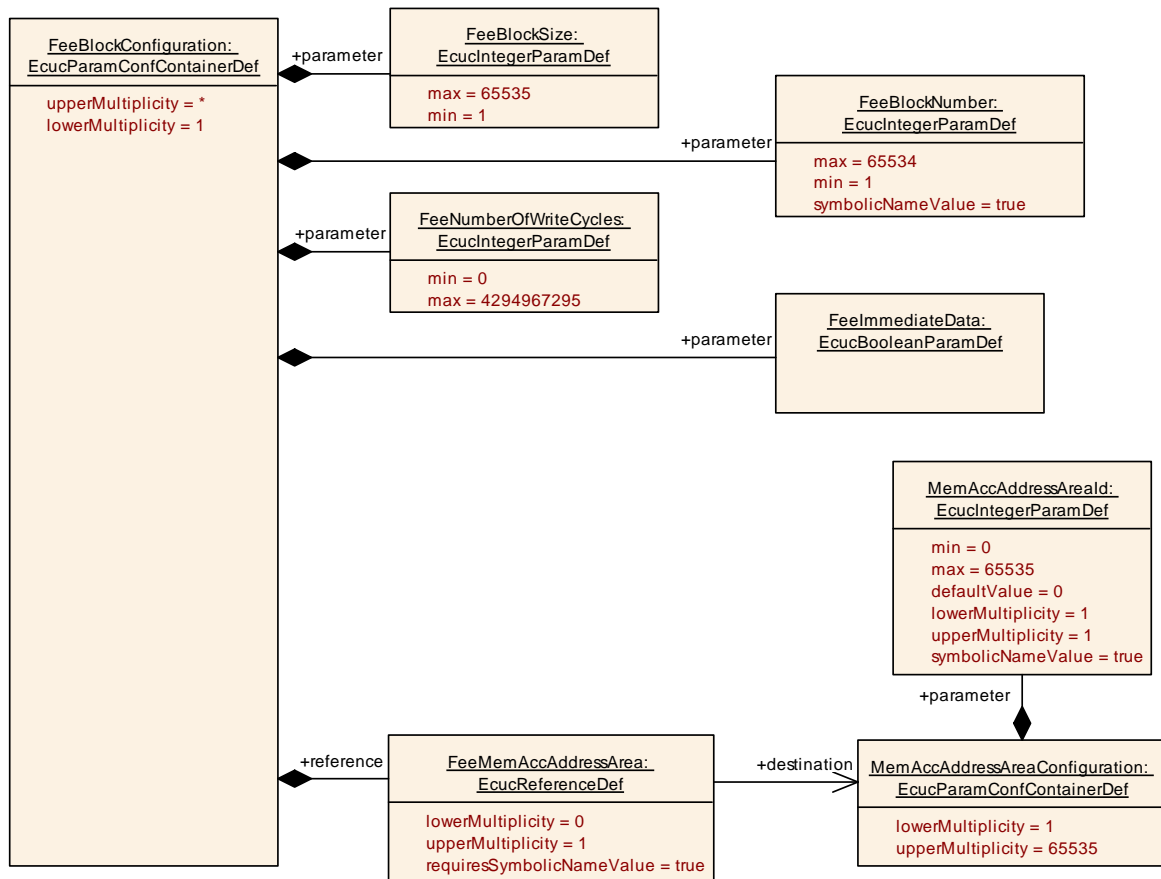| No Included Containers |
|---|

Figure 8: Overview of configuration parameters of FeeBlockConfiguration

10.2 Published Information

10.2.1 FeePublishedInformation

SWS Item	[ECUC_Fee_00043]
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	

SWS Item	[ECUC_Fee_00117]	
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	

SWS Item	[ECUC_Fee_00118]
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		

No Included Containers

11 Not applicable requirements

[SWS_Fee_NA_00999] ⌈ These requirements are not applicable to this specification.

⌋ (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_00323, 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_BSW_00334, SRS_MemHwAb_14017)