Document Title	Specification of FLASH EEPROM Emulation
Document Owner	AUTOSAR GbR
Document Responsibility	AUTOSAR GbR
Document Identification No	286
Document Classification	Standard

Document Version	1.2.1
Document Status	Final
Part of Release	3.1
Revision	0001

Document Change History

Date	Version	Changed by	Change Description
23.06.2008	1.2.1	AUTOSAR Administration	Legal disclaimer revised
19.11.2007	1.2.0	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
14.02.2007	1.1.0	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
23.03.2006	1.0.0	AUTOSAR Administration	Initial release

Page left intentionally blank

Disclaimer

Advice to users of AUTOSAR Specification Documents:

AUTOSAR Specification Documents 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 Specification Documents for illustration purposes only, and they themselves are not part of the AUTOSAR Standard. Neither their presence in such Specification Documents, 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**).

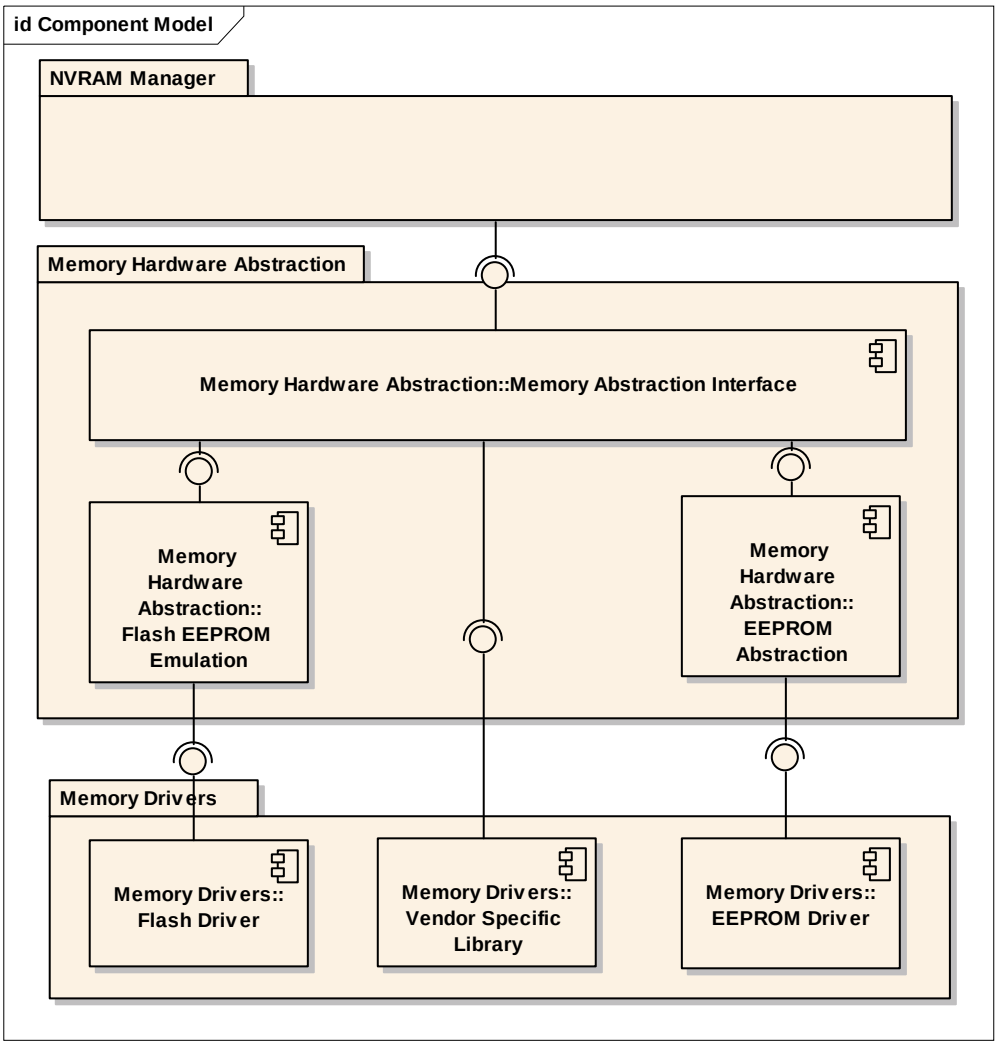

Figure 1: Module overview of memory hardware abstraction layer

FEE001: 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
https://svn2.autosar.org/repos2/22_Releases
AUTOSAR_BasicSoftwareModules.pdf

[2] Layered Software Architecture
https://svn2.autosar.org/repos2/22_Releases
AUTOSAR_LayeredSoftwareArchitecture.pdf

[3] General Requirements on Basic Software Modules
https://svn2.autosar.org/repos2/22_Releases
AUTOSAR_SRS_General.pdf

[4] General Requirements on SPAL
https://svn2.autosar.org/repos2/22_Releases
AUTOSAR_SRS_SPAL_General.pdf

[5] Requirements on Memory Hardware Abstraction Layer
https://svn2.autosar.org/repos2/22_Releases
AUTOSAR_SRS_MemHw_AbstractionLayer.doc

[6] Specification of Development Error Tracer
https://svn2.autosar.org/repos2/22_Releases
AUTOSAR_SWS_DET.pdf

[7] Specification of ECU Configuration
https://svn2.autosar.org/repos2/22_Releases
AUTOSAR_ECU_Configuration.pdf

[8] AUTOSAR Basic Software Module Description Template
https://svn2.autosar.org/repos2/22_Releases
AUTOSAR_BSW_Module_Description.pdf

3.2 Related standards and norms

[8] AUTOSAR Specification of NVRAM Manager
https://svn2.autosar.org/repos2/22_Releases
AUTOSAR_SWS_NVRAM_Manager.doc

[9] Specification of Memory Abstraction Interface
https://svn2.autosar.org/repos2/22_Releases
AUTOSAR_SWS_Mem_AbstractionInterface.pdf

[10] Specification of EEPROM Abstraction
https://svn2.autosar.org/repos2/22_Releases
AUTOSAR_SWS_EEPROM_Abstraction.pdf

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.

5.1 File structure

5.1.1 Code file structure

FEE059: The code file structure shall not be defined within this specification.

5.1.2 Header file structure

FEE002: The file include structure shall be as follows[1]:

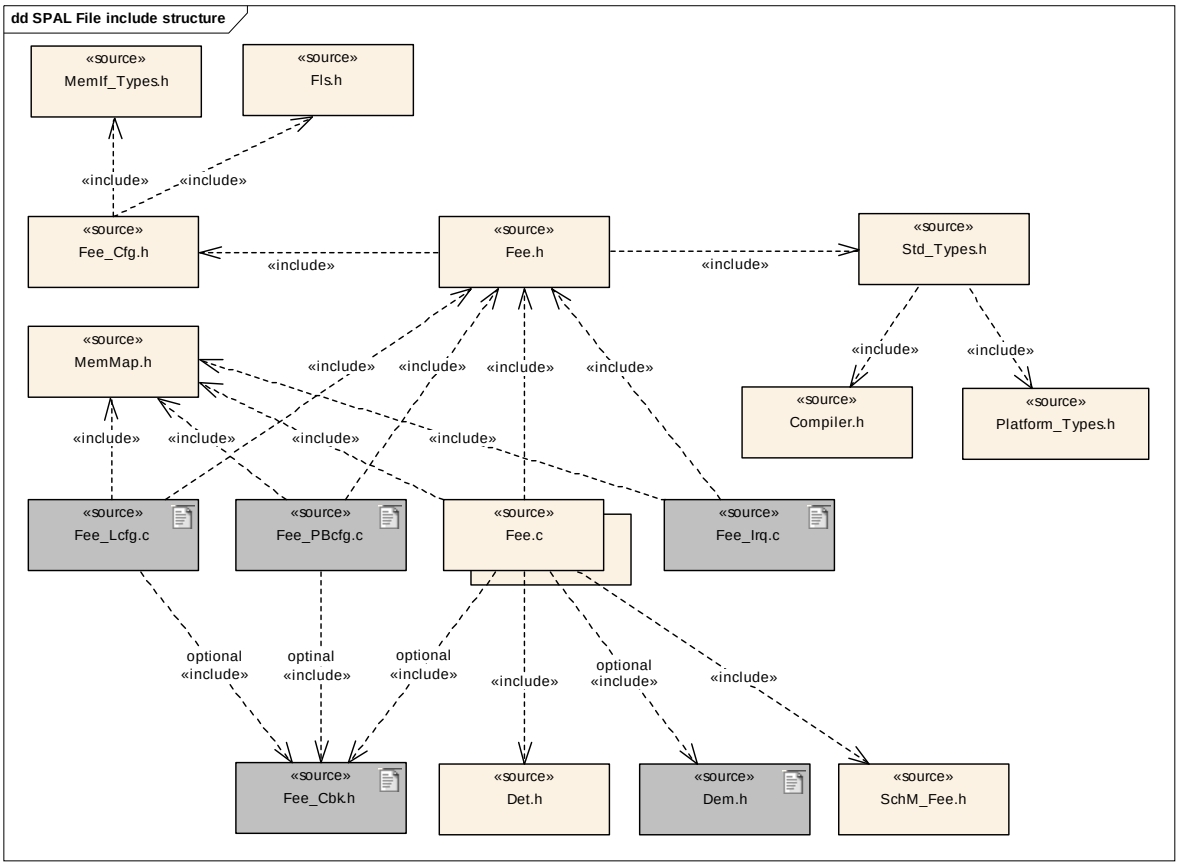

Figure 2: Flash EEPROM Emulation File Include Structure

- `Fee.h` shall include `Fls.h`
- `Fee.h` shall include `Std_Types.h` and `Fee_Cfg.h`
- `Fee_Cfg.h` shall include `MemIf_Types.h`

[1] Files shown in grey are optional and might not be needed for certain implementations and/or configurations.

- `Fee_Lcfg.c` shall include `Fee_Cfg.h`
- `Fee.c` shall include `Fee.h, MemMap.h` and other standard header files (if needed by the implementation).
- `Fee.c` shall include `Fee_Cbk.h`
- Only `Fee.h` shall be included by upper layer modules (Memory Abstraction Interface)

FEE060: The module shall include the `Dem.h` file. By this inclusion, the APIs to report errors as well as the required Event Id symbols are included. This specification defines the name of the Event Id symbols which are provided by XML to the DEM configuration tool. The DEM configuration tool assigns ECU dependent values to the Event Id symbols and publishes the symbols in `Dem_IntErrId.h`.

6 Requirements traceability

Document: General Requirements on Basic Software Modules

Requirement	*Satisfied by*
[BSW00344] Reference to link-time configuration	Not applicable (this module does not provide any post-build parameters)
[BSW00404] Reference to post build time configuration	Not applicable (this module does not provide post build time configuration)
[BSW00405] Reference to multiple configuration sets	Not applicable (this module does not support multiple configuration sets)
[BSW00345] Pre-compile-time configuration	FEE039, FEE040
[BSW159] Tool-based configuration	FEE039, FEE040
[BSW167] Static configuration checking	FEE041
[BSW171] Configurability of optional functionality	Not applicable (no optional functionality)
[BSW170] Data for reconfiguration of AUTOSAR SW-Components	Not applicable (no reconfiguration supported)
[BSW00380] Separate C-File for configuration parameters	Not applicable (no link-time or post build time configuration parameters)
[BSW00381] Separate configuration header file for pre-compile time parameters	FEE002
[BSW00412] Separate H-File for configuration parameters [approved]	Not applicable (no link-time or post build time configuration parameters)
[BSW00383] List dependencies of configuration files	FEE002
[BSW00384] List dependencies to other modules	Chapter 5
[BSW00387] Specify the configuration class of callback function	Chapter 0
[BSW00388] Introduce containers	Chapter 10.1
[BSW00389] Containers shall have names	Chapter 10.1
[BSW00390] Parameter content shall be unique within the module	Chapter 8, Chapter10.2.2, Chapter 10.2.3,
[BSW00391] Parameter shall have unique names	Chapter 8, Chapter10.2.2, Chapter 10.2.3,
[BSW00392] Parameters shall have a type	Chapter 8, Chapter10.2.2, Chapter 10.2.3,
[BSW00393] Parameters shall have a range	Chapter 8, Chapter10.2.2, Chapter 10.2.3,
[BSW00394] Specify the scope of the parameters	Chapter10.2.2
[BSW00395] List the required parameters (per parameter)	Chapter10.2.2
[BSW00396] Configuration classes	Chapter10.2.2
[BSW00397] Pre-compile-time parameters	Chapter10.2.2
[BSW00398] Link-time parameters	Not applicable (no link-time configuration parameters)
[BSW00399] Loadable Post-build time parameters	Not applicable (no post build time configuration parameters)
[BSW00400] Selectable Post-build time parameters	Not applicable (no post build time configuration parameters)
[BSW00402] Published information	Chapter 10.3
[BSW00375] Notification of wake-up reason	Not applicable (this module does not provide wakeup capabilities)
[BSW101] Initialization interface	FEE017

[BSW00416] Sequence of Initialization	Not applicable (requirement on system design, not a single module)
[BSW00406] Check module initialization	Not applicable (no parameters to check during initialization)
[BSW168] Diagnostic Interface of SW components	Not applicable (this module does not provide special diagnostics support)
[BSW00407] Function to read out published parameters	Chapter8.3.9, FEE043
[BSW00423] Usage of SW-C template to describe BSW modules with AUTOSAR Interfaces	Not applicable (this module does not provide an AUTOSAR interface)
[BSW00424] BSW main processing function task allocation	Not applicable (requirement on system design, not on a single module)
[BSW00425] Trigger conditions for schedulable objects	Not applicable (requirement on the BSW module description template)
[BSW00426] Exclusive areas in BSW modules	Not applicable (no exclusive areas defined in this module)
[BSW00427] ISR description for BSW modules	Not applicable (this module does not implement any ISRs)
[BSW00428] Execution order dependencies of main processing functions	Not applicable (only one main processing function in this module)
[BSW00429] Restricted BSW OS functionality access	Not applicable (this module does not use any OS functionality)
[BSW00431] The BSW Scheduler module implements task bodies	Not applicable (requirement on the BSW scheduler)
[BSW00432] Modules should have separate main processing functions for read/receive and write/transmit data path	Not applicable (only one main processing function in this module)
[BSW00433] Calling of main processing functions	Not applicable (requirement on system design, not on a single module)
[BSW00434] The Schedule Module shall provide an API for exclusive areas	Not applicable (requirement on the schedule module - this is not it)
[BSW00336] Shutdown interface	Not applicable (this module does not provide shutdown capabilities)
[BSW00337] Classification of errors	FEE010
[BSW00338] Detection and Reporting of development errors	FEE011, FEE012
[BSW00369] Do not return development error codes via API	FEE045
[BSW00339] Reporting of production relevant error status	Not applicable (no production relevant errors defined for this module)
[BSW00421] Reporting of production relevant error events	Not applicable (no production relevant errors defined for this module)
[BSW00422] Debouncing of production relevant error status	Not applicable (requirement on the DEM, not this module)
[BSW00420] Production relevant error event rate detection	Not applicable (requirement on the DEM, not this module)
[BSW00417] Reporting of Error Events by Non-Basic Software	Not applicable (requirement on non BSW modules)
[BSW00323] API parameter checking	Not applicable

	(no parameter check specified for this module)
[BSW004] Version check	FEE013, FEE043
[BSW00409] Header files for production code error IDs	FEE047
[BSW00385] List possible error notifications	Chapter 8.6
[BSW00386] Configuration for detecting an error	FEE010, FEE011, FEE045
[BSW161] Microcontroller abstraction	Not applicable (requirement on AUTOSAR architecture, not a single module)
[BSW162] ECU layout abstraction	Not applicable (requirement on AUTOSAR architecture, not a single module)
[BSW00324] Do not use HIS I/O Library	Not applicable (architecture decision)
[BSW005] No hard coded horizontal interfaces within MCAL	Not applicable (requirement on AUTOSAR architecture, not a single module)
[BSW00415] User dependent include files	Not applicable (only one user for this module)
[BSW164] Implementation of interrupt service routines	Not applicable (this module does not implement any ISRs)
[BSW00325] Runtime of interrupt service routines	FEE069
[BSW00326] Transition from ISRs to OS tasks	Not applicable (requirement on implementation, not on specification)
[BSW00342] Usage of source code and object code	Not applicable (requirement on AUTOSAR architecture, not a single module)
[BSW00343] Specification and configuration of time	FEE070
[BSW160] Human-readable configuration data	Not applicable (requirement on documentation, not on specification)
[BSW007] HIS MISRA C	Not applicable (requirement on implementation, not on specification)
[BSW00300] Module naming convention	Not applicable (requirement on implementation, not on specification)
[BSW00413] Accessing instances of BSW modules	Requirement can not be implemented in R2.0 timeframe.
[BSW00347] Naming separation of different instances of BSW drivers	Not applicable (requirement on the implementation, not on the specification)
[BSW00305] Self-defined data types naming convention	Chapter 8.2
[BSW00307] Global variables naming convention	Not applicable (requirement on the implementation, not on the specification)
[BSW00310] API naming convention	Chapter 8.3
[BSW00373] Main processing function naming convention	Chapter 8.5.1
[BSW00327] Error values naming convention	FEE010, FEE012
[BSW00335] Status values naming convention	Chapter 8.1
[BSW00350] Development error detection keyword	FEE011, FEE062, FEE039
[BSW00408] Configuration parameter naming convention	Chapter 10.1
[BSW00410] Compiler switches shall have de-	Chapter 10.1

fined values	
[BSW00411] Get version info keyword	Chapter 10.2.3
[BSW00346] Basic set of module files	FEE002
[BSW158] Separation of configuration from implementation	FEE002
[BSW00314] Separation of interrupt frames and service routines	Not applicable (this module does not implement any ISRs)
[BSW00370] Separation of callback interface from API	Chapter 8.4
[BSW00348] Standard type header	Not applicable (requirement on the standard header file)
[BSW00353] Platform specific type header	Not applicable (requirement on the platform specific header file)
[BSW00361] Compiler specific language extension header	Not applicable (requirement on the compiler specific header file)
[BSW00301] Limit imported information	FEE002
[BSW00302] Limit exported information	Not applicable (requirement on the implementation, not on the specification)
[BSW00328] Avoid duplication of code	Not applicable (requirement on the implementation, not on the specification)
[BSW00312] Shared code shall be reentrant	Not applicable (requirement on the implementation, not on the specification)
[BSW006] Platform independency	Not applicable (this is a module of the microcontroller abstraction layer)
[BSW00357] Standard API return type	Chapter 8.3.3, Chapter 8.3.4. Chapter 8.3.8, Chapter 8.3.10
[BSW00377] Module specific API return types	Chapter8.3.6, Chapter 8.3.7
[BSW00304] AUTOSAR integer data types	Not applicable (requirement on implementation, not for specification)
[BSW00355] Do not redefine AUTOSAR integer data types	Not applicable (requirement on implementation, not for specification)
[BSW00378] AUTOSAR boolean type	Not applicable (requirement on implementation, not for specification)
[BSW00306] Avoid direct use of compiler and platform specific keywords	Not applicable (requirement on implementation, not for specification)
[BSW00308] Definition of global data	Not applicable (requirement on implementation, not for specification)
[BSW00309] Global data with read-only constraint	Not applicable (requirement on implementation, not for specification)
[BSW00371] Do not pass function pointers via API	Not applicable (no function pointers in this specification)
[BSW00358] Return type of init() functions	Chapter 8.3.1
[BSW00414] Parameter of init function	Chapter 8.3.1, FEE072
[BSW00376] Return type and parameters of main processing functions	Chapter 8.5.1
[BSW00359] Return type of callback functions	Not applicable (this module does not provide any callback routines)
[BSW00360] Parameters of callback functions	Not applicable (this module does not provide any callback rou-

	tines)
[BSW00329] Avoidance of generic interfaces	Chapter 8.3 (explicit interfaces defined)
[BSW00330] Usage of macros / inline functions instead of functions	Not applicable (requirement on implementation, not for specification)
[BSW00331] Separation of error and status values	FEE010, FEE045
[BSW009] Module User Documentation	Not applicable (requirement on documentation, not on specification)
[BSW00401] Documentation of multiple instances of configuration parameters	Not applicable (all configuration parameters are single instance only)
[BSW172] Compatibility and documentation of scheduling strategy	Not applicable (no internal scheduling policy)
[BSW010] Memory resource documentation	Not applicable (requirement on documentation, not on specification)
[BSW00333] Documentation of callback function context	Not applicable (requirement on documentation, not for specifcia-tion)
[BSW00374] Module vendor identification	FEE043
[BSW00379] Module identification	FEE043
[BSW003] Version identification	FEE043
[BSW00318] Format of module version numbers	FEE043
[BSW00321] Enumeration of module version numbers	Not applicable (requirement on implementation, not for specification)
[BSW00341] Microcontroller compatibility documentation	Not applicable (requirement on documentation, not on specification)
[BSW00334] Provision of XML file	Not applicable (requirement on documentation, not on specification)

Document: General Requirements on SPAL

Requirement	***Satisfied by***
[BSW12263] Object code compatible configuration concept	Not applicable (this module does not provide any post-build parameters)
[BSW12056] Configuration of notification mechanisms	Not applicable (this module does not provide any notification mechanisms)
[BSW12267] Configuration of wake-up sources	Not applicable (this module does not provide any wakeup capabilities)
[BSW12057] Driver module initialization	FEE017
[BSW12125] Initialization of hardware resources	Not applicable (this module has no direct hardware access)
[BSW12163] Driver module de-initialization	Not applicable (this module does not provide any shutdown capabilities)
[BSW12058] Individual initialization of overall registers	Not applicable (this module has no direct hardware access)
[BSW12059] General initialization of overall registers	Not applicable (this module has no direct hardware access)

[BSW12060] Responsibility for initialization of one-time writable registers	Not applicable (this module has no direct hardware access)
[BSW12461] Responsibility for register initialization [approved]	Not applicable (this module has no direct hardware access)
[BSW12462] Provide settings for register initialization [approved]	Not applicable (this module has no direct hardware access)
[BSW12463] Combine and forward settings for register initialization	Not applicable (this module has no direct hardware access)
[BSW12062] Selection of static configuration sets	Not applicable (no selectable of configuration sets)FEE019
[BSW12068] MCAL initialization sequence	Not applicable (this module belongs to the ECU abstraction layer)
[BSW12069] Wake-up notification of ECU State Manager	Not applicable (this module does not provide any wakeup capabilities)
[BSW157] Notification mechanisms of drivers and handlers	Not applicable (this module does not provide any notification mechanisms)
[BSW12155] Prototypes of callback functions	Not applicable (this module does not implement any callback routines)
[BSW12169] Control of operation mode	FEE020
[BSW12063] Raw value mode	Not applicable (this module does not handle or mishandle any data)
[BSW12075] Use of application buffers	Chapter 8.3.3, Chapter 8.3.4
[BSW12129] Resetting of interrupt flags	Not applicable (this module does not implement any ISRs)
[BSW12064] Change of operation mode during running operation	Not applicable (this module has no internal operation mode)
[BSW12448] Behavior after development error detection	FEE068
[BSW12067] Setting of wake-up conditions	Not applicable (this module does not provide any wakeup capabilities)
[BSW12077] Non-blocking implementation	Not applicable (this module does not implement any schedulable services)
[BSW12078] Runtime and memory efficiency	Not applicable (requirement on implementation, not on specification)
[BSW12092] Access to drivers	Not applicable (this module is the flash driver's "manager")
[BSW12265] Configuration data shall be kept constant	Not applicable (no configuration data passed for initialization)
[BSW12264] Specification of configuration items	FEE039, FEE040, FEE043
[BSW12081] Use HIS requirements as input	Not applicable (no corresponding HIS requirements available)

Document: Requirements on Memory Hardware Abstraction Layer

Requirement	***Satisfied by***
BSW14001 Configuration of address alignment	FEE077, FEE078, FEE039
BSW14002 Configuration of number of required write cycles	FEE008, FEE040
BSW14003 Configuration of maximum blocking time	FEE039

BSW14004 Configuration of “immediate” data blocks	FEE040
BSW14026 Don’t use certain block numbers	FEE006
BSW14027 Publish overhead for internal management data per block	FEE043
BSW14005 Virtual linear address space and segmentation	FEE003
BSW14006 Alignment of block erase / write addresses	FEE077, FEE078, FEE024
BSW14007 Alignment of block read addresses	FEE021
BSW14008 Checking block read addresses	FEE038
BSW14009 Conversion of logical to physical addresses	FEE007
BSW14010 Block-wise write service	Chapter 8.3.4
BSW14029 Block-wise read service	Chapter 8.3.3,
BSW14031 Service to cancel an ongoing asynchronous operation	Chapter 8.3.5
BSW14028 Service to invalidate a memory block	Chapter 8.3.8
BSW14012 Spreading of write access	FEE008
BSW14013 Writing of “immediate” data must not be delayed	FEE009
BSW14032 Block-wise erase service for immediate data	FEE066, FEE067, FEE068
BSW14014 Detection of data inconsistencies	FEE023, FEE049, FEE050
BSW14015 Reporting of data inconsistencies	FEE023
BSW14016 Don’t return inconsistent data to the caller	FEE023
BSW14017 Scope of EEPROM Abstraction Layer	Not applicable (this is the FEE modules specification)
BSW14018 Scope of Flash EEPROM Emulation	FEE001

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`.

FEE076: 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.

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.

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.

FEE005: Each configured logical block shall take up an integer multiple of the configured virtual page size (see also chapter 10.2.3,configuration parameter `FeeVirtualPageSize`)..

FEE071: Logical blocks must not overlap each other and must not be contained within one another.

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 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.

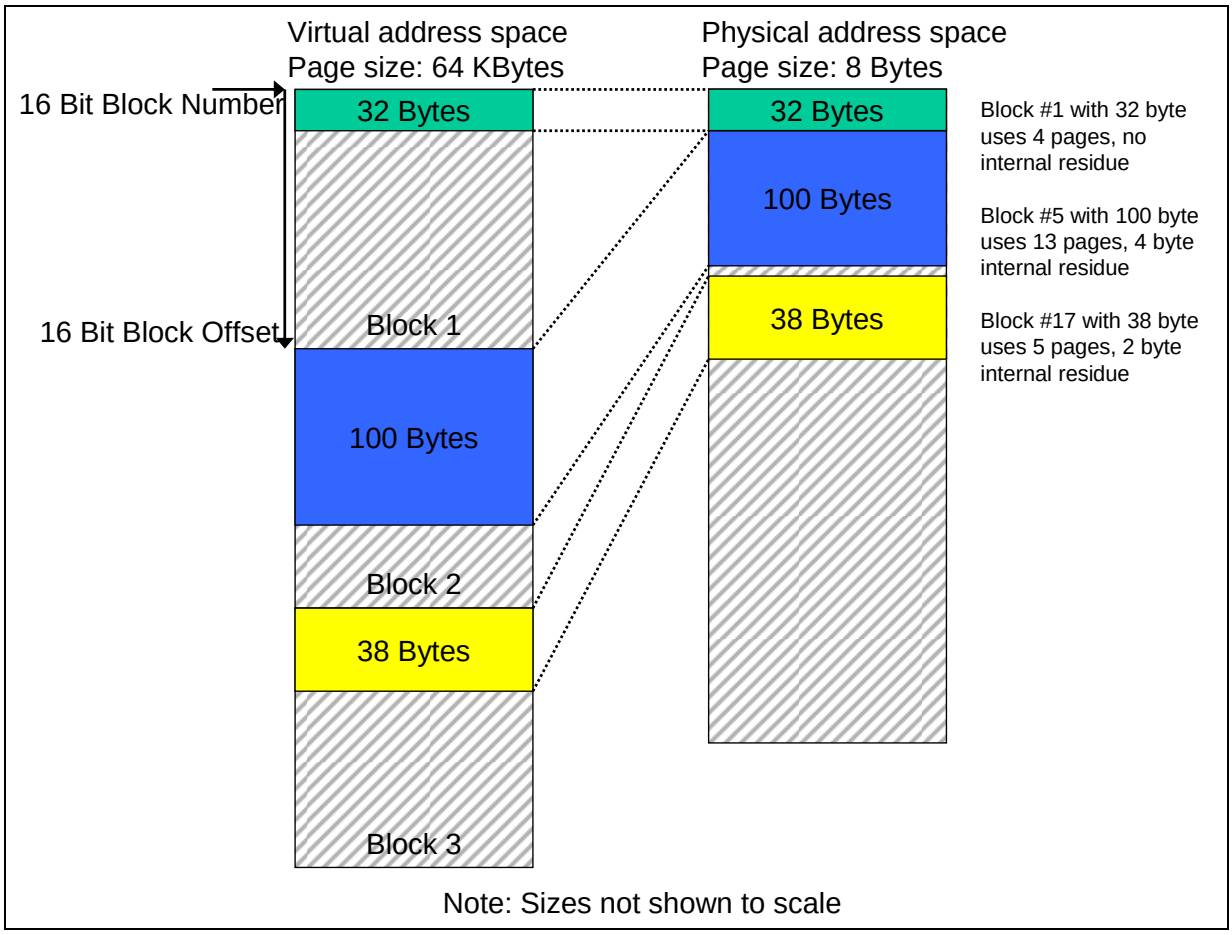

Figure 3: Virtual vs. physical memory layout

FEE006: The block numbers 0x0000 and 0xFFFF shall not be configurable for a logical block.

7.1.2 Address calculation

FEE007: Depending on the implementation of the FEE module and the exact address format used, the functions of the FEE module shall combine the 16bit block number and 16bit address offset to derive the physical flash address needed for the underlying flash driver.

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

FEE100: Only those bits of the 16bit block number, that do not denote a specific dataset or redundant copy shall be used for address calculation.

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

Example:
Dataset information is configured to be encoded in the four LSB's of the 16bit block number (allowing for a maximum of 16 datasets per NVRAM block and a total of 4094 NVRAM blocks). An implementer decides to store all datasets of a NVRAM block directly adjacent and using the length of the block and a pointer to access each dataset. To calculate the start address of the block (the address of the first dataset) she/he uses only the 12 MSB's, to access a specific dataset she/he adds the size of the block multiplied by the dataset index (the four MSB's) to this start address (Figure 4).

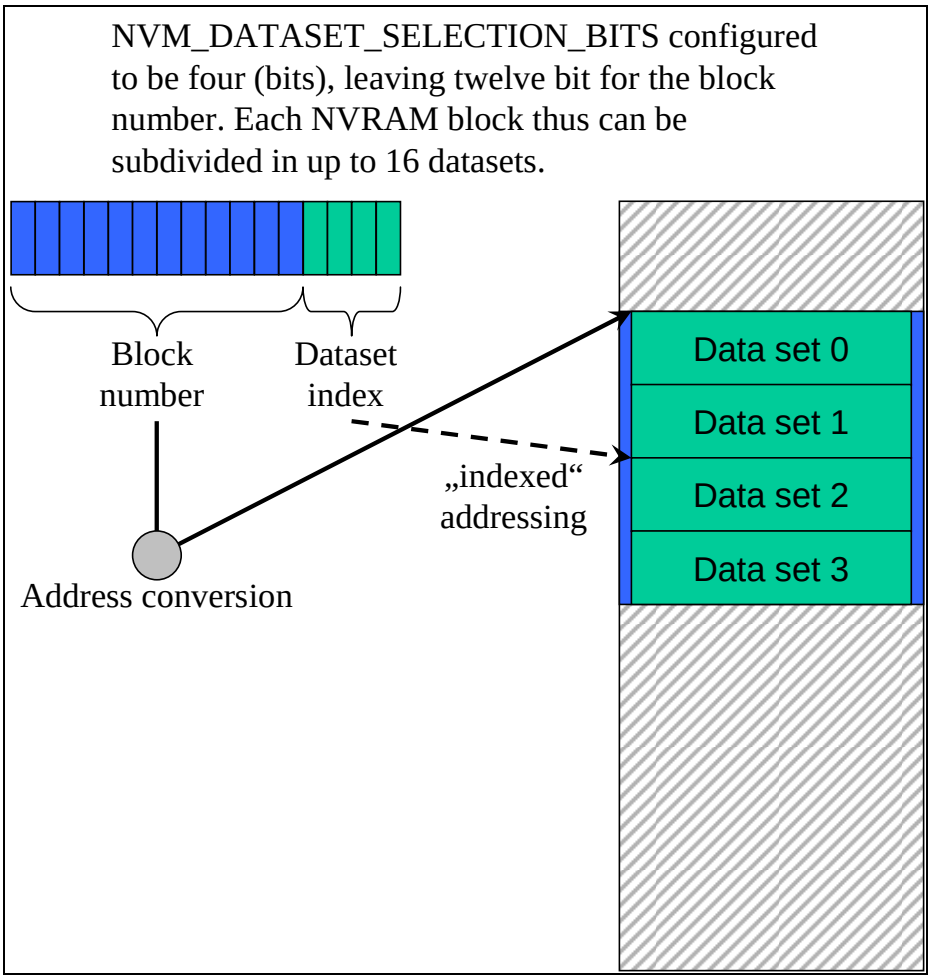

Figure 4: Block number and dataset index

7.1.3 Limitation of erase cycles

FEE102: The configuration of the Fee module shall define the expected number of erase/write cycles for each logical block in the configuration parameter `FeeNumberOfWriteCycles`.

FEE103: If the underlying flash device or device driver does not provide at least the configured number of erase/write cycles per physical memory cell, the FEE module shall provide mechanisms to spread the write access such that the physical device is

not overstressed. This shall also apply to all management data used internally by the FEE module.

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

7.1.4 Handling of "immediate" data

FEE009: Blocks containing immediate data have to be written instantaneously, i.e. the FEE module has to ensure that it can write such blocks without the need to erase the corresponding memory area (e.g. by using pre-erased memory) and that the write request is not delayed by currently running module internal management operations.

Note: An ongoing lower priority read / erase / write or compare job shall be cancelled by the NVRAM manager before immediate data is written. The FEE module has only to ensure that this write 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 cancellation 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 respectively the underlying 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 consistency information

FEE049: The FEE module shall manage for each block the information, whether this block is "correct" from the point of view of the FEE module or not. This consistency information shall only concern the internal handling of the block, not the block's contents.

FEE050: When a block write operation is started, the FEE module shall mark the corresponding block as inconsistent[2]. Upon the successful end of the block write operation, the block shall be marked as consistent (again).

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

7.2 Error classification

FEE047: Values for production code Event Ids are assigned externally by the configuration of the Dem. They are published in the file `Dem_IntErrId.h` and included via `Dem.h`.

FEE048: Development error values are of type uint8.

FEE010: 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 with invalid block number	Development	FEE_E_INVALID_BLOCK_NO	0x02

7.3 Error detection

FEE011: The detection of development errors is configurable (*ON* / *OFF*) at precompile time. The switch `FeeDevErrorDetect` (see Chapter 10) shall activate or deactivate the detection of all development errors.

FEE062: If the `FeeDevErrorDetect` switch is enabled, API parameter checking is enabled. The detailed description of the detected errors can be found in chapter 7.2 and chapter 0.

FEE063: The detection of production code errors cannot be switched off.

FEE012: Additional errors that are detected because of specific implementation and/or specific hardware properties shall be added in the FEE module's implementation documentation. The classification and enumeration shall be compatible with the errors listed above.

[2] 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.

7.4 Error notification

FEE045: Detected development errors shall be reported to the `Det_ReportError` service of the Development Error Tracer (DET) if the pre-processor switch `FeeDevErrorDetect` is set (see Chapter 10).

FEE106: Production errors shall be reported to Diagnostic Event Manager.

7.5 Consistency checks

FEE013: The FEE module's implementation shall check its version numbers against the version information given in the module's header files to ensure compatibility between implementation and configuration of the module.

FEE038: The FEE module shall not implement any kind of parameter checks during runtime. Instead the parameter check of the underlying driver shall be enabled if needed.

Note: The configuration tool shall check all configuration parameters for being within the expected bounds. Also the dependencies between configuration parameters shall be checked by the configuration tool during system generation or during the build process (for details see chapter 10).

8 API specification

8.1 Imported Types

FEE015: The FEE module shall import the types mentioned in FEE084 from the header files `Fls.h`, `Std_Types.h` respectively `MemIf_Types.h`.

FEE016: The types mentioned in FEE084 shall not be changed or extended for a specific FEE module or hardware platform.

FEE084:

Header file	*Imported Type*
MemIf_Types.h	MemIf_StatusType
	MemIf_ModeType
	MemIf_JobResultType
Fls.h	Fls_LengthType
	Fls_AddressType
Std_Types.h	Std_VersionInfoType
	Std_ReturnType

8.2 Type definitions

No local type definitions needed for this module.

8.3 Function definitions

8.3.1 Fee_Init

FEE085:

Service name:	Fee_Init

Syntax:	`void Fee_Init(`
	
	`)`

Service ID[hex]:	0x00
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (in-out):	None
Parameters (out):	None
Return value:	None
Description:	Service to initialize the FEE module.

FEE017: The function `Fee_Init` shall initialize the Flash EEPROM Emulation module.

FEE079: The FEE module's environment shall not call the function `Fee_Init` shall during a running operation of the FEE module.

8.3.2 Fee_SetMode

FEE086:

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 (in-out):	None	
Parameters (out):	None	
Return value:	None	
Description:	Service to call the Fls_SetMode function of the underlying flash driver.	

FEE020: 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.

8.3.3 Fee_Read

FEE087:

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 (in-out):	None	
Parameters (out):	DataBufferPtr	Pointer to data buffer
Return value:	Std_ReturnType	E_OK: The read job was accepted by the underlying memory driver. E_NOT_OK: The read job has not been accepted by the underlying memory driver.
Description:	Service to initiate a read job.	

FEE021: The function `Fee_Read` shall take the block start address and offset and calculate the corresponding memory read address.

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

FEE022: The function `Fee_Read` shall 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`.

FEE073: The FEE module shall execute the job of the function `Fee_Read` asynchronously within the FEE module's main function.

8.3.4 Fee_Write

FEE088:

Service name:	Fee_Write	
Syntax:	`Std_ReturnType Fee_Write( uint16 BlockNumber, 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 (in-out):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: The write job was accepted by the underlying memory driver. E_NOT_OK: The write job has not been accepted by the underlying memory driver.
Description:	Service to initiate a write job.	

FEE024: 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.

FEE025: The function `Fee_Write` shall 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`.

FEE026: The FEE module shall execute the write job of the function `Fee_Write` asynchronously within the FEE module's main function.

8.3.5 Fee_Cancel

FEE089:

Service name:	Fee_Cancel

Syntax:	`void Fee_Cancel(`
	
	`)`

Service ID[hex]:	0x04
Sync/Async:	Asynchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (in-out):	None
Parameters (out):	None
Return value:	None
Description:	Service to call the cancel function of the underlying flash driver.

FEE080: The function `Fee_Cancel` shall call the cancel function of the underlying flash driver.

FEE081: The function `Fee_Cancel` shall reset the FEE module's internal variables to make the module ready for a new job request.

Note: The function `Fee_Cancel` and the cancel function of the underlying flash driver 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 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).

8.3.6 Fee_GetStatus

FEE090:

Service name:	Fee_GetStatus	

Syntax:	`MemIf_StatusType Fee_GetStatus(`
	
	`)`	

Service ID[hex]:	0x05	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	None	
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	MemIf_StatusType	MEMIF_UNINIT: The underlying flash driver has not been initialized. MEMIF_IDLE: The underlying flash driver is currently idle. MEMIF_BUSY: The underlying flash driver is currently busy. MEMIF_BUSY_INTERNAL: The FEE module is busy with internal management operations.
Description:	Service to call the GetStatus function of the underlying flash driver.	

FEE034: : If no internal operation is currently ongoing, the function Fee_GetStatus shall call the “GetStatus” function of the underlying flash driver and pass its return value back to the caller

FEE074: The function Fee_GetStatus shall return MEMIF_BUSY_INTERNAL, if an internal operation is currently ongoing. In this case the “GetStatus” function of the underlying driver shall not be called

8.3.7 Fee_GetJobResult

FEE091:

Service name:	Fee_GetJobResult	

Syntax:	`MemIf_JobResultType Fee_GetJobResult(`
	
	`)`	

Service ID[hex]:	0x06	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	None	
Parameters (in-out):	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_CANCELLED: The last job has been cancelled (which means it failed). MEMIF_JOB_FAILED: The last read/erase/write/compare job 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 call the GetJobResult function of the underlying flash driver.	

FEE035: The function `Fee_GetJobResult` shall call the “GetJobResult” function of the underlying flash driver and pass the return value back to the caller.

8.3.8 Fee_InvalidateBlock

FEE092:

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 (in-out):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: The job was accepted by the underlying memory driver E_NOT_OK: The job has not been accepted by the underlying memory driver
Description:	Service to invalidate a logical block.	

FEE036: The function `Fee_InvalidateBlock` shall take the block number and calculate the corresponding memory block address.

FEE037: The function `Fee_InvalidateBlock` shall invalidate block <BlockNumber> by either calling the erase function of the underlying device driver or changing some module internal management information accordingly.

Note: This 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 is not be further detailed by this specification.

8.3.9 Fee_GetVersionInfo

FEE093:

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

FEE064: The function `Fee_GetVersionInfo` shall return the version information of the FEE module. The version information includes:

- Module Id
- Vendor Id
- Vendor specific version numbers (BSW00407).

FEE065: The function `Fee_GetVersionInfo` shall be pre-compile time configurable On/Off by the configuration parameter `FeeVersionInfoApi`.

FEE082: If source code for caller and callee of the function `Fee_GetVersionInfo` is available, the FEE module should realize this function as a macro. The FEE module should define this macro in the module's header file.

8.3.10 Fee_EraseImmediateBlock

FEE094:

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 (in-out):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: The job was accepted by the underlying memory driver E_NOT_OK: The job has not been accepted by the underlying memory driver.
Description:	Service to erase a logical block.	

FEE066: The function `Fee_EraseImmediateBlock` shall take the block number and calculate the corresponding memory block address.

FEE067: 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.

FEE068: If development error detection for the FEE module is enabled, the function `Fee_EraseImmediateBlock` shall check whether the addressed logical block is configured as containing immediate data (configuration parameter `FeeImmediateData == TRUE`). If not, the function `Fee_EraseImmediateBlock` shall report the error code `FEE_E_INVALID_BLOCK_NO` to the DET, shall not erase the addressed logical block and shall return `E_NOT_OK`.

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.

8.4 Call-back notifications

This is a list of functions provided for lower layer modules.

FEE069: The FEE module shall provide function prototypes of the callback functions in the file `Fee_Cbk.h`

Note: Depending on the implementation of the modules making up the NV memory stack, callback routines provided and/or invoked by the FEE module may be called on interrupt level. The FEE module providing those routines therefore has to make sure that their runtime is reasonably short, i.e. since callbacks may be propagated upward through several software layers.

Note: 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

FEE095:

Service name:	Fee_JobEndNotification

Syntax:	`void Fee_JobEndNotification(`
	
	`)`

Service ID[hex]:	0x10
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (in-out):	None
Parameters (out):	None
Return value:	None
Description:	Service to report the FEE module the successful end of an asynchronous operation.

FEE051: The underlying flash driver shall call the function `Fee_JobEndNotification` to report the successful end of an asynchronous operation.

FEE052: The function `Fee_JobEndNotification` shall perform any necessary block management operations and shall call the corresponding callback routine of the upper layer module.

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

8.4.2 Fee_JobErrorNotification

FEE096:

Service name:	Fee_JobErrorNotification

Syntax:	`void Fee_JobErrorNotification(`
	
	`)`

Service ID[hex]:	0x11
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (in-out):	None
Parameters (out):	None
Return value:	None
Description:	Service to report the FEE module the failure of an asynchronous operation.

FEE053: The underlying flash driver shall call the function `Fee_JobErrorNotification` to report the failure of an asynchronous operation.

FEE054: The function `Fee_JobErrorNotification` shall perform any necessary block management and error handling operations and shall call the corresponding callback routine of the upper layer module.

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

FEE097:

Service name:	Fee_MainFunction

Syntax:	`void Fee_MainFunction(`
	
	`)`

Service ID[hex]:	0x12
Timing:	ON_PRE_CONDITION
Description:	Service to handle the requested read / write / erase jobs respectively the internal management operations.

FEE057: The function `Fee_MainFunction` shall asynchronously handle the requested read / write / erase jobs respectively the internal management operations.

FEE075: 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`, call the job error notification function if configured.

FEE023: 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, the function `Fee_MainFunction` shall set the job result to `MEMIF_BLOCK_INCONSISTENT` and call the error notification routine of the upper layer.

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.

FEE105:

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

8.6.2 Optional Interfaces

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

FEE104:

API function	***Description***
Det_ReportError	Service to report development errors.

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 these kind of interfaces is not fixed because they are configurable.

FEE098:

Service name:	NvM_JobEndNotification

Syntax:	`void NvM_JobEndNotification(`
	
	`)`

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

FEE055: 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 FEE067)

FEE107: The function defined in the configuration parameter `FeeNvmJobEndNotification` shall be callable on interrupt level.

FEE099:

Service name:	NvM_JobErrorNotification

Syntax:	`void NvM_JobErrorNotification(`
	
	`)`

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

FEE056: 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 FEE067)

FEE108: 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 resp. 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.

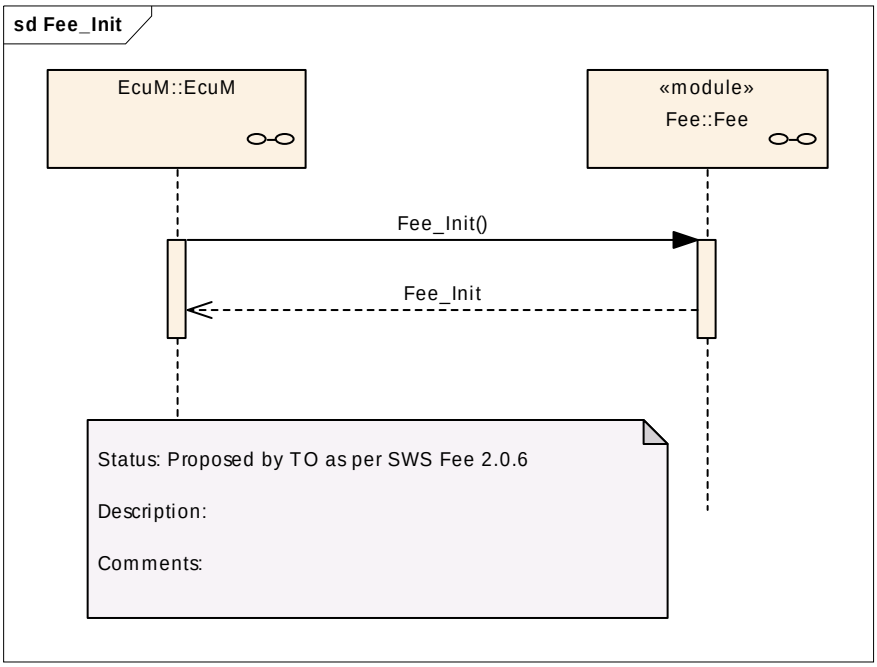

Figure 5: Sequence diagram of "Fee_Init" service

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).

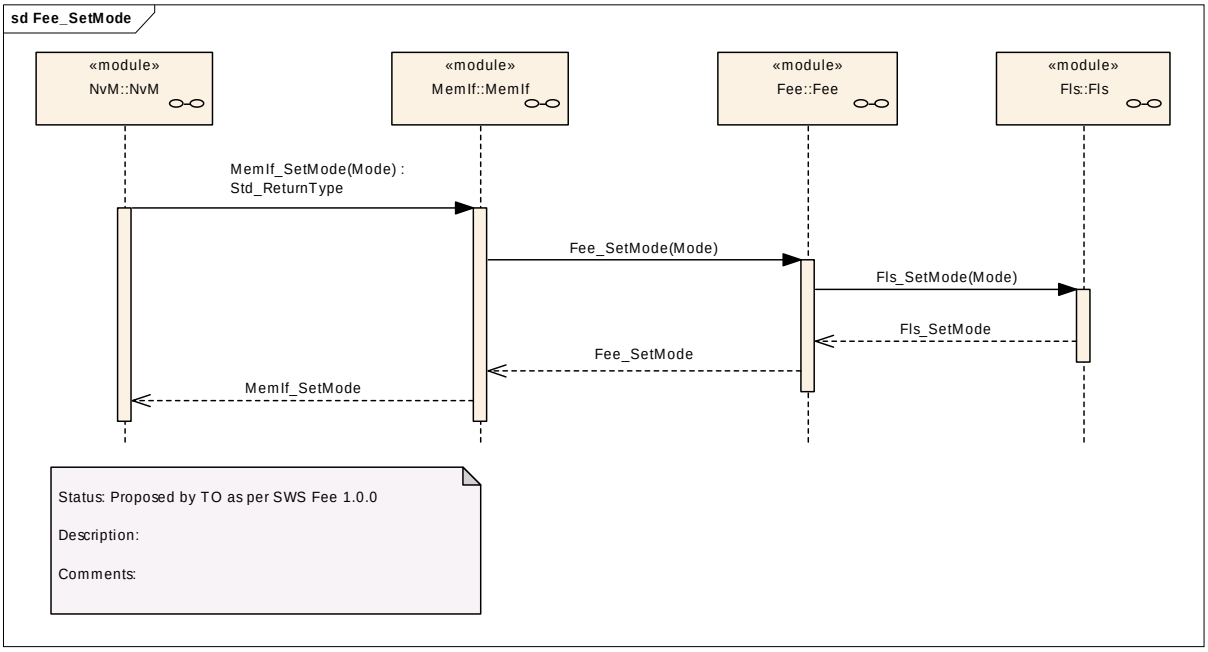

Figure 6: Sequence diagram of the "Fee_SetMode" service

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.

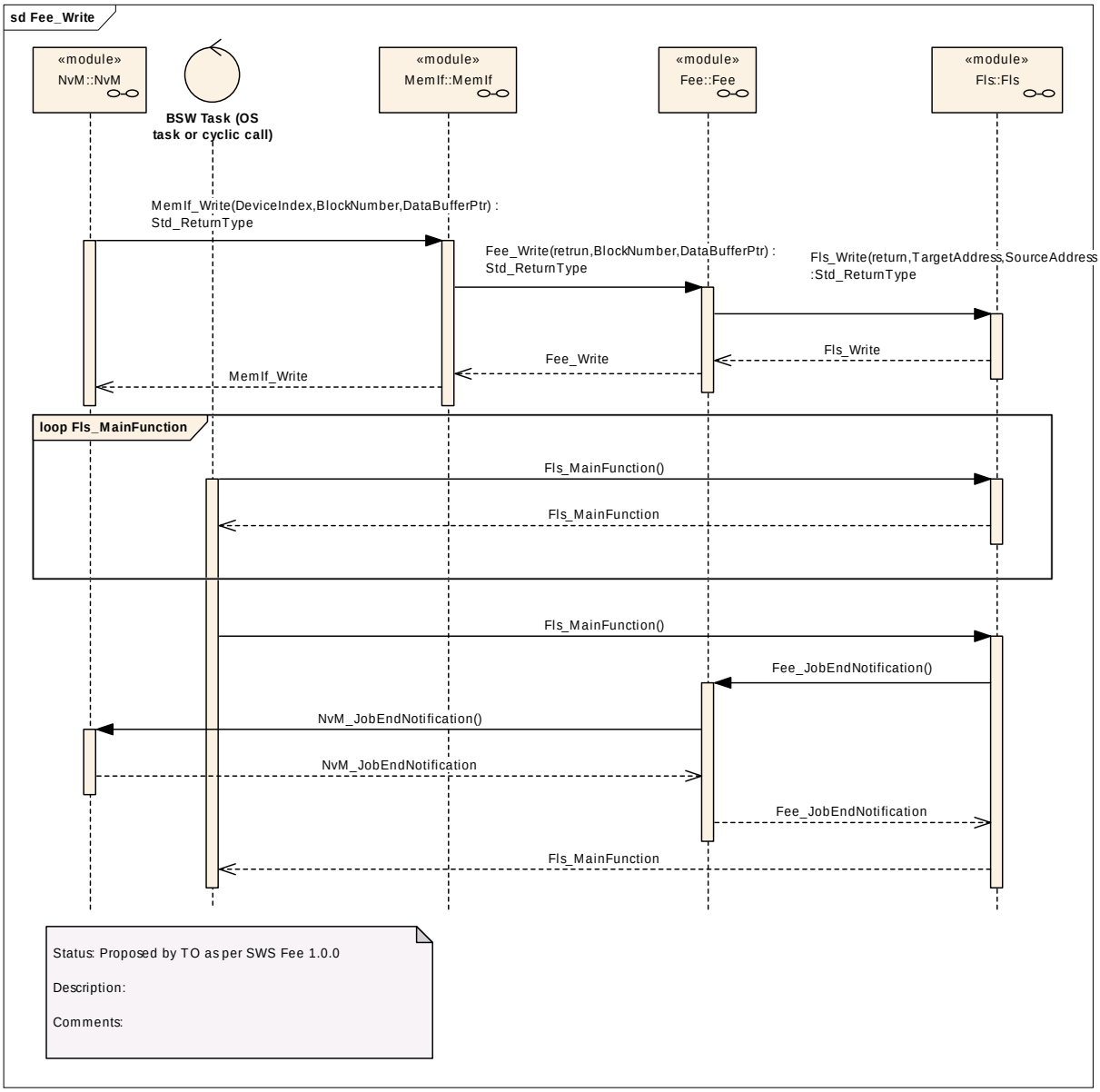

Figure 7: Sequence diagram "Fee_Write"

9.4 Fee_Cancel

The following figure shows as an example the call sequence for a cancelled `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.

10 Configuration specification

10.1 How to read this chapter

In addition to this section, it is highly recommended to read the documents:

- AUTOSAR Layered Software Architecture [2]
- AUTOSAR ECU Configuration Specification [7]
  This document describes the AUTOSAR configuration methodology and the AUTOSAR configuration metamodel in detail.

The following is only a short survey of the topic and it will not replace the ECU Configuration Specification document.

10.1.1 Configuration and configuration parameters

Configuration parameters define the variability of the generic part(s) of an implementation of a module. This means that only generic or configurable module implementation can be adapted to the environment (software/hardware) in use during system and/or ECU configuration.

The configuration of parameters can be achieved at different times during the software process: before compile time, before link time or after build time. In the following, the term “configuration class” (of a parameter) shall be used in order to refer to a specific configuration point in time.

10.1.2 Containers

Containers structure the set of configuration parameters. This means:
- *all* configuration parameters are kept in containers.
- (sub-) containers can reference (sub-) containers. It is possible to assign a multiplicity to these references. The multiplicity then defines the possible number of instances of the contained parameters.

10.1.3 Specification template for configuration parameters

The following tables consist of three sections:
- the general section
- the configuration parameter section
- the section of included/referenced containers

Pre-compile time - specifies whether the configuration parameter shall be of configuration class *Pre-compile time* or not

Label	***Description***
X	The configuration parameter shall be of configuration class *Pre-compile time*.
--	The configuration parameter shall never be of configuration class *Pre-compile time*.

Link time - specifies whether the configuration parameter shall be of configuration class *Link time* or not

Label	*Description*
x	The configuration parameter shall be of configuration class *Link time*.
--	The configuration parameter shall never be of configuration class *Link time*.

Post Build - specifies whether the configuration parameter shall be of configuration class *Post Build* or not

Label	*Description*
x	The configuration parameter shall be of configuration class *Post Build* and no specific implementation is required.
L	*Loadable* - the configuration parameter shall be of configuration class *Post Build* and only one configuration parameter set resides in the ECU.
M	*Multiple* - the configuration parameter shall be of configuration class *Post Build* and is selected out of a set of multiple parameters by passing a dedicated pointer to the init function of the module.
--	The configuration parameter shall never be of configuration class *Post Build*.

10.2 Containers and configuration parameters

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

10.2.1 Variants

No variants specified.

10.2.2 Fee

Module Name	**Fee**
Module Description	Configuration of the Fee (Flash EEPROM Emulation) module.

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.

10.2.3 FeeGeneral

SWS Item	**FEE039 :**		
Container Name	FeeGeneral{FEE_ModuleConfiguration}		
Description	Container for general parameters. These parameters are not specific to a block.		
Configuration Parameters			

SWS Item	**FEE111 :**		
Name	FeeDevErrorDetect {FEE_DEV_ERROR_DETECT}		
Description	Pre-processor switch to enable and disable development error detection. true: Development error detection enabled. false: Development error detection disabled.		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**FEE109 :**		
Name	FeeIndex		
Description	Specifies the InstanceId of this module instance. If only one instance is present it shall have the Id 0.		
Multiplicity	1		
Type	IntegerParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**FEE112 :**		
Name	FeeNvmJobEndNotification {FEE_NVM_JOB_END_NOTIFICATION}		
Description	Mapped to the job end notification routine provided by the upper layer module (NvM_JobEndNotification).		
Multiplicity	1		
Type	FunctionNameDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**FEE113 :**		
Name	FeeNvmJobErrorNotification {FEE_NVM_JOB_ERROR_NOTIFICATION}		
Description	Mapped to the job error notification routine provided by the upper layer module (NvM_JobErrorNotification).		
Multiplicity	1		
Type	FunctionNameDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	

Scope / Dependency	scope: module

SWS Item	**FEE114 :**		
Name	FeePollingMode {FEE_POLLING_MODE}		
Description	Pre-processor switch to enable and disable the polling mode for this module. true: Polling mode enabled. false: Polling mode disabled.		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**FEE115 :**		
Name	FeeVersionInfoApi {FEE_VERSION_INFO_API}		
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	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**FEE116 :**		
Name	FeeVirtualPageSize {FEE_VIRTUAL_PAGE_SIZE}		
Description	The size in bytes to which logical blocks shall be aligned.		
Multiplicity	1		
Type	IntegerParamDef		
Range	0 .. 65535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

No Included Containers

10.2.4 FeeBlockConfiguration

SWS Item	**FEE040 :**
Container Name	FeeBlockConfiguration{FEE_BlockConfiguration}
Description	Configuration of block specific parameters for the Flash EEPROM Emulation module.
Configuration Parameters	

SWS Item	**FEE107 :**
Name	FeeBlockNumber {FEE_BLOCK_NUMBER}
Description	Block identifier (handle). 0x0000 and 0xFFFF shall not be used for block

	numbers (see FEE006). Range: min = 2^NVM_DATA_SELECTION_BITS max = 0xFFFF -2^NVM_DATA_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	IntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 65535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**FEE108 :**		
Name	FeeBlockSize {FEE_BLOCK_SIZE}		
Description	Size of a logical block in bytes.		
Multiplicity	1		
Type	IntegerParamDef		
Range	0 .. 65535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**FEE109 :**		
Name	FeeImmediateData {FEE_IMMEDIATE_DATA}		
Description	Marker for high priority data. true: Block contains immediate data. false: Block does not contain immediate data.		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**FEE110 :**		
Name	FeeNumberOfWriteCycles {FEE_NUMBER_OF_WRITE_CYCLES}		
Description	Number of write cycles required for this block.		
Multiplicity	1		
Type	IntegerParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**FEE106 :**		
Name	FeeDeviceIndex {FEE_DEVICE_INDEX}		
Description	Device index (handle). Range: 0 .. 254 (0xFF reserved for broadcast call to GetStatus function).		
Multiplicity	1		
Type	Reference to FlsGeneral		
ConfigurationClass	***Pre-compile time***	X	All Variants

	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module 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

10.3 Published Information

Published information contains data defined by the implementer of the SW module that does not change when the module is adapted (i.e. configured) to the actual HW/SW environment. It thus contains version and manufacturer information.

The standard common published information like

- vendorId (FEE_VENDOR_ID),
- moduleId (FEE_MODULE_ID),
- arMajorVersion (FEE_AR_MAJOR_VERSION),
- arMinorVersion (FEE_ AR_MINOR_VERSION),
- arPatchVersion (FEE_ AR_PATCH_VERSION),
- swMajorVersion (FEE_SW_MAJOR_VERSION),
- swMinorVersion (FEE_ SW_MINOR_VERSION),
- swPatchVersion (FEE_ SW_PATCH_VERSION),
- vendorApiInfix (FEE_VENDOR_API_INFIX)

is provided in the BSW Module Description Template (see [8], Figure 4.1 and Figure 7.1). Additional published parameters are listed below if applicable for this module.

10.3.1 FeePublishedInformation

SWS Item	**FEE043 :**
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	**FEE117 :**		
Name	FeeBlockOverhead {FEE_BLOCK_OVERHEAD}		
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	IntegerParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	--	
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**FEE070 :**
Name	FeeMaximumBlockingTime {FEE_MAXIMUM_BLOCKING_TIME}
Description	The maximum time the FEE module's API routines shall be blocked (delayed) by internal operations. Note: Internal operations in that case means operations that are not explicitly invoked from the upper layer module but need to be handled for proper operation of this module or the underlying memory driver.
Multiplicity	1

Type	FloatParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	--	
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**FEE118 :**		
Name	FeePageOverhead {FEE_PAGE_OVERHEAD}		
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	IntegerParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	--	
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

No Included Containers

11 Changes during SWS Improvements by Technical Office

11.1 Deleted SWS Items

SWS Item	*Rationale*

11.2 Replaced SWS Items

SWS Item of Release 1	*replaced by SWS Item*	*Rationale*
FEE004	FEE077, FEE078	Made requirement atomic.
FEE033	FEE080, FEE081	Made requirement atomic.
FEE061	FEE100, FEE101	Made requirement atomic.
FEE008	FEE102, FEE103	Made requirement atomic.

11.3 Changed SWS Items

Many requirements have been changed to improve understandability without changing the technical contents.

SWS Item	*Rationale*
FEE059	RfC #17181: Replaced with appropriate text from SWS template.

11.4 Added SWS Items

SWS Item	*Rationale*
FEE079	Caveat Fee_Init
FEE082	Hint from Fee_GetVersionInfo
FEE083	Caveat Fee_EraseImmediateBlock
FEE084	UML Model linking of imported types
FEE085	UML Model linking of Fee_Init
FEE086	UML Model linking of Fee_SetMode
FEE087	UML Model linking of Fee_Read
FEE088	UML Model linking of Fee_Write
FEE089	UML Model linking of Fee_Cancel
FEE090	UML Model linking of Fee_GetStatus
FEE091	UML Model linking of Fee_GetJobResult
FEE092	UML Model linking of Fee_InvalidateBlock
FEE093	UML Model linking of Fee_GetVersionInfo
FEE094	UML Model linking of Fee_EraseImmediateBlock
FEE095	UML Model linking of Fee_JobEndNotification
FEE096	UML Model linking of Fee_JobErrorNotification
FEE097	UML Model linking of Fee_MainFunction
FEE098	UML Model linking of NvM_JobEndNotification
FEE099	UML Model linking of NvM_JobErrorNotification
FEE104	UML Model linking of optional interfaces
FEE105	UML Model linking of mandatory interfaces

FEE106	RfC18948: Statement from SWS Template added.