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

Document Version	2.0.0
Document Status	Final
Part of Release	4.0
Revision	3

Document Change History

Date	Version	Changed by	Change Description
03.11.2011	2.0.0	AUTOSAR Administration	• DET errors added / removed • Handling of internal management operations detailed • Module short name changed • Consistency checking reformulated
13.10.2010	1.4.0	AUTOSAR Administration	• Inter-module checks clarified (FEE013) • Sequence diagram for Fee_Cancel replaced for generated one • Naming in FEE150_Conf corrected to NVM_DATASET_SELECTION_BITS • Sequence diagram for Fee_Init extended • Handling of internal management operations refined (FEE022, FEE025, FEE173, FEE174, FEE183) • Inter module checks detailed (FEE013) • NvM_Cbk.h added to file include structure (FEE002) • Ranges for FeeBlockNumber (FEE150_Conf) and FeeBlockSize (FEE148_Conf) adjusted • Initialization might not be finished within Fee_Init, state machine adapted accordingly (FEE120, FEE168, FEE169) • Handling of internal management operations refined (FEE170 .. FEE182 e.a.)
03.12.2009	1.3.0	AUTOSAR Administration	• Configuration variants clarified • Job result handling re-formulated • Range of configuration parameters restricted • Legal disclaimer revised
23.06.2008	1.2.1	AUTOSAR Administration	Legal disclaimer revised

Document Change History			
Date	**Version**	**Changed by**	**Change Description**
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

Disclaimer

Advice for users

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**).

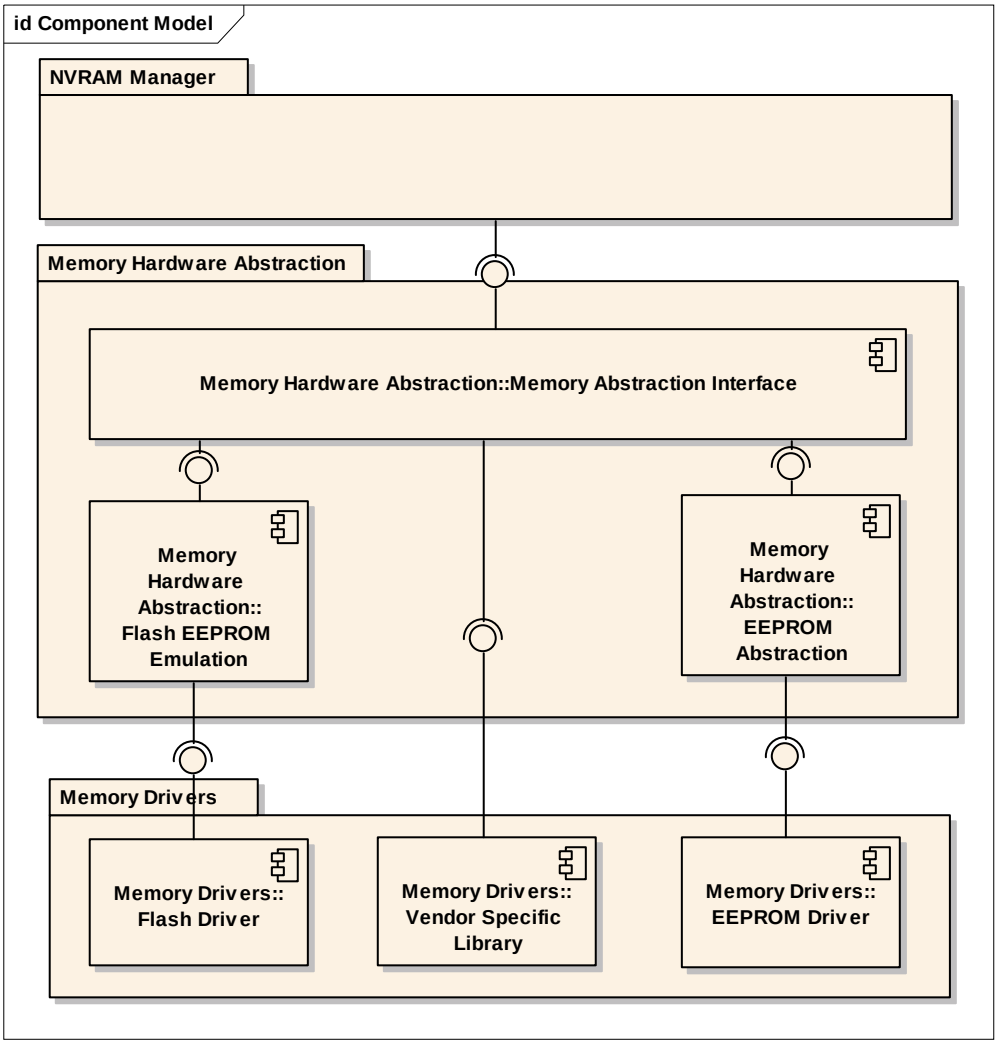

Figure 1: Module overview of memory hardware abstraction layer

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

2 Acronyms and abbreviations

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

Abbreviation / Acronym:	***Description:***
EA	EEPROM Abstraction
EEPROM	Electrically Erasable and Programmable ROM (Read Only Memory)
FEE	Flash EEPROM Emulation
LSB	Least significant bit / byte (depending on context). Here, “bit” is meant.
MemIf	Memory Abstraction Interface
MSB	Most significant bit / byte (depending on context). Here, “bit” is meant.
NvM	NVRAM Manager
NVRAM	Non-volatile RAM (Random Access Memory)
NVRAM block	Management unit as seen by the NVRAM Manager
(Logical) block	Smallest writable / erasable unit as seen by the modules user. Consists of one or more virtual pages.
Virtual page	May consist of one or several physical pages to ease handling of logical blocks and address calculation.
Internal residue	Unused space at the end of the last virtual page if the configured block size isn’t an integer multiple of the virtual page size (see Figure 3)).
Virtual address	Consisting of 16 bit block number and 16 bit offset inside the logical block.
Physical address	Address information in device specific format (depending on the underlying EEPROM driver and device) that is used to access a logical block.
Dataset	Concept of the NVRAM manager: A user addressable array of blocks of the same size. E.g. could be used to provide different configuration settings for the CAN driver (CAN IDs, filter settings, …) to an ECU which has otherwise identical application software (e.g. door module).
Redundant copy	Concept of the NVRAM manager: Storing the same information twice to enhance reliability of data storage.

3 Related documentation

3.1 Input documents

[1] List of Basic Software Modules
AUTOSAR_TR_BSWModuleList.pdf

[2] Layered Software Architecture
AUTOSAR_EXP_LayeredSoftwareArchitecture..pdf

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

[4] General Requirements on SPAL
AUTOSAR_SRS_SPALGeneral.pdf

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

[6] Specification of Development Error Tracer
AUTOSAR_SWS_DevelopmentErrorTracer.pdf

[7] Specification of ECU Configuration
AUTOSAR_TPS_ECUConfiguration.pdf

[8] Basic Software Module Description Template
AUTOSAR_TPS_BSWModuleDescriptionTemplate.pdf

3.2 Related standards and norms

[8] AUTOSAR Specification of NVRAM Manager
AUTOSAR_SWS_NVRAMManager.doc

[9] Specification of Memory Abstraction Interface
AUTOSAR_SWS_MemoryAbstractionInterface.pdf

[10] Specification of EEPROM Abstraction
AUTOSAR_SWS_EEPROMAbstraction.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:

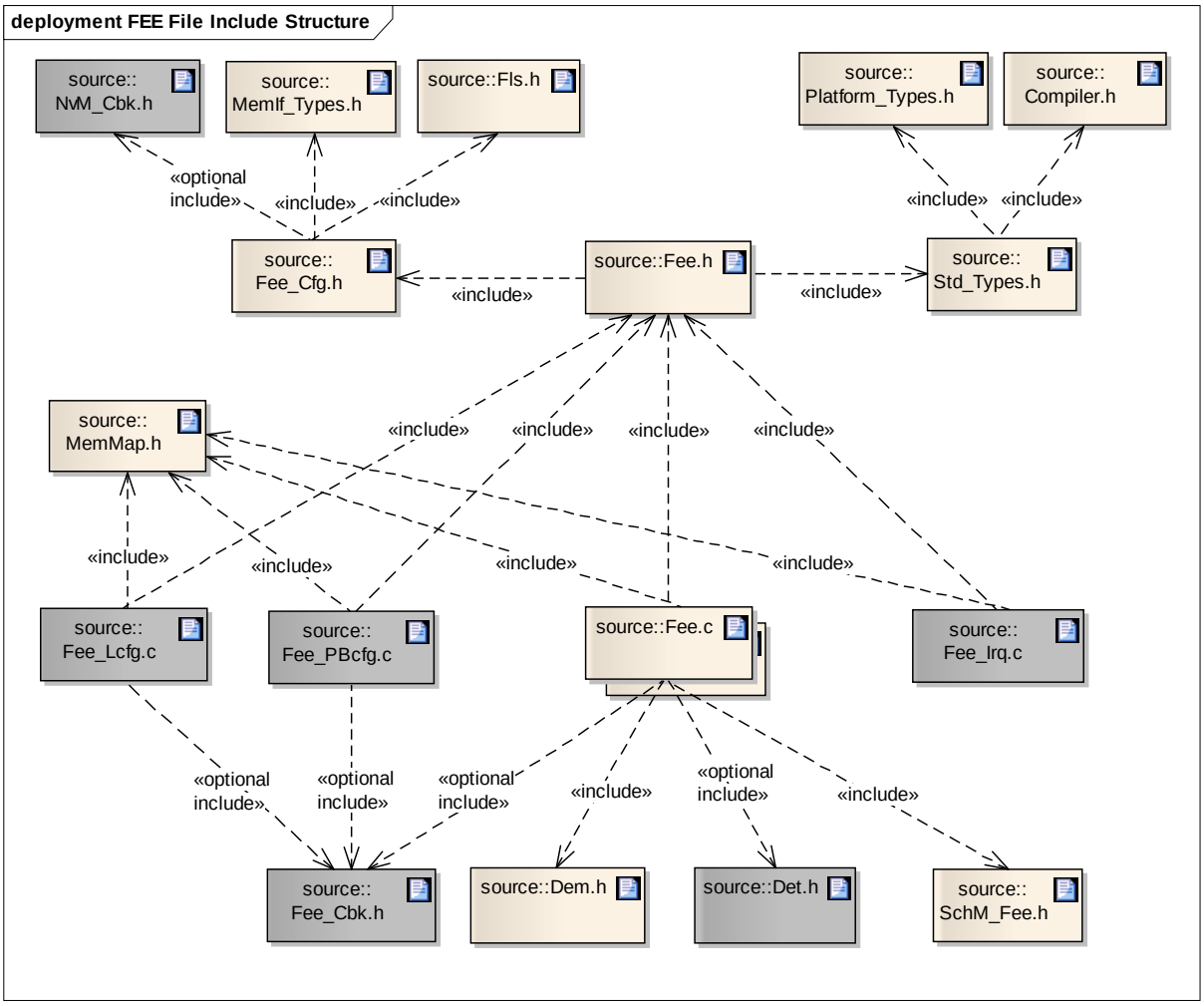

Figure 2: Flash EEPROM Emulation File Include Structure ⌋(BSW167, BSW00383, BSW00346, BSW158, BSW00301)

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

[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

Requirement	Satisfied by
-	FEE170
-	FEE166
-	FEE128
-	FEE136
-	FEE026
-	FEE141
-	FEE132
-	FEE133
-	FEE096
-	FEE182
-	FEE081
-	FEE091
-	FEE052
-	FEE160
-	FEE066
-	FEE175
-	FEE060
-	FEE145
-	FEE059
-	FEE073
-	FEE140
-	FEE134
-	FEE054
-	FEE184
-	FEE080
-	FEE057
-	FEE144
-	FEE065
-	FEE022
-	FEE064
-	FEE180
-	FEE131
-	FEE155
-	FEE086
-	FEE100
-	FEE138
-	FEE181

-	FEE095
-	FEE056
-	FEE173
-	FEE025
-	FEE074
-	FEE179
-	FEE147
-	FEE093
-	FEE183
-	FEE165
-	FEE172
-	FEE099
-	FEE036
-	FEE164
-	FEE168
-	FEE137
-	FEE129
-	FEE037
-	FEE055
-	FEE105
-	FEE158
-	FEE156
-	FEE084
-	FEE159
-	FEE135
-	FEE177
-	FEE176
-	FEE035
-	FEE130
-	FEE104
-	FEE016
-	FEE157
-	FEE169
-	FEE162
-	FEE067
-	FEE090
-	FEE143
-	FEE174
-	FEE075
-	FEE106
-	FEE098
-	FEE034

-	FEE163
-	FEE097
-	FEE142
-	FEE146
-	FEE082
-	FEE178
-	FEE048
-	FEE171
-	FEE063
-	FEE139
BSW00301	FEE002
BSW00325	FEE069
BSW00327	FEE010
BSW00331	FEE045, FEE010
BSW00337	FEE010
BSW00338	FEE011
BSW00346	FEE002
BSW00350	FEE062, FEE011
BSW00369	FEE045
BSW00383	FEE002
BSW00386	FEE045, FEE011, FEE010
BSW004	FEE013
BSW00406	FEE124, FEE123, FEE126, FEE125, FEE120, FEE122, FEE121, FEE127, FEE010
BSW00409	FEE047
BSW101	FEE085
BSW12057	FEE085
BSW12169	FEE020
BSW12448	FEE068
BSW14001	FEE071, FEE076, FEE005
BSW14002	FEE103, FEE102
BSW14005	FEE076
BSW14006	FEE024
BSW14007	FEE021
BSW14008	FEE021
BSW14009	FEE007
BSW14010	FEE088
BSW14012	FEE103, FEE102
BSW14013	FEE009
BSW14014	FEE049, FEE023, FEE154, FEE153
BSW14015	FEE023
BSW14016	FEE023
BSW14026	FEE006

BSW14028	FEE092
BSW14029	FEE087
BSW14031	FEE089
BSW14032	FEE094
BSW158	FEE002
BSW167	FEE002
BWS00300	FEE999
BWS00302	FEE999
BWS00304	FEE999
BWS00306	FEE999
BWS00307	FEE999
BWS00308	FEE999
BWS00309	FEE999
BWS00312	FEE999
BWS00314	FEE999
BWS00321	FEE999
BWS00323	FEE999
BWS00324	FEE999
BWS00326	FEE999
BWS00328	FEE999
BWS00330	FEE999
BWS00333	FEE999
BWS00334	FEE999
BWS00336	FEE999
BWS00339	FEE999
BWS00341	FEE999
BWS00342	FEE999
BWS00344	FEE999
BWS00347	FEE999
BWS00348	FEE999
BWS00353	FEE999
BWS00355	FEE999
BWS00359	FEE999
BWS00360	FEE999
BWS00361	FEE999
BWS00371	FEE999
BWS00375	FEE999
BWS00378	FEE999
BWS00380	FEE999
BWS00398	FEE999
BWS00399	FEE999
BWS00400	FEE999

BWS00401	FEE999
BWS00404	FEE999
BWS00405	FEE999
BWS00412	FEE999
BWS00415	FEE999
BWS00416	FEE999
BWS00417	FEE999
BWS00420	FEE999
BWS00421	FEE999
BWS00422	FEE999
BWS00423	FEE999
BWS00424	FEE999
BWS00425	FEE999
BWS00426	FEE999
BWS00427	FEE999
BWS00428	FEE999
BWS00429	FEE999
BWS00431	FEE999
BWS00432	FEE999
BWS00433	FEE999
BWS00434	FEE999
BWS005	FEE999
BWS006	FEE999
BWS007	FEE999
BWS009	FEE999
BWS010	FEE999
BWS12056	FEE999
BWS12058	FEE999
BWS12059	FEE999
BWS12060	FEE999
BWS12062	FEE999
BWS12063	FEE999
BWS12064	FEE999
BWS12067	FEE999
BWS12068	FEE999
BWS12069	FEE999
BWS12077	FEE999
BWS12078	FEE999
BWS12081	FEE999
BWS12092	FEE999
BWS12125	FEE999
BWS12129	FEE999

BWS12155	FEE999
BWS12163	FEE999
BWS12263	FEE999
BWS12265	FEE999
BWS12267	FEE999
BWS12461	FEE999
BWS12462	FEE999
BWS12463	FEE999
BWS14003	FEE999
BWS14017	FEE999
BWS157	FEE999
BWS160	FEE999
BWS161	FEE999
BWS164	FEE999
BWS168	FEE999
BWS170	FEE999
BWS171	FEE999
BWS172	FEE999

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	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 08.5.1
[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, Chapter 10.2
[BSW00391] Parameter shall have unique names	Chapter 8, Chapter 10.2
[BSW00392] Parameters shall have a type	Chapter 8, Chapter 10.2
[BSW00393] Parameters shall have a range	Chapter 8, Chapter 10.2
[BSW00394] Specify the scope of the parameters	Chapter 8, Chapter 10.2
[BSW00395] List the required parameters (per parameter)	Chapter 8, Chapter 10.2
[BSW00396] Configuration classes	Chapter 8, Chapter 10.2
[BSW00397] Pre-compile-time parameters	Chapter 8, Chapter 10.2
[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	FEE085
[BSW00416] Sequence of Initialization	Not applicable (requirement on system design, not a single module)
[BSW00406] Check module initialization	FEE120, FEE121, FEE122, FEE123, FEE124, FEE125, FEE126, FEE127, FEE010
[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_Conf
[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
[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_Conf
[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
[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 defined values	Chapter 10.1
[BSW00411] Get version info keyword	Chapter 8.3.9
[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 8.3.7, 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 routines)
[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 specifciation)
[BSW00374] Module vendor identification	FEE043_Conf
[BSW00379] Module identification	FEE043_Conf
[BSW003] Version identification	FEE043_Conf
[BSW00318] Format of module version numbers	FEE043_Conf
[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	FEE085
[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	Not applicable (this module has no direct hardware access)
[BSW12462] Provide settings for register initialization	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)
[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_Conf
[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	FEE076, FEE005, FEE071, FEE116_Conf
BSW14002 Configuration of number of required write cycles	FEE102, FEE103, FEE110_Conf
BSW14003 Configuration of maximum blocking time	Not applicable (any more) Maximum blocking time has been converted into a published parameter (see FEE070_Conf)
BSW14004 Configuration of "immediate" data blocks	FEE151_Conf
BSW14026 Don't use certain block numbers	FEE006
BSW14027 Publish overhead for internal management data per block	FEE117_Conf, FEE118_Conf
BSW14005 Virtual linear address space and segmentation	FEE076
BSW14006 Alignment of block erase / write addresses	FEE024
BSW14007 Alignment of block read addresses	FEE021 and note below
BSW14008 Checking block read addresses	FEE021 and note below
BSW14009 Conversion of logical to physical addresses	FEE007
BSW14010 Block-wise write service	FEE088
BSW14029 Block-wise read service	FEE087
BSW14031 Service to cancel an ongoing asynchronous operation	FEE089
BSW14028 Service to invalidate a memory block	FEE092
BSW14012 Spreading of write access	FEE102, FEE103
BSW14013 Writing of "immediate" data must not be delayed	FEE009
BSW14032 Block-wise erase service for immediate data	FEE094
BSW14014 Detection of data inconsistencies	FEE023, FEE049, FEE153, FEE154
BSW14015 Reporting of data inconsistencies	FEE023
BSW14016 Don't return inconsistent data to the caller	Note below FEE023
BSW14017 Scope of EEPROM Abstraction Layer	Not applicable (this is the FEE modules specification)
BSW14018 Scope of Flash EEPROM Emulation	Chapter 1

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. ⌋(BSW14001, BSW14005)

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.

[FEE005] ⌈ Each configured logical block shall take up an integer multiple of the configured virtual page size (see also Chapter 10.2 configuration parameter `FeeVirtualPageSize`). ⌋(BSW14001)

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.

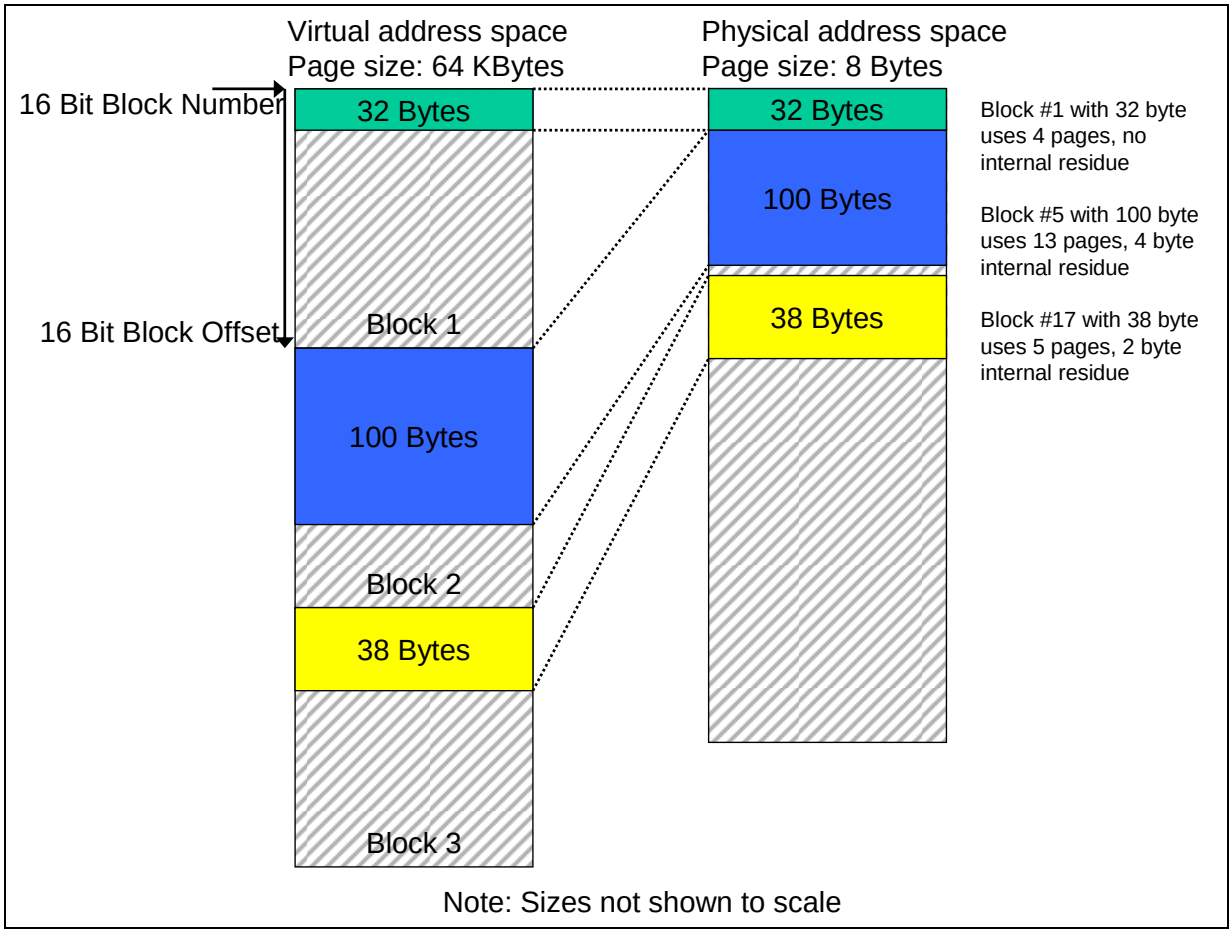

Figure 3: Virtual vs. physical memory layout

[FEE071] ⌈ Logical blocks must not overlap each other and must not be contained within one another. ⌋(BSW14001)

[FEE006] ⌈ The block numbers 0x0000 and 0xFFFF shall not be configurable for a logical block. ⌋(BSW14026)

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. ⌋(BSW14009)

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).

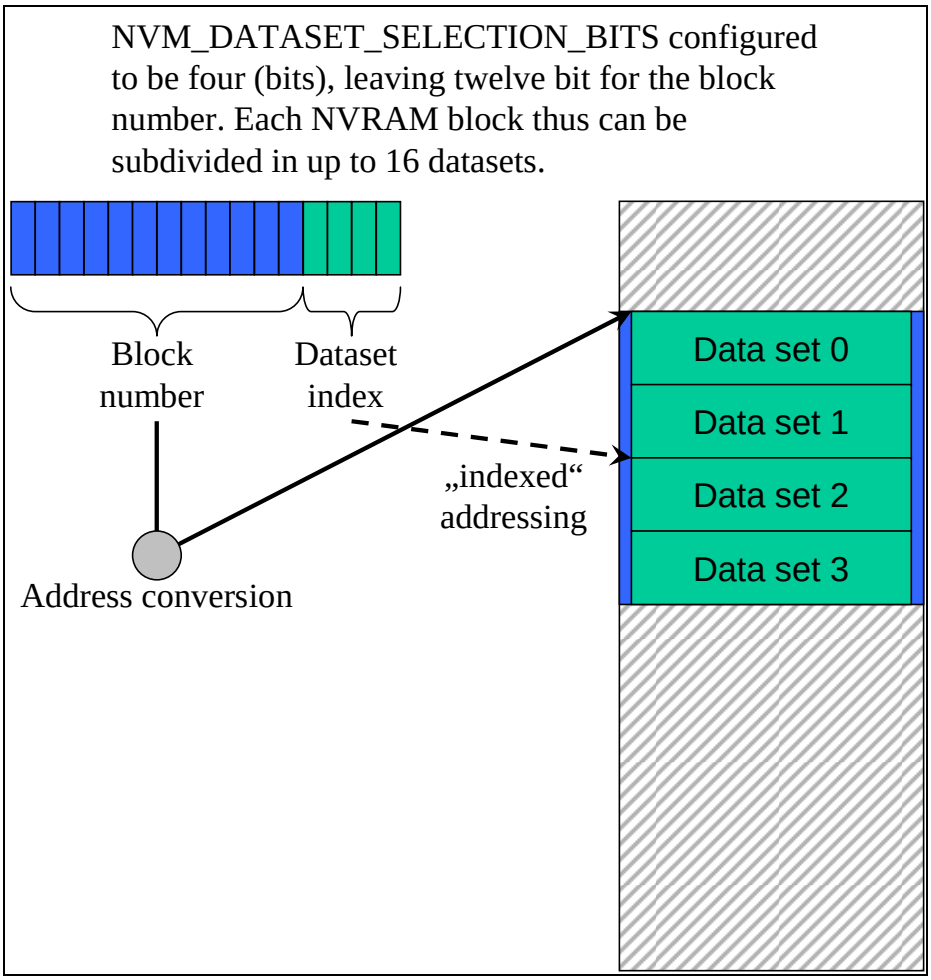

Figure 4: Block number and dataset index

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`. ⌋(BSW14002, BSW14012)

[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. ⌋(BSW14002, BSW14012)

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. ⌋(BSW14013)

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

[FEE049] ⌈ 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. ⌋(BSW14014)

[FEE153] ⌈ When a block write operation is started, the FEE module shall mark the corresponding block as “corrupted”[1]. ⌋(BSW14014)

[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.

[FEE154] ⌈ Upon the successful end of the block write operation, the block shall be marked as “not corrupted” (again). ⌋(BSW14014)

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

7.2 Error classification

[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`. ⌋(BSW00409)

[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 when module was not initialized	Development	FEE_E_UNINIT	0x01
API service called with invalid block number	Development	FEE_E_INVALID_BLOCK_NO	0x02
API service called with invalid block offset	Development	FEE_E_INVALID_BLOCK_OFS	0x03
API service called with invalid data pointer	Development	FEE_E_INVALID_DATA_PTR	0x04
API service called with invalid length information	Development	FEE_E_INVALID_BLOCK_LEN	0x05
API service called while module is busy processing a user request	Development	FEE_E_BUSY	0x06
API service called while module is busy doing internal management operations.	Development	FEE_E_BUSY_INTERNAL	0x07
Fee_Cancel called while no job was pending.	Development	FEE_E_INVALID_CANCEL	0x08

⌋(BSW00406, BSW00337, BSW00386, BSW00327, BSW00331)

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

Note: The error `FEE_BUSY_INTERNAL` *shall only be reported, if the internal management operation cannot be suspended or aborted (see e.g. FEE173). Whether an internal management operation can be suspended or aborted depends first on the underlying hardware (flash technology) and second on the implementation of the FEE (design decision of the software implementor / customer).*

7.3 Error detection

[FEE011] ⌈ The detection of development errors is configurable (*ON* / *OFF*) at pre-compile time. The switch `FeeDevErrorDetect` (see Chapter 10) shall activate or deactivate the detection of all development errors. ⌋(BSW00338, BSW00386, BSW00350)

[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 10.2. ⌋(BSW00350)

[FEE063] ⌈ The detection of production code errors cannot be switched off. ⌋()

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). ⌋(BSW00369, BSW00386, BSW00331)

[FEE106] ⌈ Production errors shall be reported to Diagnostic Event Manager. ⌋()

7.5 Consistency checks

[FEE013] ⌈ The Fee module shall perform inter module checks to avoid integration of incompatible files: all included header files shall be checked by pre-processing directives. The Fee module shall thereby verify that `<MODULENAME>_AR_RELEASE_MAJOR_VERSION` and `<MODULENAME>_AR_RELEASE_MINOR_VERSION` are identical to the expected values, where `<MODULENAME>` is the module abbreviation of the external module, which provides the included header file. If the values are not identical, an error shall be raised at compile time. ⌋(BSW004)

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).

7.6 Support for Debugging

[FEE130] ⌈ The modules status, the job result and the block meta information (see FEE049) shall be made available for debugging (reading). Therefore those variables shall be implemented as global variables. ⌋()

[FEE131] ⌈ The type definitions and declarations of all variables which shall be used for debugging shall be given in the modules header file `Fee.h`. ⌋()

[FEE132] ⌈ All variables which shall be used for debugging shall be described in detail in the modules description file. ⌋()

8 API specification

8.1 Imported Types

[FEE084]

⌈

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

⌋()

[FEE016] ⌈ The types mentioned in FEE084 shall not be changed or extended for a specific FEE module or hardware platform. ⌋()

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(`
	`  void`
	`)`

Service ID[hex]:	0x00
Sync/Async:	Asynchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None

Description:	Service to initialize the FEE module.

」(BSW101, BSW12057)

[FEE120] ⌈ The function `Fee_Init` shall set the module state from `MEMIF_UNINIT` to `MEMIF_BUSY_INTERNAL` once it starts the module's initialization. 」(BSW00406)

[FEE168] ⌈ If initialization is finished within `Fee_Init`, the function Fee_Init shall set the module state from `MEMIF_BUSY_INTERNAL` to `MEMIF_IDLE` once initialization has been successfully finished. 」()

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

8.3.2 Fee_SetMode

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

」()

[FEE020] ⌈ If the current module status is `MEMIF_IDLE` and if supported by the underlying hardware and device driver, the function `Fee_SetMode` shall call the function `Fls_SetMode` of the underlying flash driver with the given "Mode" parameter. 」(BSW12169)

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

[FEE121] ⌈ If development error detection is enabled for the module: the function `Fee_SetMode` shall check if the module status is `MEMIF_UNINIT`. If this is the case,

the function Fee_SetMode shall raise the development error FEE_E_UNINIT and return to the caller without executing the mode switch. ⌋(BSW00406)

[FEE170] ⌈ If development error detection is enabled for the module: the function Fee_SetMode shall check if the module state is MEMIF_BUSY. If this is the case, the function Fee_SetMode shall raise the development error FEE_E_BUSY and return to the caller without executing the mode switch. ⌋()

[FEE171] ⌈ If development error detection is enabled for the module: the function Fee_SetMode shall check if the module state is MEMIF_BUSY_INTERNAL. If this is the case, the function Fee_SetMode shall raise the development error FEE_E_BUSY_INTERNAL and return to the caller without executing the mode switch. ⌋()

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 (inout):	None	
Parameters (out):	DataBufferPtr	Pointer to data buffer
Return value:	Std_ReturnType	E_OK: The requested job has been accepted by the module. E_NOT_OK: The requested job has not been accepted by the module.
Description:	Service to initiate a read job.	

⌋(BSW14029)

[FEE021] ⌈ The function Fee_Read shall take the block start address and offset and calculate the corresponding memory read address. ⌋(BSW14007, BSW14008)

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] ⌈ If the current module status is MEMIF_IDLE or if the current module status is MEMIF_BUSY INTERNAL and the internal management operation can be suspended or aborted, 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. ⌋()

[FEE172] ⌈ If the current module status is MEMIF_UNINIT or MEMIF_BUSY or MEMIF_BUSY_INTERNAL and the internal management operation can't be suspended or aborted, the function Fee_Read shall reject the job request and return with E_NOT_OK. ⌋()

[FEE073] ⌈ The FEE module shall execute the read operation asynchronously within the FEE module's main function. ⌋()

[FEE122] ⌈ If development error detection is enabled for the module: the function Fee_Read shall check if the module state is MEMIF_UNINIT. If this is the case, the function Fee_Read shall reject the read request, raise the development error FEE_E_UNINIT and return with E_NOT_OK. ⌋(BSW00406)

[FEE133] ⌈ If development error detection is enabled for the module: the function Fee_Read shall check if the module state is MEMIF_BUSY. If this is the case, the function Fee_Read shall reject the read request, raise the development error FEE_E_BUSY and return with E_NOT_OK. ⌋()

[FEE173] ⌈ If development error detection is enabled for the module: if the current module status is MEMIF_BUSY_INTERNAL and if it is not possible to suspend or abort the internal management operation (because of data consistency / module implementation / hardware restrictions), the function Fee_Read shall reject the read request, raise the development error FEE_E_BUSY_INTERNAL and return with E_NOT_OK. ⌋()

[FEE134] ⌈ If development error detection is enabled for the module: the function Fee_Read shall check that the given block number is valid (i.e. it has been configured). If this is not the case, the function Fee_Read shall reject the read request, raise the development error FEE_E_INVALID_BLOCK_NO and return with E_NOT_OK. ⌋()

[FEE135] ⌈ If development error detection is enabled for the module: the function Fee_Read shall check that the given block offset is valid (i.e. that it is less than the block length configured for this block). If this is not the case, the function Fee_Read

shall reject the read request, raise the development error FEE_E_INVALID_BLOCK_OFS and return with E_NOT_OK. ⌋()

[FEE136] ⌈ If development error detection is enabled for the module: the function Fee_Read shall check that the given data pointer is valid (i.e. that it is not NULL). If this is not the case, the function Fee_Read shall reject the read request, raise the development error FEE_E_INVALID_DATA_PTR and return with E_NOT_OK. ⌋()

[FEE137] ⌈ If development error detection is enabled for the module: the function Fee_Read shall check that the given length information is valid, i.e. that the requested length information plus the block offset do not exceed the block end address (block start address plus configured block length). If this is not the case, the function Fee_Read shall reject the read request, raise the development error FEE_E_INVALID_BLOCK_LEN and return with E_NOT_OK. ⌋()

[FEE162] ⌈ If a read request is rejected by the function Fee_Read, i.e. requirements FEE122, FEE133, FEE134, FEE135, FEE136, FEE137 or FEE173 apply, the function Fee_Read shall not change the current module status or job result. ⌋()

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 (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: The requested job has been accepted by the module. E_NOT_OK: The requested job has not been accepted by the module.
Description:	Service to initiate a write job.	

⌋(BSW14010)

[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. ⌋(BSW14006)

[FEE025] ⌈ If the current module status is `MEMIF_IDLE` or if the current module status is `MEMIF_BUSY INTERNAL` and the internal management operation can be suspended or aborted, 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`. ⌋()

[FEE174] ⌈ If the current module status is `MEMIF_UNINIT` or `MEMIF_BUSY` or `MEMIF_BUSY_INTERNAL` and the internal management operation can't be suspended or aborted, the function `Fee_Write` shall reject the job request and return with `E_NOT_OK`. ⌋()

[FEE183] ⌈ If the write request addresses a block containing immediate data, the function `Fee_Write` shall accept the write request, even if the current module status is `MEMIF_BUSY_INTERNAL` and the internal management operation can't be suspended or aborted. ⌋()

Note: In this case the internal management operation shall be aborted without the chance to restart it and with the risk of unrecoverable errors for the "normal" data.

[FEE026] ⌈ The FEE module shall execute the write operation asynchronously within the FEE module's main function. ⌋()

[FEE123] ⌈ If development error detection is enabled for the module: the function `Fee_Write` shall check if the module state is `MEMIF_UNINIT`. If this is the case, the function `Fee_Write` shall reject the write request, raise the development error `FEE_E_UNINIT` and return with `E_NOT_OK`. ⌋(BSW00406)

[FEE144] ⌈ If development error detection is enabled for the module: the function `Fee_Write` shall check if the module state is `MEMIF_BUSY`. If this is the case, the function `Fee_Write` shall reject the write request, raise the development error `FEE_E_BUSY` and return with `E_NOT_OK`. ⌋()

[FEE175] ⌈ If development error detection is enabled for the module: if the current module status is `MEMIF_BUSY_INTERNAL` and if it is not possible to suspend or abort the internal management operation (because of data consistency / module implementation / hardware restrictions), the function `Fee_Write` shall reject the write

request, raise the development error FEE_E_BUSY_INTERNAL and return with E_NOT_OK. ⌋()

[FEE138] ⌈ If development error detection is enabled for the module: the function Fee_Write shall check that the given block number is valid (i.e. it has been configured). If this is not the case, the function Fee_Write shall reject the write request, raise the development error FEE_E_INVALID_BLOCK_NO and return with E_NOT_OK. ⌋()

[FEE139] ⌈ If development error detection is enabled for the module: the function Fee_Write shall check that the given data pointer is valid (i.e. that it is not NULL). If this is not the case, the function Fee_Write shall reject the write request, raise the development error FEE_E_INVALID_DATA_PTR and return with E_NOT_OK. ⌋()

[FEE163] ⌈ If a write request is rejected by the function Fee_Write, i.e. requirements FEE123, FEE144, FEE138, FEE139 or FEE175 apply, the function Fee_Write shall not change the current module status or job result. ⌋()

8.3.5 Fee_Cancel

[FEE089]
⌈

Service name:	Fee_Cancel

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

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

⌋(BSW14031)

Note: The function Fee_Cancel and the cancel function of the underlying flash driver are – from their behaviour – synchronous functions but they are asynchronous w.r.t. an ongoing read, erase or write job in the flash memory. The cancel functions shall only reset their modules internal variables so that a new job can be accepted by

the modules. They do not cancel an ongoing job in the hardware and they do not wait for an ongoing job to be finished by the hardware. This might lead to the situation in which the module's state is reported as `MEMIF_IDLE` while there is still an ongoing job being executed by the hardware. Therefore, the flash driver's main function shall check that the hardware is indeed free before starting a new job (see chapter 9.4 for a detailed sequence diagram).

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

[FEE124] ⌈ If development error detection is enabled for the module: the function Fee_Cancel shall check if the module state is `MEMIF_UNINIT`. If this is the case the function Fee_Cancel shall raise the development error `FEE_E_UNINIT` and return to the caller without changing any internal variables. ⌋(BSW00406)

[FEE080] ⌈ 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. ⌋()

[FEE081] ⌈ 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`. ⌋()

[FEE164] ⌈ 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. ⌋()

[FEE184] ⌈ If the current module status is not `MEMIF_BUSY` (i.e. there is no job to cancel and therefore the request to cancel a pending job is rejected by the function `Fee_Cancel`), the function `Fee_Cancel` shall raise the development error `FEE_E_INVALID_CANCEL`. ⌋()

8.3.6 Fee_GetStatus

[FEE090]

⌈

Service name:	Fee_GetStatus

Syntax:	`MemIf_StatusType Fee_GetStatus(`
	`  void`
	`)`

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

」()

[FEE034] 「 The function `Fee_GetStatus` shall return `MEMIF_UNINIT` if the module has not (yet) been initialized. 」()

[FEE128] 「 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. 」()

[FEE129] 「 The function `Fee_GetStatus` shall return `MEMIF_BUSY` if it is currently processing a request from the upper layer. 」()

[FEE074] 「 The function `Fee_GetStatus` shall return `MEMIF_BUSY_INTERNAL`, if an internal management operation is currently ongoing. 」()

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

8.3.7 Fee_GetJobResult

[FEE091]

「

Service name:	Fee_GetJobResult	

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

Service ID[hex]:	0x06	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	None	
Return value:	MemIf_JobResultType	MEMIF_JOB_OK: The last job has been finished

		successfully. MEMIF_JOB_PENDING: The last job is waiting for execution or currently being executed. MEMIF_JOB_CANCELED: The last job has been canceled (which means it failed). MEMIF_JOB_FAILED: The last job has not been finished successfully (it failed). MEMIF_BLOCK_INCONSISTENT: The requested block is inconsistent, it may contain corrupted data. MEMIF_BLOCK_INVALID: The requested block has been invalidated, the requested read operation can not be performed.
Description:	Service to query the result of the last accepted job issued by the upper layer software.	

」()

[FEE035] 「 The function `Fee_GetJobResult` shall return `MEMIF_JOB_OK` if the last job has been finished successfully. 」()

[FEE156] 「 The function `Fee_GetJobResult` shall return `MEMIF_JOB_PENDING` if the requested job is still waiting for execution or is currently being executed. 」()

[FEE157] 「 The function `Fee_GetJobResult` shall return `MEMIF_JOB_CANCELED` if the last job has been canceled by the upper layer. 」()

[FEE158] 「 The function `Fee_GetJobResult` shall return `MEMIF_JOB_FAILED` if the last job has failed. 」()

[FEE159] 「 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). 」()

[FEE160] 「 The function `Fee_GetJobResult` shall return `MEMIF_BLOCK_INVALID` if the requested block has been invalidated by the upper layer. 」()

[FEE155] 「 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. 」()

[FEE125] 「 If development error detection is enabled for the module: the function `Fee_GetJobResult` shall check if the module state is `MEMIF_UNINIT`. If this is the

case, the function `Fee_GetJobResult` shall raise the development error `FEE_E_UNINIT` and return with `MEMIF_JOB_FAILED`. ⌋(BSW00406)

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 (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: The requested job has been accepted by the module. E_NOT_OK - only if DET is enabled: The requested job has not been accepted by the module.
Description:	Service to invalidate a logical block.	

⌋(BSW14028)

[FEE036] ⌈ The function `Fee_InvalidateBlock` shall take the block number and calculate the corresponding memory block address. ⌋()

[FEE037] ⌈ The function `Fee_InvalidateBlock` shall invalidate the requested block <BlockNumber> by calling the erase function of the underlying device driver and / or by changing some module internal management information accordingly. ⌋()

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

[FEE176] ⌈ If the current module status is not `MEMIF_IDLE`, the function `Fee_InvalidateBlock` shall reject the invalidation request and return with `E_NOT_OK`. ⌋()

[FEE126] ⌈ If development error detection is enabled for the module: the function `Fee_InvalidateBlock` shall check if the module status is `MEMIF_UNINIT`. If this is the case, the function `Fee_InvalidateBlock` shall reject the invalidation

request, raise the development error FEE_E_UNINIT and return with E_NOT_OK. ⌋(BSW00406)

[FEE145] ⌈ If development error detection is enabled for the module: the function Fee_InvalidateBlock shall check if the module status is MEMIF_BUSY. If this is the case, the function Fee_InvalidateBlock shall reject the request, raise the development error FEE_E_BUSY and return with E_NOT_OK. ⌋()

[FEE177] ⌈ If development error detection is enabled for the module: if the current module status is MEMIF_BUSY_INTERNAL and if it is not possible to suspend or abort the internal management operation (because of data consistency / module implementation / hardware restrictions), the function Fee_InvalidateBlock shall reject the invalidation request, raise the development error FEE_E_BUSY_INTERNAL and return with E_NOT_OK. ⌋()

[FEE140] ⌈ If development error detection is enabled for the module: the function Fee_InvalidateBlock shall check that the given block number is valid (i.e. it has been configured). If this is not the case, the function Fee_InvalidateBlock shall reject the request, raise the development error FEE_E_INVALID_BLOCK_NO and return with E_NOT_OK. ⌋()

[FEE165] ⌈ If an invalidation request is rejected by the function Fee_InvalidateBlock, i.e. requirements FEE126, FEE140, FEE145 or FEE177 apply, the function Fee_InvalidateBlock shall not change the current module status or job result. ⌋()

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:	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.	

⌋()

[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. ⌋()

[FEE147] ⌈ 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_INVALID_DATA_PTR`. ⌋()

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 (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: The requested job has been accepted by the module. E_NOT_OK - only if DET is enabled: The requested job has not been accepted by the module.
Description:	Service to erase a logical block.	

⌋(BSW14032)

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.

[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 of the module. ⌋()

[FEE127] ⌈ If development error detection is enabled for the module: the function `Fee_EraseImmediateBlock` shall check if the module state is `MEMIF_UNINIT`. If this is the case, the function `Fee_EraseImmediateBlock` shall reject the erase request, raise the development error `FEE_E_UNINIT` and return with `E_NOT_OK`. ⌋(BSW00406)

[FEE146] ⌈ If development error detection is enabled for the module: the function `Fee_EraseImmediateBlock` shall check if the module state is `MEMIF_BUSY`. If this is the case, the function `Fee_EraseImmediateBlock` shall reject the erase request, raise the development error `FEE_E_BUSY` and return with `E_NOT_OK`. ⌋()

[FEE178] ⌈ If development error detection is enabled for the module: if the current module status is `MEMIF_BUSY_INTERNAL` and if it is not possible to suspend or abort the internal management operation (because of data consistency / module implementation / hardware restrictions), the function `Fee_EraseImmediateBlock` shall reject the request, raise the development error `FEE_E_BUSY_INTERNAL` and return with `E_NOT_OK`. ⌋()

[FEE068] ⌈ If development error detection is enabled for the module: the function `Fee_EraseImmediateBlock` shall check whether the addressed logical block is configured as containing immediate data (`FeeImmediateData == TRUE`). If not, the function `Fee_EraseImmediateBlock` shall raise the development error `FEE_E_INVALID_BLOCK_NO` and return `E_NOT_OK` without erasing the addressed logical block. ⌋(BSW12448)

[FEE141] ⌈ If development error detection is enabled for the module: the function `Fee_EraseImmediateBlock` shall check that the given block number is valid (i.e. it has been configured). If this is not the case, the function `Fee_EraseImmediateBlock` shall reject the erase request, raise the development error `FEE_E_INVALID_BLOCK_NO` and return with `E_NOT_OK`. ⌋()

[FEE166] ⌈ If a erase request is rejected by the function Fee_EraseImmediateBlock, i.e. requirements FEE068, FEE127, FEE141, FEE146 or FEE178 apply, the function `Fee_EraseImmediateBlock` shall not change the current module status or job result. ⌋()

8.4 Call-back notifications

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

[FEE069] ⌈ The FEE module shall provide function prototypes of the callback functions in the file `Fee_Cbk.h` ⌋(BSW00325)

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

[FEE095]

⌈

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.

⌋()

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 subsequently call the job end notification routine of the upper layer module if configured. ⌋()

[FEE142] ⌈ 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. ⌋()

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

⌋()

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 subsequently call the job error notification routine of the upper layer module if configured. ⌋()

[FEE143] ⌈ If the job result is currently `MEMIF_JOB_PENDING`, the function Fee_JobErrorNotification shall set the job result to `MEMIF_JOB_FAILED`, else it shall leave the job result untouched. ⌋()

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(`
	`  void`
	`)`

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

⌋()

[FEE169] ⌈ If the module initialization (started in the function `Fee_Init`) is completed in the module's main function, the function `Fee_MainFunction` shall set the module status from `MEMIF_BUSY_INTERNAL` to `MEMIF_IDLE` once initialization of the module has been successfully finished. ⌋()

[FEE057] ⌈ The function `Fee_MainFunction` shall asynchronously handle the read / write / erase / invalidate jobs requested by the upper layer and 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` and call the error notification routine of the upper layer 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 if configured. ⌋(BSW14014, BSW14015, BSW14016)

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

[FEE179] ⌈ If the current module status is `MEMIF_BUSY_INTERNAL` and if the internal management operation can be suspended without jeopardizing the data consistency: the function `Fee_MainFunction` shall save all information which is

necessary to resume the internal management operation, suspend the internal management operation and start processing the job requested by the upper layer. ⌋()

[FEE180] ⌈ If the current module status is `MEMIF_BUSY_INTERNAL` and if the internal management operation can be aborted without jeopardizing the data consistency: the function `Fee_MainFunction` shall save all information which is necessary to restart the internal management operation, abort the internal management operation and start processing the job requested by the upper layer. ⌋()

Note: Whether an internal management operation can be suspended or aborted depends on the type of management operation, the implementation of the FEE module and the capabilities of the underlying hardware and thus cannot be determined in this document.

[FEE181] ⌈ If an internal management operation has been suspended because of a job request from the upper layer, the function `Fee_MainFunction` shall resume this internal management operation once the job requested by the upper layer has been finished. ⌋()

[FEE182] ⌈ If an internal management operation has been aborted because of a job request from the upper layer, the function `Fee_MainFunction` shall restart this internal management operation once the job requested by the upper layer has been finished. ⌋()

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

」()

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 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.

[FEE098]

「

Service name:	NvM_JobEndNotification

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

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

」()

[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)
- Invalidation of memory block finished & OK ⌋()

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

[FEE099]

⌈

Service name:	NvM_JobErrorNotification
Syntax:	`void NvM_JobErrorNotification( void )`
Sync/Async:	true
Reentrancy:	Don't care
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	--

⌋()

[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)
- Invalidation of memory block finished & failed ⌋()

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.

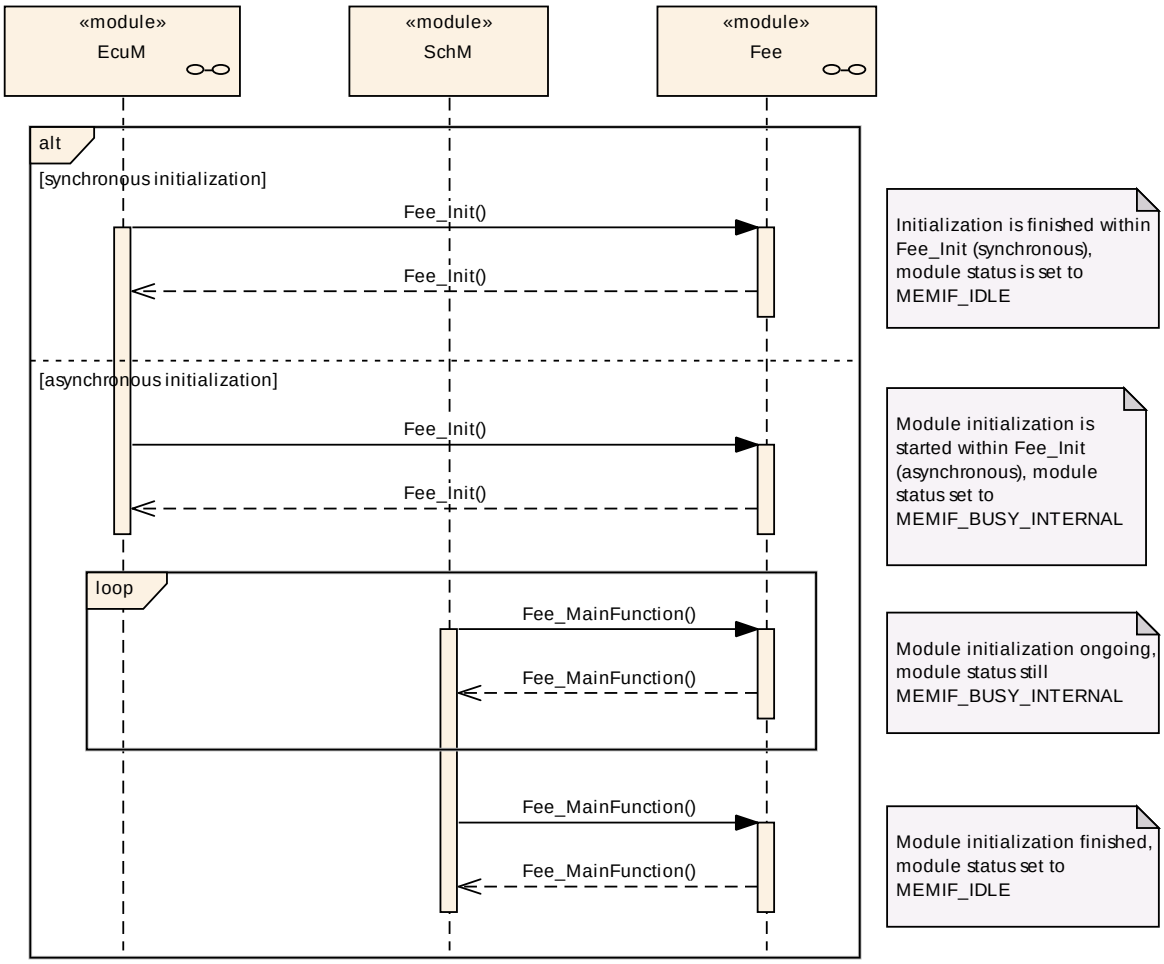

Figure 5: Sequence diagram of “Fee_Init” service

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).

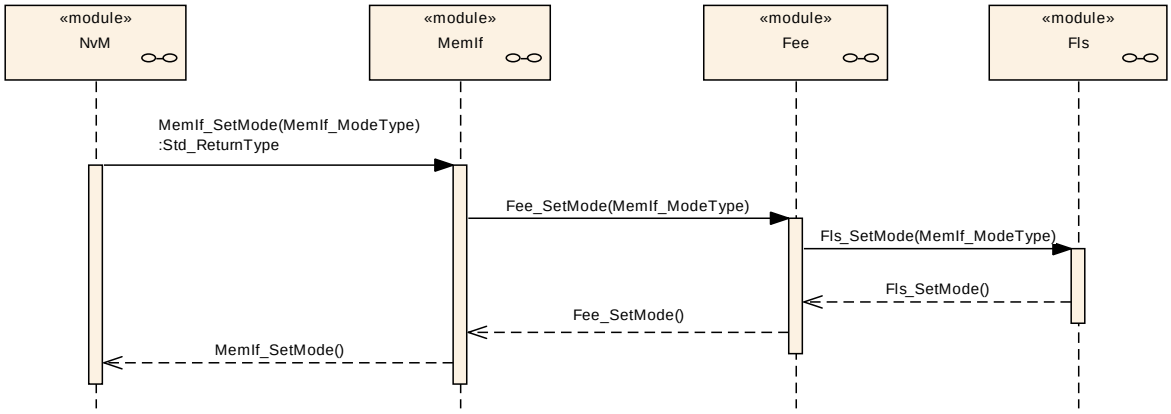

Figure 6: Sequence diagram of the "Fee_SetMode" service

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.

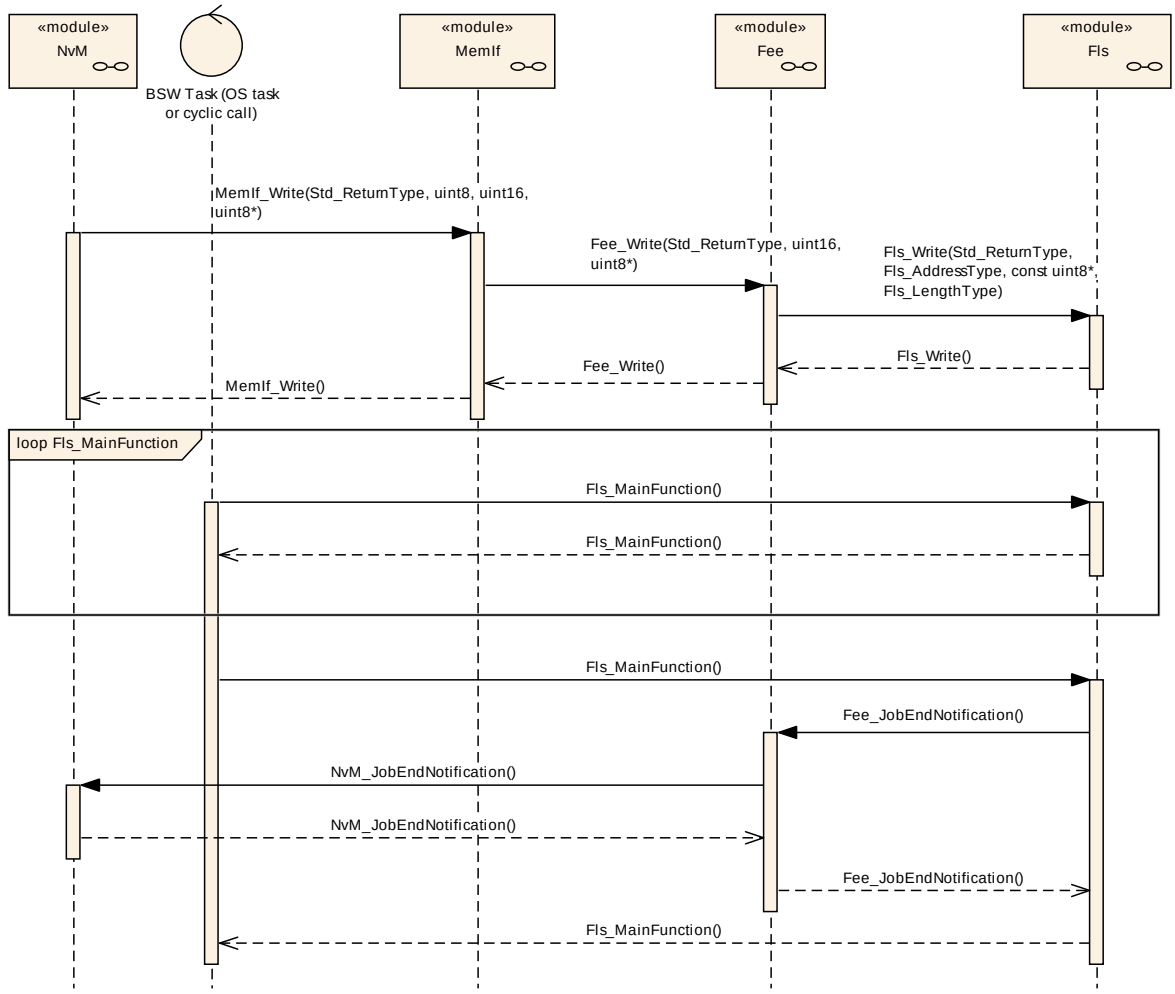

Figure 7: Sequence diagram “Fee_Write”

9.4 Fee_Cancel

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

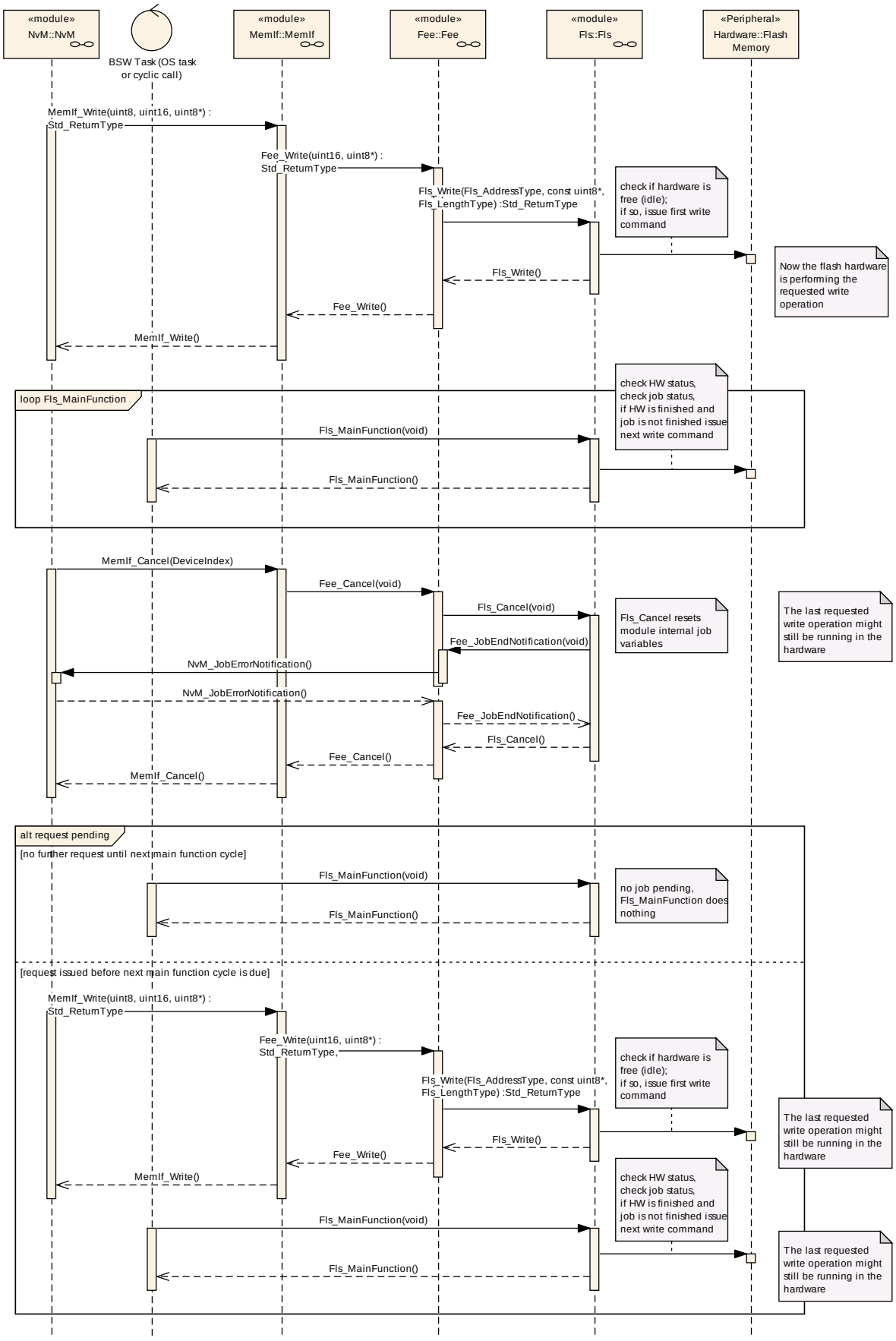

Figure 8: Sequence diagram „Fee_Cancel“

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

[FEE167] ⌈ The FEE module shall support (only) the following configuration variants:

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

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

SWS Item	**FEE111_Conf :**		
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	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**FEE152_Conf :**		
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	EcucIntegerParamDef		
Range	0 .. 254		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**FEE112_Conf :**		
Name	FeeNvmJobEndNotification {FEE_NVM_JOB_END_NOTIFICATION}		
Description	Mapped to the job end notification routine provided by the upper layer module (NvM_JobEndNotification).		
Multiplicity	0..1		
Type	EcucFunctionNameDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**FEE113_Conf :**		
Name	FeeNvmJobErrorNotification {FEE_NVM_JOB_ERROR_NOTIFICATION}		
Description	Mapped to the job error notification routine provided by the upper layer module (NvM_JobErrorNotification).		
Multiplicity	0..1		
Type	EcucFunctionNameDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**FEE114_Conf :**		
Name	FeePollingMode {FEE_POLLING_MODE}		
Description	Pre-processor switch to enable and disable the polling mode for this module. true: Polling mode enabled, callback functions (provided to FLS module) disabled. false: Polling mode disabled, callback functions (provided to FLS module) enabled.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**FEE119_Conf :**		
Name	FeeSetModeSupported {FEE_SET_MODE_SUPPORTED}		
Description	Compiler switch to enable/disable the 'SetMode' functionality of the FEE module. TRUE: SetMode functionality supported / code present, FALSE: SetMode functionality not supported / code not present. Note: This configuration setting has to be consistent with that of all underlying flash device drivers (configuration parameter FlsSetModeApi).		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**FEE115_Conf :**		
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	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants

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

SWS Item	**FEE116_Conf :**		
Name	FeeVirtualPageSize {FEE_VIRTUAL_PAGE_SIZE}		
Description	The size in bytes to which logical blocks shall be aligned.		
Multiplicity	1		
Type	EcucIntegerParamDef		
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_Conf :**
Container Name	FeeBlockConfiguration{FEE_BlockConfiguration}
Description	Configuration of block specific parameters for the Flash EEPROM Emulation module.
Configuration Parameters	

SWS Item	**FEE150_Conf :**		
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_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	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**FEE148_Conf :**		
Name	FeeBlockSize {FEE_BLOCK_SIZE}		
Description	Size of a logical block in bytes.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 65535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	

Scope / Dependency	scope: module

SWS Item	**FEE151_Conf :**		
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	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**FEE110_Conf :**		
Name	FeeNumberOfWriteCycles {FEE_NUMBER_OF_WRITE_CYCLES}		
Description	Number of write cycles required for this block.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**FEE149_Conf :**		
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

[**FEE185**] ⌈ The standardized common published parameters as required by BSW00402 in the General Requirements on Basic Software Modules [3] shall be published within the header file of this module and need to be provided in the BSW Module Description. The according module abbreviation can be found in the List of Basic Software Modules [1]. ⌋()

Additional module-specific published parameters are listed below if applicable.

10.3.1 FeePublishedInformation

SWS Item	**FEE043_Conf :**
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_Conf :**		
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	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	--		
ConfigurationClass	***Published Information***	X	All Variants
Scope / Dependency	scope: module		

SWS Item	**FEE070_Conf :**		
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	EcucFloatParamDef		
Range	`0 .. INF`		
Default value	--		
ConfigurationClass	***Published Information***	X	All Variants
Scope / Dependency	scope: module		

SWS Item	**FEE118_Conf :**
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	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	--		
ConfigurationClass	***Published Information***	X	All Variants
Scope / Dependency	scope: module		

No Included Containers

11 Changes w.r.t. Release 4.0

11.1 Deleted SWS Items

SWS Item	*Rationale*
FEE038	No longer appropriate

11.2 Replaced SWS Items

SWS Item	*replaced by SWS Item*	*Rationale*

11.3 Changed SWS Items

SWS Item	*Rationale*
FEE002	NvM_Cbk.h added to file include structure in chapter 5.1.2
FEE150_Conf, FEE148_Conf	Ranges for FeeBlockNumber and FeeBlockSize adjusted
FEE120	Initialization might not be finished withing Fee_Init, state machine adapted accordingly.
FEE010, FEE020, FEE022, FEE025, FEE133, FEE144, FEE080, FEE081, FEE164, FEE128, FEE129, FEE145, FEE146, FEE052, FEE054, FEE075, FEE023	Handling of internal management operations refined.
FEE013	Inter module checks detailed
FEE150_Conf	Naming corrected to NVM_DATASET_SELECTION_BITS
9.1	Sequence diagram for Fee_Init extended.
9.4	Sequence diagram for Fee_Cancel replaced for generated one.
FEE013	Inter-modul checks clarified.
FEE013	
FEE010	RfC #46721: Note added regarding internal management operations.
FEE085	RfC #46718: Fee_Init changed to asynchronous
FEE087, FEE088	RfC #46722: Removed dependency to DET.

11.4 Added SWS Items

SWS Item	*Rationale*
FEE168, FEE169	Initialization might not be finished withing Fee_Init, state machine adapted accordingly.
FEE170, FEE171, FEE172, FEE173, FEE174, FEE175, FEE176, FEE177,	Handling of internal management operations refined.

FEE178, FEE179, FEE180, FEE181, FEE182, FEE183	
FEE184	

12 Not applicable requirements

[FEE999] ⌈ These requirements are not applicable to this specification. ⌋ (BWS00344, BWS00404, BWS00405, BWS171, BWS170, BWS00380, BWS00412, BWS00398, BWS00399, BWS00400, BWS00375, BWS00416, BWS168, BWS00423, BWS00424, BWS00425, BWS00426, BWS00427, BWS00428, BWS00429, BWS00431, BWS00432, BWS00433, BWS00434, BWS00336, BWS00339, BWS00421, BWS00422, BWS00420, BWS00417, BWS00323, BWS161, BWS00324, BWS005, BWS00415, BWS164, BWS00326, BWS00342, BWS160, BWS007, BWS00300, BWS00347, BWS00307, BWS00314, BWS00348, BWS00353, BWS00361, BWS00302, BWS00328, BWS00312, BWS006, BWS00304, BWS00355, BWS00378, BWS00306, BWS00308, BWS00309, BWS00371, BWS00359, BWS00360, BWS00330, BWS009, BWS00401, BWS172, BWS010, BWS00333, BWS00321, BWS00341, BWS00334, BWS12263, BWS12056, BWS12267, BWS12125, BWS12163, BWS12058, BWS12059, BWS12060, BWS12461, BWS12462, BWS12463, BWS12062, BWS12068, BWS12069, BWS157, BWS12155, BWS12063, BWS12129, BWS12064, BWS12067, BWS12077, BWS12078, BWS12092, BWS12265, BWS12081, BWS14003, BWS14017)