User Manual

for S32K3 FEE Driver

Document Number: UM34FEEASR4.4 Rev0000R3.0.0 P01 Rev. 1.0

Chapter 1

Revision History

Revision	Date	Author	Description
1.0	31.03.2023	NXP RTD Team	S32K3 Real-Time Drivers AUTOSAR 4.4 Version 3.0.0 P01

Chapter 2

Introduction

This User Manual describes NXP Semiconductor AUTOSAR Flash EEPROM Emulation (FEE) for S32K3XX. AUTOSAR FEE driver configuration parameters and deviations from the specification are described in FEE Driver chapter of this document. AUTOSAR FEE driver requirements and APIs are described in the AUTOSAR FEE driver software specification document.

2.1 Supported Derivatives

The software described in this document is intended to be used with the following microcontroller devices of NXP Semiconductors:

- s32k310_mqfp100
- s32k310_lqfp48
- s32k311_mqfp100 / MWCT2015S_mqfp100
- s32k311_lqfp48
- s32k312_mqfp100 / MWCT2016S_mqfp100
- s32k312_mqfp172 / MWCT2016S_mqfp172
- s32k314_mqfp172
- s32k314_mapbga257
- s32k322_mqfp100 / MWCT2D16S_mqfp100
- s32k322_mqfp172 / MWCT2D16S_mqfp172

- s32k324_mqfp172 / MWCT2D17S_mqfp172
- s32k324_mapbga257
- s32k341_mqfp100
- s32k341_mqfp172
- s32k342_mqfp100
- s32k342_mqfp172
- s32k344_mqfp172
- s32k344_mapbga257
- s32k394_mapbga289
- s32k396_mapbga289
- s32k358_mqfp172
- s32k358_mapbga289
- s32k328_mqfp172
- s32k328_mapbga289
- s32k338_mqfp172
- s32k338_mapbga289
- s32k348_mqfp172
- s32k348_mapbga289
- s32m274_lqfp64
- s32m276_lqfp64

All of the above microcontroller devices are collectively named as S32K3.

Note: MWCT part numbers contain NXP confidential IP for Qi Wireless Power.

2.2 Overview

AUTOSAR (AUTomotive Open System ARchitecture) is an industry partnership working to establish standards for software interfaces and software modules for automobile electronic control systems.

AUTOSAR:

- paves the way for innovative electronic systems that further improve performance, safety and environmental friendliness.
- is a strong global partnership that creates one common standard: "Cooperate on standards, compete on implementation".
- is a key enabling technology to manage the growing electrics/electronics complexity. It aims to be prepared for the upcoming technologies and to improve cost-efficiency without making any compromise with respect to quality.
- facilitates the exchange and update of software and hardware over the service life of the vehicle.

2.3 About This Manual

This Technical Reference employs the following typographical conventions:

- **Boldface** style: Used for important terms, notes and warnings.
- *Italic* style: Used for code snippets in the text. Note that C language modifiers such "const" or "volatile" are sometimes omitted to improve readability of the presented code.

Notes and warnings are shown as below:

Note

This is a note.

Warning

This is a warning

2.4 Acronyms and Definitions

Term	Definition
API	Application Programming Interface
ASM	Assembler
AUTOSAR	Automotive Open System Architecture
DET	Development Error Tracer
ECU	Electronic Control Unit
MCU	Micro Controller Unit
OS	Operating System
MSB	Most Significant Bit
N/A	Not Applicable
SWS	Software Specification
VLE	Variable Length Encoding
XML	Extensible Markup Language
ISR	Interrupt Service Routine
IVOR	Interrupt Vector Offset Register
ECC	Error Correcting Code
DFO	Data Flash Optimized
DW	Double Word
EEPROM	Electrically Erasable Programmable Read-Only Memory
FLS	Flash memory driver
FEE	Flash EEPROM Emulation
RTD	Real Time Drivers

2.5 Reference List

#	Title	Version
1	Specification of Fee Driver	AUTOSAR Release 4.4.0
2	S32K3XX Reference Manual	Rev.6, Draft B, 01/2023
3	S32K3xx Data Sheet	Rev. 6, 11/2022
4	S32K39 and S32K37 Reference Manual	Rev. 2 Draft A, 11/2022
5	S32K396 Data Sheet	Rev. 1.1 — 08/2022
6	S32M27x Reference Manual	Rev.2, Draft A, — 02/2023
7	S32M2xx Data Sheet	Rev. 2 RC — 12/2022
8	S32K358_0P14E Mask Set Errata	Rev. 28, 9/2022
9	S32K396_0P40E Mask Set Errata	Rev. DEC2022, 12/2022
10	S32K311_0P98C Mask Set Errata	Rev. 6/March/2023, 3/2023
11	S32K312: Mask Set Errata for Mask 0P09C	Rev. 25/April/2022
12	S32K342: Mask Set Errata for Mask 0P97C	Rev. 10, 11/2022
13	S32K3x4: Mask Set Errata for Mask 0P55A/1P55A	Rev. 14/Oct/2022

Chapter 3

Driver

3.1 Requirements

The driver deviates from the AUTOSAR Fee driver software specification (See Table Reference List).

AUTOSAR deviations from requirements are described in Deviations from Requirements chapter of this document..

3.2 Driver Design Summary

EEPROM (electrically erasable programmable read only memory), which can be byte or word programmed and erased, is often used in automotive electronic control units (ECUs). This flexibility for program and erase operations makes it suitable for data storage of application variables.

For the devices without EEPROM memory, the block-erasable (or sector-erasable) flash memory can be used to emulate the EEPROM through EEPROM emulation software. The Flash EEPROM Emulation module implements emulation of variable-length blocks. Two or more FEE clusters are used to implement such software emulation scheme. The Flash EEPROM Emulation (FEE) provides the upper layer a virtual addressing scheme as well as a "virtually" unlimited number of erase/program cycles. The Flash EEPROM emulation module provides services for reading, writing, erasing and invalidating emulated EEPROM blocks apart from other basic features specified by the software specification.

During the FEE module configuration, each FEE block is assigned to specific FEE cluster group where the FEE block will be physically emulated. Each FEE cluster group consists of at least two FEE clusters, where each FEE cluster consists of at least one FLS logical sector. The list of available FLS logical sectors that can be used by the FEE module for emulation depends on actual FLS driver logical sector list configuration.

The memory operations (read, write etc.) are performed asynchronously. Their respective APIs store actual parameters into internal data structures and immediately return. The *job* is performed by means of state machines, which are driven by repetitive calls to the so called *main functions* of the FEE and FLS drivers (Fee_MainFunction, Fls_MainFunction). These executive functions accomplish their tasks in predefined chunks of data only, and shall be called by the application repeatedly (e.g. periodically in a dedicated task).

Note: For correct FEE operation the underlying FLS driver has to have configured the job-notification callbacks to FEE module:

```
FlsJobEndNotification = Fee_JobEndNotification
FlsJobErrorNotification = Fee_JobErrorNotification
```

These callbacks control the FEE driver state machine transitions.

3.3 Hardware Resources

None

3.4 Deviations from Requirements

The driver deviates from the AUTOSAR FEE Driver software specification in some places. The table below identifies the AUTOSAR requirements that are not implemented or out of scope for the FEE Driver.

Term	Definition
N/S	Out of scope
N/I	Not implemented
N/F	Not fully implemented

Below table identifies the AUTOSAR requirements that are not fully implemented, implemented differently or out of scope for the FEE driver.

Requirement	Status	Description	Notes
SWS_Fee_00084	N/S	Module - Header File - Imported Type - Fls - Fls.h - Fls_AddressType - Fls.h - Fls_LengthType - MemIf - MemIf.h - MemIf_JobResultType - MemIf.h - MemIf_ModeType - MemIf.h - MemIf_StatusType - Std_Types - StandardTypes.h - Std_ReturnType - StandardTypes.h - Std_VersionInfoType -	Not an FEE module requirement.
SWS_Fee_00187	N/S	If the function Fls_BlankCheck is configured (in the flash driver), the function Fee_Read shall call the function Fls_BlankCheck to determine in advance whether a given memory area can be read without encountering e.g. ECC errors due to trying to read erased but not programmed flash cells.	Not applicable
SWS_Fee_00007	N/S	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.	Unclear implementation of dataset concept. This FEE uses a different mapping algorithm.
SWS_Fee_00100	N/S	Only those bits of the 16bit block number, that do not denote a specific dataset or redundant copy shall be used for address calculation.	Unclear implementation of dataset concept. Not used in FEE driver.
SWS_Fee_00102	N/S	The configuration of the FEE module shall define the expected number of erase/write cycles for each logical block in the configuration parameter Fee_NumberOfWriteCycles.	Not used due to usage of the two or more clusters emulation algorithm.
SWS_Fee_00103	N/S	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.	Emulation algorithm uses multiple flash blocks regardless on configured erase/write cycles.
SWS_Fee_00168	N/S	If initialization is finished within Fee_Init, the function Fee_Init shall set the module state from MEMIF_UNINIT to MEMIF_IDLE once initialization has been successfully finished.	Initialization Phase needs Fee_MainFunction and Fls_MainFunction calles to be completed. Also, the state transition is not performed by the Fee_Init() function itself, but as a part of state machine execution driven by main function calls.

Requirement	Status	Description	Notes
SWS_Fee_00074	N/S	The function Fee_GetStatus shall return MEMIF_BUSY_INTERNAL, if an internal management operation is currently ongoing.	Fee driver does not support MEMIF↩ _BUSY INTERNAL
SWS_Fee_00105	N/S	API function - Header File - Description - Det_ReportRuntime↩ Error - Det.h - Service to report runtime errors. If a callout has been configured then this callout shall be called. - Fls_Cancel - Fls.h - Cancels an ongoing job. - Fls_Compare - Fls↩ _Com.h - Compares the contents of an area of flash memory with that of an application data buffer. - Fls_Erase - Fls.h - Erases flash sector(s). - Fls↩ _GetJobResult - Fls.h - Returns the result of the last job. - Fls_GetStatus - Fls.h - Returns the driver state. - Fls_Read - Fls.h - Reads from flash memory. - Fls_SetMode - Fls.h - Sets the flash driver's operation mode. - Fls_Write - Fls.h - Writes one or more complete flash pages. -	Not a FEE module requirement
SWS_Fee_00104	N/S	API function - Header File - Description - Det_ReportError - Det.h - Service to report development errors. - Fls_BlankCheck - Fls.h - The function Fls_BlankCheck shall verify, whether a given memory area has been erased but not (yet) programmed. The function shall limit the maximum number of checked flash cells per main function cycle to the configured value FlsMaxReadNormalMode or FlsMaxReadFastMode respectively. -	Not a FEE module requirement
SWS_Fee_00098	N/S	Service name: - NvM_JobEnd↩ Notification - 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: - – - Available via: - Nvm.h -	Not a FEE module requirement

Requirement	Status	Description	Notes
SWS_Fee_00099	N/S	Service name: - NvM_JobError↩ Notification - 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: - – - Available via: - Nvm.h -	Not a FEE module requirement
SWS_Fee_00999	N/S	These requirements are not applicable to this specification.	This is not a requirement.
ECUC_Fee_00153	N/S	Name - FeeMainFunctionPeriod - Parent Container - FeeGeneral - Description - The period between successive calls to the main function in seconds. - Multiplicity - 1 - Type - EcucFloatParamDef - Range -]0 .. INF[- - Default value - – - Post-Build Variant Value - false - Value Configuration Class - Pre-compile time - X - All Variants - Link time - – - - Post-build time - – - - Scope / Dependency - scope: ECU -	Not needed in the current Fee implementation.
ECUC_Fee_00114	N/S	Name - FeePollingMode - Parent Container - FeeGeneral - Description - Pre-processor switch to enable and disable the polling mode for this module.true: Polling mode enabled, callback functions (provided to FLS module) disabled.false: Polling mode disabled, callback functions (provided to FLS module) enabled. - Multiplicity - 1 - Type - EcucBooleanParamDef - Default value - – - Post-Build Variant Value - false - Value Configuration Class - Pre-compile time - X - All Variants - Link time - – - - Post-build time - – - - Scope / Dependency - scope: local -	Not supported by FEE module, unclear polling concept.
ECUC_Fee_00110	N/S	Name - FeeNumberOfWriteCycles - Parent Container - FeeBlock↩ Configuration - Description - Number of write cycles required for this block. - Multiplicity - 1 - Type - EcucIntegerParamDef - Range - 0 .. 4294967295 - - Default value - – - Post-Build Variant Value - false - Value Configuration Class - Pre-compile time - X - All Variants - Link time - – - - Post-build time - – - - Scope / Dependency - scope: local -	Not supported by FEE module due to emulation scheme.

3.5 Driver Limitations

None.

3.6 Driver usage and configuration tips

- FEE Data Organization details
- Memory Dump Example
- FEE Block Always Available
- Managing Cluster and Block Consistency
- Cluster Swap
- Block Update
- Immediate Block Update
- Det Errors Description
- Endurance calculation
- Configuration tips
- Sector managerment and sector retirement
- FEE Swap Foreign Blocks feature

3.6.1 FEE Data Organization details

The FEE module provides upper layers with a 32bit virtual linear address space and uniform segmentation scheme. This virtual 32bit address 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 organization of flash area reserved for FEE driver is described here below. The memory area is organized in:

- **Cluster Group**: A group is made by at least 2 Clusters
- **Cluster**: One or more flash physical sectors containing FEE blocks
- **Block**: Area of flash containing application data

More clusters could be present in the area but just one is active and contains valid data while the others are not used.

Note

In the example below FeeVirtualPageSize is set to 8. Header valid flag and invalid flag are aligned each to FeeVirtualPageSize boundary. Each cluster/block has:

- an header
- data

Table 3.3 Data Organization Detail

4 bytes		4 bytes	4 bytes	4 bytes	Description
ClrID		**Start address**	**Cluster size**	**Checksum**	*Cluster header*
Valid flag			not used	not used	*Cluster status*
Block id	**Length**	**Target address**	**Checksum**	**Block Assignment(1byte)**	*Block 1 header*
Valid flag			**Invalid flag**		*Block 1 status*
Block id	Length	Target address	Checksum	Block Assignment(1byte)	*Block 2 header*
Valid flag			Invalid flag		Block 2 status
..					
...					
Block id	Length	Target address	Checksum	Block Assignment(1byte)	*Block n-1 header*
Valid flag			Invalid flag		*Block n-1 status*
Block id	Length	Target address	Checksum	Block Assignment(1byte)	*Block n header*
Valid flag			Invalid flag		*Block n status*
(padding)					*16 byte*
(padding)					*16 byte*
BLOCK n DATA					*Block n Data*
BLOCK n DATA					*Block n Data*
BLOCK n DATA					*Block n Data*
BLOCK n DATA					*Block n Data*
BLOCK n-1 DATA					*Block n-1 Data*
BLOCK n-1 DATA					*Block n-1 Data*
..					
...					
BLOCK 2 DATA					*Block 2 Data*
BLOCK 2 DATA					*Block 2 Data*
BLOCK 1 DATA					*Block 1 Data*
BLOCK 1 DATA					*Block 1 Data*
BLOCK 1 DATA					*Block 1 Data*
BLOCK 1 DATA					*Block 1 Data*

Table 3.4 ClusterHdr Type

uint32 (4 bytes)	uint32 (4 bytes)	uint32 (4 bytes)	uint32 (4 bytes)
ClrID	StartAddress	ClusterSize	checkSum

uint32 (4 bytes)	uint32 (4 bytes)	uint32 (4 bytes)	uint32 (4 bytes)
valFlag	blank1	invalFlag	blank2

- **ClrID**: (uint32) Integer number uniquely identifying the cluster. The number is incremented whenever a new cluster becomes active, i.e. the cluster with highest ClrID is the active one.
- **StartAddress**: (uint32) Configuration data: Start address of the cluster (logical start address of the first flash sector belonging to this cluster).
- **ClusterSize**: (uint32) Configuration data: Length of the cluster.
- **checkSum**: (uint32) Sum of the ClrID, StartAddress and ClusterSize fields.
- **val Flag**: (uint8) 0x81 for a valid cluster. The field is padded with blank bytes (0xFF) to the virtual page size boundary.
- **invalFlag**: (uint8) not-used. The field is padded with blank bytes (0xFF) to the virtual page size boundary.

Table 3.5 BlockHdr Type

uint32 (4 bytes)	uint32 (4 bytes)	uint32 (4 bytes)	uint32 (4 bytes)
BlockNumber:Length	TargetAddress	checkSum	assignment(1byte)
valFlag	blank2	invalFlag	blank3

- **BlockNumber**: (uint16) Configuration data: Integer number uniquely identifying the block.
- **Length**: (uint16) Configuration data: Length of the block.
- **TargetAddress**: (uint32) Logical address of the beginning of data area of this block.
- **checkSum**: (uint32) Sum of the BlockNumber, Length and TargetAddress fields.
- **assignment(1byte)**: (uint8)Block assignment.Used only for FeeSwapForeignBlocksEnabled=True mode.
- **valFlag**: (uint8) 0x81 for a valid block. The field is padded with blank bytes (0xFF) to the virtual page size boundary.
- **invalFlag**: (uint8) Value 0x18 in this field indicates that the block was invalidated. The field is padded with blank bytes (0xFF) to the virtual page size boundary.

3.6.2 Memory Dump Example

The table below shows an example of the cluster dump:

- One group of two clusters is configured.
- The first cluster has start address 0x2000 (logical address), length: 0x4000.
- The second cluster has start address 0x8000 (logical address), length: 0x2000.
- Two blocks are written.
- The length of the first block is 4.
- The length of the second block is 64.

Table 3.6 Dump memory of first cluster example (Fee_VirtualPageSize = 8)

Offset	4 bytes				4 bytes				4 bytes				4 bytes				Description
0000	00	00	00	01	00	00	20	00	00	00	40	00	00	01	00	01	Clr Hdr
0010	81	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	Clr Sts
0020	00	01	00	04	00	00	FF	F8	00	01	FF	FD	FF	FF	FF	FF	Blk1 Hdr
0030	81	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	Blk1 Sts
0040	00	02	00	40	00	00	FF	B8	00	01	FF	FA	FF	FF	FF	FF	Blk2 Hdr
0050	81	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	Blk2 Sts
	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	...
	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	...
	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	...
	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	...
	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	...
	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	...
	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	...
FFB0	FF	FF	FF	FF	FF	FF	FF	FF	01	01	01	01	01	01	01	01	Blk2 Data
FFC0	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	...
FFD0	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	...
FFE0	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	...
FFF0	01	01	01	01	01	01	01	01	00	00	00	00	FF	FF	FF	FF	Blk1 Data

Note: Values 0x00 and 0x01 represent example application data stored in blocks 1 and 2 respectively, and 0xFF is the value of unprogrammed (erased, blank) flash memory byte.

3.6.3 FEE Block Always Available

To be consistent with AUTOSAR Requirement **SWS_Fee_00153** (*When a block write operation is started, the FEE module shall mark the corresponding block as inconsistent.*) and **SWS_Fee_00154** (*Upon the successful end of the block write operation, the block shall be marked as consistent (again)*) in case of reset, power loss etc. occur between the writing of the first part of the header (including the checksum) and the writing of the valid flag of the header, neither newly nor previously written data is available.

This is the default behavior of the driver if "FEE Block Always Available" configuration parameter is set to FALSE.

This behaviour can be modified (thus losing compliance with AUTOSAR requirement SWS_Fee_00153 and SWS↩ _Fee_00154) and if a previous valid instance of the block exists, it is always possible to recover it.

3.6.4 Managing Cluster and Block Consistency

The FEE module shall manage the consistency of blocks when catastrophic events occur. If such event occurs (i.e. a power down or reset when an erase/write operation is ongoing), FEE driver shall be able to recover the last valid instance of the blocks stored in flash, ignoring the possible last block update interrupted by the reset. **In case of reset or power down, the flash peripheral aborts any high voltage operation, it can lead to ECC errors in some flash locations.** During start-up, the FEE driver, will scan the memory (header region) in order to restore the cluster and blocks status. If some blocks header contain ECC errors an IVOR exception is thrown during the read operation and FLS driver will manages it.

To achieve the same robustness as in the previous platforms, the FeeVirtualPageSize in 57*xx*s should be updated accordingly:

- If optimized ECC handling is available in the FLS driver **and** if only code flash segments are configured, the FeeVirtualPageSize can be set to 8 bytes (1 DW).
- Otherwise, 4 DW (32 bytes) must be used.

This is due to the fact that the EER bit (used to verify if ECC is present) is affected by 4 DW, regardless on the ECC size which is just 1 DW.

The driver behaves differently depending on which operation was interrupted:

- a cluster swap is ongoing,
- a block update is ongoing,
- an immediate block update is ongoing.

3.6.5 Cluster Swap

The cluster swap is a way how unlimited erase/write cycles required by AUTOSAR are implemented. A cluster swap occurs in the following cases:

- when the active cluster has not enough free space to host the writing of new block.
- when a flash job has failed during FEE initialization stage.
- when the last header is corrupted∗ (wrong checksum, parsed block doesn't match with configuration, unknown block number).

- when trying to execute write or erase immediate jobs on address damaged for aging of the flash.

The swap consists of the following steps (stages):

- Erase the next cluster (ERASING stage).
- Write the first part of the cluster header (16 byte: incremented ClrID, StartAddress, ClusterSize, Checksum) (FORMATTING stage).
- Copy the last valid instances of all blocks (header, data and status) also the block that generated the cluster swap (old instance) (COPYING stage).
- Write the second part of cluster header (16 bytes: valid/invalid flag) (ACTIVE stage);
- Write the block that generated the cluster swap (new instance) (UPDATING stage) to the newly allocated cluster.

Table 3.7 Cluster Swap Stages

CLUSTER	STAGE 1	STAGE 2	STAGE 3	STAGE 4	STAGE 5
ID 0001	ACTIVE	ACTIVE	ACTIVE	OLD	OLD
ID 0002	ERASING	FORMATTING	COPYING	ACTIVE	UPDATING

A system reset happens during STAGE 1
Since an erase operation may have been interrupted, the next cluster could be affected by ECC error.

A system reset happens during STAGE 2 and STAGE 4.
Since a program operation may have been interrupted, the next cluster header could be affected by ECC error.

A system reset happens during STAGE 3
Since a program operation may have been interrupted, the next cluster area could be affected by ECC error.

A system reset happens during STAGE 5.
Since a program operation may have been interrupted, the next cluster area could be affected by ECC error. The active cluster is the one with ID 0002.
If the cluster swap process is broken in any of the preceding stages, the following happens during next startup:

- The application should runs the FEE initialization phase by calling Fee_Init and then repeatedly Fee_Main↩ Function and Fls_MainFunction until the driver is in the idle state.
- During this phase the active cluster is recognized and it is not affected by ECC error.
- Only the block that caused cluster swap is lost. It the application writes any block, a new cluster swap is initiated. This will erase the cluster again and remove the ECC error wherever it is.

3.6.6 Block Update

During normal execution (without any catastrophic event) the consistency of data block is assured by the order in which the block fields are written to the memory. The block updating process consists of the following steps:

- writing the BlkId, StartAddress Length and CheckSum fields in the header area;

- writing Data in the data area;
- updating the Status of block from INCONSISTENT to VALID.

In case of a catastrophic event during block updating after powering the system again a FEE Initialization phase should be re-executed after power up the system. During this phase:

- the active cluster will be selected;
- the header blocks zone will be scanned in order to restore the status of blocks before the power down.

If more instances of the same block are present in the cluster, only the instance with highest address is kept as valid.

If FEE_MARK_EMPTY_BLOCKS_INVALID is ON: If an ECC error occurs, Fee_Init considers the block affected invalid, and keeps the previous instance. If there is no block instance, the block is considered invalid either.
If FEE_MARK_EMPTY_BLOCKS_INVALID is OFF: If an ECC error occurs, Fee_Init considers the block affected inconsistent, and keeps the previous instance. If there is no block instance, the block is considered inconsistent.

3.6.7 Immediate Block Update

Immediate data are used for fast write operations, because no swap operation can occur during write (when used properly). Typical use case is storage of crash; related data. Unfortunately the AutoSAR specification is not 100% unambiguous regarding this feature and its use rules.

A part of each cluster is reserved for this kind of data. The size of this area is computed from the configuration to be able to hold one instance of all immediate data blocks. It is not located in any predefined static area.

Immediate data blocks are usually stored exactly the same way as standard blocks. No pre-allocation of block private data area is performed in the Fee_EraseImmediateBlock function. Only if the given immediate block has already been stored in the reserved area, cluster swap is performed. Standard blocks cannot be saved to the reserved area at all.

During normal execution (without a catastrophic event) immediate data blocks are updated in two steps:

1. **First phase**: By calling Fee_EraseImmediateBlock (and appropriate amount of Fee_MainFunction/Fls_↩ MainFunction calls), the cluster usage is checked.
   - If the affected immediate data block has not been stored in the reserved area, it is safe to continue without a cluster swap (there is a space reserved for a single future write operation).
   - If there is already an instance of this block in the reserved area, a cluster swap is performed. As a result, a copy of this old block instance will be stored in the unreserved area, and a new instance can be later written without any delay.

   Note: This step is mandatory.
2. **Second phase**: Write the actual data by calling the Fee_Write function. The write operation is exactly the same as for the standard data blocks.

To enhance compatibility with the legacy mode, it is possible to configure the Fee_EraseImmediateBlock function to explicitly invalidate the previous instance of the given block. It is effectively the same as hiding the old instance

3.6.8 Det Errors Description

Table 3.8 Det Errors detailed description

Name	Value [hex]	Description
FEE_E_UNINIT	0x01	API service called when module was not initialized. Please see the AUTOSAR specifications for further details.
FEE_E_INVALID_BLOCK_NO	0x02	API service called when FEE Block Number is invalid. Please see the AUTOSAR specifications for further details.
FEE_E_INVALID_BLOCK_OFS	0x03	API service called when Block Offset is invalid. Please see the AUTOSAR specifications for further details.
FEE_E_PARAM_POINTER	0x04	API service called when input Data Pointer is invalid. Please see the AUTOSAR specifications for further details.
FEE_E_INVALID_BLOCK_LEN	0x05	API service called when input Block Length is invalid. Please see the AUTOSAR specifications for further details.
FEE_E_BUSY	0x06	API service called when FEE module is busy processing a user request. Please see the AUTOSAR specifications for further details.
FEE_E_BUSY_INTERNAL	0x07	API service called when FEE module is busy doing internal management operations. Please see the AUTOSAR specifications for further details.
FEE_E_INVALID_CANCEL	0x08	API service called when no job was pending. Please see the AUTOSAR specifications for further details.
FEE_E_INIT_FAILED	0x09	API API Fee_init failed.
FEE_E_CANCEL_API	0x0A	User called the Fee_Cancel() function but the "Fee Cancel API" configuration parameter is set to off
FEE_E_CLUSTER_GROUP_IDX	0x0B	Error reported by APIs Fee_GetRunTimeInfo and Fee_ForceSwapOnNextWrite when they are called with invalid(out of range) parameter.
FEE_E_FOREIGN_BLOCKS_OVF	0x0C	Error reported during scanning of the data flash as part of processing of Fee_Init job when Fee finds in data flash more blocks than the maximum number statically configured. It happens when the number of foreign blocks found in data flash is equal or higher than FEE_MAX_NR_OF_BLOCKS - FEE_CRT↩ _CFG_NR_OF_BLOCKS. The error code is only reported with FEE_SWAP_FOREIGN_BLOCKS_↩ ENABLED=ON.To fix this DET error the configurator must configure FEE_MAX_NR_OF_BLOCKS to a higher value.

3.6.9 Endurance calculation

Below is an example of calculating the FEE endurance. **Assumption:** The number of write times is the same for all blocks.

Table 3.9 INPUT VALUES

Name	Value
BLOCK1_SIZE	32
BLOCK2_SIZE	64
BLOCK3_SIZE	16
FLASH_ALLOCATED_SIZE	65536
FEE_VIRTUAL_PAGE_SIZE	8
FLASH_CELL_ENDURANCE	1000

Table 3.10 ASSUMED INPUT

Name	Value
NO_OF_CLUSTER_GROUPS	1
NO_OF_CLUSTERS	2
RSRVD_CLUSTER_GAP	64

Table 3.11 INTERMEDIATE OUTPUT

Name	Value
FEE_BLOCK_OVERHEAD	32
FEE_CLUSTER_OVERHEAD	32
ALIGNED_BLOCK1_SIZE	32
ALIGNED_BLOCK2_SIZE	64
ALIGNED_BLOCK3_SIZE	16
BLOCK1_INSTANCE_TOTAL_SIZE	64
BLOCK2_INSTANCE_TOTAL_SIZE	96
BLOCK3_INSTANCE_TOTAL_SIZE	48

Intermediate output formula:
FEE_BLOCK_OVERHEAD = *ceiling(12 / FEE_VIRTUAL_PAGE_SIZE + 2) ∗ FEE_VIRTUAL_↩ PAGE_SIZE*
FEE_CLUSTER_OVERHEAD = *ceiling(16 / FEE_VIRTUAL_PAGE_SIZE + 2) ∗ FEE_VIRTUAL_↩ PAGE_SIZE*

ALIGNED_BLOCKx_SIZE = *floor((BLOCK_SIZE + FEE_VIRTUAL_PAGE_SIZE - 1)/FEE_↩ VIRTUAL_PAGE_SIZE) ∗ FEE_VIRTUAL_PAGE_SIZE*
BLOCKx_INSTANCE_TOTAL_SIZE = *FEE_BLOCK_OVERHEAD + ALIGNED_BLOCKx_SIZE*
RSRVD_CLUSTER_GAP = *2 ∗ FEE_BLOCK_OVERHEAD*
BLOCKS_INSTANCE_TOTAL_SIZE = *Total of BLOCKx_INSTANCE_TOTAL_SIZE*

Table 3.12 ENDURANCE

Name	Value
AVAILABLE_WRITE_CYCLES	312,144.00

Endurance formula:
AVAILABLE_WRITE_CYCLES = *floor(((FLASH_ALLOCATED_SIZE -(FEE_CLUSTER_OVERHEAD + RSRVD_CLUSTER_GAP + BLOCKS_INSTANCE_TOTAL_SIZE + 1) ∗NO_OF_CLUSTERS∗NO_OF_↩ CLUSTER_GROUPS)/BLOCKS_INSTANCE_TOTAL_SIZE)∗FLASH_CELL_ENDURANCE)*

3.6.10 Configuration tips

Task scheduling

Make sure that no FEE/FLS functions are interrupted by another FEE/FLS function except Fee_Cancel.

Example 1: Fee_MainFunction is called from 10 ms OS task and Fee_Write function is called from 20 ms OS task. It has to be ensured that Fee_Write function is not interrupted by Fee_MainFunction.

Example2: Fee_MainFunction is called from several places in the application. It has to be ensured that Fee_Main↩ Function is not interrupted by another Fee_MainFunction.

Time consumption

1. Be aware of the maximum time consumption of the FEE/FLS functions (the best is to use actual configuration for performance analysis).

   Example 1: If Fee_MainFunction execution takes up to 5 ms (e.g. when a cluster swap occurs), it is risky to call it from an 1 ms task.

   Example 2: Write operation takes 5 ms to complete. If a cluster swap occurs during the write operation, it may take 500 ms.

2. Be aware of the maximum number of FEE/FLS main function calls or overall time needed for operations (read, write, . . .) to finish. Again, the best is to use actual configuration for performance analysis.

   Example 1: If "FLS erase blank check" is enabled, 64 KiB clusters are used and "Max erase blank check" is set to 256 bytes, it may take up to 600 FEE/FLS main function calls to complete the swap operation. I.e. if the main functions are called within a 10 ms task, it takes up to 6 s to finish the swap operation.

   Note: presented timing numbers are just an example, please refer to the profiling report to get real numbers.

FEE Virtual Page Size

This parameter must reflect the granularity at which the Fls driver is able to distinguish ECC events. The ECC page is 8 bytes in standard flash memory, and 4 bytes in DFO memory. To speed up flash read access, the flash array is accessed using a wider bus (*read page*), which typically consists of 2 or more ECC pages.

For robustness reasons, the Fee management data and user blocks must be aligned at the virtual page boundary. Otherwise an ECC-affected memory location may inadvertently spread the damage to areas belonging to another block.

Note: On 57_xx_ platforms (55nm device) IVOR exception has been suppressed only in Data Flash and a new ECC management has been implemented in such situation, as described also in integration/user manuals for the Fls driver.

To achieve the same robustness as in the previous platforms, the FeeVirtualPageSize in 57_xx_s should be updated accordingly:
If only code flash segments are configured, the FeeVirtualPageSize can be set to 8 bytes (1 DW). Otherwise, 4 DW (32 bytes) must be used.
This is due to the fact that the EER bit (used to verify if ECC is present) is affected by 4 DW, regardless on the ECC size which is just 1 DW.

Also: If data cache is enabled, even if the flash region is configured as cache-inhibited in the memory protection unit, an ECC event might be reported, if there is an ECC affected location inside the same cache line as the location read by driver. This means that an ECC affected location might falsely trigger and report ECC errors for any other location inside the same cache line. To achieve the same robustness as in the previous platforms, the FeeVirtualPageSize should be updated accordingly, to the same size as the cache line (4 DW, 32 bytes).

FEE Block Always Available

According to the AUTOSAR requirement, when the write operation starts, corresponding block is marked as inconsistent. It is marked as valid after successful write. It means that if the write operation is interrupted (canceled, by device reset, power down ...), application cannot access the block data anymore. There is a configuration parameter (FEE Block Always Available) which allows to set the module behavior against this requirement. As a result, the FEE will always provide the last information which is not corrupted(this means block status FEE_BLOCK_VALID or FEE_BLOCK_INVALID).

Example 1: Driver behavior is set according ASR requirement. One instance of the block is successfully written to the memory. Another write operation of this block is scheduled but it is interrupted before finish (by Fee_Cancel or power down). Block can be valid containing old data (operation was interrupted before write process started), block can be valid containing new data (operation was interrupted just before returning success information), or block can be invalid/inconsistent (ongoing write process was interrupted) .

Example 2: Driver behavior is set to violate ASR requirement (not supported by all versions). One instance of the block is successfully written to the memory. Another write operation of this block is scheduled but it is interrupted before finish (by Fee_Cancel or power down). Previous instance of the block is still accessible.

Immediate Data Usage (General)

Immediate data are used for fast write operations, because no swap operation can occur during write (when used properly). Typical use-case is storage of crash-related data. Unfortunately the AutoSAR specification is not 100 unambiguous regarding this feature and its use rules. The following sequence shall be used for the immediate data usage:

1. *Fee_EraseImmediateBlock*: to allocate space for the block in advance. During its processing a swap operation can occur.
2. *Fee_Cancel*: to interrupt any internal processing in case the driver is not idle.
3. *Fee_Write* of an immediate block: write block data; no swap can occur because space is already allocated.

In the course of time, a 2nd slightly different approach has been developed for this Fee module codebase. The older one is called *Legacy Mode* to distinguish it from the current default (standard) one.

Immediate Data Usage in the Non-Legacy Mode (Default)

To streamline immediate data handling and to get rid of wasted space due to calls to the Fee_EraseImmediateBlock function, a part of each cluster is reserved for this kind of data. Size of this area is computed from the configuration to be able to hold one instance of all immediate data blocks. During the ECU lifetime, immediate data blocks are stored exactly the same way as standard blocks. The cluster swap is triggered only if either A standard block does not fit into the remaining free unreserved area, or Given immediate data block has already been stored in the reserved area, or A write request is issued in case some inconsistencies have been detected in the non-volatile data structures stored in the flash memory configured for the Fee module.

Immediate Data Usage in the Legacy Mode

In the *Legacy Mode*, memory area for an immediate data block is allocated as a part of the Fee_EraseImmediateBlock function invocation that must precede write operation for this block.

After the immediate block is written, new space shall be allocated for further usage of this block (Fee_Erase↩ ImmediateBlock function). Note that after Fee_EraseImmediateBlock function is called, new space is allocated and the previous instance of the immediate block is no longer accessible.

Example of usage 1:

```
FEE/FLS initialization.
Space reservation for immediate blocks (Fee_EraseImmediateBlock).
Execution of the application code.
Abnormal situation occurs.
All fee operations are stopped (Fee_Cancel).
Immediate blocks are written.
Power down/reset...
FEE/FLS initialization.
Check if immediate data are written. If so, some abnormal situation occurred. The stored
information can be used.
Space reservation for immediate blocks - all previous data of these blocks are lost.
```

Example of usage 2:

```
Space reservation is done using flash image.
App is running and writing immediate data in case of some special situation (each
immediate block is written just once).
When all immediate blocks are written, unit is not writing any immediate data anymore
and report error state.
```

Code flash erase Be aware of the read-while-write support in the device when erasing the code flash from code flash. If it is not supported, the access code shall be loaded into the RAM (the "FLS Load Access Code On Job Start" parameter must be turned on). In this case the write/erase operation shall be set as synchronous (if asynchronous write/erase is enabled, access code in the RAM is ignored).

Meaning of the FEE block states (result of the Fee_GetJobResult operation)

Table 3.13 FEE Job Information

MemIf_JobResultType	Non-immediate block	Immediate block
MEMIF_JOB_OK	block is valid	block is valid
MEMIF_BLOCK_INVALID	block was invalidated, or block is not in the cluster(if FEE↩ _MARK_EMPTY_BLOCKS_↩ INVALID = ON)	block was invalidated, or block is not in the cluster(if FEE↩ _MARK_EMPTY_BLOCKS_↩ INVALID = ON) space for the block is reserved
MEMIF_BLOCK_↩ INCONSISTENT	block has corrupted header (wrong checksum, address, ...), or block is not in the cluster(if FEE↩ _MARK_EMPTY_BLOCKS_↩ INVALID = OFF) or block doesn't match the configuration (length, immediate, ...)	block has corrupted header (wrong checksum, address, ...), or block is not in the cluster(if FEE↩ _MARK_EMPTY_BLOCKS_↩ INVALID = OFF) or block doesn't match the configuration (length, immediate, ...)

Note: If FEE_BLOCK_ALWAYS_AVAILABLE == STD_ON last valid block will be returned, it means that inconsistent blocks will not be considered.

Note: For immediate data blocks MEMIF_BLOCK_INCONSISTENT is actually one of the working (acceptable) states – the block has been allocated but not written yet, in this situation a requested read operation cannot be performed.

Fls Configuration Constraints: Fls_Cancel() and Asynchronous Mode

The following has been identified during robustness analysis for the Fls / Fee drivers:

- *Fls write/erase in synchronous mode is faster than in asynchronous one (fewer Fee/Fls_MainFunctions are needed).*
- *If Fls asynchronous write is used, then the Fls_Cancel() can trigger some robustness issues. As a counter-measure, if the Fls_Cancel() is supported (enabled), it is not possible to configure flash sectors with asynchronous write for the Fee clusters. Either Fls_Cancel() must be disabled,or a sector with synchronous write mode selected.*
- *The previous item does not apply to asynchronous erase (i.e. it is possible to use asynchronous erase mode for better performance even if the Fls_Cancel is enabled).*

Cluster versus total block size configuration

When configuring a cluster, the integrator must consider the following: the size of the cluster must be greater than the sum of:

- one instance size for each block configured in the respective cluster plus
- the size of the largest block instance configured in the respective cluster plus
- all the management information required by the FEE to store those blocks.

This is the minimum cluster size required relative to the blocks size, but it is recommended to the integrator to design the memory layout in such a way that the cluster swap is performed as rarely as possible (eg. there is enough space in the cluster to write several block instances before there is the need to swap clusters). The disadvantages of the cluster swap are the time consumption (user operations such as read/write might get delayed until the swap is finished), the excessive consumption of available erase/program cycles of the flash memory, and the fact that because of the long duration of a swap there is a higher probability to get interrupted by power-on resets which can lead to ECC errors.Another important factor to consider when designing the memory layout is the write frequency of data.

3.6.11 Sector managerment and sector retirement

The "FEE sector managerment and sector retirement" feature can be enabled or disabled by the parameter **FeeSectorRetirement**

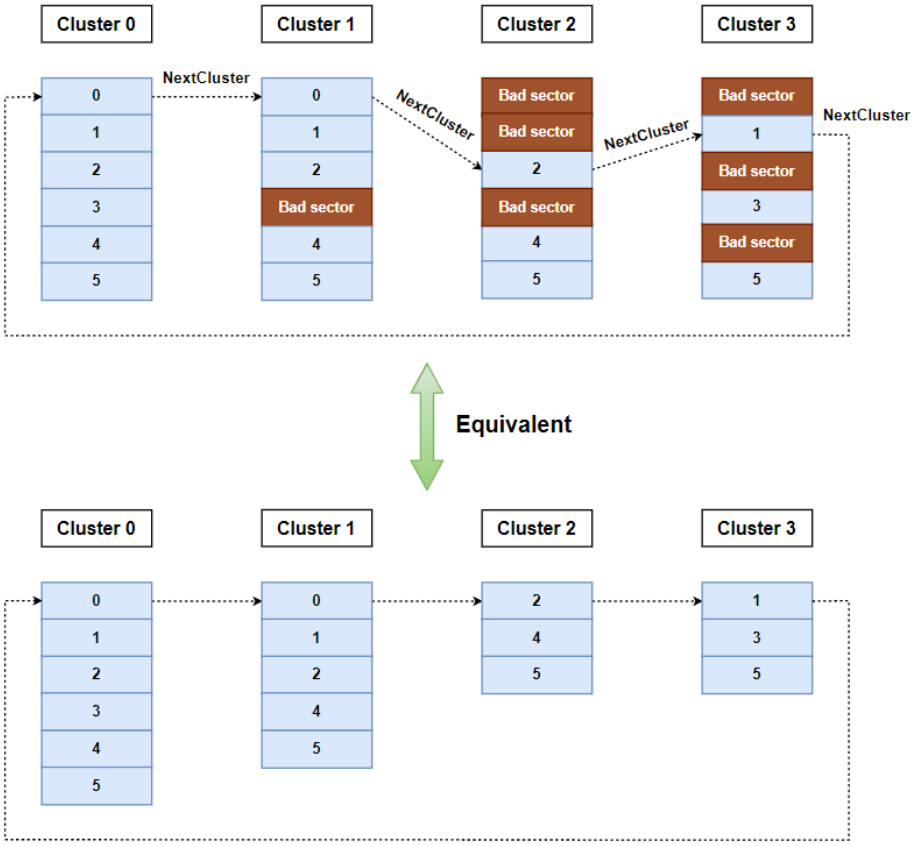
- This feature was developed to support the following customer use case: "CPR_RTD_00505.fee"
- Add support for bad Sector Management and Sector Retirement
- Some flash memory specs have low endurance/write cycles availability, which needs to be improved by the ability to retire bad sectors or locations affected by permanent ECC, at the expense of allocated flash size - the endurance being a ppm specification
- This involves removing Sectors from configuration when they become unusable

The following parameters are used in **FeeSectorRetirement** mode:

- **FeeSectorEraseRetries**
  - Number of attempts to erase each sector in the cluster when swapping in sector retirement mode.
  - A sector will be marked as a bad sector after a number of attemps to erase all have failed.
- **FeeSectorToRecover**
  - Container for the list of sectors to recover
  - Specifies which sectors will be restored after replacing them on the configuration

3.6.11.1 Introduction

In this mode, the Fee driver can have ability to detect a bad sectors and skip them in runtime:



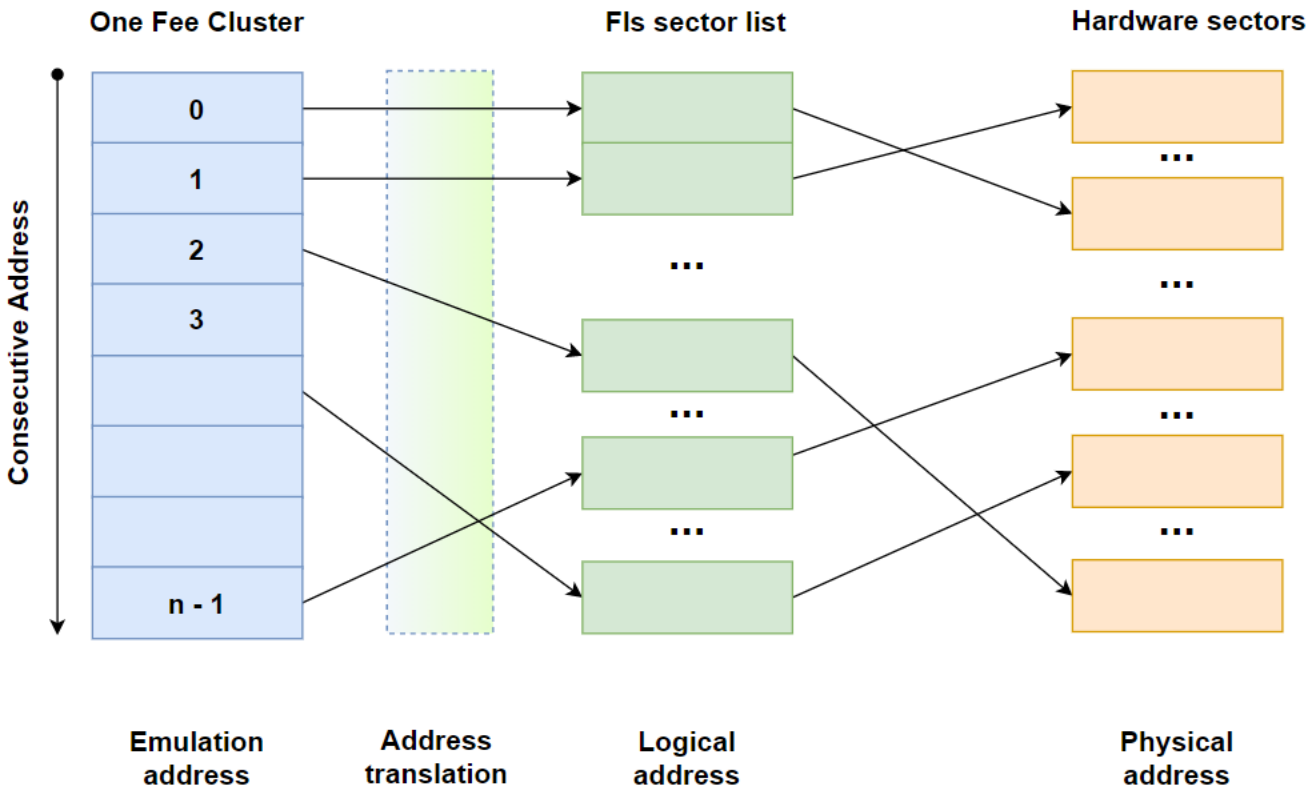
In order to support this feature, the following items have been implemented:

1. Support for in-consecutive sectors addresses in each cluster
   - Because of the sector retirement, the logical address cannot be consecutive
2. Bad sectors management
   - Bad sector criteria
   - Store and retrieve the information of sectors status

3.6.11.2 Discontinuous logical sector address

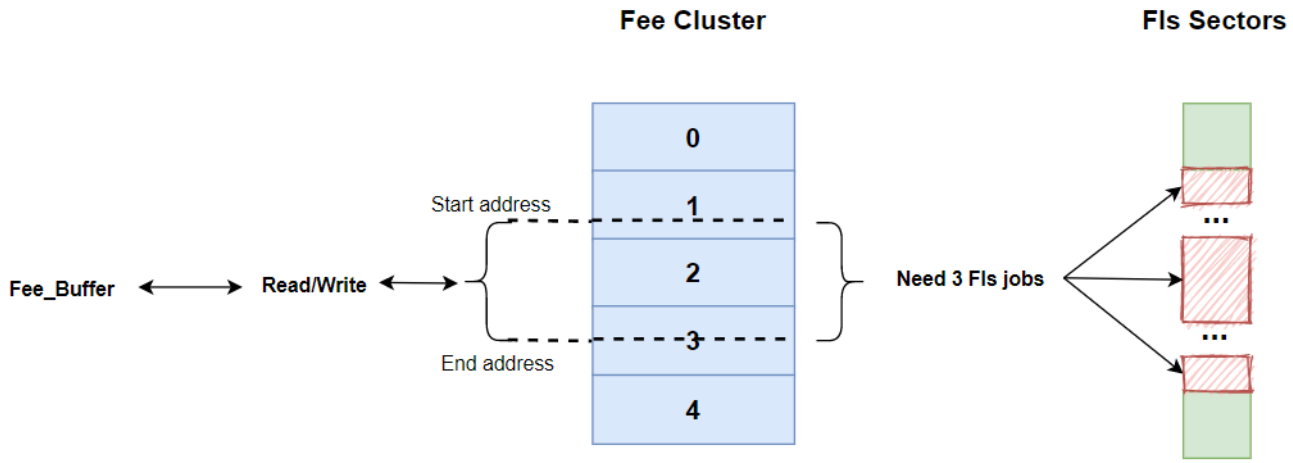
From Fee's point of view:

- The cluster address (called emulation address) is always consecutive: starts from 0 to (cluster size – 1)
- An extra layer will handle the address translation

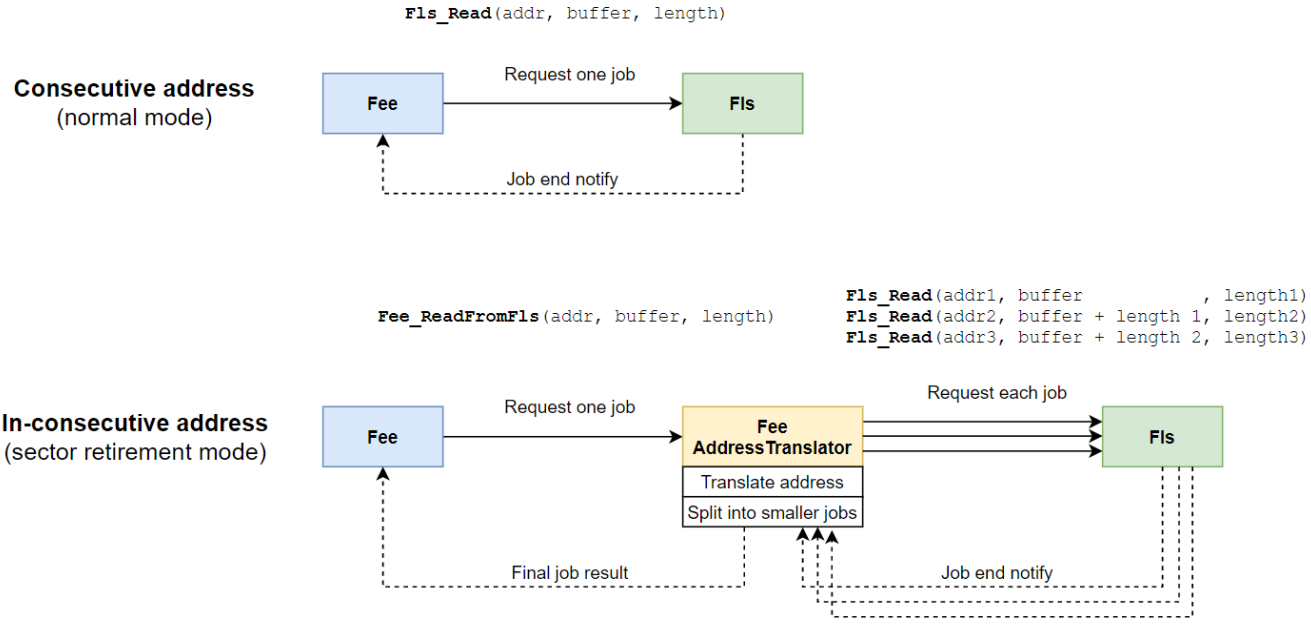


To support this, the Fee driver needs to:

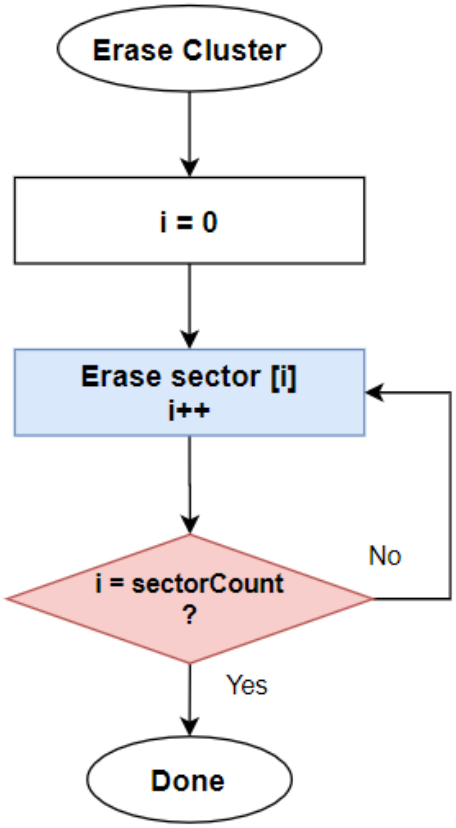
- Generate the information for each sector, including the start address and size in Fls layer
- Translate address from one Fee cluster to Fls sectors
- Calculate the logical address corresponding to the emulation address (based on the generated sectors configuration)
  - Each part will be linked to its sector located in Fls layer
  - In this example, the data Fee requested is located at 3 discontinuous sectors
  - The translation layer needs to calculate the start address and length for each job



- A job (read/write) in a cluster might be splitted into smaller jobs to handle the fragmented data localed in discontinuous sectors. The translation layer will be responsible for:
  - Process each job result
  - Stop immediately when a part of job fails
  - Report the final job result to Fee



- For erase operation:
  - Erase sectors one by one
  - The erase operation results will be used to detect and consider each sector is bad or not (bad sectors criteria will be explained in the next section)



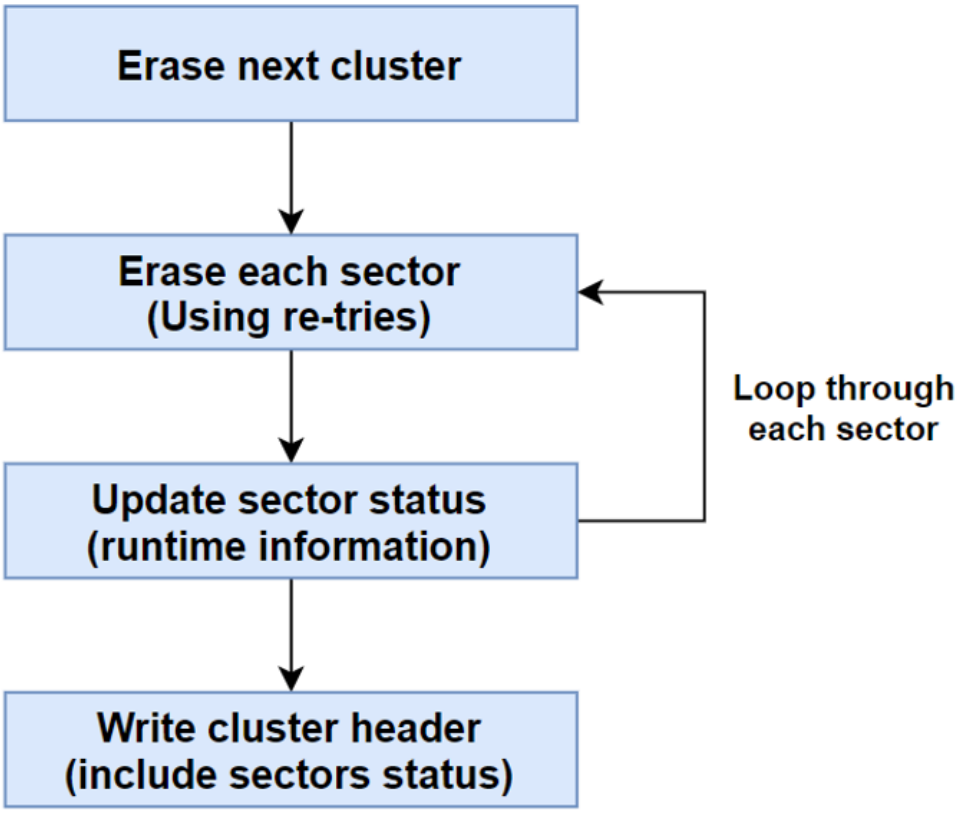
3.6.11.3 Bad sectors criteria

This section describes how the FEE driver manages bad sectors:

- How to detect a bad sector (Which criteria?)
- Where to store information of sectors status (Where?)
- How to retrieve this information? (If the power loss or reset)

Item	Implemented solution
Bad sector criteria	Perfom in the swap stage: when erasing the next cluster
	Use erase operation results to verify a sector is good or bad
	Add a new configuration parameter **FeeSectorEraseRetries** (maximum number of re-erase a sector)
Store sectors status	Store in the cluster header
	- Use 2 byte for the sector count (the number of sectors in the cluster)
	- Use 1 bit / sector to mark its status (1=good, 0=bad)
	This information will be updated each swap stage, and written into the new cluster header
Retrieve sectors status	Perfom in the init stage
	- After the valid cluster header was found
	- Read the sector status section in the cluster header and update the runtime data structure

- A sector will be marked as a bad sector after a number of attemps to erase all have failed:



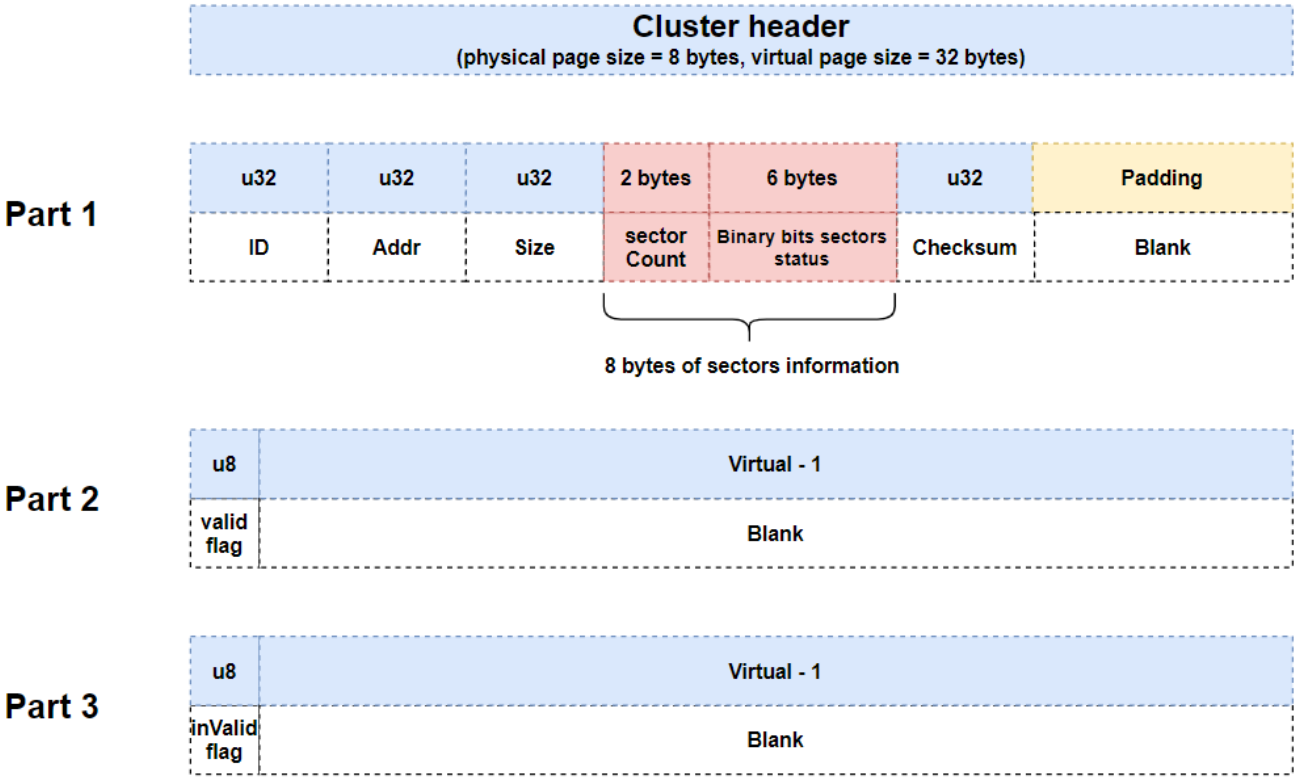
Note

- A cluster will be marked as a bad cluster if all its sectors were bad sectors or its size is not big enough to contain any block data
- Bad cluster will be skipped by the Fee driver in runtime

3.6.11.4 Sectors information

3.6.11.4.1 Storing the sectors information

- Fee driver uses memory space in each cluster header to store the sectors information
- In the example below
  - The Fls physical page size is 8 bytes and the FeeVirtualPageSize is set to 32
  - The number os sectors is 6

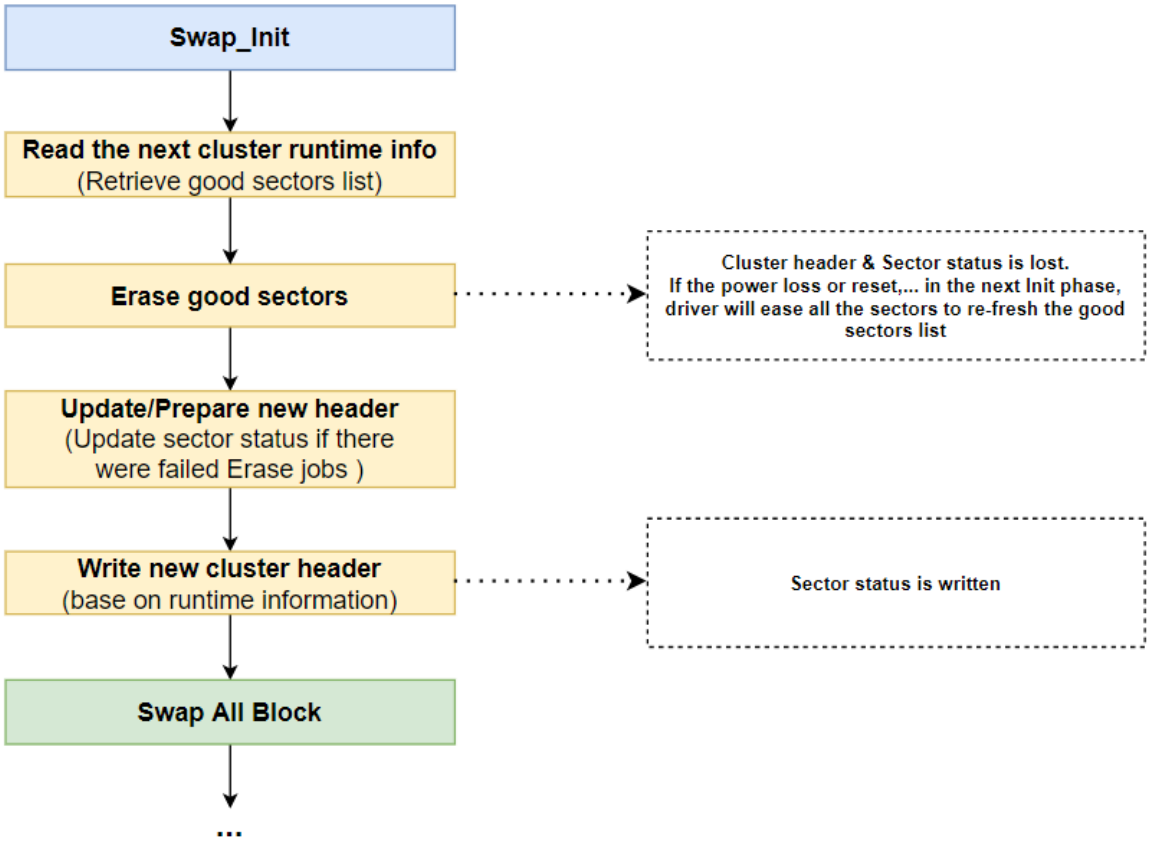


Note

- The size of the sectors information will be rounded up to align with the Fls physical page size
- In the example above, 06 sectors need only 06 bit in the first byte of the binary bit status
- But the cluster header will be expanded to 8 bytes to align with the Fls physical page size (8 bytes)
- The unused bits and unused bytes will be filled with zero value

- The sequence of updating sector status information when erasing an Fee cluster

  1. Read cluster runtime information
  2. Erase all good sector (include the cluster header)
  3. Update sector information
  4. Write new header into flash

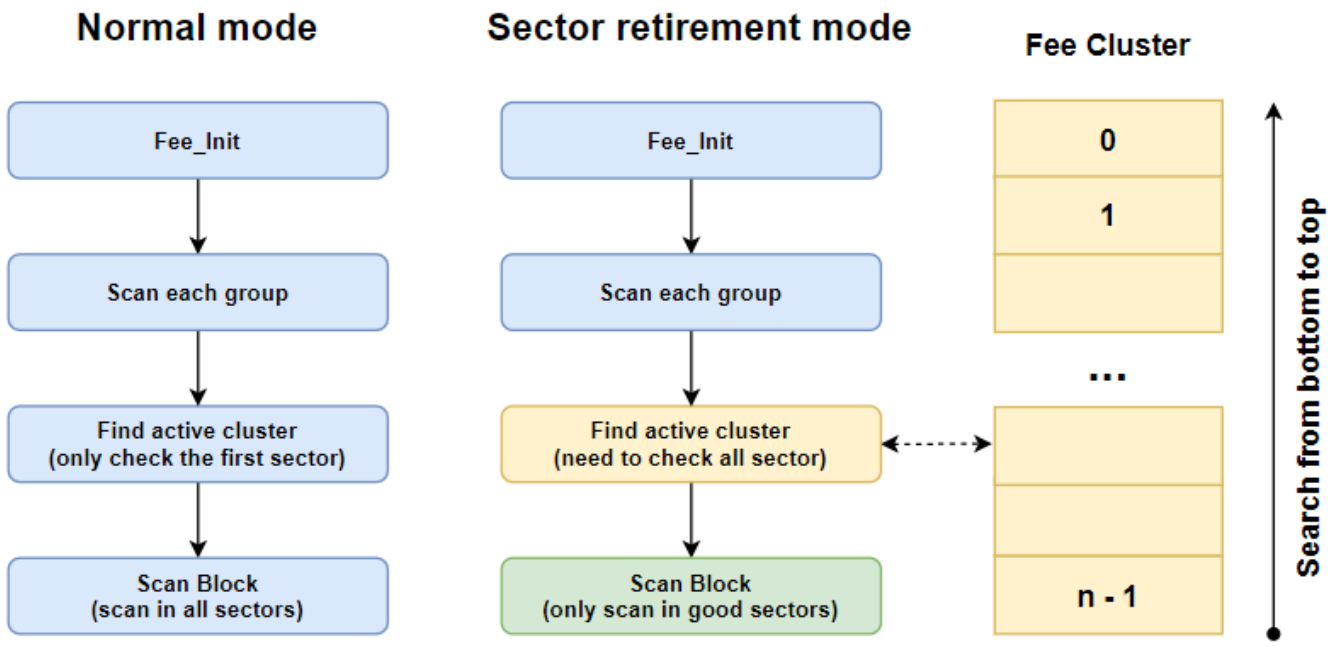


Note

If **FeeSectorRetirement** is ON, the sectors information must be kept in the cluster header, so the data flash layout will not be compatible with layout with previous FEE versions. This means that the first time when this mode is switched to ON, projects must start with a clean erased data flash

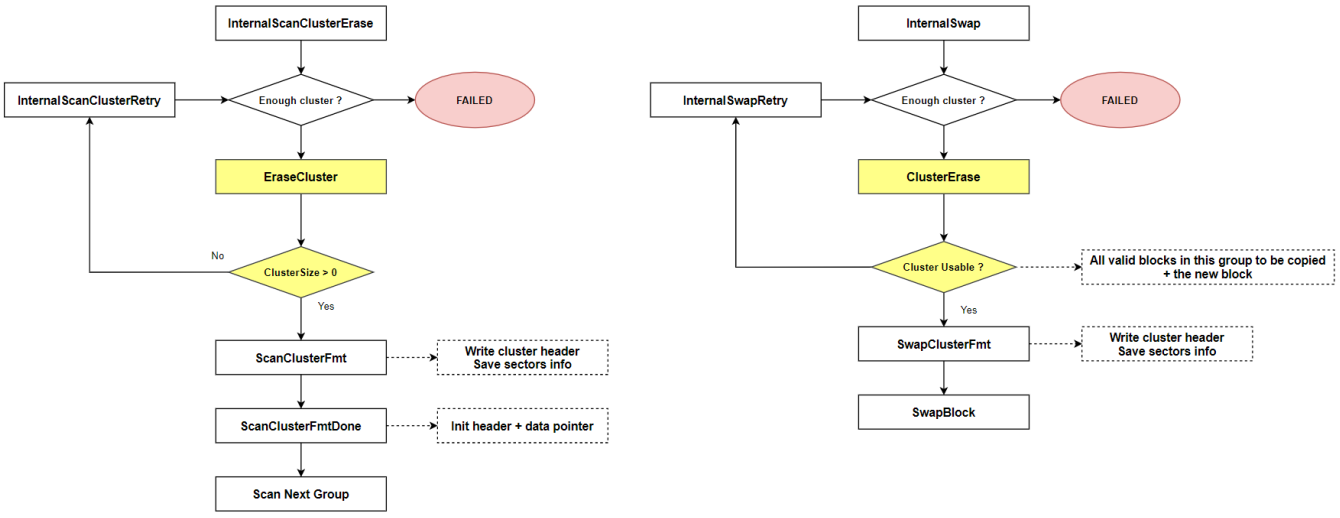
3.6.11.4.2 Achieving the sectors information

- In the initialization stage, when scanning the valid cluster header in each cluster, Fee driver will:
  - Scan every sector from the bottom until find the valid header
  - Parse the cluster header to get the sectors status
  - Only scan block data in good sectors in the active cluster



Note

- In the **FeeSectorRetirement** mode, Fee driver will automatically ignore bad clusters after erasing a cluster in Init and Swap phases
- Fee driver will only notify the upper layer and mark the current job as failed in case there is not enough cluster left in the cluster group

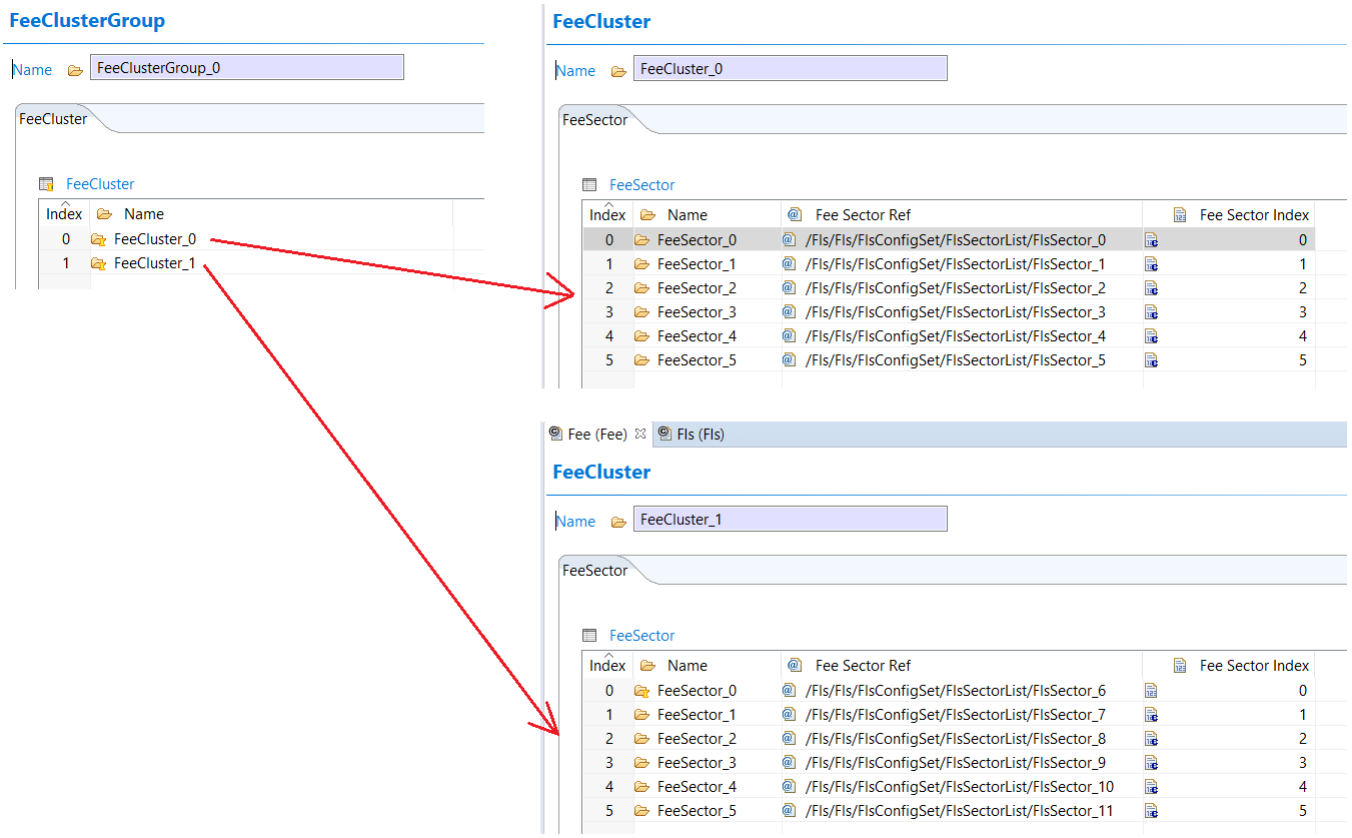


3.6.11.5 Bad sectors replacement

- Bad sectors can be replaced from the configuration by doing the following steps:

1. Get the current runtime information to identify which sectors are bad
2. Replace bad sectors by other good sectors in the configuration tool
   (a) Replace bad sector
   (b) Add the sector to the **FeeSectorToRecover** list
3. Re-initialize the Fee driver
4. Get the runtime information to check if the bad sectors were replaced or not

- The example below is using 01 group with 02 clusters, each cluster has 06 sectors (12 sectors in total)



Step 1: Get runtime information

- In order to get the information of Fee group 0, we need to create the runtime information data structure
- Then allocate the memory array to hold the whole 12 sectors information in this group
- After that, call the function **Fee_GetRunTimeInfo**

Note

- Users have to take care about the memory size of the sector runtime array passing to the function **Fee_GetRunTimeInfo**
- Because the driver will write the exact the number of sectors of the requested group into that array
- In case users do not want to get the sectors information, this must be a null pointer

Code example:

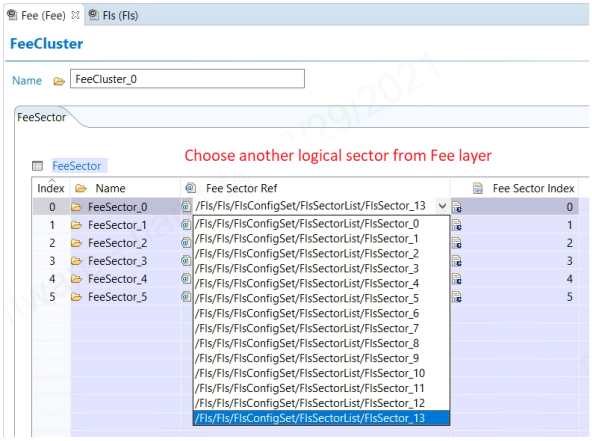
```
/* Allocate memory to contain sectors information for group 0 */

Fee_SectorRuntimeInfoType clusterGroup_0_sectorInfo[12U];

/* Create the runtime information data structure */

Fee_ClusterGroupRuntimeInfoType clusterGroup_0;

/* Point to the array will hold the sector information */

clusterGroup_0.sectorInfo = clusterGroup_0_sectorInfo;

/* Get the runtime information from FEE driver */

Fee_GetRunTimeInfo(0U, &clusterGroup_0);
```

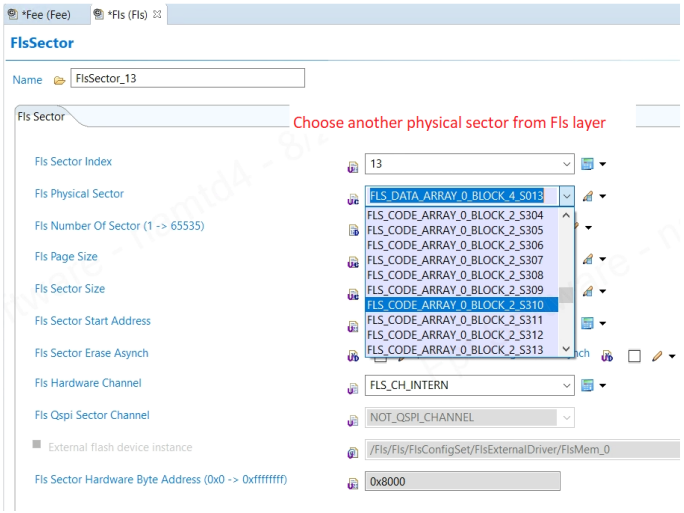
- An example of the data runtime information
  - As we can see the sector 0 of cluster 0 is a bad sector, because its size is zero

Expression	Type	Value
clusterGroup_0	Fee_ClusterGroupRuntimeInfoType	{...}
clusterTotalSpace	Fls_AddressType	49152
clusterFreeSpace	Fls_AddressType	19872
blockHeaderOverhead	uint16	96
virtualPageSize	uint16	32
numberOfSwap	uint32	3
sectorInfo	Fee_SectorRuntimeInfoType *	0x2001ef8c
clusterGroup_0_sectorInfo	Fee_SectorRuntimeInfoType [12]	0x2001ef8c
clusterGroup_0_sectorInfo[0]	Fee_SectorRuntimeInfoType	{...}
sectorAddr	Fls_AddressType	65536
sectorSize	Fls_LengthType	0
clusterGroup_0_sectorInfo[1]	Fee_SectorRuntimeInfoType	{...}
sectorAddr	Fls_AddressType	73728
sectorSize	Fls_LengthType	8192
clusterGroup_0_sectorInfo[2]	Fee_SectorRuntimeInfoType	{...}
sectorAddr	Fls_AddressType	32768
sectorSize	Fls_LengthType	8192
clusterGroup_0_sectorInfo[3]	Fee_SectorRuntimeInfoType	{...}
sectorAddr	Fls_AddressType	0
sectorSize	Fls_LengthType	8192
clusterGroup_0_sectorInfo[4]	Fee_SectorRuntimeInfoType	{...}
sectorAddr	Fls_AddressType	49152
sectorSize	Fls_LengthType	8192
clusterGroup_0_sectorInfo[5]	Fee_SectorRuntimeInfoType	{...}
sectorAddr	Fls_AddressType	40960
sectorSize	Fls_LengthType	8192
clusterGroup_0_sectorInfo[6]	Fee_SectorRuntimeInfoType	{...}
sectorAddr	Fls_AddressType	90112
sectorSize	Fls_LengthType	8192
clusterGroup_0_sectorInfo[7]	Fee_SectorRuntimeInfoType	{...}
sectorAddr	Fls_AddressType	81920
sectorSize	Fls_LengthType	8192
clusterGroup_0_sectorInfo[8]	Fee_SectorRuntimeInfoType	{...}
sectorAddr	Fls_AddressType	24576

Step 2.1: Replace the bad sector by another good sector

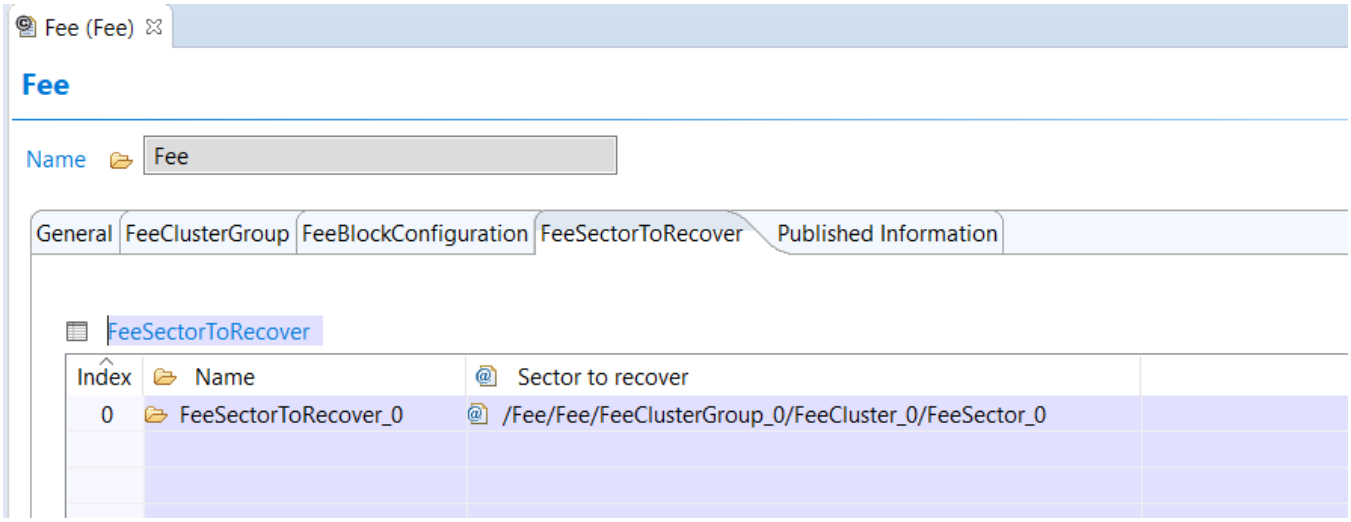
- There are two way: choosing another **logical** or **physical** Fls sector





Step 2.2: Add that sector to the list **FeeSectorToRecover**

- The purpose is tell the Fee driver to recover this sector in the next swap phase



Step 3: Re-initialize the Fee driver by calling **Fee_Init()**

Step 4: Repeat the step 1 to check if the bad sector was replaced or not

Note

- Fee driver will replace bad sectors only in the swap phase, before erasing the next cluster
- Optional: After the bad sectors were recovered, users can remove them from the list **FeeSectorTo↩ Recover**

3.6.12 FEE Swap Foreign Blocks feature

The "FEE Swap foreign blocks" feature can be enabled or disabled by the parameter FeeSwapForeignBlocks↩ Enabled.

The "FEE Swap foreign blocks" feature was developed to support the following customer use case: the customer has two different projects, BOOTLOADER and APPLICATION, which are separately compiled, built into different executable files which are run on the same ECU in different scenarios. This means the two executables share the same data flash emulated for calibration data. For the two projects the customer uses two different FEE block configurations: for the BOOTLOADER project some blocks, for APPLICATION project other blocks, maybe some blocks are common.

In the previous FEE implementation it was not possible to run over same data flash with different FEE configurations because FEE used to keep at cluster swap only blocks which were present in the current configuration. This behavior had the disadvantage that each time a new block was added in the APPLICATION project, the customer was required to update also the FEE configuration from the BOOTLOADER project, even if BOOTLOADER blocks were not added or deleted.

The following parameters are used in FeeSwapForeignBlocksEnabled mode:

- *FeeBlockAssignment Defines in which project this block is used.*
  1. BOOTLOADER - Block is used ONLY by the BOOTLOADER project
  2. APPLICATION - Block is used ONLY by the APPLICATION project
  3. Shared - Block is used for BOTH APPLICATION and BOOTLOADER projects
- *FeeConfigAssignment Defines for which project the current Fee configuration is used.*
  1. BOOTLOADER - Configuration is used only by the BOOTLOADER project
  2. APPLICATION - Configuration is used only by the APPLICATION project
- *FeeMaximumNumberBlocks This must be configured to total number of BOOTLOADER,APPLICATION blocks and also some buffer for blocks added in the future project versions. This parameter will be used to statically allocate space for the foreign block information, so it should be configured to accommodate the total number of blocks running on the ECU in all project versions. Example of configuration: If APPLICATION uses 2 blocks, boot loader 1 block and another 1 is shared between APPLICATION and BOOTLOADER then this parameter must be configured to 2(only appl) + 1(only boot loader) + 1(shared) + X, where x is the maximum number of blocks that it is estimated to be added in future project versions.*

With FEE in the mode "FeeSwapForeignBlocksEnabled" it is possible to have different block configuration for the BOOTLOADER and APPLICATION projects. This is possible by swapping also the foreign blocks. A block is foreign if it has FeeBlockAssignment=BOOTLOADER if the FeeConfigAssignment is APPLICATION or if it has FeeBlockAssignment=APPLICATION if the FeeConfigAssignment is BOOTLOADER.

The mode "FeeSwapForeignBlocksEnabled" is useful in the development phase, **but it is not recommended to be used in production phase** , as it has the following disadvantages:

- It increases the cluster swap overall timing
- It increases the RAM consumption because it needs to allocate management data for the foreign blocks as well
- It adds the possibility of a new error FEE_E_FOREIGN_BLOCKS_OVF in case the configuration of Fee↩ MaximumNumberBlocks is incorrect or in case Fee finds in data flash more blocks than it is allowed by configuration FeeMaximumNumberBlocks. This can happen for example in case of some eccs in data flash block header block assignment data and Fls_DsiHandler is not used. More information about Fls_DsiHandler usage can be found in the FLS driver IM.

Nevertheless, it can be used in the production phase if the above points are not critical for the project.

Restrictions for configuring APPLICATION and BOOTLOADER projects:

- If the configurator changes a block which affects the configuration of the other project then he must apply this change to both configurations. This means:
  - if a SHARED block is changed to APPLICATION only in the APPLICATION configuration or BOOTLOADER only in the BOOTLOADER configuration, then the change must be applied in the other configuration also(delete block from the other configuration)
  - if a non-SHARED block becomes SHARED, then the change must be applied to both configurations The configurator must ensure that block numbers used for BOOTLOADER-only blocks and APPLICATION-only blocks are different.
- If a block is used in both BOOTLOADER and APPLICATION(shared), the block must have the same attributes(block number, size, immediate/non-immediate) in both configurations.
- A block is defined by the following attributes: number, size, immediate/not immediate characteristic. If a part of this information changes in the configuration then the block will not be copied during the cluster swap.
- All configurations for BOOTLOADER and APPLICATION projects must be the same, except configuration of not-shared blocks which must be different.
- All immediate blocks must be defined as shared between APPLICATION and BOOTLOADER if FEE_↩ LEGCY_MODE=OFF is used. The reason is that the FEE must have the same reserved area for both configurations.
- The configurator must ensure that the total APPLICATION and BOOTLOADER blocks size with FEE management information included can be accommodated by the cluster size. This might be managed in 2 ways:
  - Use only SHARED and APPLICATION blocks, don't used BOOTLOADER blocks. This means all BOOTLOADER blocks are included in the APPLICATION configuration, even if they are not written/read by the application project. If the bootloader configuration needs to change, application needs to be reconfigured as well. This will ensure that the size restriction will be checked by the TRESOS tool at APPLICATION configuration time.
  - Temporarily add BOOTLOADER blocks to the application configuration only to check the size restriction

If FeeSwapForeignBlocksEnabled is ON, the block assignment information must be kept in the block header, so the data flash layout will not be compatible with layout with previous FEE versions. This means that the first time when this mode is switched to ON, projects must start with a clean erased data flash.

3.7 Runtime errors

- The driver supports runtime generation of the errors listed in the table:

Error code	Function	Condition triggering the error
FEE_E_BUSY	Fee_Init()	This function called while module is busy
	Fee_SetMode()	
	Fee_Read()	
	Fee_Write()	
	Fee_InvalidateBlock()	
	Fee_EraseImmediateBlock()	
	Fee_GetRunTimeInfo()	
FEE_E_INVALID_CANCEL	Fee_Cancel()	Fee_Cancel called while no job was pending

3.8 Symbolic Names Disclaimer

All containers having symbolicNameValue set to TRUE in the AUTOSAR schema will generate defines like:

#define <Mip>Conf_<Container_ShortName>_<Container_ID>

For this reason it is forbidden to duplicate the names of such containers across the RTD configurations or to use names that may trigger other compile issues (e.g. match existing `#ifdefs` arguments).

Chapter 4

Tresos Configuration Plug-in

This chapter describes the Tresos configuration plug-in for the driver. All the parameters are described below.

- Module Fee
  - Container FeeClusterGroup
    - Container FeeCluster
      - Container FeeSector
      - Parameter FeeSectorIndex
      - Reference FeeSectorRef
  - Container FeeBlockConfiguration
    - Parameter FeeBlockNumber
    - Parameter FeeBlockSize
    - Parameter FeeImmediateData
    - Parameter FeeNumberOfWriteCycles
    - Parameter FeeBlockAssignment
    - Reference FeeClusterGroupRef
    - Reference FeeDeviceIndex
  - Container FeeSectorToRecover
    - Reference FeeSectorToRecoverRef
  - Container FeeGeneral
    - Parameter FeeDevErrorDetect
    - Parameter FeeEnableUserModeSupport
    - Parameter FeeMainFunctionPeriod
    - Parameter FeeNvmJobEndNotification
    - Parameter FeeNvmJobErrorNotification
    - Parameter FeeClusterFormatNotification
    - Parameter FeePollingMode
    - Parameter FeeSetModeSupported
    - Parameter FeeVersionInfoApi
    - Parameter FeeVirtualPageSize

        * Parameter FeeDataBufferSize
        * Parameter FeeBlockAlwaysAvailable
        * Parameter FeeLegacyEraseMode
        * Parameter FeeSwapForeignBlocksEnabled
        * Parameter FeeMarkEmptyBlocksInvalid
        * Parameter FeeConfigAssignment
        * Parameter FeeMaximumNumberBlocks
        * Parameter FeeSectorRetirement
        * Parameter FeeSectorEraseRetries
    - Container CommonPublishedInformation
        * Parameter ArReleaseMajorVersion
        * Parameter ArReleaseMinorVersion
        * Parameter ArReleaseRevisionVersion
        * Parameter ModuleId
        * Parameter SwMajorVersion
        * Parameter SwMinorVersion
        * Parameter SwPatchVersion
        * Parameter VendorApiInfix
        * Parameter VendorId
    - Container FeePublishedInformation
        * Parameter FeeBlockOverhead
        * Parameter FeePageOverhead

4.1 Module Fee

Configuration of the Fee (Flash EEPROM Emulation) module.

Included containers:

- FeeClusterGroup
- FeeBlockConfiguration
- FeeSectorToRecover
- FeeGeneral
- CommonPublishedInformation
- FeePublishedInformation

Property	Value
type	ECUC-MODULE-DEF
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantSupport	false
supportedConfigVariants	VARIANT-PRE-COMPILE

4.2 Container FeeClusterGroup

Vendor specific: Configuration of cluster group specific

parameters for the Flash EEPROM Emulation module.

Included subcontainers:

- FeeCluster

Property	Value
type	ECUC-PARAM-CONF-CONTAINER-DEF
lowerMultiplicity	1
upperMultiplicity	Infinite
postBuildVariantMultiplicity	false
multiplicityConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE

4.3 Container FeeCluster

Vendor specific: Configuration of cluster specific

parameters for the Flash EEPROM Emulation module.

Included subcontainers:

- FeeSector

Property	Value
type	ECUC-PARAM-CONF-CONTAINER-DEF
lowerMultiplicity	2
upperMultiplicity	Infinite
postBuildVariantMultiplicity	false
multiplicityConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE

4.4 Container FeeSector

Vendor specific: Configuration of sector specific

parameters for the Flash EEPROM Emulation module.

Included subcontainers:

- None

Property	Value
type	ECUC-PARAM-CONF-CONTAINER-DEF
lowerMultiplicity	1
upperMultiplicity	Infinite
postBuildVariantMultiplicity	false
multiplicityConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE

4.5 Parameter FeeSectorIndex

Vendor specific: Fee Sector Index is an invariant index, used to order Fee sectors and loop over them in the correct, configured order. Its value should be equal with the position of the configured sector inside the configured sector list (the same value as the shown index).

Rationale: The generated .epc configuration might reorder the flash sectors(alphabetically), thus the index parameter changes, becoming out of sync with the real intended order .

Range:

min = 0

max = 65534

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	0
max	65534
min	0

4.6 Reference FeeSectorRef

Vendor specific: Reference to a logical Fls sector

the Fee cluster consist of.

Note: If the Fls_Cancel API is enabled in the FLS driver used by the FEE, the Fls Page Write Asynch

mode cannot be used for flash sectors. Disable Fls_Cancel API or use synchronous write.

If the Fls_Cancel API is enabled in the FLS driver used by the FEE, the FlsMaxWriteNormalMode/FlsMaxWriteFastMode

parameter value cannot be smaller than the size of the FEE Block/Cluster Header or the size of the FEE virtual page, whichever

is larger. Disable Fls_Cancel API or choose the compatible FlsMaxWriteNormalMode/FlsMaxWriteFastMode parameter value in the given FlsConfigSet.

Property	Value
type	ECUC-REFERENCE-DEF
origin	NXP
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
requiresSymbolicNameValue	False
destination	/AUTOSAR/EcucDefs/Fls/FlsConfigSet/FlsSectorList/FlsSector

4.7 Container FeeBlockConfiguration

Configuration of block specific parameters for the Flash EEPROM Emulation module.

Included subcontainers:

- None

Property	Value
type	ECUC-PARAM-CONF-CONTAINER-DEF
lowerMultiplicity	1
upperMultiplicity	Infinite
postBuildVariantMultiplicity	false
multiplicityConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE

4.8 Parameter FeeBlockNumber

Block identifier (handle).

0x0000 and 0xFFFF shall not be used for block

numbers (see FEE006).

Range:

min = 2^NVM_DATA_SELECTION_BITS

max = 0xFFFF - 2^NVM_DATA_SELECTION_BITS

Note: Depending on the number of bits set aside for dataset selection

several other block numbers shall also be left out to ease implementation.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	AUTOSAR_ECUC
symbolicNameValue	true
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	1
max	65534
min	1

4.9 Parameter FeeBlockSize

Size of a logical block in bytes.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	AUTOSAR_ECUC
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A

Property	Value
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	1
max	65535
min	1

4.10 Parameter FeeImmediateData

Marker for high priority data.

true: Block contains immediate data.

false: Block does not contain immediate data.

Property	Value
type	ECUC-BOOLEAN-PARAM-DEF
origin	AUTOSAR_ECUC
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	false

4.11 Parameter FeeNumberOfWriteCycles

Number of write cycles required for this block.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	AUTOSAR_ECUC
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE

Property	Value
defaultValue	0
max	4294967295
min	0

4.12 Parameter FeeBlockAssignment

Choose to which project this block is assigned.

BootLoader - Block is used only by the boot loader project

Application - Block is used only by the application project

Shared - Block is used for the Application and Boot Loader projects

Property	Value
type	ECUC-ENUMERATION-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	APPLICATION
literals	['APPLICATION', 'BOOTLOADER', 'SHARED']

4.13 Reference FeeClusterGroupRef

Vendor specific: Reference to the Fee cluster group which the Fee

block belongs to. In other words, FeeClusterGroupRef assigns the Fee block

to particular Fee cluster group.

Property	Value
type	ECUC-REFERENCE-DEF
origin	NXP
lowerMultiplicity	1
upperMultiplicity	1

Property	Value
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
requiresSymbolicNameValue	False
destination	/TS_T40D34M30I0R0/Fee/FeeClusterGroup

4.14 Reference FeeDeviceIndex

Device index (handle).

Range: 0 .. 254 (0xFF reserved for broadcast call to

GetStatus function).

Property	Value
type	ECUC-REFERENCE-DEF
origin	AUTOSAR_ECUC
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
requiresSymbolicNameValue	true
destination	/AUTOSAR/EcucDefs/Fls/FlsGeneral

4.15 Container FeeSectorToRecover

Vendor specific: Container for the list of sectors to recover, only used in Sector Retirement mode.

For more information, please refer to the chapter Sector Management and Sector Retirement in the User manual.

Included subcontainers:

- None

Property	Value
type	ECUC-PARAM-CONF-CONTAINER-DEF

Property	Value
lowerMultiplicity	0
upperMultiplicity	Infinite
postBuildVariantMultiplicity	false
multiplicityConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE

4.16 Reference FeeSectorToRecoverRef

Vendor specific: Reference to the sector which will be recovered from the bad state.

For more information, please refer to the chapter Sector Management and Sector Retirement in the User manual.

Property	Value
type	ECUC-REFERENCE-DEF
origin	NXP
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
requiresSymbolicNameValue	False
destination	/TS_T40D34M30I0R0/Fee/FeeClusterGroup/FeeCluster/FeeSector

4.17 Container FeeGeneral

Container for general parameters. These parameters are not specific to a block.

Included subcontainers:

- None

Property	Value
type	ECUC-PARAM-CONF-CONTAINER-DEF
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A

4.18 Parameter FeeDevErrorDetect

Pre-processor switch to enable and disable development error detection.

true: Development error detection enabled.

false: Development error detection disabled.

Property	Value
type	ECUC-BOOLEAN-PARAM-DEF
origin	AUTOSAR_ECUC
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	true

4.19 Parameter FeeEnableUserModeSupport

Fee driver is an independent hardware module, so it can run in user mode without any specific measures.

The parameter is not used in the Fee implementation.

Property	Value
type	ECUC-BOOLEAN-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	false

4.20 Parameter FeeMainFunctionPeriod

The period between successive calls to the main function in seconds.

Property	Value
type	ECUC-FLOAT-PARAM-DEF
origin	AUTOSAR_ECUC
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	0.005
max	100000.0
min	1.0E-7

4.21 Parameter FeeNvmJobEndNotification

Mapped to the job end notification routine provided by the upper layer

module (NvM_JobEndNotification).

Note: Disable the end notification to have it set as NULL_PTR.

Property	Value
type	ECUC-FUNCTION-NAME-DEF
origin	AUTOSAR_ECUC
symbolicNameValue	false
lowerMultiplicity	0
upperMultiplicity	1
postBuildVariantMultiplicity	false
multiplicityConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	NvM_JobEndNotification

4.22 Parameter FeeNvmJobErrorNotification

Mapped to the job error notification routine provided by the upper layer

module (NvM_JobErrorNotification).

Note: Disable the error notification to have it set as NULL_PTR.

Property	Value
type	ECUC-FUNCTION-NAME-DEF
origin	AUTOSAR_ECUC
symbolicNameValue	false
lowerMultiplicity	0
upperMultiplicity	1
postBuildVariantMultiplicity	false
multiplicityConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	NvM_JobErrorNotification

4.23 Parameter FeeClusterFormatNotification

Fee calls this notification to inform the user in case a cluster erase and write cluster header is performed during the Fee initialization.

Note: Disable the error notification to have it set as NULL_PTR.

Property	Value
type	ECUC-FUNCTION-NAME-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	0
upperMultiplicity	1
postBuildVariantMultiplicity	false
multiplicityConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	NULL_PTR

4.24 Parameter FeePollingMode

Pre-processor switch to enable and disable the polling mode for this module

Note: This paramater is not utilized in this BSW module implementation

Property	Value
type	ECUC-BOOLEAN-PARAM-DEF
origin	AUTOSAR_ECUC
symbolicNameValue	false

Property	Value
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	false

4.25 Parameter FeeSetModeSupported

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).

Property	Value
type	ECUC-BOOLEAN-PARAM-DEF
origin	AUTOSAR_ECUC
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	true

4.26 Parameter FeeVersionInfoApi

Pre-processor switch to enable/disable the API to read out the module

version information.

true: Version info API enabled.

false: Version info API disabled.

Property	Value
type	ECUC-BOOLEAN-PARAM-DEF
origin	AUTOSAR_ECUC
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	true

4.27 Parameter FeeVirtualPageSize

The size in bytes to which logical blocks shall be aligned.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	AUTOSAR_ECUC
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	32
max	65535
min	1

4.28 Parameter FeeDataBufferSize

Vendor specific: Size of the data buffer in bytes.

The data buffer is used to buffer data when Fee is copying data

from one cluster to another and when Fee is reading the block header

information on startup.

Size of the data buffer affects number of Fls_MainFunction cycles.

Bigger data buffer improves performance of the Fee cluster management

operations and speeds up the startup phase as Fee can read more data

in one cycle of Fls_MainFunction

Note: FeeDataBufferSize must be equal or greater than

FEE_CLUSTER_OVERHEAD. Where FEE_CLUSTER_OVERHEAD is management overhead

per logical cluster in bytes and can be calculated using the following

formula:

  ceiling(16 / FEE_VIRTUAL_PAGE_SIZE + 2) * FEE_VIR

Note: In sector management mode, the buffer size will be increased to store sector information in the cluster header.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	96
max	65535
min	0

4.29 Parameter FeeBlockAlwaysAvailable

Vendor specific: According to the AUTOSAR requirements, when the

write operation starts, corresponding block is marked as inconsistent.

It is marked as consistent after successful write.

It means that when write operation is interrupted (cancel, reset)

application cannot access the block data anymore.

Enabling this parameter allows to set the behavior against the AUTOSAR

requirements. In this case, the last valid information is always provided.

Valid information means:

FEE_BLOCK_INVALID if the block was invalidated or the block is

not present in the cluster at all.

FEE_BLOCK_VALID in case the previous instance of the block exists and

was not invalidated.

Property	Value
type	ECUC-BOOLEAN-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	false

4.30 Parameter FeeLegacyEraseMode

If enabled, the Fee_EraseImmediateBlock function invalidates the referred

block if it has been already written. Otherwise, the block is left intact.

Property	Value
type	ECUC-BOOLEAN-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	false

4.31 Parameter FeeSwapForeignBlocksEnabled

The recommendation is not to use this mode Enabled for the production phase as it increases the cluster swap overall

timing.

This mode may be useful for developement phase if Fee is used in 2 projects running on the same ECU and the integrator wants

to be able to change configuration for the application project without changing it for the second project.

If disabled,the Fee driver will swap only blocks existing in the current configuration.

If enabled, the Fee driver will swap foreign blocks. This is a limited support when using 2 subprojects with different Fee configurations

running on the same ECU(example:boot loader and applications projects).For example if this paramters is enabled and Fee is running in application 1 mode,

it will swap blocks assigned to application 2, even if those blocks are not present in application 1 configuration.This allow to update the Fee configuration for application 2 without reconfiguring application 1.

Nevertheless, this mode will result in a longer time for cluster swap.

Property	Value
type	ECUC-BOOLEAN-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	false

4.32 Parameter FeeMarkEmptyBlocksInvalid

If the parameter is enabled, Fee will mark the never written blocks as INVALID,

otherwise Fee will mark the never written blocks as INCONSISTENT. Having this define set as false is compatible with AUTOSAR 4.2.2 version.

Previous AUTOSAR versions do not specify which status must be returned for empty blocks.

Property	Value
type	ECUC-BOOLEAN-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1

Property	Value
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	false

4.33 Parameter FeeConfigAssignment

Choose for which project this Fee configuration is used.

BootLoader - Configuration is used only by the boot loader project

Application - Configuration is used only by the application project

Property	Value
type	ECUC-ENUMERATION-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	APPLICATION
literals	['APPLICATION', 'BOOTLOADER']

4.34 Parameter FeeMaximumNumberBlocks

This must be configured to total number of boot loader, application blocks and also some buffer for blocks added in the future.

This parameter will be used to statically allocate space for the block runtime information,

so it should be configured to accommodate the total number of blocks running on the ECU in all project versions.

The parameter is used to support Fee Swap Foreign Blocks Feature.

Example of configuration:

If application uses 2 blocks, boot loader 1 block and another 1 is shared between application and bootloader then

this parameter must be configured to 2(only appl) + 1(only boot loader) + 1(shared) + X,

where x is the maximum number of blocks we estimate it will be added in future versions.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP
symbolicNameValue	true
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	1
max	65534
min	1

4.35 Parameter FeeSectorRetirement

Vendor specific: Enables the bad Sector Management and Sector Retirement feature.

Fee driver can have the ability to manage the sectors in each cluster, detect a bad sector and skip it in runtime.

For more information, please refer to the chapter Sector Management and Sector Retirement in the User manual.

Property	Value
type	ECUC-BOOLEAN-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	false

4.36 Parameter FeeSectorEraseRetries

Vendor specific: Number of attempts to erase each sector in the cluster when swapping in sector retirement mode.

A sector will be marked as a bad sector after a number of attemps to erase all have failed.

For more information, please refer to the chapter Sector Management and Sector Retirement in the User manual.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	3
max	255
min	0

4.37 Container CommonPublishedInformation

CommonPublishedInformation

Common container, aggregated by all modules. It contains published

information about vendor and versions.

Included subcontainers:

- None

Property	Value
type	ECUC-PARAM-CONF-CONTAINER-DEF
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A

4.38 Parameter ArReleaseMajorVersion

AUTOSAR Major Version

Vendor specific: Major version number of AUTOSAR specification

on which the appropriate implementation is based on.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PUBLISHED-INFORMATION
defaultValue	4
max	4
min	4

4.39 Parameter ArReleaseMinorVersion

AUTOSAR Minor Version

Vendor specific: Minor version number of AUTOSAR specification

on which the appropriate implementation is based on.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PUBLISHED-INFORMATION
defaultValue	7
max	7
min	7

4.40 Parameter ArReleaseRevisionVersion

AUTOSAR Patch Version

Vendor specific: Revision version number of AUTOSAR specification

on which the appropriate implementation is based on.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PUBLISHED-INFORMATION
defaultValue	0
max	0
min	0

4.41 Parameter ModuleId

Module ID

Vendor specific: Module ID of this module from Module List.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PUBLISHED-INFORMATION
defaultValue	21
max	21
min	21

4.42 Parameter SwMajorVersion

Software Major Version

Vendor specific: Major version number of the vendor specific

implementation of the module. The numbering is vendor specific.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PUBLISHED-INFORMATION
defaultValue	3
max	3
min	3

4.43 Parameter SwMinorVersion

Software Minor Version

Vendor specific: Minor version number of the vendor specific

implementation of the module. The numbering is vendor specific.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PUBLISHED-INFORMATION
defaultValue	0
max	0
min	0

4.44 Parameter SwPatchVersion

Software Patch Version

Vendor specific: Patch level version number of the vendor

specific implementation of the module. The numbering is vendor specific.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PUBLISHED-INFORMATION
defaultValue	0
max	0
min	0

4.45 Parameter VendorApiInfix

Vendor Api Infix

Vendor specific: In driver modules which can be instantiated several

times on a single ECU, BSW00347 requires that the name of APIs is extended

by the VendorId and a vendor specific name.

This parameter is used to specify the vendor specific name. In total, the

implementation specific name is generated as follows:

<ModuleName>_<VendorId>_<VendorApiInfix>.

E.g. assuming that the VendorId of the implementor is 123 and the

implementer chose a VendorApiInfix of "v11r456" a API name

Can_Write defined in the SWS will translate to Can_123_v11r456Write.

This parameter is mandatory for all modules with

upper multiplicity > 1.

It shall not be used for modules with upper multiplicity = 1.

Property	Value
type	ECUC-STRING-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	0
upperMultiplicity	1
postBuildVariantMultiplicity	false
multiplicityConfigClasses	VARIANT-PRE-COMPILE: PUBLISHED-INFORMATION

4.46 Parameter VendorId

Vendor ID

Vendor specific: Vendor ID of the dedicated implementation of this module according to the AUTOSAR vendor list.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PUBLISHED-INFORMATION
defaultValue	43
max	43
min	43

4.47 Container FeePublishedInformation

Additional published parameters not covered by CommonPublishedInformation container.

Note:

That these parameters do not have any configuration class setting, since they are published information.

Included subcontainers:

- None

Property	Value
type	ECUC-PARAM-CONF-CONTAINER-DEF
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A

4.48 Parameter FeeBlockOverhead

Management overhead per logical block in bytes

Note:

The logical block management overhead depends on FeeVirtualPageSize and can be

calculated using the following formula:

  ceiling(12 / FEE_VIRTUAL_PAGE_SIZE + 2) * FEE_VIR

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	AUTOSAR_ECUC
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PUBLISHED-INFORMATION
defaultValue	0
max	65535
min	0

4.49 Parameter FeePageOverhead

Management overhead per page in bytes

Note:

The page management overhead is 0 bytes

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	AUTOSAR_ECUC
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PUBLISHED-INFORMATION
defaultValue	0
max	65535
min	0

Chapter 5

Module Index

5.1 Software Specification

Here is a list of all modules:

Chapter 6

Module Documentation

6.1 FEE

6.1.1 Detailed Description

Data Structures

- struct Fee_ClusterGroupInfoType

  Fee cluster group run-time status. *More...*
- struct Fee_BlockInfoType

  Fee block run-time status. *More...*
- struct Fee_SectorRuntimeInfoType

  Fee sectors run-time status. *More...*
- struct Fee_ClusterRuntimeInfoType

  Fee clusters run-time status. *More...*
- struct Fee_ClusterGroupRuntimeInfoType

  Fee cluster group run-time Information. *More...*
- struct Fee_BlockConfigType

  Fee block configuration structure. *More...*
- struct Fee_ClusterType

  Fee cluster configuration structure. *More...*
- struct Fee_ClusterGroupType

  Fee cluster group configuration structure. *More...*
- struct Fee_BlockType

  Fee block header configuration structure. This consists of block number and length of block and Type of Fee block. *More...*
- struct Fee_ClusterHeaderType

  Fee cluster header configuration structure. *More...*

Macros

- #define FEE_INIT_ID

  service ID of function: Fee_Init. (passed to DET)
- #define FEE_SETMODE_ID

  service ID of function: Fee_SetMode. (passed to DET)
- #define FEE_READ_ID

  service ID of function: Fee_Read. (passed to DET)
- #define FEE_WRITE_ID

  service ID of function: Fee_Write. (passed to DET)
- #define FEE_CANCEL_ID

  service ID of function: Fee_Cancel. (passed to DET)
- #define FEE_GETSTATUS_ID

  service ID of function: Fee_GetStatus. (passed to DET)
- #define FEE_GETJOBRESULT_ID

  service ID of function: Fee_GetJobResult. (passed to DET)
- #define FEE_INVALIDATEBLOCK_ID

  service ID of function: Fee_InvalidateBlock. (passed to DET)
- #define FEE_GETVERSIONINFO_ID

  service ID of function: Fee_GetVersionInfo. (passed to DET)
- #define FEE_ERASEIMMEDIATEBLOCK_ID

  service ID of function: Fee_EraseImmediateBlock. (passed to DET)
- #define FEE_JOBENDNOTIFICATION_ID

  service ID of function: Fee_JobEndNotification. (passed to DET)
- #define FEE_JOBERRORNOTIFICATION_ID

  service ID of function: Fee_JobErrorNotification.(passed to DET)
- #define FEE_MAINFUNCTION_ID

  service ID of function: Fee_MainFunction. (passed to DET)
- #define FEE_GETRUNTIMEINFO_ID

  service ID of function: Fee_GetRunTimeInfo. (passed to DET)
- #define FEE_FORCESWAPONNEXTWRITE_ID

  service ID of function: Fee_ForceSwapOnNextWrite. (passed to DET)
- #define FEE_E_UNINIT

  API called when module was not initialized.
- #define FEE_E_INVALID_BLOCK_NO

  API called with invalid block number.
- #define FEE_E_INVALID_BLOCK_OFS

  API called with invalid block offset.
- #define FEE_E_PARAM_POINTER

  API called with invalid data pointer.
- #define FEE_E_INVALID_BLOCK_LEN

  API called with invalid length information.
- #define FEE_E_BUSY

  API called while module is busy processing a user request.
- #define FEE_E_INVALID_CANCEL

  API called while module is not busy because there is no job to cancel.
- #define FEE_E_INIT_FAILED

API Fee_init failed.

- #define FEE_E_CANCEL_API

  API called when underlying driver has cancel API disabled.
- #define FEE_E_CLUSTER_GROUP_IDX

  API called with invalid cluster group index.
- #define FEE_E_FOREIGN_BLOCKS_OVF

  API number of foreign blocks from data flash exceeds the total number of blocks allowed which is FEE_MAX_NR↩ _OF_BLOCKS.

Types Reference

- typedef Fee_BlockConfigType Fee_ConfigType

  Fee Configuration type is a stub type, not used, but required by ASR 4.2.2.

Enum Reference

- enum Fee_BlockStatusType

  Status of Fee block header.
- enum Fee_ClusterStatusType

  Status of Fee cluster header.
- enum Fee_JobType

  Type of job currently executed by Fee_MainFunction.
- enum Fee_TranslationJobType

  Type of translation jobs to translate emulation address to logical address.
- enum Fee_BlockAssignmentType

  Fee block assignment type.

Function Reference

- void Fee_Init (const Fee_ConfigType *ConfigPtr)

  Service to initialize the FEE module.
- Std_ReturnType Fee_Read (uint16 BlockNumber, uint16 BlockOffset, uint8 *DataBufferPtr, uint16 Length)

  Service to initiate a read job.
- Std_ReturnType Fee_Write (uint16 BlockNumber, const uint8 *DataBufferPtr)

  Service to initiate a write job.
- void Fee_Cancel (void)

  Service to call the cancel function of the underlying flash driver.
- MemIf_StatusType Fee_GetStatus (void)

  Return the Fee module state.
- MemIf_JobResultType Fee_GetJobResult (void)

  Return the result of the last job.
- Std_ReturnType Fee_InvalidateBlock (uint16 BlockNumber)

  Service to invalidate a logical block.
- void Fee_GetVersionInfo (Std_VersionInfoType *VersionInfoPtr)

Return the version information of the Fee module.

- Std_ReturnType Fee_EraseImmediateBlock (uint16 BlockNumber)

  Service to erase a logical block.
- void Fee_GetRunTimeInfo (uint8 ClrGrpIndex, Fee_ClusterGroupRuntimeInfoType *ClrGrpRTInfo)

  Service to read runtime information in the selected cluster.
- Std_ReturnType Fee_ForceSwapOnNextWrite (uint8 ClrGrpIndex)

  Service to prepare the driver for a cluster swap in the selected cluster group.

6.1.2 Data Structure Documentation

6.1.2.1 struct Fee_ClusterGroupInfoType

Fee cluster group run-time status.

Definition at line 186 of file Fee_InternalTypes.h.

Data Fields

Type	Name	Description
Fls_AddressType	DataAddrIt	Address of current Fee data block in flash.
Fls_AddressType	HdrAddrIt	Address of current Fee block header in flash.
uint32	ActClrID	ID of active cluster.
uint8	ActClr	Index of active cluster.
Fee_ClusterRuntimeInfoType *	ClrInfo	Pointer to array of Fee cluster runtime infomation.

6.1.2.2 struct Fee_BlockInfoType

Fee block run-time status.

Definition at line 201 of file Fee_InternalTypes.h.

Data Fields

Type	Name	Description
Fls_AddressType	DataAddr	Address of Fee block data in flash.
Fls_AddressType	InvalidAddr	Address of Fee block invalidation field in flash.
Fee_BlockStatusType	BlockStatus	Current status of Fee block.

6.1.2.3 struct Fee_SectorRuntimeInfoType

Fee sectors run-time status.

Definition at line 106 of file Fee_Types.h.

Data Fields

Type	Name	Description
Fls_AddressType	SectorAddr	Logical address of sector on Fls.
Fls_LengthType	SectorSize	Size of sector in bytes.

6.1.2.4 struct Fee_ClusterRuntimeInfoType

Fee clusters run-time status.

Definition at line 116 of file Fee_Types.h.

Data Fields

Type	Name	Description
Fls_LengthType	Length	Size of Fee cluster in bytes.
uint16	SectorCount	Number of sector in cluster.
Fee_SectorRuntimeInfoType *	SectorList	Pointer to array of sector configurations.

6.1.2.5 struct Fee_ClusterGroupRuntimeInfoType

Fee cluster group run-time Information.

Definition at line 142 of file Fee_Types.h.

Data Fields

Type	Name	Description
Fls_AddressType	ClusterTotalSpace	Total space in the selected cluster group.
Fls_AddressType	ClusterFreeSpace	Free space in the selected cluster group.
uint16	BlockHeaderOverhead	Block Overhead (header valid and inval flag)
uint16	VirtualPageSize	Fee Virtual Page Size.
uint32	NumberOfSwap	Number of cluster swap performed in the selected cluster group.
Fee_SectorRuntimeInfoType *	SectorInfo	Pointer to the sector runtime information data structure.

6.1.2.6 struct Fee_BlockConfigType

Fee block configuration structure.

Definition at line 172 of file Fee_Types.h.

Data Fields

- uint16 BlockNumber
  Fee block number.
- uint16 BlockSize
  Size of Fee block in bytes.
- uint8 ClrGrp
  Index of cluster group the Fee block belongs to.
- boolean ImmediateData
  TRUE if immediate data block.
- Fee_BlockAssignmentType BlockAssignment
  specifies which project uses this block

6.1.2.6.1 Field Documentation

6.1.2.6.1.1 BlockNumber `uint16 BlockNumber`

Fee block number.

Definition at line 175 of file Fee_Types.h.

6.1.2.6.1.2 BlockSize `uint16 BlockSize`

Size of Fee block in bytes.

Definition at line 178 of file Fee_Types.h.

6.1.2.6.1.3 ClrGrp `uint8 ClrGrp`

Index of cluster group the Fee block belongs to.

Definition at line 179 of file Fee_Types.h.

6.1.2.6.1.4 ImmediateData `boolean ImmediateData`

TRUE if immediate data block.

Definition at line 182 of file Fee_Types.h.

6.1.2.6.1.5 BlockAssignment `Fee_BlockAssignmentType BlockAssignment`

specifies which project uses this block

Definition at line 183 of file Fee_Types.h.

6.1.2.7 struct Fee_ClusterType

Fee cluster configuration structure.

Definition at line 190 of file Fee_Types.h.

Data Fields

Type	Name	Description
Fls_AddressType	StartAddr	Address of Fee cluster in flash.

6.1.2.8 struct Fee_ClusterGroupType

Fee cluster group configuration structure.

Definition at line 203 of file Fee_Types.h.

Data Fields

- const Fee_ClusterType *const ClrPtr
  Pointer to array of Fee cluster configurations.
- uint32 ClrCount
  Number of clusters in cluster group.
- Fls_LengthType ReservedSize
  Size of reserved area in the given cluster group (memory occupied by immediate blocks)

6.1.2.8.1 Field Documentation

6.1.2.8.1.1 ClrPtr `const Fee_ClusterType* const ClrPtr`

Pointer to array of Fee cluster configurations.

Definition at line 205 of file Fee_Types.h.

6.1.2.8.1.2 ClrCount `uint32 ClrCount`

Number of clusters in cluster group.

Definition at line 206 of file Fee_Types.h.

6.1.2.8.1.3 ReservedSize `Fls_LengthType ReservedSize`

Size of reserved area in the given cluster group (memory occupied by immediate blocks)

Definition at line 207 of file Fee_Types.h.

6.1.2.9 struct Fee_BlockType

Fee block header configuration structure. This consists of block number and length of block and Type of Fee block.

Definition at line 221 of file Fee_Types.h.

Data Fields

Type	Name	Description
uint16	BlockNumber	Number of block.
uint16	Length	Length of block.
boolean	ImmediateBlock	Type of Fee block. Set to TRUE for immediate block.

6.1.2.10 struct Fee_ClusterHeaderType

Fee cluster header configuration structure.

Definition at line 232 of file Fee_Types.h.

Data Fields

Type	Name	Description
uint32	ClrID	32-bit cluster ID
Fls_AddressType	StartAddr	Start address of Fee cluster in Fls address space.
Fls_LengthType	Length	Length of Fee cluster in bytes.
Fee_ClusterRuntimeInfoType *	ClrInfo	Pointer to the cluster runtime information.

6.1.3 Macro Definition Documentation

6.1.3.1 FEE_INIT_ID

```
#define FEE_INIT_ID
```

service ID of function: Fee_Init. (passed to DET)

Definition at line 100 of file Fee.h.

6.1.3.2 FEE_SETMODE_ID

```
#define FEE_SETMODE_ID
```

service ID of function: Fee_SetMode. (passed to DET)

Definition at line 105 of file Fee.h.

6.1.3.3 FEE_READ_ID

```
#define FEE_READ_ID
```

service ID of function: Fee_Read. (passed to DET)

Definition at line 110 of file Fee.h.

6.1.3.4 FEE_WRITE_ID

```
#define FEE_WRITE_ID
```

service ID of function: Fee_Write. (passed to DET)

Definition at line 115 of file Fee.h.

6.1.3.5 FEE_CANCEL_ID

```
#define FEE_CANCEL_ID
```

service ID of function: Fee_Cancel. (passed to DET)

Definition at line 120 of file Fee.h.

6.1.3.6 FEE_GETSTATUS_ID

```
#define FEE_GETSTATUS_ID
```

service ID of function: Fee_GetStatus. (passed to DET)

Definition at line 125 of file Fee.h.

6.1.3.7 FEE_GETJOBRESULT_ID

```
#define FEE_GETJOBRESULT_ID
```

service ID of function: Fee_GetJobResult. (passed to DET)

Definition at line 130 of file Fee.h.

6.1.3.8 FEE_INVALIDATEBLOCK_ID

```
#define FEE_INVALIDATEBLOCK_ID
```

service ID of function: Fee_InvalidateBlock. (passed to DET)

Definition at line 135 of file Fee.h.

6.1.3.9 FEE_GETVERSIONINFO_ID

```
#define FEE_GETVERSIONINFO_ID
```

service ID of function: Fee_GetVersionInfo. (passed to DET)

Definition at line 140 of file Fee.h.

6.1.3.10 FEE_ERASEIMMEDIATEBLOCK_ID

```
#define FEE_ERASEIMMEDIATEBLOCK_ID
```

service ID of function: Fee_EraseImmediateBlock. (passed to DET)

Definition at line 145 of file Fee.h.

6.1.3.11 FEE_JOBENDNOTIFICATION_ID

```
#define FEE_JOBENDNOTIFICATION_ID
```

service ID of function: Fee_JobEndNotification. (passed to DET)

Definition at line 150 of file Fee.h.

6.1.3.12 FEE_JOBERRORNOTIFICATION_ID

```
#define FEE_JOBERRORNOTIFICATION_ID
```

service ID of function: Fee_JobErrorNotification.(passed to DET)

Definition at line 155 of file Fee.h.

6.1.3.13 FEE_MAINFUNCTION_ID

```
#define FEE_MAINFUNCTION_ID
```

service ID of function: Fee_MainFunction. (passed to DET)

Definition at line 160 of file Fee.h.

6.1.3.14 FEE_GETRUNTIMEINFO_ID

```
#define FEE_GETRUNTIMEINFO_ID
```

service ID of function: Fee_GetRunTimeInfo. (passed to DET)

Definition at line 167 of file Fee.h.

6.1.3.15 FEE_FORCESWAPONNEXTWRITE_ID

```
#define FEE_FORCESWAPONNEXTWRITE_ID
```

service ID of function: Fee_ForceSwapOnNextWrite. (passed to DET)

Definition at line 172 of file Fee.h.

6.1.3.16 FEE_E_UNINIT

```
#define FEE_E_UNINIT
```

API called when module was not initialized.

Definition at line 178 of file Fee.h.

6.1.3.17 FEE_E_INVALID_BLOCK_NO

```
#define FEE_E_INVALID_BLOCK_NO
```

API called with invalid block number.

Definition at line 181 of file Fee.h.

6.1.3.18 FEE_E_INVALID_BLOCK_OFS

```
#define FEE_E_INVALID_BLOCK_OFS
```

API called with invalid block offset.

Definition at line 184 of file Fee.h.

6.1.3.19 FEE_E_PARAM_POINTER

```
#define FEE_E_PARAM_POINTER
```

API called with invalid data pointer.

Definition at line 187 of file Fee.h.

6.1.3.20 FEE_E_INVALID_BLOCK_LEN

```
#define FEE_E_INVALID_BLOCK_LEN
```

API called with invalid length information.

Definition at line 190 of file Fee.h.

6.1.3.21 FEE_E_BUSY

```
#define FEE_E_BUSY
```

API called while module is busy processing a user request.

Definition at line 193 of file Fee.h.

6.1.3.22 FEE_E_INVALID_CANCEL

```
#define FEE_E_INVALID_CANCEL
```

API called while module is not busy because there is no job to cancel.

Definition at line 196 of file Fee.h.

6.1.3.23 FEE_E_INIT_FAILED

```
#define FEE_E_INIT_FAILED
```

API Fee_init failed.

Definition at line 199 of file Fee.h.

6.1.3.24 FEE_E_CANCEL_API

```
#define FEE_E_CANCEL_API
```

API called when underlying driver has cancel API disabled.

Definition at line 205 of file Fee.h.

6.1.3.25 FEE_E_CLUSTER_GROUP_IDX

```
#define FEE_E_CLUSTER_GROUP_IDX
```

API called with invalid cluster group index.

Definition at line 209 of file Fee.h.

6.1.3.26 FEE_E_FOREIGN_BLOCKS_OVF

```
#define FEE_E_FOREIGN_BLOCKS_OVF
```

API number of foreign blocks from data flash exceeds the total number of blocks allowed which is FEE_MAX_↩ NR_OF_BLOCKS.

Definition at line 213 of file Fee.h.

6.1.4 Types Reference

6.1.4.1 Fee_ConfigType

```
typedef Fee_BlockConfigType Fee_ConfigType
```

Fee Configuration type is a stub type, not used, but required by ASR 4.2.2.

Definition at line 215 of file Fee_Types.h.

6.1.5 Enum Reference

6.1.5.1 Fee_BlockStatusType

```
enum Fee_BlockStatusType
```

Status of Fee block header.

Enumerator

FEE_BLOCK_VALID	Fee block is valid.
FEE_BLOCK_INVALID	Fee block is invalid (has been invalidated)
FEE_BLOCK_INCONSISTENT	Fee block is inconsistent (contains bogus data)
FEE_BLOCK_HEADER_INVALID	Fee block header is garbled.
FEE_BLOCK_INVALIDATED	Fee block header is invalidated by Fee_InvalidateBlock(BlockNumber)(not used when FEE_BLOCK_ALWAYS_AVAILABLE == STD_OFF)
FEE_BLOCK_HEADER_BLANK	Fee block header is blank, it is used to mark the end of Fee block header list when parsing the memory at initialization.
FEE_BLOCK_INCONSISTENT_COPY	FEE data read error during swap (ie data area was allocated)
FEE_BLOCK_NEVER_WRITTEN	FEE block was never written in data flash.

Definition at line 89 of file Fee_InternalTypes.h.

6.1.5.2 Fee_ClusterStatusType

`enum Fee_ClusterStatusType`

Status of Fee cluster header.

Enumerator

FEE_CLUSTER_VALID	Fee cluster is valid.
FEE_CLUSTER_INVALID	Fee cluster is invalid.
FEE_CLUSTER_INCONSISTENT	Fee cluster is inconsistent (contains bogus data)
FEE_CLUSTER_HEADER_INVALID	Fee cluster header is garbled.

Definition at line 111 of file Fee_InternalTypes.h.

6.1.5.3 Fee_JobType

`enum Fee_JobType`

Type of job currently executed by Fee_MainFunction.

Enumerator

FEE_JOB_READ	Read Fee block.
FEE_JOB_WRITE	Write Fee block to flash.

Enumerator

FEE_JOB_WRITE_DATA	Write Fee block data to flash.
FEE_JOB_WRITE_UNALIGNED_DATA	Write unaligned rest of Fee block data to flash.
FEE_JOB_WRITE_VALIDATE	Validate Fee block by writing validation flag to flash.
FEE_JOB_WRITE_DONE	Finalize validation of Fee block.
FEE_JOB_INVAL_BLOCK	Invalidate Fee block by writing the invalidation flag to flash.
FEE_JOB_INVAL_BLOCK_DONE	Finalize invalidation of Fee block.
FEE_JOB_ERASE_IMMEDIATE	Erase (pre-allocate) immediate Fee block.
FEE_JOB_INT_SCAN	Initialize the cluster scan job.
FEE_JOB_INT_SCAN_CLR	Scan active cluster of current cluster group.
FEE_JOB_INT_SCAN_CLR_HDR_PARSE	Parse Fee cluster header.
FEE_JOB_INT_SCAN_CLR_FMT	Format first Fee cluster.
FEE_JOB_INT_SCAN_CLR_FMT_DONE	Finalize format of first Fee cluster.
FEE_JOB_INT_SCAN_BLOCK_HDR_PARSE	Parse Fee block header.
FEE_JOB_INT_SWAP_CLR_FMT	Format current Fee cluster in current Fee cluster group.
FEE_JOB_INT_SWAP_BLOCK	Copy next block from source to target cluster.
FEE_JOB_INT_SWAP_DATA_READ	Read data from source cluster to internal Fee buffer.
FEE_JOB_INT_SWAP_DATA_WRITE	Write data from internal Fee buffer to target cluster.
FEE_JOB_INT_SWAP_CLR_VLD_DONE	Finalize cluster validation.
FEE_JOB_DONE	No more subsequent jobs to schedule.
FEE_JOB_SETMODE	Setmode to fls.

Definition at line 123 of file Fee_InternalTypes.h.

6.1.5.4 Fee_TranslationJobType

enum Fee_TranslationJobType

Type of translation jobs to translate emulation address to logical address.

Enumerator

FEE_TRANS_JOB_READ	Translation read data from Fls.
FEE_TRANS_JOB_WRITE	Translation write data to Fls.
FEE_TRANS_JOB_ERASE	Translation erase Fls sectors.

Definition at line 169 of file Fee_InternalTypes.h.

6.1.5.5 Fee_BlockAssignmentType

```
enum Fee_BlockAssignmentType
```

Fee block assignment type.

Enumerator

FEE_PROJECT_SHARED	block is used for all the projects
FEE_PROJECT_APPLICATION	block is used for the application project
FEE_PROJECT_BOOTLOADER	block is used for the bootloader project
FEE_PROJECT_RESERVED	the value is reserved

Definition at line 159 of file Fee_Types.h.

6.1.6 Function Reference

6.1.6.1 Fee_Init()

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

Service to initialize the FEE module.

The function Fee_Init shall initialize the Flash EEPROM Emulation module.

Parameters

in	*ConfigPtr*	Pointer to fee driver configuration set.

Precondition

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

Note

The function Autosar Service ID[hex]: 0x00.

Asynchronous

Non Reentrant

6.1.6.2 Fee_Read()

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

Service to initiate a read job.

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

Parameters

in	*BlockNumber*	Number of logical block, also denoting start address of that block in flash memory.
in	*BlockOffset*	Read address offset inside the block.
out	*DataBufferPtr*	Pointer to data buffer.
in	*Length*	Number of bytes to read.

Precondition

The module must be initialized, not busy, uBlockNumber must be valid, uLength != 0, pDataBufferPtr != NULL_PTR, uBlockOffset and (uBlockOffset + uLength - 1) must be in range.

Postcondition

changes Fee_eModuleStatus module status and Fee_uJobBlockOffset, Fee_uJobBlockLength, Fee_uJob↩ BlockIndex, Fee_pJobReadDataDestPtr, Fee_eJob, Fee_eJobResult job control internal variables.

Returns

Std_ReturnType

Return values

E_OK	The read job was accepted by the underlying memory driver.
E_NOT_OK	The read job has not been accepted by the underlying memory driver.

Note

The function Autosar Service ID[hex]: 0x02.

Asynchronous.

Non Reentrant.

6.1.6.3 Fee_Write()

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

Service to initiate a write job.

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. The function Fee_Write shall copy the given or 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.The FEE module shall execute the write job of the function Fee_Write asynchronously within the FEE module' s main function.

Parameters

in	*BlockNumber*	Number of logical block, also denoting start address of that block in emulated EEPROM.
out	*DataBufferPtr*	Pointer to data buffer.

Returns

Std_ReturnType

Return values

E_OK	The write job was accepted by the underlying memory driver.
E_NOT_OK	The write job has not been accepted by the underlying memory driver.

Precondition

The module must be initialized, not busy, uBlockNumber must be valid, and pDataBufferPtr != NULL_PTR. Before call the function "Fee_Write" for immediate date must be called the function "Fee_EraseImmediate↩ Block".

Postcondition

changes Fee_eModuleStatus module status and Fee_uJobBlockIndex, Fee_pJobWriteDataDestPtr, Fee_eJob, Fee_eJobResult job control internal variables.

Note

The function Autosar Service ID[hex]: 0x03.

Asynchronous.

Non Reentrant.

6.1.6.4 Fee_Cancel()

```
void Fee_Cancel (
            void  )
```

Service to call the cancel function of the underlying flash driver.

The function Fee_Cancel and the cancel function of the underlying flash driver are asynchronous w.r.t. an ongoing read, erase or write job in the flash memory.

Precondition

The module must be initialized.

Postcondition

Changes Fee_eModuleStatus module status and job result Fee_eJobResult internal variables.

Note

The function Autosar Service ID[hex]: 0x04.

Synchronous.

Non Reentrant.

6.1.6.5 Fee_GetStatus()

```
MemIf_StatusType Fee_GetStatus (
            void  )
```

Return the Fee module state.

Return the Fee module state synchronously.

Note

The function Autosar Service ID[hex]: 0x05.

Synchronous

Non Reentrant

Returns

Fee_eModuleStatus

Return values

MEMIF_UNINIT	Module has not been initialized (yet).
MEMIF_IDLE	Module is currently idle.
MEMIF_BUSY	Module is currently busy.
MEMIF_BUSY_INTERNAL	Module is busy with internal management operations.

6.1.6.6 Fee_GetJobResult()

```
MemIf_JobResultType Fee_GetJobResult (
            void  )
```

Return the result of the last job.

Return the result of the last job synchronously.

Returns

MemIf_JobResultType

Return values

MEMIF_JOB_OK	The job has been finished successfully.
MEMIF_JOB_FAILED	The job has not been finished successfully.
MEMIF_JOB_PENDING	The job has not yet been finished.
MEMIF_JOB_CANCELED	The job has been canceled.
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.

Note

The function Autosar Service ID[hex]: 0x06.

Synchronous.

Non Reentrant.

6.1.6.7 Fee_InvalidateBlock()

```
Std_ReturnType Fee_InvalidateBlock (
            uint16 BlockNumber )
```

Service to invalidate a logical block.

Parameters

in	*BlockNumber*	Number of logical block, also denoting start address of that block in flash memory

Returns

Std_ReturnType

Return values

E_OK	The job was accepted by the underlying memory driver.
E_NOT_OK	The job has not been accepted by the underlying memory driver.

Precondition

The module must be initialized, not busy, and uBlockNumber must be valid

Postcondition

changes Fee_eModuleStatus module status and Fee_uJobBlockIndex, Fee_eJob, and Fee_eJobResult job control internal variables. EEPROM.

Note

The function Autosar Service ID[hex]: 0x07.

Asynchronous.

Non Reentrant.

6.1.6.8 Fee_GetVersionInfo()

```
void Fee_GetVersionInfo (
            Std_VersionInfoType * VersionInfoPtr )
```

Return the version information of the Fee module.

The version information includes: Module Id, Vendor Id, Vendor specific version numbers.

Parameters

out	*VersionInfoPtr*	Pointer to where to store the version information of this module .

Precondition

pVersionInfoPtr must not be NULL_PTR.

Note

The function Autosar Service ID[hex]: 0x08.

Synchronous.

Non Reentrant.

6.1.6.9 Fee_EraseImmediateBlock()

```
Std_ReturnType Fee_EraseImmediateBlock (
            uint16 BlockNumber )
```

Service to erase a logical block.

The function Fee_EraseImmediateBlock shall take the block number and calculate the corresponding memory block address. 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. If development error detection for the FEE module is enabled, the function Fee_EraseImmediateBlock shall check whether the addressed logical block is configured as containing immediate data (configuration parameter FeeImmediateData == TRUE). If not, the function Fee_EraseImmediateBlock shall report the error code FEE_E_INVALID_BLOCK_NO.

Parameters

in	*BlockNumber*	Number of logical block, also denoting.

Returns

Std_ReturnType

Return values

E_OK	The job was accepted by the underlying memory driver.
E_NOT_OK	The job has not been accepted by the underlying memory driver. start address of that block in emulated EEPROM.

Precondition

The module must be initialized, not busy, uBlockNumber must be valid, and type of Fee block must be immediate.

Postcondition

changes Fee_eModuleStatus module status and Fee_uJobBlockIndex, Fee_eJob, and Fee_eJobResult job control internal variables.

Note

The function Autosar Service ID[hex]: 0x09.

Asynchronous.

Non Reentrant.

6.1.6.10 Fee_GetRunTimeInfo()

```
void Fee_GetRunTimeInfo (
            uint8 ClrGrpIndex,
            Fee_ClusterGroupRuntimeInfoType * ClrGrpRTInfo )
```

Service to read runtime information in the selected cluster.

Parameters

in	*ClrGrpIndex*	Index of the selected cluster group
in	*ClrGrpRTInfo*	Pointer to point Fee cluster group run-time Information

Returns

Fee_ClusterGroupRuntimeInfoType

Return values

ClusterTotalSpace	current cluster total size
ClusterFreeSpace	current cluster available space.
BlockHeaderOverhead	the block header overhead (blk header, valid flag and inval flag),
VirtualPageSize	the virtual page size (the size of an allocation unit)
NumberOfSwap	number of cluster swap already performed
SectorInfo	sector information from all clusters in the selected cluster group (in sector retirement mode only)

Precondition

The module must be initialized, not busy and uClrGrpIndex must be valid

6.1.6.11 Fee_ForceSwapOnNextWrite()

```
Std_ReturnType Fee_ForceSwapOnNextWrite (
            uint8 ClrGrpIndex )
```

Service to prepare the driver for a cluster swap in the selected cluster group.

@detailes While the computed amount of memory is allocated as a result of Fee_Write call for plain data blocks, for immediate data blocks memory gets completely pre-allocated through Fee_EraseImmediateBlock function (i.e. Fee_Write does not change the remaining space). As a result, swaps triggered by the planned Fee_ForceSwapOn↩ NextWrite function behave the same way, or in other words, an operation that really activates the pysical swap must be either Fee_Write on plain FEE block or Fee_EraseImmediateBlock on immediate data block.

Parameters

in	*ClrGrpIndex*	Index of the selected cluster group

Returns

Std_ReturnType

Return values

E_NOT_OK	module is not initialized, busy or uClrGrpIndex is not in the valid range.
E_OK	No more space available in the selected cluster.

Precondition

The module must be initialized, not busy and uClrGrpIndex must be valid.

Note

As this API manipulates the internal driver state, it has to be claimed non-reentrant and colliding with other FEE ASR APIs

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support