# User Manual

## for S32K14X FEE Driver

# Contents

## Chapter 4
## Tresos Configuration Plug-in

**Chapter 5**
**Driver Information**

**Chapter 6**

## How to Configure

# Chapter 1
# Revision History

**Table 1-1.   Revision History**

| Revision | Date | Author | Description |
|---|---|---|---|
| 1.0 | 13/07/2018 | NXP MCAL Team | Updated version for ASR 4.2.2S32K14X1.0.1 Release |

# Chapter 2
# Introduction

This User Manual describes the platform independent NXP Semiconductors AUTOSAR 4.2 Rev0002  Flash EEPROM Emulation (FEE) module.

AUTOSAR 4.2 Rev0002  FEE API are described in the Driver chapter of this document.

AUTOSAR 4.2 Rev0002  FEE driver configuration parameters are described in the Tresos Configuration Plug-in chapter of this document.

## 2.1  Supported Derivatives

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

**Table 2-1.  S32K14X Derivatives**

| NXP Semiconductors | s32k148_lqfp144, s32k148_lqfp176, s32k148_mapbga100, s32k146_lqfp144, s32k146_lqfp100, s32k146_lqfp64, s32k146_mapbga100, s32k144_lqfp100, s32k144_lqfp64, s32k144_mapbga100, s32k142_lqfp100, s32k142_lqfp64, s32k118_lqfp48, s32k118_lqfp64 |
| --- | --- |

All of the above microcontroller devices are collectively named as S32K14X .

## 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** type: Bold is used for important terms, notes and warnings.

*Italic* font: Italic typeface is 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.

## 2.4  Acronyms and Definitions

**Table 2-2.  Acronyms and Definitions**

| Term | Definition |
|---|---|
| API | Application Programming Interface |
| ASM | Assembler |
| AUTOSAR | Automotive Open System Architecture |
| BSMI | Basic Software Make file Interface |
| BSW | Basic Software |
| C/CPP | C and C++ Source Code |
| CAN | Controller Area Network |
| DEM | Diagnostic Event Manager |
| DET | Development Error Tracer |
| DFO | Data Flash Optimized |
| DW | Double Word |

*Table continues on the next page...*

**User Manual, Rev. 1.0**

## Table 2-2.   Acronyms and Definitions (continued)

| Term | Definition |
|------|------------|
| ECC | Error Correcting Code |
| ECU | Electronic Control Unit |
| EcuM | ECU Manager Module |
| EEPROM | Electrically Erasable Programmable Read-Only Memory |
| FEE | Flash EEPROM Emulation Module/Driver |
| FLS | Flash memory driver |
| ISR | Interrupt Service Routine |
| IVOR | Interrupt Vector Offset Register |
| *job* | A FEE block operation passed to the module |
| MCU | Microcontroller Unit |
| MemIf | Memory Interface Module |
| N/A | Not Applicable |
| NvM | NVRAM Manager |
| NVRAM | Non-volatile RAM memory |
| OS | Operating System |
| RAM | Random Access Memory |
| SchM | Schedule Manager |
| VLE | Variable Length Encoding |
| VSMD | Vendor-Specific Module Definition |
| XML | Extensible Markup Language |

# 2.5   Reference List

## Table 2-3.   Reference List

| # | Title | Version |
|---|-------|---------|
| 1 | Specification of FEE Driver | AUTOSAR Release 4.2.2 |
| 2 | S32K14X Reference Manual | Reference Manual, Rev. 4, 06/2017 |
| 3 | S32K142 Mask Set Errata for Mask 0N33V (0N33V) | 30/11/2017 |
| 4 | S32K144 Mask Set Errata for Mask 0N57U (0N57U) | 30/11/2017 |
| 5 | S32K146 Mask Set Errata for Mask 0N73V (0N73V) | 30/11/2017 |
| 6 | S32K148 Mask Set Errata for Mask 0N20V (0N20V) | 30/11/2017 |
| 7 | S32K118 Mask Set Errata for Mask 0N97V (0N97V) | 26/02/2018 |

**User Manual, Rev. 1.0**

# Chapter 3
# Driver

## 3.1 Requirements

Requirements for this driver are detailed in the AUTOSAR 4.2 Rev0002 Flash EEPROM Emulation Driver Software Specification document (see the Reference List table).

AUTOSAR FEE deviations from requirements are described in Deviation 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  Deviation from Requirements

The driver deviates from the AUTOSAR FEE driver software specification (see the Reference List) in several details.

There are also some additional requirements (on top of requirements detailed in AUTOSAR FEE Driver software specification) which must be satisfied for correct operation.

Table Table 3-1 provides Status column description.

**Table 3-1.  Description of Deviation Status Column**

| Term | Definition |
|---|---|
| N/A | Not Available |
| N/T | Not Testable |

*Table continues on the next page...*

**User Manual, Rev. 1.0**

**Table 3-1.  Description of Deviation Status Column (continued)**

| Term | Definition |
|------|-----------|
| N/S | Out of Scope |
| N/R | Unclear Requirement |
| N/I | Not Implemented |
| N/F | Not Fully Implemented |
| I/D | Implemented with Deviations |

Table 3-2 identifies the AUTOSAR requirements that are not fully implemented, implemented differently, or out of scope of the FEE driver.

**Table 3-2.  Driver Deviations Table**

| Requirement | Status | Description | Notes |
|------------|--------|-------------|-------|
| SWS_Fee_00084 | N/R | Imported Type<br>• Fls_AddressType<br>• Fls_LengthType<br>• MemIf_JobResultType<br>• MemIf_ModeType<br>• MemIf_StatusType<br>• Std_ReturnType<br>• Std_VersionInfoType | |
| SWS_Fee_00013 | I/D | The FEE module shall perform inter module checks to avoid integration of incompatible files: all included header files shall be checked by pre-processing directives. The FEE module shall thereby verify that <MODULENAME>_AR_RELEASE_MAJOR_VERSION and <MODULENAME>_AR_RELEASE_MINOR_VERSION are identical to the expected values, where <MODULENAME> is the module short name of the external module, which provides the included header file. If the values are not identical, the FEE module shall raise the development error FEE_E_INCOMPATIBLE_VERSIONS. | Compile-time error is raised in case the verification fails. |
| SWS_Fee_00009 | I/D | Blocks containing immediate data have to be written instantaneously, i.e. the FEE module has to ensure that it can write such blocks without the need to erase the corresponding memory area (e.g. by using pre-erased memory) and that the write request is not delayed by currently running module internal management operations. | Memory for an immediate data block must be pre-allocated by an Fee_EraseImmediateBlock() call and possible ongoing operation must be explicitly cancelled before a write request can be issued for any block holding immediate data. |
| SWS_Fee_00022 | N/F | If the current module status is MEMIF_IDLE or if the current module status is MEMIF_BUSY INTERNAL, the function Fee_Read shall accept the read request, copy the given / computed parameters to module internal variables, initiate a read job, set the FEE module status to MEMIF_BUSY, set the job result to MEMIF_JOB_PENDING and return with E_OK. | MEMIF_BUSY INTERNAL is used only for the Fee_Init job.In case the status is MEMIF_BUSY INTERNAL Fee will reject the job.For more details see iTWG ticket ENGR00385266. A change request to the AUTOSAR consortium was also created. |

*Table continues on the next page...*

**Table 3-2.   Driver Deviations Table (continued)**

| Requirement | Status | Description | Notes |
|---|---|---|---|
| SWS_Fee_00172 | I/D | If the current module status is MEMIF_UNINIT or MEMIF_BUSY, the function Fee_Read shall reject the job request and return with E_NOT_OK. | In case the status is MEMIF_BUSY INTERNAL Fee will also reject the job.For more details see iTWG ticket ENGR00385266. A change request to the AUTOSAR consortium was also created to clarify the behavior. |
| SWS_Fee_00025 | N/F | If the current module status is MEMIF_IDLE or if the current module status is MEMIF_BUSY INTERNAL, the function Fee_Write shall accept the write request, copy the given / computed parameters to module internal variables, initiate a write job, set the FEE module status to MEMIF_BUSY, set the job result to MEMIF_JOB_PENDING and return with E_OK. | MEMIF_BUSY INTERNAL is used only for the Fee_Init job.In case the status is MEMIF_BUSY INTERNAL Fee will reject the job.For more details see iTWG ticket ENGR00385266. A change request to the AUTOSAR consortium was also created. |
| SWS_Fee_00174 | I/D | If the current module status is MEMIF_UNINIT or MEMIF_BUSY, the function Fee_Write shall reject the job request and return with E_NOT_OK. | In case the status is MEMIF_BUSY INTERNAL Fee will also reject the job.For more details see iTWG ticket ENGR00385266. A change request to the AUTOSAR consortium was also created to clarify the behavior. |
| SWS_Fee_00100 | N/R | 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. |
| SWS_Fee_00102 | N/I | The configuration of the FEE module shall define the expected number of erase/write cycles for each logical block in the configuration parameter FeeNumberOfWriteCycles. | Not used due to usage of the two or more clusters emulation algorithm. |
| SWS_Fee_00103 | I/D | 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. |
| ECUC_Fee_00110 | N/I | Number of write cycles required for this block. | Not supported by FEE module due to emulation scheme. |
| SWS_Fee_00034 | N/R | If no internal operation is currently ongoing, the function Fee_GetStatus shall call the "GetStatus" function of the underlying flash driver and pass its return value back to the caller. | Misleading requirement, it shall return actual FEE status (which depends on internal FEE states). |
| SWS_Fee_00035 | N/R | The function Fee_GetJobResult shall call the "GetJobResult" function of the underlying flash driver and pass the return value back to the caller. | Misleading requirement, it shall return actual FEE job result. |
| SWS_Fee_00093 | N/R | The function Fee_GetVersionInfo shall return the version information of the FEE module. The version information includes: Module Id, Vendor Id, Vendor specific version numbers (BSW00407) | The implementation also fills VersionInfoPtr->instanceID |

*Table continues on the next page...*

**User Manual, Rev. 1.0**

## Table 3-2.   Driver Deviations Table (continued)

| Requirement | Status | Description | Notes |
|---|---|---|---|
| SWS_Fee_00023 | I/D | The function Fee_MainFunction shall check the consistency of the logical block being read before notifying the caller. If an inconsistency of the read data is detected or if the requested block can't be found, the function Fee_MainFunction shall set the job result to MEMIF_BLOCK_INCONSISTENT and call the error notification routine of the upper layer if configured. | MEMIF_BLOCK_INCONSISTENT is return value also for not-existing Fee blocks in NVM in case parameter FEE_MARK_EMPTY_BLOCKS_INVALID is OFF and in case block is pre-erased in the LEGACY mode. |
| SWS_Fee_00075 | N/R | The function Fee_MainFunction shall check, whether the block requested for reading has been invalidated by the upper layer module. If so, the function Fee_MainFunction shall set the job result to MEMIF_BLOCK_INVALID, call the job error notification function if configured. | MEMIF_BLOCK_INVALID is return value also for not-existing Fee blocks in NVM in case parameter FEE_MARK_EMPTY_BLOCKS_INVALID is ON. |
| SWS_Fee_00098 | N/R | NvM_JobEndNotification | The NvM_JobEndNotification is called whenever appropriate (if configured). |
| SWS_Fee_00099 | N/R | NvM_JobErrorNotification | The NvM_JobErrorNotification is called whenever appropriate (if configured). |
| SWS_Fee_00007 | N/R | 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 different mapping algorithm. |
| ECUC_Fee_00114 | N/I | 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. | Not supported by FEE module due to unclear polling concept. |
| ECUC_Fee_00149 | N/R | FeeDeviceIndex | Not used by FEE module. |
| ECUC_Fee_00153 | N/R | FeeMainFunctionPeriod | Not used by FEE module. |
| SWS_Fee_00130 | N/I | The modules status, the job result and the block meta information (see SWS_Fee_00049) shall be made available for debugging (reading). (RS_BRF_00420) | The AUTOSAR specified debug support shall not be implemented according to PR-MCAL-3330.fee. |
| SWS_Fee_00168 | N/I | If initialization is finished within Fee_Init, the function Fee_Init shall set the module state from MEMIF_BUSY_INTERNAL to MEMIF_IDLE once initialization has been successfully finished. | 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. |
| SWS_Fee_00187 | N/I | 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 | Not applicable. |

*Table continues on the next page...*

**User Manual, Rev. 1.0**

**Table 3-2.   Driver Deviations Table (continued)**

| Requirement | Status | Description | Notes |
|---|---|---|---|
| | | errors due to trying to read erased but not programmed flash cells. | |
| SWS_Fee_00010 | I/D | | The error code FEE_E_BUSY_INTERNAL was removed in ASR 4.2.2, but we decided to keep it in the smcal implementation in order to make sure that no other API is called until Fee initialization is finished.For more details see iTWG ticket ENGR00385266. A change request to the AUTOSAR consortium was also created. |

## 3.5   Driver Limitations

None.

## 3.6   Driver usage and 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_MainFunction 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

To achieve the same robustness as in the previous platforms, due to ECC page restriction and CACHE coherency issues, it is recommended to set the FeeVirtualPageSize to 16 bytes.

## FEE Block Always Available

According to the AUTOSAR FEE153 and FEE154 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 (cancelled, 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 FEE153 and FEE154 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 FEE153 and FEE154. 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 FEE153 and FEE154 (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 <u>one instance</u> 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 <u>unreserved</u> 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_EraseImmediateBlock function). Note that after Fee_EraseImmediateBlock function is called, new space is allocated and <u>the previous instance of the immediate block is no longer accessible</u>.

*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 3-3.  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<br><br>block is not in the cluster(if FEE_MARK_EMPTY_BLOCKS_INVALID = ON) | block was invalidated, or<br><br>block is not in the cluster(if FEE_MARK_EMPTY_BLOCKS_INVALID = ON)<br><br>space for the block is reserved |
| MEMIF_BLOCK_INCONSISTENT | block has corrupted header (wrong checksum, address, ...), or<br><br>block is not in the cluster(if FEE_MARK_EMPTY_BLOCKS_INVALID = OFF) or<br><br>block doesn't match the configuration (length, immediate, ...) | block has corrupted header (wrong checksum, address, ...), or<br><br>block is not in the cluster(if FEE_MARK_EMPTY_BLOCKS_INVALID = OFF) or<br><br>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.

## FEE Swap Foreign Blocks feature

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

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. BOOTLOADER - Block is used ONLY by the BOOTLOADER project APPLICATION - Block is used ONLY by the APPLICATION project Shared - Block is used for BOTH APPLICATION and BOOTLOADER projects
   - **FeeConfigAssignment** Defines for which project the current Fee configuration is used. BOOTLOADER - Configuration is used only by the BOOTLOADER project 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 FeeMaximumNumberBlocks 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

None.

## 3.8  Software specification

The following sections contains driver software specifications.

### 3.8.1  Define Reference

Constants supported by the driver are as per AUTOSAR FEE Driver software specification Version 4.2 Rev0002 .

#### 3.8.1.1  Macro FEE_DESERIALIZE

Deserialize scalar parameter from the buffer.

**Pre:** pDeserialPtr must be valid pointer.

**Post:** increments the pDeserialPtr by sizeof(ParamType).

**Prototype:** `FEE_DESERIALIZE(pDeserialPtr,ParamVal,ParamType)`

**User Manual, Rev. 1.0**

<p align="center">**Table 3-4.  FEE_DESERIALIZE**<br>**Arguments**</p>

| Name | Direction | Description |
|---|---|---|
| pDeserialPtr | **in/out** | Pointer to source buffer. |
| ParamVal | **output** | Deserialized parameter. |
| ParamType | **in** | Type of serialized parameter. |

## 3.8.1.2   Macro FEE_SERIALIZE

Serialize scalar parameter into the buffer.

<u>**Pre:**</u> pSerialPtr must be valid pointer.

<u>**Post:**</u> increments the pSerialPtr by sizeof(ParamType).

<u>**Prototype:**</u> `FEE_SERIALIZE(ParamVal,ParamType,pSerialPtr)`

<p align="center">**Table 3-5.  FEE_SERIALIZE**<br>**Arguments**</p>

| Name | Direction | Description |
|---|---|---|
| ParamVal | **input** | Serialized parameter. |
| ParamType | **input** | Type of serialized parameter. |
| pSerialPtr | **in/out** | Pointer to target buffer. |

## 3.8.2   Enum Reference

Enumeration of all constants supported by the driver are as per AUTOSAR FEE Driver software specification Version 4.2 Rev0002 .

## 3.8.2.1   Enumeration Fee_BlockStatusType

Status of Fee block header.

<p align="center">**User Manual, Rev. 1.0**</p>

## Table 3-6.  Enumeration Fee_BlockStatusType Values

| Name | Initializer | Description |
|------|-------------|-------------|
| FEE_BLOCK_VALID | 0 | Fee block is valid. |
| FEE_BLOCK_INVALID | | Fee block is invalid (has been invalidated). |
| FEE_BLOCK_INCONSISTENT | | Fee block is inconsistent (contains corrupted data). |
| FEE_BLOCK_HEADER_INVALID | | Fee block header is contains corrupted data. In case FEE_MARK_EMPTY_BLOCKS_INVALID is OFF, the status will be changed at cluster swap to FEE_BLOCK_NEVER_WRITTEN because invalid and invalidated blocks are discarded during cluster swap. |
| FEE_BLOCK_INVALIDATED | | Fee block header is invalidated by Fee_InvalidateBlock(BlockNumber)(not used when FEE_BLOCK_ALWAYS_AVAILABLE == STD_OFF). In case FEE_MARK_EMPTY_BLOCKS_INVALID is OFF, the status will be changed at cluster swap to FEE_BLOCK_NEVER_WRITTEN because invalid and invalidated blocks are discarded during cluster swap. |
| FEE_BLOCK_HEADER_BLANK | | Fee block header is blank (end of Fee block header list). |
| FEE_BLOCK_INCONSISTENT_COPY | | FEE data read error during swap (i.e. data area was allocated but could not be properly written). |
| FEE_BLOCK_NEVER_WRITTEN | | FEE block was never written in data flash. |

## 3.8.2.2   Enumeration Fee_BlockAssignmentType

Block assignment to a project.

## Table 3-7.   Enumeration Fee_BlockAssignmentType Values

| Name | Initializer | Description |
|------|-------------|-------------|
| FEE_PROJECT_SHARED | 0x01 | Fee block is used by the APPLICATION and BOOTLOADER projects. |
| FEE_PROJECT_APPLICATION | 0x02 | Fee block is used only by the APPLICATION project. |
| FEE_PROJECT_BOOTLOADER | 0x03 | Fee block is used only by the BOOTLOADER project. |
| FEE_PROJECT_RESERVED | 0xFF | The value is reserved. |

User Manual, Rev. 1.0

## 3.8.2.3   Enumeration Fee_ClusterStatusType

Status of Fee cluster header.

**Table 3-8.   Enumeration Fee_ClusterStatusType Values**

| Name | Initializer | Description |
|---|---|---|
| FEE_CLUSTER_VALID | 0 | 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. |

## 3.8.2.4   Enumeration Fee_JobType

Type of job currently executed by Fee_MainFunction.

**Table 3-9.   Enumeration Fee_JobType Values**

| Name | Initializer | Description |
|---|---|---|
| FEE_JOB_READ | 0 | Read Fee block. |
| FEE_JOB_WRITE | | Write Fee block to flash. |
| 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_ERASE_IMMEDIATE_DONE | | Finalize erase (pre-allocation) of Fee block. |
| FEE_JOB_INT_SCAN | | Initialize the cluster scan job. |
| FEE_JOB_INT_SCAN_CLR_HDR_PARSE | | Parse Fee cluster header. |
| FEE_JOB_INT_SCAN_CLR | | Scan active cluster of current cluster group. |

*Table continues on the next page...*

**User Manual, Rev. 1.0**

**Table 3-9.   Enumeration Fee_JobType Values (continued)**

| Name | Initializer | Description |
|---|---|---|
| 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_BLOCK | | Copy next block from source to target cluster. |
| FEE_JOB_INT_SWAP_CLR_FMT | | Format current Fee cluster in current Fee cluster group. |
| 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. |

## 3.8.3   Function Reference

Functions of all functions supported by the driver are as per AUTOSAR FEE Driver software specification Version 4.2 Rev0002 .

### 3.8.3.1   Function Fee_Cancel

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

**Details:**

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.

**Pre:** The module must be initialized.

**Post:** Changes Fee_ModuleStatus module status and job result Fee_JobResult internal variables.

**Note**

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

**User Manual, Rev. 1.0**

**Prototype:** `void Fee_Cancel(void);`

## 3.8.3.2  Function Fee_EraseImmediateBlock

Service to erase a logical block.



**Figure 3-1. Function Fee_EraseImmediateBlock References.**

**Details:**

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.

**Return:** Std_ReturnType.

**Pre:** The module must be initialized, not busy, BlockNumber must be valid, and type of Fee block must be immediate.

**Post:** changes Fee_ModuleStatus module status and Fee_JobBlockIndex, Fee_Job, and Fee_JobResult job control internal variables.

**Note**

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

**Prototype:** `Std_ReturnType Fee_EraseImmediateBlock(uint16 BlockNumber);`

**Table 3-10.  Fee_EraseImmediateBlock Arguments**

| Type | Name | Direction | Description |
|------|------|-----------|-------------|
| uint16 | BlockNumber | **input** | Number of logical block, also denoting. |

**Table 3-11.  Fee_EraseImmediateBlock Return Values**

| Name | Description |
|---|---|
| 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. |

## 3.8.3.3  Function Fee_ForceSwapOnNextWrite

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

**Details:**

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_ForceSwapOnNextWrite 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.

**Return:** Std_ReturnType.

**Pre:** The module must be initialized, not busy and clrGrpIndex 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

**Prototype:** `Std_ReturnType Fee_ForceSwapOnNextWrite(uint8 clrGrpIndex);`

**Table 3-12.  Fee_ForceSwapOnNextWrite Arguments**

| Type | Name | Direction | Description |
|---|---|---|---|
| uint8 | clrGrpIndex | **input** | Index of the selected cluster group. |

**Table 3-13.  Fee_ForceSwapOnNextWrite Return Values**

| Name | Description |
|---|---|
| E_NOT_OKmodule | Is not initialized, busy or clrGrpIndex is not in the valid range. |
| E_OKNo | More space available in the selected cluster. |

**User Manual, Rev. 1.0**

## 3.8.3.4   Function Fee_GetJobResult

Return the result of the last job.

**Details:**

Return the result of the last job synchronously.

**Return:** MemIf_JobResultType.

### Note
The function Autosar Service ID[hex]: 0x06.Synchronous.Non Reentrant.

**Prototype:** `MemIf_JobResultType Fee_GetJobResult(void);`

### Table 3-14.   Fee_GetJobResult Return Values

| Name | Description |
|---|---|
| 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) or the block is empty (if FEE_MARK_EMPTY_BLOCKS_INVALID is OFF). |
| MEMIF_BLOCK_INVALID | The requested block has been invalidated (the requested read operation can not be performed) or the block is empty (if FEE_MARK_EMPTY_BLOCKS_INVALID is ON). |

## 3.8.3.5   Function Fee_GetRunTimeInfo

Service to read runtime information in the selected cluster.

**Pre:** The module must be initialized, not busy, clrGrpIndex must be valid and pClrGrpRTInfo must be not NULL_PTR

**Prototype:** `void Fee_GetRunTimeInfo(uint8 clrGrpIndex, Fee_ClusterGroupRuntimeInfoType *pClrGrpRTInfo);`

**Table 3-15.   Fee_GetRunTimeInfo Arguments**

| Type | Name | Direction | Description |
|---|---|---|---|
| uint8 | clrGrpIndex | **input** | Index of the selected cluster group. |
| Fee_ClusterGroupRuntimeInfoType * | pClrGrpRTInfo | **input** | Pointer to where to store the runtime information of the selected cluster group |

## 3.8.3.6   Function Fee_GetStatus

Return the Fee module state.

### Details:

Return the Fee module state synchronously.

### Note
The function Autosar Service ID[hex]: 0x05.SynchronousNon Reentrant

**Return:** Fee_ModuleStatus.

**Prototype:** `MemIf_StatusType Fee_GetStatus(void);`

**Table 3-16.   Fee_GetStatus Return Values**

| Name | Description |
|---|---|
| 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. |

## 3.8.3.7   Function Fee_GetVersionInfo

Return the version information of the Fee module.

### Details:

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

**Pre:** VersionInfoPtr must not be NULL_PTR.

**User Manual, Rev. 1.0**

### Note

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

**Prototype:** `void Fee_GetVersionInfo(Std_VersionInfoType *VersionInfoPtr);`

**Table 3-17.  Fee_GetVersionInfo Arguments**

| Type | Name | Direction | Description |
|------|------|-----------|-------------|
| Std_VersionInfoType * | VersionInfoPtr | **output** | Pointer to where to store the version information of this module . |

## 3.8.3.8   Function Fee_Init

Service to initialize the FEE module.

### Details:

The function Fee_Init shall initialize the Flash EEPROM Emulation module. The
parameter ConfigPtr must always be NULL_PTR.

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

### Note

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

**Prototype:** `void Fee_Init( const Fee_ConfigType* ConfigPtr );`

## 3.8.3.9   Function Fee_InvalidateBlock

Service to invalidate a logical block.



**Figure 3-2. Function Fee_InvalidateBlock References.**

**Return:** Std_ReturnType.

**Pre:** The module must be initialized, not busy, and BlockNumber must be valid.

**User Manual, Rev. 1.0**

**Post:** changes Fee_ModuleStatus module status and Fee_JobBlockIndex, Fee_Job, and Fee_JobResult job control internal variables. EEPROM.

### Note

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

**Prototype:** `Std_ReturnType Fee_InvalidateBlock(uint16 BlockNumber);`

#### Table 3-18.   Fee_InvalidateBlock Arguments

| Type | Name | Direction | Description |
|------|------|-----------|-------------|
| uint16 | BlockNumber | **input** | Number of logical block, also denoting start address of that block in flash memory. |

#### Table 3-19.   Fee_InvalidateBlock Return Values

| Name | Description |
|------|-------------|
| 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. |

## 3.8.3.10   Function Fee_JobEndNotification

Service to report the FEE module the successful end of an asynchronous operation.



**Figure 3-3. Function Fee_JobEndNotification References.**

**Details:**

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

**Pre:** The module must be initialized.

**Post:** Changes Fee_ModuleStatus module status and Fee_JobResult internal variables.

### Note

The function Autosar Service ID[hex]: 0x10.Synchronous.Non Reentrant

**Prototype:** `void Fee_JobEndNotification(void);`

**User Manual, Rev. 1.0**

### 3.8.3.11  Function Fee_JobErrorNotification

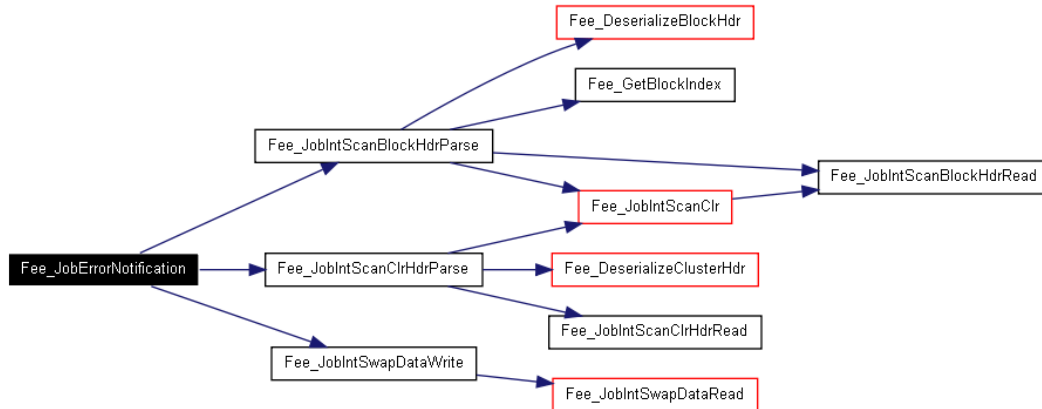Service to report the FEE module the failure of an asynchronous operation.



**Figure 3-4. Function Fee_JobErrorNotification References.**

<u>**Details**</u>**:**

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

<u>**Pre**</u>**:** The module must be initialized.

<u>**Post**</u>**:** Changes Fee_ModuleStatus module status and Fee_JobResult internal variables.

**Note**

The function Autosar Service ID[hex]: 0x11.Synchronous.Non Reentrant.

<u>**Prototype**</u>**:** `void Fee_JobErrorNotification(void);`

### 3.8.3.12  Function Fee_MainFunction

Service to handle the requested read / write / erase jobs respectively the internal management operations.



**Figure 3-5. Function Fee_MainFunction References.**

<u>**Details**</u>**:**

**User Manual, Rev. 1.0**

The function shall asynchronously handle the requested read / write / erase jobs respectively the internal management operations. The function shall check, whether the block requested for reading has been invalidated by the upper layer module. If so, the function shall set the job result to MEMIF_BLOCK_INVALID and call the error notification routine of the upper layer if configured. The function shall check the consistency of the logical block being read before notifying the caller. If an inconsistency of the read data is detected, the function shall set the job result to MEMIF_BLOCK_INCONSISTENT and call the error notification routine of the upper layer.

**Pre:** The module must be initialized.

**Note**

The function Autosar Service ID[hex]: 0x12.

**Prototype:** `void Fee_MainFunction(void);`

## 3.8.3.13 Function Fee_Read

Service to initiate a read job.



**Figure 3-6. Function Fee_Read References.**

**Details:**

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

**Pre:** The module must be initialized, not busy, BlockNumber must be valid, Length != 0, DataBufferPtr != NULL_PTR, BlockOffset and (BlockOffset + Length - 1) must be in range.

**Post:** changes Fee_ModuleStatus module status and Fee_JobBlockOffset, Fee_JobBlockLength, Fee_JobBlockIndex, Fee_JobDataDestPtr, Fee_Job, Fee_JobResult job control internal variables.

**Return:** Std_ReturnType.

**Note**

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

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

**Table 3-20. Fee_Read Arguments**

| Type | Name | Direction | Description |
|------|------|-----------|-------------|
| uint16 | BlockNumber | **input** | Number of logical block, also denoting start address of that block in flash memory. |
| uint16 | BlockOffset | **input** | Read address offset inside the block. |
| uint8 * | DataBufferPtr | **output** | Pointer to data buffer. |
| uint16 | Length | **input** | Number of bytes to read. |

**Table 3-21. Fee_Read Return Values**

| Name | Description |
|------|-------------|
| 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. |

## 3.8.3.14  Function Fee_SetMode

Set the Fee module' s operation mode to the given Mode.

**Details:**

Call the Fls_SetMode function of the underlying flash driver.

**Pre:** The module must be initialized and not busy.

**Note**

The function Autosar Service ID[hex]: 0x01.SynchronousNon Reentrant

**Prototype:** `void Fee_SetMode(MemIf_ModeType Mode);`

**Table 3-22. Fee_SetMode Arguments**

| Type | Name | Direction | Description |
|------|------|-----------|-------------|
| MemIf_ModeType | Mode | **input** | (Either MEMIF_MODE_FAST or MEMIF_MODE_SLOW). |

**User Manual, Rev. 1.0**

### 3.8.3.15  Function Fee_Write

Service to initiate a write job.



**Figure 3-7. Function Fee_Write References.**

**<u>Details:</u>**

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 / 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.

**<u>Return:</u>** Std_ReturnType.

**<u>Pre:</u>** The module must be initialized, not busy, BlockNumber must be valid, and DataBufferPtr != NULL_PTR. Before call the function "Fee_Write" for immediate date must be called the function "Fee_EraseImmediateBlock".

**<u>Post:</u>** changes Fee_ModuleStatus module status and Fee_JobBlockIndex, Fee_JobDataDestPtr, Fee_Job, Fee_JobResult job control internal variables.

**Note**
The function Autosar Service ID[hex]:
0x03.Asynchronous.Non Reentrant.

**<u>Prototype:</u>** `Std_ReturnType Fee_Write(uint16 BlockNumber,const uint8 *DataBufferPtr);`

**Table 3-23.  Fee_Write Arguments**

| Type | Name | Direction | Description |
|---|---|---|---|
| uint16 | BlockNumber | **input** | Number of logical block, also denoting start address of that block in emulated EEPROM. |
| const uint8 * | DataBufferPtr | **output** | Pointer to data buffer. |

**User Manual, Rev. 1.0**

**Table 3-24. Fee_Write Return Values**

| Name | Description |
|---|---|
| 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. |

## 3.8.4  Structs Reference

Data structures supported by the driver are as per AUTOSAR FEE Driver software specification Version 4.2 Rev0002 .

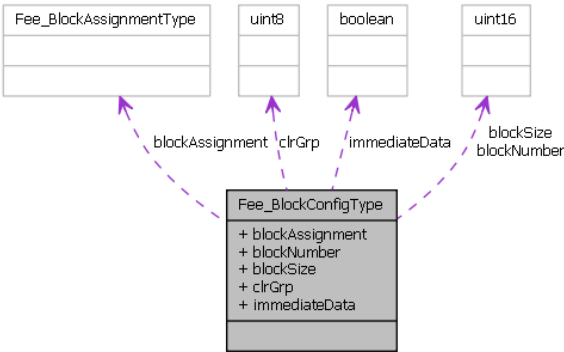### 3.8.4.1  Structure Fee_BlockConfigType

Fee block configuration structure.



**Figure 3-8. Struct Fee_BlockConfigType**

### Declaration:

```
typedef struct
                    {
                      uint16 blockNumber,
                            uint16 blockSize,
                            uint8 clrGrp,
                            boolean immediateData,
                            Fee_BlockAssignmentType blockAssignment
                    } Fee_BlockConfigType;
```

**Table 3-25. Structure Fee_BlockConfigType member description**

| Member | Description |
|---|---|
| blockNumber | Fee block number. |
| blockSize | Size of Fee block in bytes. |
| clrGrp | Index of cluster group the Fee block belongs to. |
| immediateData | TRUE if immediate data block. |
| blockAssignment | block assignment to a project. |

### 3.8.4.2  Structure Fee_BlockInfoType

Fee block run-time status.



**Figure 3-9. Struct Fee_BlockInfoType**

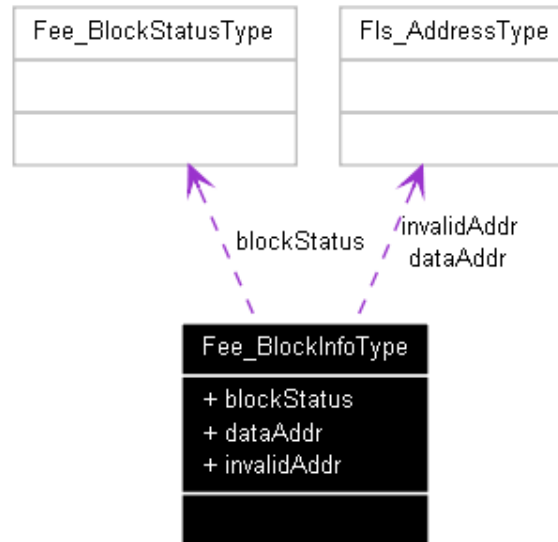## Declaration:

```
typedef struct
                        {
                          Fee_BlockStatusType blockStatus,
                                Fls_AddressType dataAddr,
                                Fls_AddressType invalidAddr
                        } Fee_BlockInfoType;
```

**Table 3-26.  Structure Fee_BlockInfoType member description**

| Member | Description |
|---|---|
| blockStatus | Current status of Fee block. |
| dataAddr | Address of Fee block data in flash. |
| invalidAddr | Address of Fee block invalidation field in flash. |

### 3.8.4.3  Structure Fee_ClusterGroupInfoType

Fee cluster group run-time status.

**Figure 3-10. Struct Fee_ClusterGroupInfoType**

## Declaration:

```
typedef struct
                    {
                       uint8 actClr,
                              uint32 actClrID,
                              Fls_AddressType dataAddrIt,
                              Fls_AddressType hdrAddrIt
                    } Fee_ClusterGroupInfoType;
```
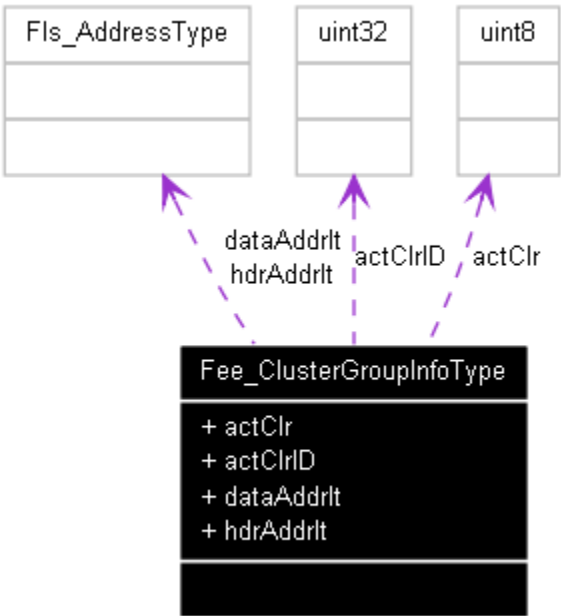
**Table 3-27.   Structure Fee_ClusterGroupInfoType member description**

| Member | Description |
|--------|-------------|
| actClr | Index of active cluster. |
| actClrID | ID of active cluster. |
| dataAddrIt | Address of current Fee data block in flash. |
| hdrAddrIt | Address of current Fee block header in flash. |

## 3.8.4.4   Structure Fee_ClusterGroupRuntimeInfoType

Fee cluster group run-time Information.

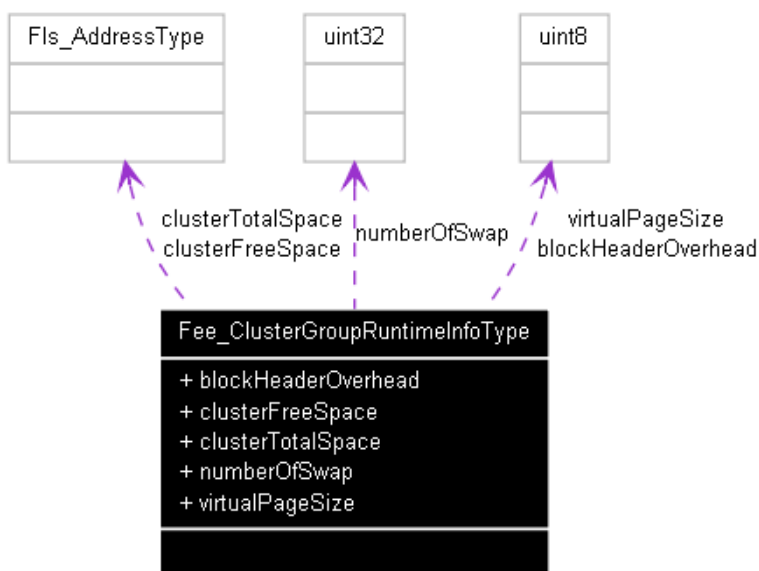**Figure 3-11. Struct Fee_ClusterGroupRuntimeInfoType**

## Declaration:

```
typedef struct
                    {
                       uint8 blockHeaderOverhead,
                               Fls_AddressType clusterFreeSpace,
                               Fls_AddressType clusterTotalSpace,
                               uint32 numberOfSwap,
                               uint8 virtualPageSize
                    } Fee_ClusterGroupRuntimeInfoType;
```

**Table 3-28.  Structure Fee_ClusterGroupRuntimeInfoType member description**

| Member | Description |
|---|---|
| blockHeaderOverhead | Block Overhead (header valid and inval flag). |
| clusterFreeSpace | Free space in the selected cluster group. |
| clusterTotalSpace | Total space in the selected cluster group. |
| numberOfSwap | Number of cluster swap performed in the selected cluster group. |
| virtualPageSize | Fee Virtual Page Size. |

## 3.8.4.5   Structure Fee_ClusterGroupType

Fee cluster group configuration structure.

## Declaration:

```
typedef struct
                    {
                       const Fee_ClusterType * const clrPtr;
                       uint32 clrCount;
                    Fls_LengthType reservedSize;
                    } Fee_ClusterGroupType;
```

**User Manual, Rev. 1.0**

**Table 3-29.   Structure Fee_ClusterGroupType member description**

| Member | Description |
|---|---|
| clrCount | Pointer to array of Fee cluster configurations Number of clusters in cluster group. |
| clrPtr | Number of clusters in cluster group |
| reservedSize | Size of reserved area in the given cluster group (memory occupied by immediate blocks) |

## 3.8.4.6   Structure Fee_ClusterType
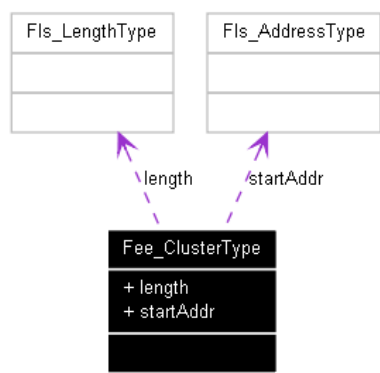
Fee cluster configuration structure.



**Figure 3-12. Struct Fee_ClusterType**

### Declaration:

```
typedef struct
                          {
                            Fls_LengthType length,
                                  Fls_AddressType startAddr
                          } Fee_ClusterType;
```

**Table 3-30.   Structure Fee_ClusterType member description**

| Member | Description |
|---|---|
| length | Size of Fee cluster in bytes. |
| startAddr | Address of Fee cluster in flash. |

## 3.8.5   Types Reference

Types supported by the driver are as per AUTOSAR FEE Driver software specification Version 4.2 Rev0002 .

### 3.8.5.1   Fee_ConfigType

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

## 3.9   Symbolic Names Disclaimer

All containers having the symbolic name tag set as true in the Autosar schema will generate defines like:

#define <Container_Short_Name> <Container_ID>

For this reason it is forbidden to duplicate the name of such containers across the MCAL configuration, 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 FEE Driver. The most of the parameters are described below.

## 4.1  Configuration elements of Fee

**Included forms:**

- IMPLEMENTATION_CONFIG_VARIANT
- FeeGeneral
- CommonPublishedInformation
- FeePublishedInformation
- FeeClusterGroup
- FeeBlockConfiguration

## 4.2  Form IMPLEMENTATION_CONFIG_VARIANT

**VariantPreCompile**: Precompile configuration parameters.
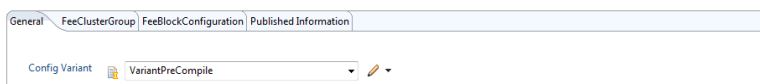


**Figure 4-1. Tresos Plugin snapshot for IMPLEMENTATION_CONFIG_VARIANT form.**

**Table 4-1.  Attribute IMPLEMENTATION_CONFIG_VARIANT detailed description**

| Property | Value |
|---|---|
| Label | Config Variant |
| Default | VariantPreCompile |

# 4.3  Form FeeGeneral

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



**Figure 4-2. Tresos Plugin snapshot for FeeGeneral form.**

## 4.3.1  FeeDevErrorDetect (FeeGeneral)

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

**true**: Development error detection enabled.

**false**: Development error detection disabled.

**Table 4-2.  Attribute FeeDevErrorDetect (FeeGeneral) detailed description**

| Property | Value |
|---|---|
| Label | FEE Development Error Detect |
| Type | BOOLEAN |
| Origin | AUTOSAR_ECUC |
| Symbolic Name | false |
| Default | true |

## 4.3.2 FeeEnableUserModeSupport (FeeGeneral)

Vendor specific: 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.

**Table 4-3.  Attribute FeeEnableUserModeSupport (FeeGeneral) detailed description**

| Property | Value |
| --- | --- |
| Label | FEE Enable User Mode Support |
| Type | BOOLEAN |
| Origin | NXP |
| Symbolic Name | false |
| Default | false |

## 4.3.3 FeeMainFunctionPeriod (FeeGeneral)

The period between successive calls to the main function in seconds

**Table 4-4.  Attribute FeeMainFunctionPeriod (FeeGeneral) detailed description**

| Property | Value |
| --- | --- |
| Label | Fee Main Function Period |
| Type | FLOAT |
| Origin | AUTOSAR_ECUC |
| Symbolic Name | false |
| Default | 5.0E-3 |
| Invalid | Range<br>        <=100000<br>        >=1.0E-7 |

## 4.3.4 FeeNvmJobEndNotification (FeeGeneral)

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.

**User Manual, Rev. 1.0**

**Table 4-5.   Attribute FeeNvmJobEndNotification (FeeGeneral) detailed description**

| Property | Value |
|---|---|
| Label | FEE Nvm Job End Notification |
| Type | FUNCTION-NAME |
| Origin | AUTOSAR_ECUC |
| Symbolic Name | false |
| Default | NvM_JobEndNotification |
| Enable | false |

## 4.3.5   FeeNvmJobErrorNotification (FeeGeneral)

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.

**Table 4-6.   Attribute FeeNvmJobErrorNotification (FeeGeneral) detailed description**

| Property | Value |
|---|---|
| Label | FEE Nvm Job Error Notification |
| Type | FUNCTION-NAME |
| Origin | AUTOSAR_ECUC |
| Symbolic Name | false |
| Default | NvM_JobErrorNotification |
| Enable | false |

## 4.3.6   FeeClusterFormatNotification (FeeGeneral)

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 this notification to have it set as NULL_PTR.

**Table 4-7.   Attribute FeeClusterFormatNotification (FeeGeneral) detailed description**

| Property | Value |
|---|---|
| Label | Fee Cluster Format During Init Notification |
| Type | FUNCTION-NAME |
| Origin | NXP |
| Symbolic Name | false |

*Table continues on the next page...*

**Table 4-7. Attribute FeeClusterFormatNotification (FeeGeneral) detailed description (continued)**

| Property | Value |
|---|---|
| Default | NULL_PTR |
| Enable | false |

## 4.3.7 FeePollingMode (FeeGeneral)

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

### Note

This paramater is not utilized in this BSW module implementation

**Table 4-8. Attribute FeePollingMode (FeeGeneral) detailed description**

| Property | Value |
|---|---|
| Label | FEE Polling Mode |
| Type | BOOLEAN |
| Origin | AUTOSAR_ECUC |
| Symbolic Name | false |
| Default | false |

## 4.3.8 FeeSetModeSupported (FeeGeneral)

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).

**Table 4-9. Attribute FeeSetModeSupported (FeeGeneral) detailed description**

| Property | Value |
|---|---|
| Label | FEE Set Mode Supported |

*Table continues on the next page...*

**Table 4-9. Attribute FeeSetModeSupported (FeeGeneral) detailed description (continued)**

| Property | Value |
| --- | --- |
| Type | BOOLEAN |
| Origin | AUTOSAR_ECUC |
| Symbolic Name | false |
| Default | true |

## 4.3.9   FeeVersionInfoApi (FeeGeneral)

Pre-processor switch to enable / disable the API to read out the modules version information.

**true**: Version info API enabled.

**false**: Version info API disabled.

**Table 4-10. Attribute FeeVersionInfoApi (FeeGeneral) detailed description**

| Property | Value |
| --- | --- |
| Label | FEE Version Info Api |
| Type | BOOLEAN |
| Origin | AUTOSAR_ECUC |
| Symbolic Name | false |
| Default | true |

## 4.3.10   FeeVirtualPageSize (FeeGeneral)

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

**Table 4-11. Attribute FeeVirtualPageSize (FeeGeneral) detailed description**

| Property | Value |
| --- | --- |
| Label | FEE Virtual Page Size |
| Type | INTEGER |
| Origin | AUTOSAR_ECUC |
| Symbolic Name | false |
| Default | 32 |
| Invalid | Range<br>        <=65535<br>        >=8 |

**User Manual, Rev. 1.0**

NXP Semiconductors

## 4.3.11  FeeDataBufferSize (FeeGeneral)

**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_VIRTUAL_PAGE_SIZE

**Table 4-12.  Attribute FeeDataBufferSize (FeeGeneral) detailed description**

| Property | Value |
|---|---|
| Label | FEE Data Buffer Size |
| Type | INTEGER |
| Origin | Custom |
| Symbolic Name | false |
| Invalid | Range<br>    >=0<br>    <=65535 |

## 4.3.12  FeeBlockAlwaysAvailable (FeeGeneral)

**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:

**User Manual, Rev. 1.0**

- FEE_BLOCK_INVALID if the block was invalidated or the block is not present in the cluster at all(if FEE_MARK_EMPTY_BLOCKS_INVALID is ON).
- FEE_BLOCK_VALID in case the previous instance of the block exists and was not invalidated.

**Table 4-13.  Attribute FeeBlockAlwaysAvailable (FeeGeneral) detailed description**

| Property | Value |
|---|---|
| Label | FEE Block Always Available |
| Type | BOOLEAN |
| Origin | Custom |
| Symbolic Name | false |
| Default | false |

## 4.3.13   FeeLegacyMode (FeeGeneral)

**Vendor specific:** This parameter allows to specify a way how the immediate data blocks and allocation of their data areas are handled.

- **false**: Part of each cluster is dynamically reserved for this kind of data. Size of this area is computed from the configuration to be able to hold <u>one instance</u> of all immediate data blocks. Contrary, standard blocks cannot be stored in this reserved area. It is ensured that unless the given immediate data block is also stored in the reserved area, its reservation operation (Fee_EraseImmediateBlock) won't trigger a cluster swap. In other words, in an (almost) empty cluster, the Fee_EraseImmediateBlock does not pre-allocate any memory.
- **true**: Data area for immediate blocks is reserved upon each call to the Fee_EraseImmediateBlock. Previous instance of a given block (if any) is thus effectively lost (hidden).

**Table 4-14.  Attribute FeeLegacyMode (FeeGeneral) detailed description**

| Property | Value |
|---|---|
| Label | Fee Legacy Mode |
| Type | BOOLEAN |
| Origin | Custom |
| Symbolic Name | false |
| Default | false |

## 4.3.14   FeeLegacyEraseMode (FeeGeneral)

**Vendor specific:** In the non-legacy mode of immediate data handling, it is possible to specify whether the Fee_EraseImmediateBlock function invalidates the already present instance of the given block.

This option is valid only in the default (non-legacy) mode.

- **false**: The Fee_EraseImmediateBlock does not perform invalidation of the previous block.
- **true**: The Fee_EraseImmediateBlock invalidates the previous block thus mimicking the behaviour in the Legacy Mode.

**Table 4-15.   Attribute FeeLegacyEraseMode (FeeGeneral) detailed description**

| Property | Value |
| --- | --- |
| Label | Fee EraseImmediate Legacy Mode |
| Type | BOOLEAN |
| Origin | Custom |
| Symbolic Name | false |
| Default | false |

## 4.3.15   FeeSwapForeignBlocksEnabled (FeeGeneral)

**Vendor specific:** With FEE in the mode "FeeSwapForeignBlocksEnabled" it is possible to have different block configurations for the BOOTLOADER and APPLICATION projects. This is possible by swapping also the foreign blocks. For more details please see chapter **3.6 Driver usage and configuration tips** . The mode FeeSwapForeignBlocksEnabled = ON is not recommended for the production phase.

- **false**: Fee does not consider the foreign blocks at cluster swap.
- **true**: Fee considers the foreign blocks at cluster swap.

**Table 4-16.   Attribute FeeSwapForeignBlocksEnabled (FeeGeneral) detailed description**

| Property | Value |
| --- | --- |
| Label | Fee Swap Foreign Blocks Enabled |
| Type | BOOLEAN |
| Origin | Custom |
| Symbolic Name | false |
| Default | false |

## 4.3.16   FeeConfigAssignment (FeeGeneral)

**Vendor specific:** Defines for which project the current Fee configuration is used.
  - **APPLICATION**: Configuration is used only by the APPLICATION project.
  - **BOOTLOADER**: Configuration is used only by the BOOTLOADER project.

**Table 4-17.   Attribute FeeConfigAssignment (FeeGeneral) detailed description**

| Property | Value |
|---|---|
| Label | Fee Configuration Assignment |
| Type | ENUMERATION |
| Origin | Custom |
| Symbolic Name | false |
| Default | APPLICATION |

## 4.3.17   FeeMaximumNumberBlocks (FeeGeneral)

**Vendor specific:** 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.

**Table 4-18.   Attribute FeeMaximumNumberBlocks (FeeGeneral) detailed description**

| Property | Value |
|---|---|
| Label | FeeMaximumNumberBlocks |
| Type | INTEGER |
| Origin | Custom |
| Symbolic Name | false |
| Default | 1 |

## 4.3.18  FeeMarkEmptyBlocksInvalid (FeeGeneral)

**Vendor specific:** 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.

- **false**: Fee will mark the never written blocks as INCONSISTENT.
- **true**: Fee will mark the never written blocks as INVALID.

**Table 4-19.  Attribute FeeMarkEmptyBlocksInvalid (FeeGeneral) detailed description**

| Property | Value |
|---|---|
| Label | Fee Mark Empty Blocks As Invalid |
| Type | BOOLEAN |
| Origin | Custom |
| Symbolic Name | false |
| Default | false |

## 4.4  Form CommonPublishedInformation

Common container, aggregated by all modules. It contains published information about vendor and versions.



**Figure 4-3. Tresos Plugin snapshot for CommonPublishedInformation form.**

## 4.4.1   ArReleaseMajorVersion (CommonPublishedInformation)

Major version number of AUTOSAR specification on which the appropriate implementation is based on.

**Table 4-20.   Attribute ArReleaseMajorVersion (CommonPublishedInformation) detailed description**

| Property | Value |
|---|---|
| Label | AUTOSAR Major Version |
| Type | INTEGER_LABEL |
| Origin | Custom |
| Symbolic Name | false |
| Default | 4 |
| Invalid | Range<br>    >=4<br>    <=4 |

## 4.4.2   ArReleaseMinorVersion (CommonPublishedInformation)

Minor version number of AUTOSAR specification on which the appropriate implementation is based on.

**Table 4-21.   Attribute ArReleaseMinorVersion (CommonPublishedInformation) detailed description**

| Property | Value |
|---|---|
| Label | AUTOSAR Minor Version |
| Type | INTEGER_LABEL |
| Origin | Custom |
| Symbolic Name | false |
| Default | 2 |
| Invalid | Range<br>    >=2<br>    <=2 |

## 4.4.3   ArReleaseRevisionVersion (CommonPublishedInformation)

Revision version number of AUTOSAR specification on which the appropriate implementation is based on.

**User Manual, Rev. 1.0**

**Table 4-22.   Attribute ArReleaseRevisionVersion (CommonPublishedInformation) detailed description**

| Property | Value |
|---|---|
| Label | AUTOSAR Release Revision Version |
| Type | INTEGER_LABEL |
| Origin | Custom |
| Symbolic Name | false |
| Default | 2 |
| Invalid | Range<br>    >=2<br>    <=2 |

## 4.4.4   ModuleId (CommonPublishedInformation)

Module ID of this module from Module List.

**Table 4-23.   Attribute ModuleId (CommonPublishedInformation) detailed description**

| Property | Value |
|---|---|
| Label | Module Id |
| Type | INTEGER_LABEL |
| Origin | Custom |
| Symbolic Name | false |
| Default | 21 |
| Invalid | Range<br>    >=21<br>    <=21 |

## 4.4.5   SwMajorVersion (CommonPublishedInformation)

Major version number of the vendor specific implementation of the module. The numbering is vendor specific.

**Table 4-24.   Attribute SwMajorVersion (CommonPublishedInformation) detailed description**

| Property | Value |
|---|---|
| Label | Software Major Version |
| Type | INTEGER_LABEL |
| Origin | Custom |
| Symbolic Name | false |

*Table continues on the next page...*

**Table 4-24. Attribute SwMajorVersion (CommonPublishedInformation) detailed description (continued)**

| Property | Value |
|---|---|
| Default | 1 |
| Invalid | Range<br>    >=1<br>    <=1 |

# 4.4.6  SwMinorVersion (CommonPublishedInformation)

Minor version number of the vendor specific implementation of the module. The numbering is vendor specific.

**Table 4-25. Attribute SwMinorVersion (CommonPublishedInformation) detailed description**

| Property | Value |
|---|---|
| Label | Software Minor Version |
| Type | INTEGER_LABEL |
| Origin | Custom |
| Symbolic Name | false |
| Default | 0 |
| Invalid | Range<br>    >=0<br>    <=0 |

# 4.4.7  SwPatchVersion (CommonPublishedInformation)

Patch level version number of the vendor specific implementation of the module. The numbering is vendor specific.

**Table 4-26. Attribute SwPatchVersion (CommonPublishedInformation) detailed description**

| Property | Value |
|---|---|
| Label | Software Patch Version |
| Type | INTEGER_LABEL |
| Origin | Custom |
| Symbolic Name | false |
| Default | 1 |
| Invalid | Range<br>    >=1<br>    <=1 |

## 4.4.8   VendorApiInfix (CommonPublishedInformation)

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><Api name from SWS>. 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.

**Table 4-27.   Attribute VendorApiInfix (CommonPublishedInformation) detailed description**

| Property | Value |
|---|---|
| Label | Vendor Api Infix |
| Type | STRING_LABEL |
| Origin | Custom |
| Symbolic Name | false |
| Default | |
| Enable | false |

## 4.4.9   VendorId (CommonPublishedInformation)

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

**Table 4-28.   Attribute VendorId (CommonPublishedInformation) detailed description**

| Property | Value |
|---|---|
| Label | Vendor Id |
| Type | INTEGER_LABEL |
| Origin | Custom |
| Symbolic Name | false |
| Default | 43 |
| Invalid | Range<br>        >=43<br>        <=43 |

**User Manual, Rev. 1.0**

# 4.5  Form 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.
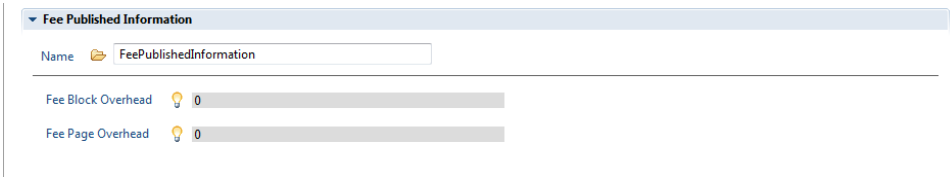


**Figure 4-4. Tresos Plugin snapshot for FeePublishedInformation form.**

## 4.5.1  FeeBlockOverhead (FeePublishedInformation)

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_VIRTUAL_PAGE_SIZE

**Table 4-29.  Attribute FeeBlockOverhead (FeePublishedInformation) detailed description**

| Property | Value |
|---|---|
| Label | FEE Block Overhead |
| Type | INTEGER_LABEL |
| Origin | AUTOSAR_ECUC |
| Symbolic Name | false |
| Default | 0 |
| Invalid | Range<br>        <=65535<br>        >=0 |

## 4.5.2  FeePageOverhead (FeePublishedInformation)

Management overhead per page in bytes

**User Manual, Rev. 1.0**

**Note**

The page management overhead is 0 bytes

**Table 4-30. Attribute FeePageOverhead (FeePublishedInformation) detailed description**

| Property | Value |
|---|---|
| Label | FEE Page Overhead |
| Type | INTEGER_LABEL |
| Origin | AUTOSAR_ECUC |
| Symbolic Name | false |
| Default | 0 |
| Invalid | Range<br>    <=65535<br>    >=0 |

# 4.6 Form FeeClusterGroup

**Vendor specific:** Configuration of cluster group specific parameters for the Flash EEPROM Emulation module.

**Included forms:**
 • Form FeeCluster
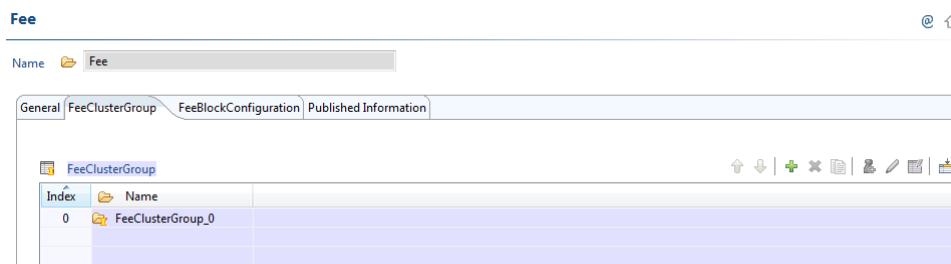


**Figure 4-5. Tresos Plugin snapshot for FeeClusterGroup form.**

# 4.6.1 Form FeeCluster

**Vendor specific:** Configuration of cluster specific parameters for the Flash EEPROM Emulation module.

**Is included by form:**Form FeeClusterGroup

**Included forms:**
 • Form FeeSector

**User Manual, Rev. 1.0**
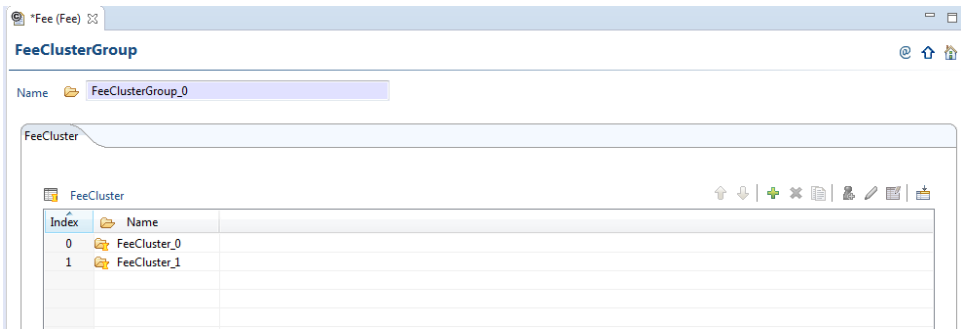
**Figure 4-6. Tresos Plugin snapshot for FeeCluster form.**

## Note

When generate to epc file, FeeCluster will be listed by name and in alphabetical order.

## 4.6.1.1 Form FeeSector

**Vendor specific:** Configuration of sector specific parameters for the Flash EEPROM Emulation module.

**Is included by form:** Form FeeCluster
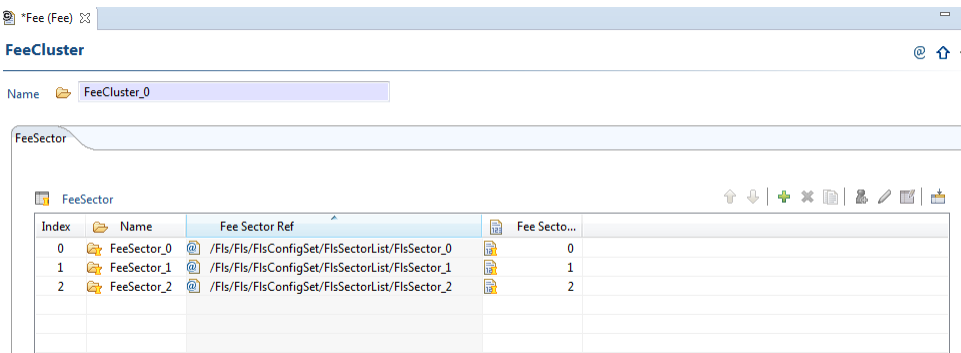


**Figure 4-7. Tresos Plugin snapshot for FeeSector form.**

## 4.6.1.1.1 FeeSectorRef (FeeSector)

**Vendor specific:** Reference to a logical FLS sector the FEE cluster consists of.

**Table 4-31. Attribute FeeSectorRef (FeeSector) detailed description**

| Property | Value |
|----------|-------|
| Label | FEE Sector Ref |
| Type | REFERENCE |
| Origin | Custom |

**Note**

When generate to epc file, FeeSectorRef will be listed by name and in alphabetical order.

### 4.6.1.1.2 FeeSectorIndex (FeeSector)

**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.

**Table 4-32.   Attribute FeeSectorIndex (FeeSector) detailed description**

| Property | Value |
|----------|-------|
| Label | FEE Sector Index |
| Type | INTEGER |
| Origin | Freescale |

# 4.7  Form FeeBlockConfiguration

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



**Figure 4-8. Tresos Plugin snapshot for FeeBlockConfiguration form.**

**User Manual, Rev. 1.0**

# 4.7.1  FeeBlockNumber (FeeBlockConfiguration)

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.

**Table 4-33.   Attribute FeeBlockNumber (FeeBlockConfiguration) detailed description**

| Property | Value |
|---|---|
| Label | FEE Block Number |
| Type | INTEGER |
| Origin | AUTOSAR_ECUC |
| Symbolic Name | true |
| Invalid | Range<br>    <=65534<br>    >=1 |

# 4.7.2  FeeBlockSize (FeeBlockConfiguration)

Size of a logical block in bytes.

**Table 4-34.   Attribute FeeBlockSize (FeeBlockConfiguration) detailed description**

| Property | Value |
|---|---|
| Label | FEE Block Size |
| Type | INTEGER |
| Origin | AUTOSAR_ECUC |
| Symbolic Name | false |
| Default | 1 |
| Invalid | Range<br>    <=65535<br>    >=1 |

**User Manual, Rev. 1.0**

NXP Semiconductors

### 4.7.3  FeeImmediateData (FeeBlockConfiguration)

Marker for high priority data.

**true**: Block contains immediate data.

**false**: Block does not contain immediate data.

**Table 4-35.   Attribute FeeImmediateData (FeeBlockConfiguration) detailed description**

| Property | Value |
|---|---|
| Label | FEE Immediate Data |
| Type | BOOLEAN |
| Origin | AUTOSAR_ECUC |
| Symbolic Name | false |
| Default | false |

### 4.7.4  FeeNumberOfWriteCycles (FeeBlockConfiguration)

Number of write cycles required for this block.

**Table 4-36.   Attribute FeeNumberOfWriteCycles (FeeBlockConfiguration) detailed description**

| Property | Value |
|---|---|
| Label | FEE Number Of Write Cycles |
| Type | INTEGER |
| Origin | AUTOSAR_ECUC |
| Symbolic Name | false |
| Default | 0 |
| Invalid | Range<br>    <=4294967295<br>    >=0 |

Note: This parameter is unused in the given implementation.

### 4.7.5  FeeClusterGroupRef (FeeBlockConfiguration)

**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.

**Table 4-37.   Attribute FeeClusterGroupRef (FeeBlockConfiguration) detailed description**

| Property | Value |
|---|---|
| Label | FEE Cluster Group Ref |
| Type | REFERENCE |
| Origin | Custom |

## 4.7.6   FeeBlockAssignment (FeeGeneral)

**Vendor specific:** Defines in which project this block is used.
- **APPLICATION**: Fee block is used only by the APPLICATION project.
- **BOOTLOADER**: Fee block is used only by the BOOTLOADER project.
- **SHARED**: Fee block is used by the APPLICATION and BOOTLOADER projects.

**Table 4-38.   Attribute FeeBlockAssignment (FeeGeneral) detailed description**

| Property | Value |
|---|---|
| Label | Fee Block Assignment |
| Type | ENUMERATION |
| Origin | Custom |
| Symbolic Name | false |
| Default | APPLICATION |

## 4.7.7   FeeDeviceIndex (FeeBlockConfiguration)

Device index (handle).

Range: 0 .. 254 (`0xFF` reserved for broadcast call to GetStatus function).

**Table 4-39.   Attribute FeeDeviceIndex (FeeBlockConfiguration) detailed description**

| Property | Value |
|---|---|
| Label | FEE Device Index |
| Type | SYMBOLIC-NAME-REFERENCE |
| Origin | AUTOSAR_ECUC |

# Chapter 5
# Driver Information

## 5.1 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 5-1.   Data Organization details

| 128 bits | | | | Description |
|---|---|---|---|---|
| **4 bytes** | **4 bytes** | **4 bytes** | **4 bytes** | |
| **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 5-2.   ClusterHdr Type

| uint32 (4 bytes) | uint32 (4 bytes) | uint32 (4 bytes) | uint32 (4 bytes) |
|---|---|---|---|

*Table continues on the next page...*

**User Manual, Rev. 1.0**

**Table 5-2.   ClusterHdr Type (continued)**

| ClrID | StartAddress | ClusterSize | checkSum |
|-------|--------------|-------------|----------|
| 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 5-3.   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.

## 5.2  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 0x10000.
- The second cluster has start address 0x18000.
- Two blocks are written.
- The length of the first block is 4.
- The length of the second block is 64.

**Table 5-4.   Dump Memory example (Fee_VirtualPageSize = 8)**

| Offset (hex) | 4 bytes (hex) | | | | 4 bytes (hex) | | | | 4 bytes (hex) | | | | 4 bytes (hex) | | | | Description |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0000 | 00 | 00 | 00 | 01 | 00 | 00 | 00 | 00 | 00 | 01 | 00 | 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.

# 5.3   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 previusly written data is available.

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

Fee Block Always Available

**Figure 5-1. SWS_Fee_00153 and SWS_Fee_00154 respected**

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.

Fee Block Always Available*

**Figure 5-2. SWS_Fee_00153 and SWS_Fee_00154 violated**

## 5.4 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.

**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 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.

## 5.5 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):

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

**Table 5-5.  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_MainFunction 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.

## 5.6  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.

## 5.7  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.

In the course of time, a $2^{nd}$ 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.

### 5.7.1  Default (Standard) Immediate Block Handling

In the default (non-legacy) mode, 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 <u>one instance</u> 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 by a pre-written header in the legacy mode.

## 5.7.2   Legacy Mode Immediate Block Handling

The two update steps of immediate data blocks differ from the default mode:

1. **First phase**: The header is written by calling Fee_EraseImmediateBlock (and appropriate amount of Fee_MainFunction/Fls_MainFunction calls)
   - the BlkId, StartAddress, Length and CheckSum fields in the header area are written, and
   - sufficient space for data is reserved.

   **Note**: This step is mandatory.

2. **Second phase**: Write the actual data by calling the Fee_Write function:

**User Manual, Rev. 1.0**

- the data portion is written, and
- the status of block is updated from MEMIF_BLOCK_INCONSISTENT to FEE_BLOCK_VALID by writing the valid flag.

**Important note**: If a power drop occurs during immediate block write operation, the next FEE initialization phase may consider the space reserved for this immediate block as correct even if there are already some data written (there is no blank check). It is responsibility of the application to use the Fee_EraseImmediateBlock function before each immediate block write operation.

# 5.8  Det Errors Description

**Table 5-6.   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 happenes 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 |

## Table 5-6.  Det Errors detailed description

| Name | Value [hex] | Description |
|---|---|---|
|  |  | FEE_SWAP_FOREIGN_BLOCKS_ENABLED=ON.To fix this DET error the configurator must configure FEE_MAX_NR_OF_BLOCKS to a higher value. |

**User Manual, Rev. 1.0**

# Chapter 6
# How to Configure

This chapter describes the configure Tresos plug-in for the FEE Driver.

## 6.1   FEE Settings Config
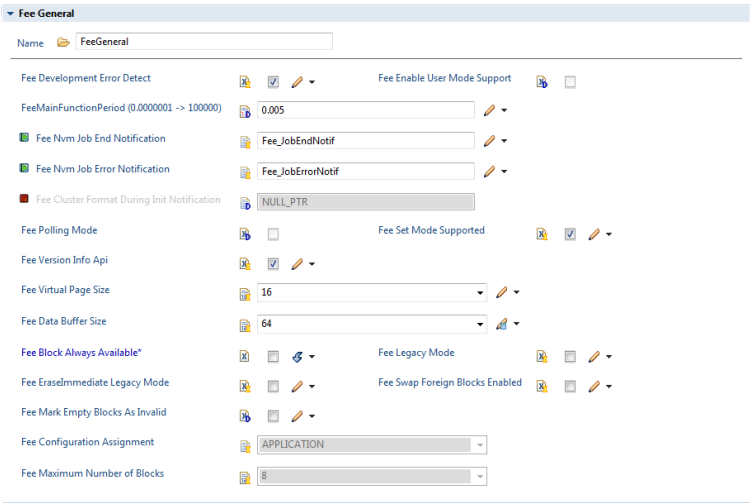


**Figure 6-1. Config FEE Settings General.**
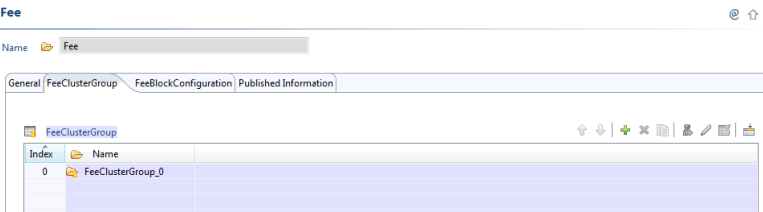


**Figure 6-2. Fee Cluster Group**

**User Manual, Rev. 1.0**
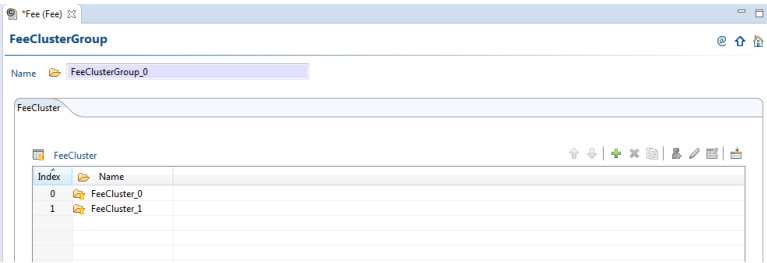
**Figure 6-3. Fee Cluster**
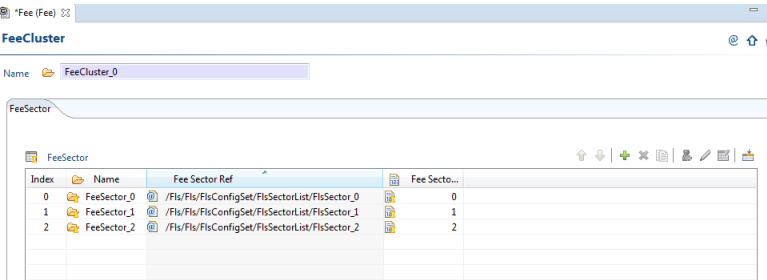


**Figure 6-4. Fee Sector**
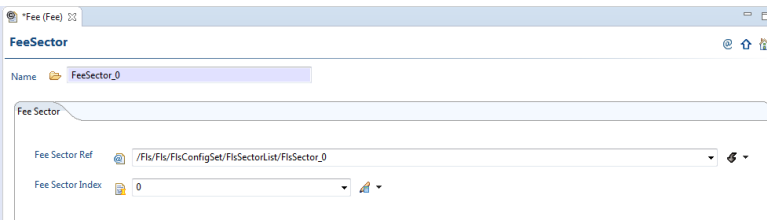


**Figure 6-5. FEE Sector Config**
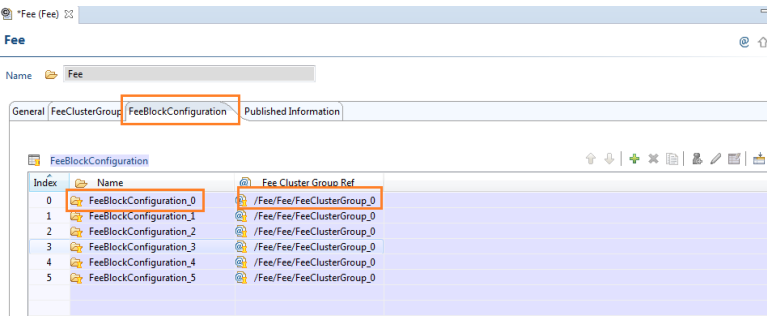


**Figure 6-6. FEE Block Config.**

**User Manual, Rev. 1.0**
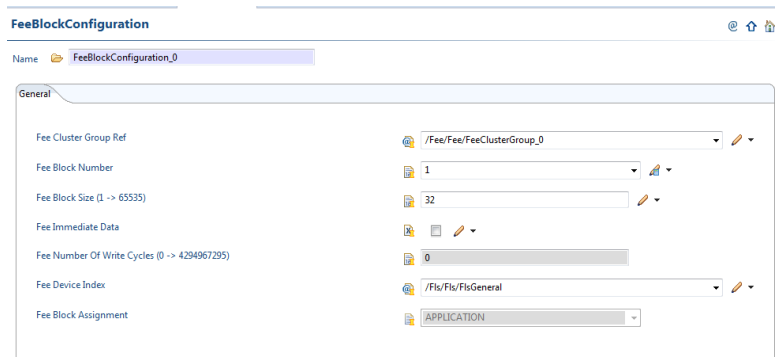
**Figure 6-7. FEE Block General Config.**

Document Number UM2FEEASR4.2 Rev0002R1.0.1
Revision 1.0