User Manual

for MPC5634M FEE Driver

Document Number: UM14FEEASR3.0R2.0.0
Rev. 2.6

freescale™

Contents

Section Number **Title** **Page**

Chapter 1
Revision History

Chapter 2
Introduction

Chapter 3
Driver

Chapter 4
Driver Information

Chapter 5
Tresos Configuration Plug-in

Chapter 1
Revision History

Table 1-1. Revision History

Revision	Date	Author	Description
2.5	03-Feb-2011	Gaetano Stabile	Update for Monaco automatic documentation
2.6	06-Dec-2011	Gaetano Stabile	Added driver information chapter

Chapter 2
Introduction

This User Manual describes Freescale Semiconductor AUTOSAR Fee module (FEE) for MPC5634M.

AUTOSAR FEE driver configuration parameters and deviations from the specification are described in FEE Driver chapter of this document. AUTOSAR FEE driver requirements and APIs are described in the AUTOSAR FEE driver software specification document.

2.1 Supported Derivatives

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

Table 2-1. MPC5634M Derivatives

Freescale Semiconductor	mpc5634m_bga208, mpc5634m_qfp144, mpc5634m_qfp176

All of the above microcontroller devices are collectively named as MPC5634M .

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
AUTOSAR	Automotive Open System Architecture
DEM	Diagnostic Event Manager
DET	Development Error Tracer
C/CPP	C and C++ Source Code
VLE	Variable Length Encoding
N/A	Not Applicable
MCU	Micro Controller Unit
ECU	Electronic Control Unit
EEPROM	Electrically Erasable Programmable Read-Only Memory
FEE	Flash EEPROM Emulation

Table continues on the next page...

Table 2-2. Acronyms and Definitions (continued)

Term	Definition
FLS	Flash
XML	Extensible Markup Language

2.5 Reference List

Table 2-3. Reference List

#	Title	Version
1	AUTOSAR AUTOSAR_SWS_Flash_EEPROM_Emulation.pdf FEE Driver Software Specification Document.	3.0 Rev0002
2	MPC5634M Reference Manual	Rev. 6, 4 October 2011

Chapter 3
Driver

3.1 Requirements

Requirements for this driver are detailed in the AUTOSAR 3.0FEE Driver Software Specification document (See Table Reference List).

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 the 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 an emulated EEPROM blocks apart from the other basic features specified by software specification.

Each Fee block is assigned during the Fee module configuration time to a 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 Fee module for emulation depends on actual Fls driver logical sector list configuration.

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

3.3 Deviation from Requirements

The driver deviates from the AUTOSAR FEE Driver software specification in some places.

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

Table Table 3-1 provides Status column description.

Table 3-1. Deviations Status Column Description

Term	Definition
N/A	Not available
N/T	Not testable
N/S	Out of scope
N/R	Unclear Requirement
N/I	Not implemented
N/F	Not fully implemented
I/D	Implemented with Deviations

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

Table 3-2. Driver Deviations Table

Requirement	Status	Description	Notes
FEE007	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.
FEE100	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.

Table continues on the next page...

Table 3-2. Driver Deviations Table (continued)

Requirement	Status	Description	Notes
FEE102	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.
FEE103	N/I	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.	Not used due to usage of the two or more clusters emulation algorithm.
FEE081	N/R	The function Fee_Cancel shall reset the FEE module's internal variables to make the module ready for a new job request.	It's in opposite to FLS230, FLS147.
FEE034	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).
FEE035	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.
FEE064	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)	implementation also fills VersionInfoPtr->instanceID
FEE082	N/F	If source code for caller and callee of the function Fee_GetVersionInfo is available, the FEE module should realize this function as a macro. The FEE module should define this macro in the module's header file.	The function will be implemented as function, not as a macro.
FEE075	N/F	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.
FEE023	N/F	The function Fee_MainFunction shall check the consistency of the logical block being read before notifying the caller. If an inconsistency of the read data is detected, the function Fee_MainFunction shall set the job result to MEMIF_BLOCK_INCONSISTENT and call the error notification routine of the upper layer.	MEMIF_BLOCK_INCONSISTENT is return value also for Fee blocks that has mismatch block-size or immediate-flag.
FEE109	N/R	FeeIndex	Not used by FEE module.

Table continues on the next page...

Table 3-2. Driver Deviations Table (continued)

Requirement	Status	Description	Notes
FEE114	N/R	FeePollingMode	Not supported by FEE module, unclear polling concept.
FEE110	N/I	FeeNumberOfWriteCycles	Not supported by FEE module due to emulation scheme.
FEE106	N/R	FeeDeviceIndex	Not used by FEE module.

3.4 Software specification

The following sections contains driver software specifications.

3.4.1 Define Reference

Constants supported by the driver are as per AUTOSAR FEE Driver software specification Version 3.0 .

3.4.2 Enum Reference

Enumeration of all constants supported by the driver are as per AUTOSAR FEE Driver software specification Version 3.0 .

3.4.2.1 Enumeration Fee_BlockStatusType

Status of Fee block header.

Table 3-3. 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 bogus data).
FEE_BLOCK_HEADER_INVALID		Fee block header is garbled.

Table continues on the next page...

Table 3-3. Enumeration Fee_BlockStatusType Values (continued)

Name	Initializer	Description
FEE_BLOCK_INVALIDATED		Fee block header is invalidated by Fee_InvalidateBlock(BlockNumber). Not used when FEE_BLOCK_ALWAYS_AVAILABLE == STD_OFF .
FEE_BLOCK_HEADER_BLANK		Fee block header is blank (end of Fee block header list).

3.4.2.2 Enumeration Fee_ClusterStatusType

Status of Fee cluster header.

Table 3-4. 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.

3.4.2.3 Enumeration Fee_JobType

Type of job currently executed by Fee_MainFunction.

Table 3-5. 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 flashn.

Table continues on the next page...

Table 3-5. Enumeration Fee_JobType Values (continued)

Name	Initializer	Description
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.
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.

3.4.3 Function Reference

Functions of all functions supported by the driver are as per AUTOSAR FEE Driver software specification Version 3.0 .

3.4.3.1 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

Prototype: `void Fee_Cancel(void);`

3.4.3.2 Function Fee_EraseImmediateBlock

Service to erase a logical block.

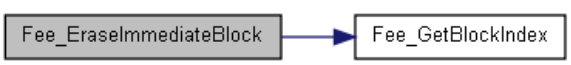

Figure 3-1. Function Fee_EraseImmediateBlock References.

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 FeeImmediate- Data == TRUE). If not, the function Fee_EraseImmediateBlock shall report the error code FEE_E_INVALID_BLOCK_NO.

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

Return: Std_ReturnType .

Prototype: `Std_ReturnType Fee_EraseImmediateBlock(uint16 BlockNumber);`

Table 3-6. Fee_EraseImmediateBlock Arguments

Type	Name	Direction	Description
uint16	BlockNumber	**input**	Number of logical block, also denoting start address of that block in emulated EEPROM.

Table 3-7. 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.

3.4.3.3 Function Fee_GetJobResult

Return the result of the last job.

Details:

Return the result of the last job synchronously.

Note

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

Return: MemIf_JobResultType .

Prototype: `MemIf_JobResultType Fee_GetJobResult(void);`

Table 3-8. Fee_GetJobResult Return Values

Name	Description
MEMIF_JOB_OK	The job has been finished successfully .

Table continues on the next page...

Table 3-8. Fee_GetJobResult Return Values (continued)

Name	Description
MEMIF_JOB_FAILED	The job has not been finished successfully .
MEMIF_JOB_PENDING	The job has not yet been finished .
MEMIF_JOB_CANCELLED	The job has been cancelled .
MEMIF_BLOCK_INCONSISTENT	The requested block is inconsistent, it may contain corrupted data.
MEMIF_BLOCK_INVALID	.

3.4.3.4 Function Fee_GetStatus

Return the Fee module state.

Details:

Return the Fee module state synchronously.

Note

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

Return: Fee_ModuleStatus .

Prototype: `MemIf_StatusType Fee_GetStatus(void);`

Table 3-9. 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 .

3.4.3.5 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 .

Note

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

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

Table 3-10. Fee_GetVersionInfo Arguments

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

3.4.3.6 Function Fee_Init

Service to initialize the FEE module.

Details:

The function Fee_Init shall initialize the Flash EEPROM Emulation module.

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

Note

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

Prototype: `FEE_NVM_JOB_END_NOTIFICATION_DECL FEE_NVM_JOB_ERROR_NOTIFICATION_DECL void Fee_Init(void);`

3.4.3.7 Function Fee_InvalidateBlock

Service to invalidate a logical block.

Figure 3-2. Function Fee_InvalidateBlock References.

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

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]: 0x07. Asynchronous Non Reentrant

Return: Std_ReturnType .

Prototype: `Std_ReturnType Fee_InvalidateBlock(uint16 BlockNumber);`

Table 3-11. Fee_InvalidateBlock Arguments

Type	Name	Direction	Description
uint16	BlockNumber	**input**	Number of logical block, also denoting vstart address of that block in emulated EEPROM.

Table 3-12. 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 .

3.4.3.8 Function Fee_JobEndNotification

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

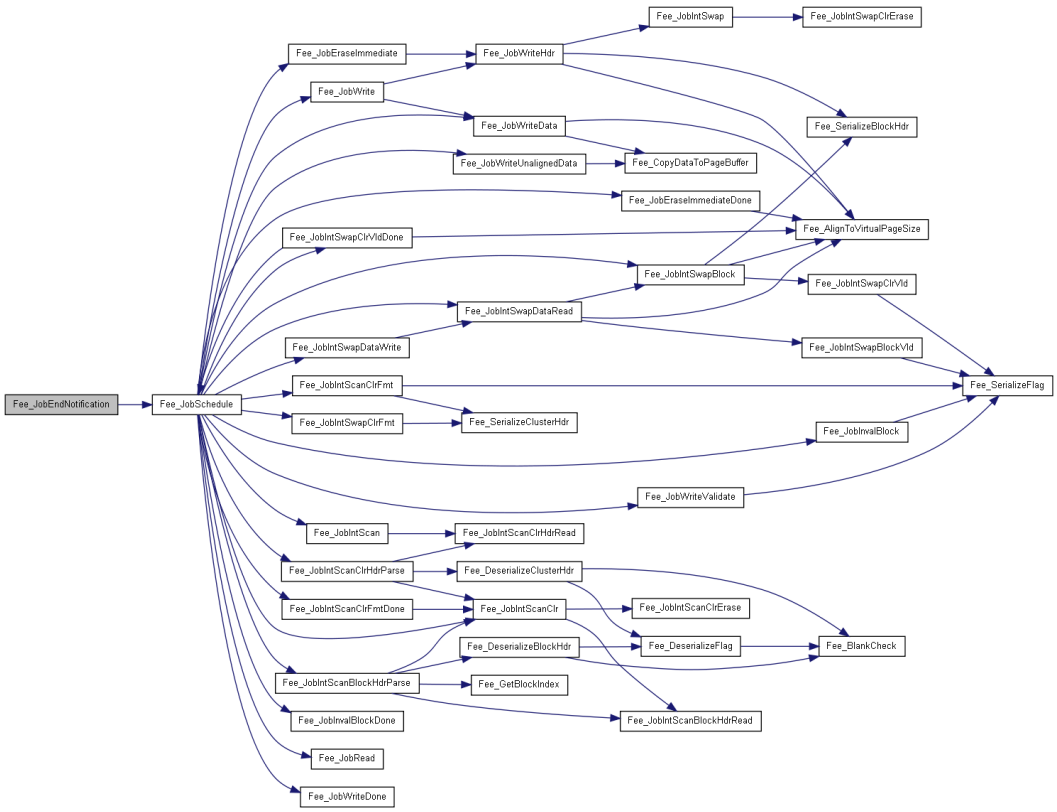

Figure 3-3. Function Fee_JobEndNotification References.

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);`

3.4.3.9 Function Fee_JobErrorNotification

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

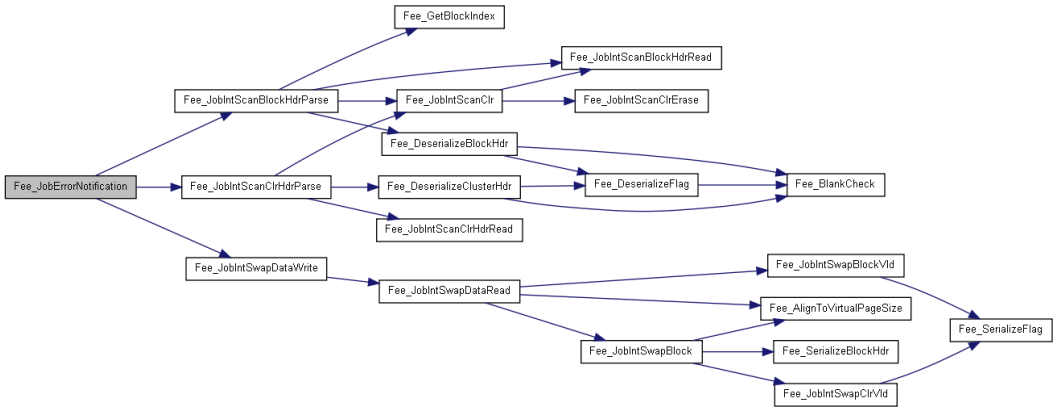

Figure 3-4. Function Fee_JobErrorNotification References.

Details:

The underlying flash driver shall call the function Fee_JobErrorNotification to report the failure 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]: 0x11. Synchronous Non Reentrant

Prototype: `void Fee_JobErrorNotification(void);`

3.4.3.10 Function Fee_MainFunction

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

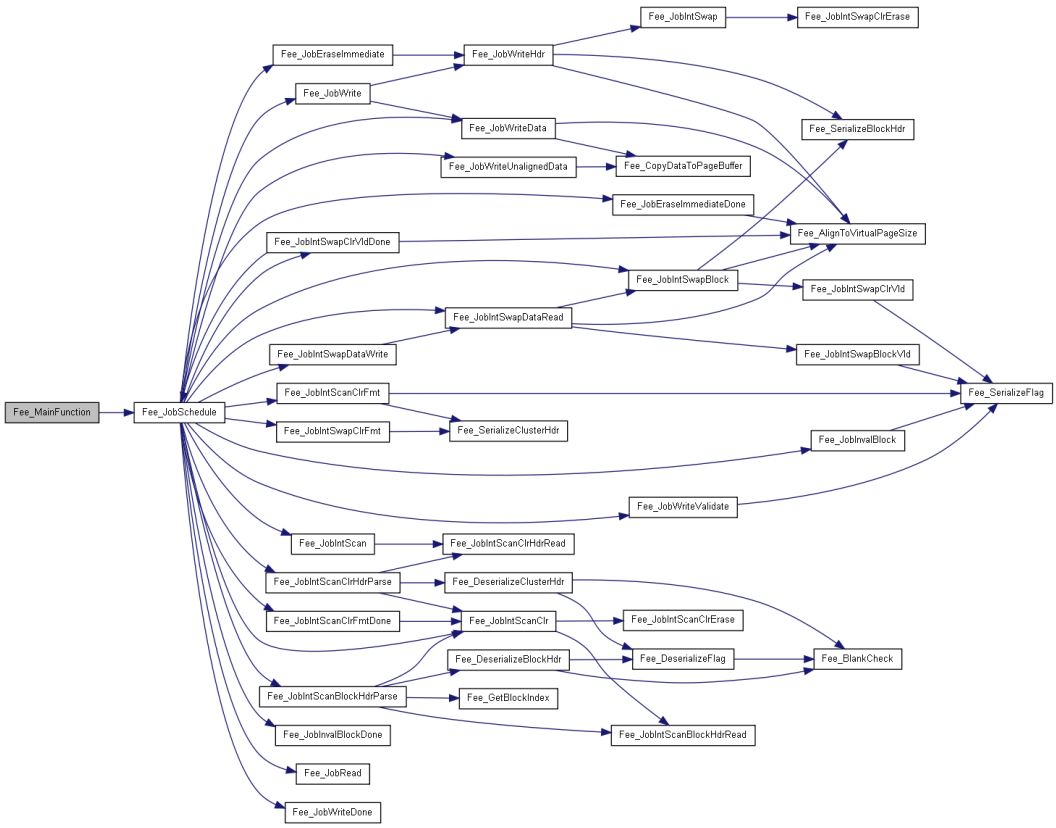

Figure 3-5. Function Fee_MainFunction References.

Details:

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, call the job error notification function 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 Specification of FLASH EEPROM Emulation 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);`

3.4.3.11 Function Fee_Read

Service to initiate a read job.

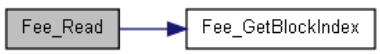

Figure 3-6. Function Fee_Read References.

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.

Note

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

Return: Std_ReturnType .

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

Table 3-13. 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 3-14. 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.

3.4.3.12 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. Synchronous Non Reentrant

Prototype: `void Fee_SetMode(MemIf_ModeType Mode);`

Table 3-15. Fee_SetMode Arguments

Type	Name	Direction	Description
MemIf_ModeType	Mode	**input**	Either MEMIF_MODE_FAST or MEMIF_MODE_SLOW .

3.4.3.13 Function Fee_Write

Service to initiate a write job.

Figure 3-7. Function Fee_Write References.

Details:

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.

Pre: 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" .

Post: 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

Return: Std_ReturnType .

Prototype: `Std_ReturnType Fee_Write(uint16 BlockNumber, uint8 *DataBufferPtr);`

Table 3-16. Fee_Write Arguments

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

Table 3-17. 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.

3.4.4 Structs Reference

Data structures supported by the driver are as per AUTOSAR FEE Driver software specification Version 3.0 .

3.4.4.1 Structure Fee_BlockConfigType

Fee block configuration structure.

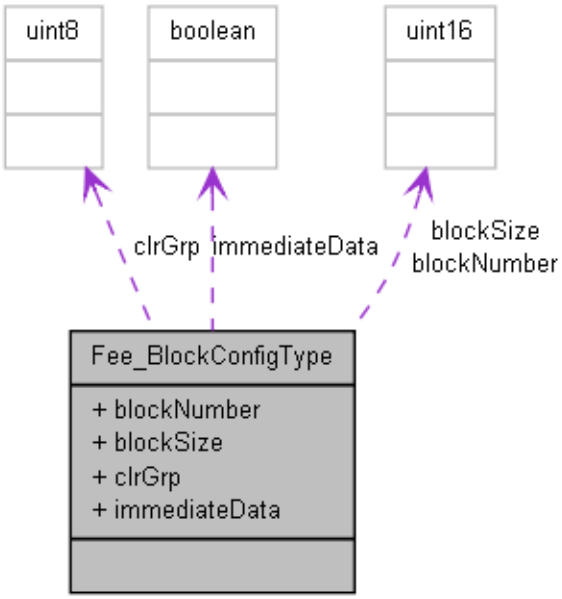

Figure 3-8. Struct Fee_BlockConfigType

Declaration:

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

Table 3-18. 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.

3.4.4.2 Structure Fee_BlockInfoType

Fee block run-time status.

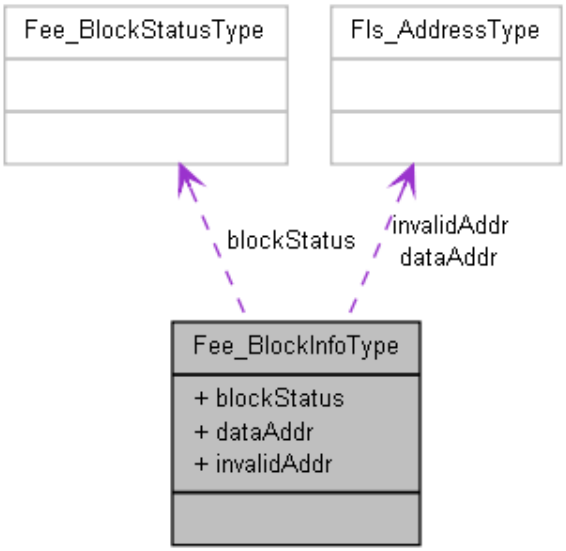

Figure 3-9. Struct Fee_BlockInfoType

Declaration:

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

Table 3-19. 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.

3.4.4.3 Structure Fee_ClusterGroupInfoType

Fee cluster group run-time status.

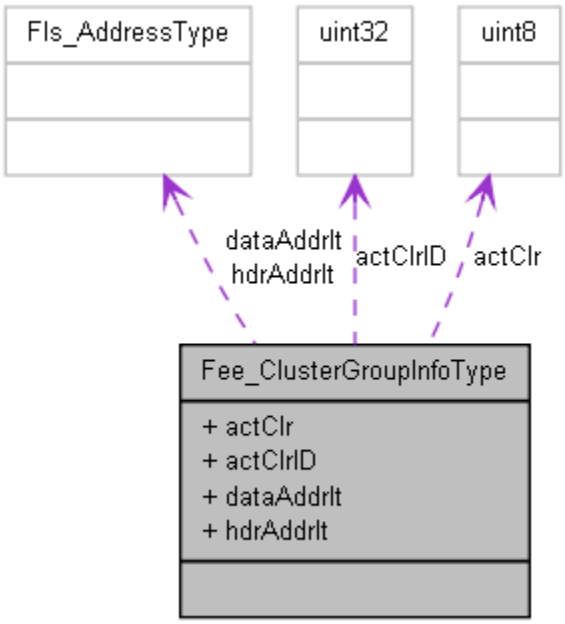

Figure 3-10. Struct Fee_ClusterGroupInfoType

Declaration:

```
typedef struct
                              {
                                uint8 actClr,
                                      uint32 actClrID,
                                      Fls_AddressType dataAddrIt,
                                      Fls_AddressType hdrAddrIt
                              } Fee_ClusterGroupInfoType;
```

Table 3-20. 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.

3.4.4.4 Structure Fee_ClusterGroupType

Fee cluster group configuration structure.

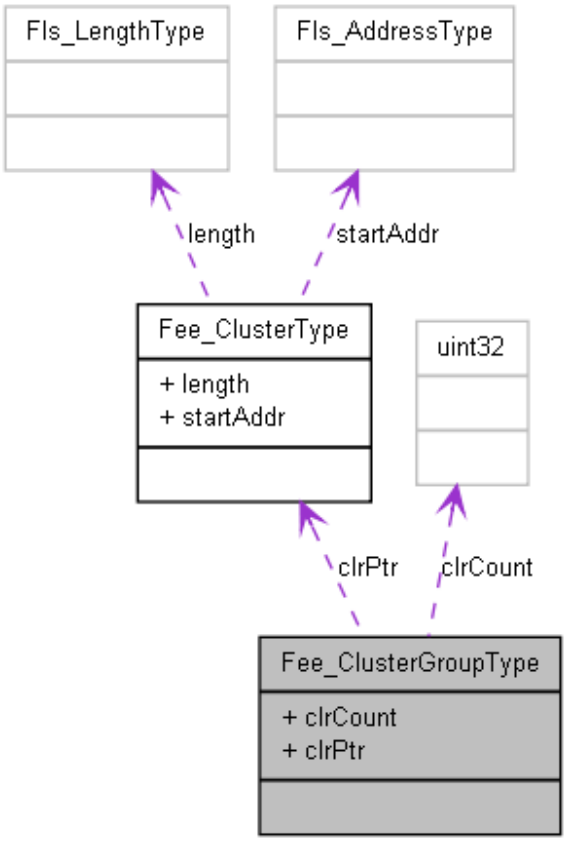

Figure 3-11. Struct Fee_ClusterGroupType

Declaration:

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

Table 3-21. Structure Fee_ClusterGroupType member description

Member	Description
clrCount	Number of clusters in cluster group.
clrPtr	Pointer to array of Fee cluster configurations.

3.4.4.5 Structure Fee_ClusterType

Fee cluster configuration structure.

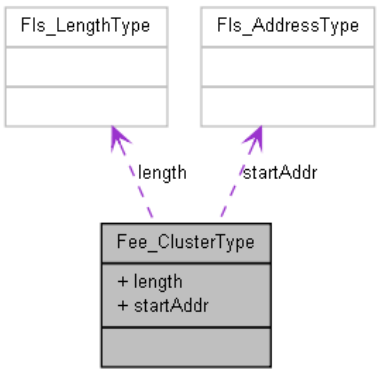

Figure 3-12. Struct Fee_ClusterType

Declaration:

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

Table 3-22. Structure Fee_ClusterType member description

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

3.4.5 Types Reference

Types supported by the driver are as per AUTOSAR FEE Driver software specification Version 3.0 .

3.4.6 Variables Reference

Variables supported by the driver are as per AUTOSAR FEE Driver software specification Version 3.0 .

3.5 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
Driver Information

4.1 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 Fee_VirtualPageSize is set to 8. Header valid flag and invalid flag are aligned each to Fee_VirtualPAgeSize

Each cluster/block has:

- an header
- data

Table 4-1. Data Organization details

128 bits					
4 byte		**4 byte**	**4 byte**	**4 byte**	**Description**
ClrID		**Start address**	**Cluster size**	**Checksum**	*Cluster header*
Valid flag		not used	not used	not used	*Cluster status*
Block id	**Length**	**Target address**	**Checksum**	not used	*Block 1 header*
Valid flag		not used	**Invalid flag**	not used	*Block 1 status*
Block id	Length	Target address	Checksum	not used	*Block 2 header*
Valid flag		not used	Invalid flag	not used	Block 2 status
...					
...					
Block id	Length	Target address	Checksum	not used	*Block n-1 header*
Valid flag		not used	Invalid flag	not used	*Block n-1 status*
Block id	Length	Target address	Checksum	not used	*Block n header*
Valid flag		not used	Invalid flag	not used	*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 4-2. ClusterHdr Type

uint32(4byte)	uint32(4byte)	uint32(4byte)	uint32(4byte)
ClrID	StartAddress	ClusterSize	checkSum
valFlag	blank1	invalFlag	blank2

- **ClrID**: (uint32) Progressive internal integer number, identify a cluster
- **StartAddress**: (uint32) Start phisical address of cluster (Cfg)
- **ClusterSize**: (uint32) Length of cluster(Cfg)
- **checkSum**: (uint32) sum of ClrID, StartAddress and ClusterSize
- **val Flag**: (uint32) 0x81 for a valid cluster
- **invalFlag**: (uint32) not-used

Table 4-3. BlockHdr Type

uint32(4byte)	uint32(4byte)	uint32(4byte)	uint32(4byte)
BlockNumber:Length	TargetAddress	checkSum	blank1
valFlag	blank2	invalFlag	blank3

- **BlockNumber**: (uint16) Integer number (Cfg)
- **Length**:(uint16) Length of block (Cfg)
- **TargetAddress**:(uint32)Address of start Data
- **checkSum**: (uint32) sum of BlockNumber, Length and TargetAddress
- **valFlag**:(uint32) 0x81 for a valid block
- **invalFlag**:(uint32) 0x18 for invalidate block (Cfg)

4.2 Dump Memory example

The Figure below show an example of Cluster dump:

- One group of two clusters are configured
- The first cluster has start physical address 0x800000
- The second cluster has start physical address 0x80C000
- Two blocks are written
- First block has 4 bytes of data size
- Second block has 64 bytes of data size

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

Offset (hex)	4byte				4byte				4byte				4byte				Desc
00	00	00	00	01	00	00	00	00	00	01	00	00	00	01	00	01	Clr Hdr
10	81	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	Clr Sts
20	00	01	00	04	00	00	FF	F8	00	01	FF	FD	FF	FF	FF	FF	Blk1 Hdr
30	81	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	Blk1 Sts
40	00	02	00	40	00	00	FF	B8	00	01	FF	FA	FF	FF	FF	FF	Blk2 Hdr
50	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

4.3 Fee Block Always Available

To be consistent with Autosar Requirement **FEE050** (*when a block write operation is started, the FEE module shall mark the corresponding block as inconsistent. 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 header (including the checksum) and the writing of valid flag to the header, neither newly nor previously written data is available.

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

Figure 4-1. FEE050 respected

This behaviour can be modified (losing compliance with Autosar requirement FEE050) and if a previous valid instance of the block exists, it is always possible to recover it.

Figure 4-2. FEE050 not respected(Fee Block Always Available)

4.4 Managing clusters and blocks consistency

The FEE module shall manage the consistency of block when catastrophic events occurs:

On reset occurrency, the flash peripheral aborts any high voltage operation.

i.e. a power down occurs or reset when an high voltage (erase/write) operation is ongoing.

When a catastrofic event occurs, Fee driver shall be able to recover the last validated instance of the blocks stored in the Flash, ignoring the eventualy last update block interrupted by the reset.

During Fee initialization phase, a reading operation on the bytes interested by the abort, could generates an ECC error.

At start-up Fee driver will scan the memory in order to restore the cluster and the blocks status. If some data is corrupted an Ivor exception will be thrown during the read operation and Fls driver will manage it.

The driver will behave differently depending on which operation was interrupted:

- a cluster swap is ongoing
- a block updating is ongoing
- a immediate block updating is ongoing

4.5 Cluster Swap

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 FEE block header is invalid.
- when a Flash job has failed during FEE initialization stage.
- when a FEE block number is invalid.
- when a FEE block header doesn't match FEE configuration.
- when the last header is corrupted.
- when trying to read/write on damaged locations.

The swap consists in the following steps:

1. Erase the next cluster (ERASING stage);
2. Write first part of cluster header (16 byte: ClrID, StartAddress, ClusterSize, Checksum) (FORMATTING stage);
3. Copy all block except a block that generated a cluster swap (header, data and status) (COPYING(n-1) stage);If FEE_BlocksAlwaysAvailable == STD_ON all block are copied.
4. Write second part of cluster header (16 byte: valid/invalid flag) (ACTIVE stage);
5. Copy block that generated a cluster swap (header, data and status) (COPYNG(n) stage);If FEE_BlocksAlwaysAvailable == STD_ON update the block that has generated the cluster swap.

Table 4-5. 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(n-1)	ACTIVE	COPYING(n)

A system reset happen during STAGE 1

Since an erase operation may have been interrupted, the next cluster could be affected by ECC error.

A system reset happen 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 happen during STAGE 3

Since a program operation may have been interrupted, the next cluster area could be affected by ECC error.

A system reset happen 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.

In all the preceding STAGEs at startup the application should start a Fee initialization phase calling Fee_Init() and MainFunctions.

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, the application should write it back generating a new cluster swap that will erase the next cluster and remove the ECC error where ever it is.

4.6 Block updating

During normal execution (without catastrophic event) the consistency of data block is assured by the order in which the block fields are written in memory.

A block updating consists in the following steps:

- writing the BlkId, StartAddress Length and CheckSum 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 power up againt the system 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 the 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 higher address is kept as valid.

If the ECC error occurs , Fee_Init considers invalid the block affected by ECC and keep as valid the previous instance of block if there are no valid block the block is considered invalid.

4.7 Immediate block updating

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

1. **first phase**: Write the Header calling Fee_EraseImmediateBlock and Fee/Fls Mainfunction's
   - write the BlkId, StartAddress Length and CheckSum in the header area
   - space for Data will be reserved

   Note: This step is mandatory
2. **second phase** Write Data calling Fee_Write and Fee/Fls Mainfunction's:
   - directly write data
   - update the status of block from MEMIF_BLOCK_INCONSISTENT to FEE_BLOCK_VALID

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.

Chapter 5
Tresos Configuration Plug-in

This chapter describes the Tresos configuration plug-in for the FEE Driver. The most of the parameters are described below.

5.1 Configuration elements of Fee

Included forms :

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

Table 5-1. Revision table

Revision	Date
revision1.0.0	2009-06-20T13:00:00

5.2 Form IMPLEMENTATION_CONFIG_VARIANT

VariantPreCompile: Precompile configuration parameters.

Figure 5-1. Tresos Plugin snapshot for IMPLEMENTATION_CONFIG_VARIANT form.

Table 5-2. Attribute IMPLEMENTATION_CONFIG_VARIANT detailed description

Property	Value
Label	Config Variant Config Variant
Default	VariantPreCompile

5.3 Form FeeGeneral

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

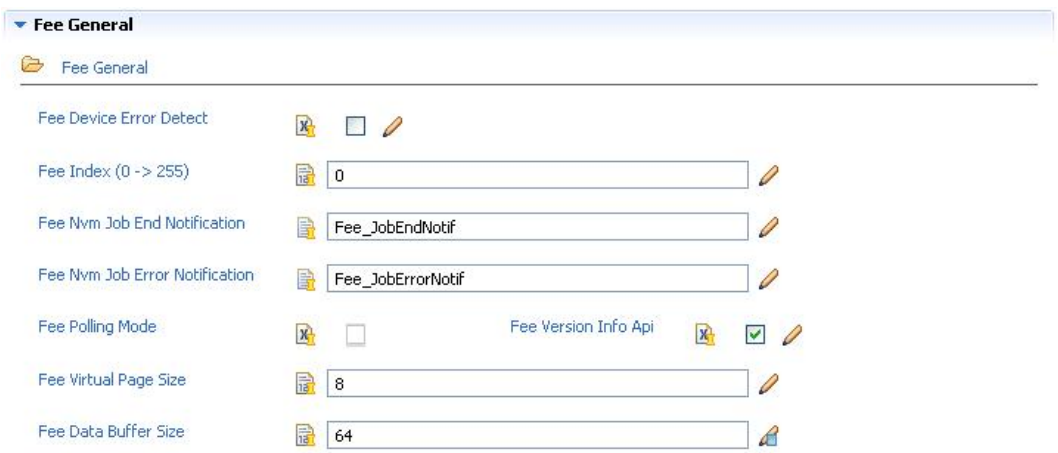

Figure 5-2. Tresos Plugin snapshot for FeeGeneral form.

5.3.1 FeeDevErrorDetect (FeeGeneral)

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

true: Development error detection enabled.

false: Development error detection disabled.

Table 5-3. Attribute FeeDevErrorDetect (FeeGeneral) detailed description

Property	Value
Label	Fee Development Error Detect
Type	BOOLEAN
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	true

5.3.2 FeeIndex (FeeGeneral)

Specifies the InstanceId of this module instance. If only one instance is present it shall have the Id 0.

Table 5-4. Attribute FeeIndex (FeeGeneral) detailed description

Property	Value
Label	Fee Index
Type	INTEGER
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	0
Invalid	Range <=255 >=0

5.3.3 FeeNvmJobEndNotification (FeeGeneral)

Mapped to the job end notification routine provided by the upper layer module (NvM_JobEndNotification).

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

Property	Value
Label	Fee Nvm Job End Notification
Type	FUNCTION-NAME
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	NULL_PTR

5.3.4 FeeNvmJobErrorNotification (FeeGeneral)

Mapped to the job error notification routine provided by the upper layer module (NvM_JobErrorNotification).

Table 5-6. Attribute FeeNvmJobErrorNotification (FeeGeneral) detailed description

Property	Value
Label	Fee Nvm Job Error Notification
Type	FUNCTION-NAME
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	NULL_PTR

5.3.5 FeePollingMode (FeeGeneral)

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

Note

Not supported by the driver.

Table 5-7. Attribute FeePollingMode (FeeGeneral) detailed description

Property	Value
Label	Fee Polling Mode
Type	BOOLEAN
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	false

5.3.6 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 5-8. Attribute FeeVersionInfoApi (FeeGeneral) detailed description

Property	Value
Label	Fee Version Info Api
Type	BOOLEAN

Table continues on the next page...

Table 5-8. Attribute FeeVersionInfoApi (FeeGeneral) detailed description (continued)

Property	Value
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	true

5.3.7 FeeVirtualPageSize (FeeGeneral)

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

Table 5-9. Attribute FeeVirtualPageSize (FeeGeneral) detailed description

Property	Value
Label	Fee Virtual Page Size
Type	INTEGER
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	8
Invalid	Range >=8 <=65535

5.3.8 FeeDataBufferSize (FeeGeneral)

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

Note

Vendor specific parameter

Table 5-10. Attribute FeeDataBufferSize (FeeGeneral) detailed description

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

5.3.9 FeeBlockAlwaysAvailable (FeeGeneral)

If reset, power loss etc. occurs here, neither newly nor previously written data is available.

Table 5-11. Attribute FeeBlockAlwaysAvailable (FeeGeneral) detailed description

Property	Value
Label	Fee Block Always Available
Type	BOOLEAN
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	false

5.4 Form CommonPublishedInformation

CommonPublishedInformation

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

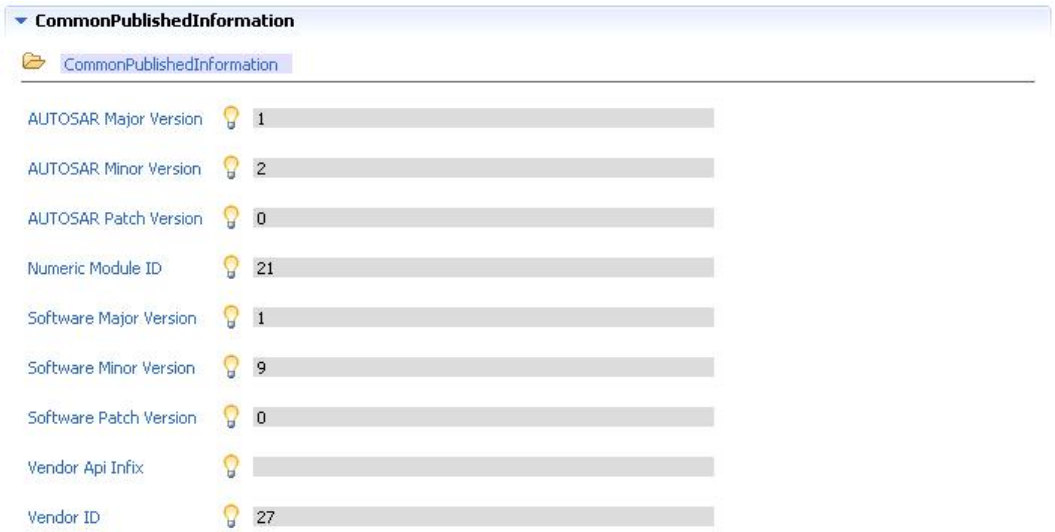

Figure 5-3. Tresos Plugin snapshot for CommonPublishedInformation form.

5.4.1 ArMajorVersion (CommonPublishedInformation)

AUTOSAR Major Version

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

Table 5-12. Attribute ArMajorVersion (CommonPublishedInformation) detailed description

Property	Value
Label	AUTOSAR Major Version
Origin	Custom
Symbolic Name	false
Default	1
Invalid	Range >=1 <=1

5.4.2 ArMinorVersion (CommonPublishedInformation)

AUTOSAR Minor Version

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

Table 5-13. Attribute ArMinorVersion (CommonPublishedInformation) detailed description

Property	Value
Label	AUTOSAR Minor Version
Origin	Custom
Symbolic Name	false
Default	2
Invalid	Range >=2 <=2

5.4.3 ArPatchVersion (CommonPublishedInformation)

AUTOSAR Patch Version

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

Table 5-14. Attribute ArPatchVersion (CommonPublishedInformation) detailed description

Property	Value
Label	AUTOSAR Patch Version
Origin	Custom
Symbolic Name	false
Default	0
Invalid	Range >=0 <=0

5.4.4 ModuleId (CommonPublishedInformation)

Module ID

Module ID of this module.

Table 5-15. Attribute ModuleId (CommonPublishedInformation) detailed description

Property	Value
Label	Numeric Module ID

Table continues on the next page...

Table 5-15. Attribute ModuleId (CommonPublishedInformation) detailed description (continued)

Property	Value
Origin	Custom
Symbolic Name	false
Default	21
Invalid	Range >=21 <=21

5.4.5 SwMajorVersion (CommonPublishedInformation)

Software Major Version

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

Table 5-16. Attribute SwMajorVersion (CommonPublishedInformation) detailed description

Property	Value
Label	Software Major Version
Origin	Custom
Symbolic Name	false
Default	0
Invalid	Range >=0 <=0

5.4.6 SwMinorVersion (CommonPublishedInformation)

Software Minor Version

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

Table 5-17. Attribute SwMinorVersion (CommonPublishedInformation) detailed description

Property	Value
Label	Software Minor Version

Table continues on the next page...

Table 5-17. Attribute SwMinorVersion (CommonPublishedInformation) detailed description (continued)

Property	Value
Origin	Custom
Symbolic Name	false
Default	9
Invalid	Range >=9 <=9

5.4.7 SwPatchVersion (CommonPublishedInformation)

Software Patch Version

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

Table 5-18. Attribute SwPatchVersion (CommonPublishedInformation) detailed description

Property	Value
Label	Software Patch Version
Origin	Custom
Symbolic Name	false
Default	1
Invalid	Range >=1 <=1

5.4.8 VendorApiInfix (CommonPublishedInformation)

Vendor Api Infix

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 5-19. Attribute VendorApiInfix (CommonPublishedInformation) detailed description

Property	Value
Label	Vendor Api Infix
Origin	Custom
Symbolic Name	false
Default	
Enable	false

5.4.9 VendorId (CommonPublishedInformation)

Vendor ID

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

Table 5-20. Attribute VendorId (CommonPublishedInformation) detailed description

Property	Value
Label	Vendor ID
Origin	Custom
Symbolic Name	false
Default	43
Invalid	Range >=43 <=43

5.5 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.

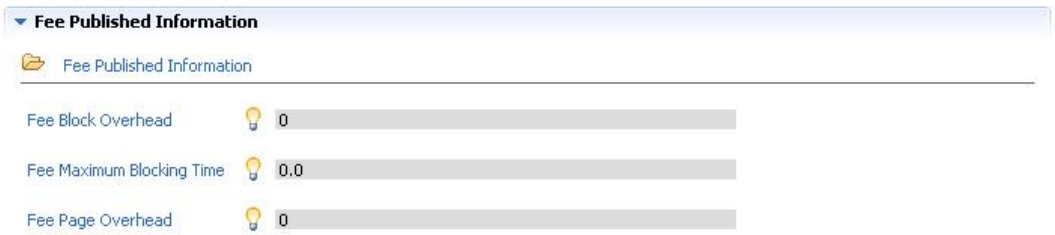

Figure 5-4. Tresos Plugin snapshot for FeePublishedInformation form.

5.5.1 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 5-21. Attribute FeeBlockOverhead (FeePublishedInformation) detailed description

Property	Value
Label	Fee Block Overhead
Type	INTEGER
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	0
Invalid	Range <=0 >=0

5.5.2 FeeMaximumBlockingTime (FeePublishedInformation)

The maximum time the FEE module's API routines shall be blocked (delayed) by internal operations.

Note

The maximum blocking time depends on various conditions such as MCU clock configuration, flash device configuration, configuration of underlying flash driver, size of flash sectors used for EEPROM emulation, etc. The maximum blocking time is not constant and can not be calculated using simple formula.

Table 5-22. Attribute FeeMaximumBlockingTime (FeePublishedInformation) detailed description

Property	Value
Label	Fee Maximum Blocking Time
Type	FLOAT
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	0
Invalid	Range <=0 >=0

5.5.3 FeePageOverhead (FeePublishedInformation)

Management overhead per page in bytes

Note

The page management overhead is 0 bytes

Table 5-23. Attribute FeePageOverhead (FeePublishedInformation) detailed description

Property	Value
Label	Fee Page Overhead
Type	INTEGER
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	0
Invalid	Range <=0 >=0

5.6 Form FeeClusterGroup

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

Note

Vendor specific parameter.

Included forms :

- Form FeeCluster

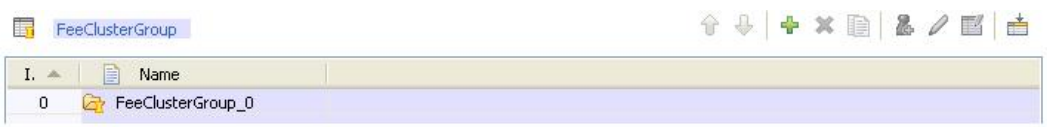

Figure 5-5. Tresos Plugin snapshot for FeeClusterGroup form.

5.6.1 Form FeeCluster

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

Note

Vendor specific parameter

Is included by form : Form FeeClusterGroup

Included forms :

- Form FeeSector

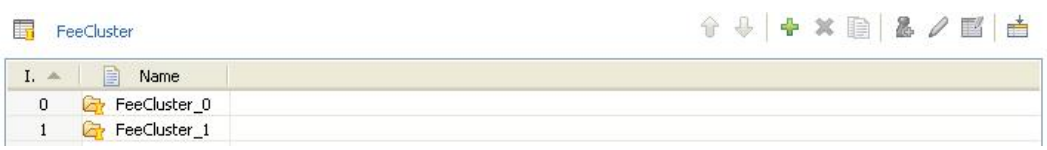

Figure 5-6. Tresos Plugin snapshot for FeeCluster form.

5.6.1.1 Form FeeSector

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

Note

Vendor specific parameter

Is included by form : Form FeeCluster

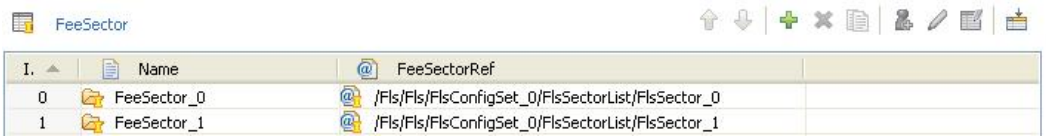

Figure 5-7. Tresos Plugin snapshot for FeeSector form.

5.6.1.1.1 FeeSectorRef (FeeSector)

Reference to a logical Fls sector the Fee cluster consist of.

Note

Vendor specific parameter

Table 5-24. Attribute FeeSectorRef (FeeSector) detailed description

Property	Value
Label	Fee Sector Ref
Type	REFERENCE
Origin	Custom

5.7 Form FeeBlockConfiguration

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

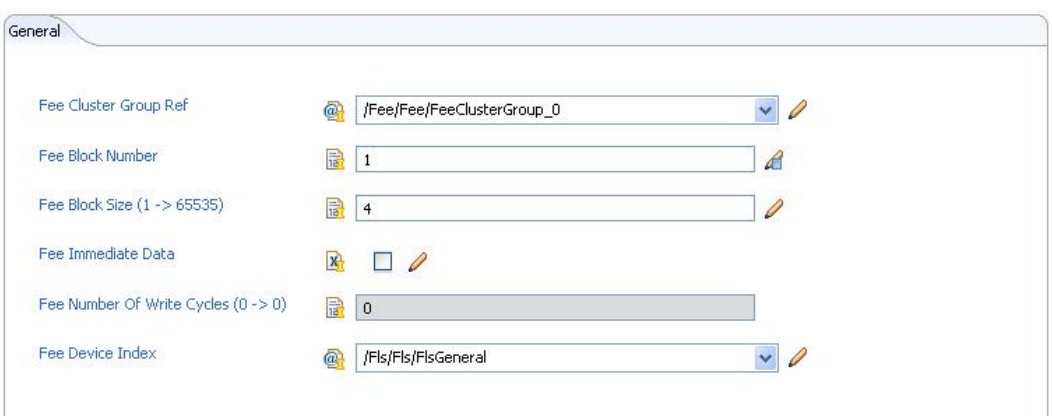

Figure 5-8. Tresos Plugin snapshot for FeeBlockConfiguration form.

5.7.1 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 5-25. Attribute FeeBlockNumber (FeeBlockConfiguration) detailed description

Property	Value
Label	Fee Block Number
Type	INTEGER
Origin	AUTOSAR_ECUC
Symbolic Name	true
Invalid	Range >=1 <=65535

5.7.2 FeeBlockSize (FeeBlockConfiguration)

Size of a logical block in bytes."/>

Table 5-26. Attribute FeeBlockSize (FeeBlockConfiguration) detailed description

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

5.7.3 FeeImmediateData (FeeBlockConfiguration)

Marker for high priority data.

true: Block contains immediate data.

false: Block does not contain immediate data.

Table 5-27. Attribute FeeImmediateData (FeeBlockConfiguration) detailed description

Property	Value
Label	Fee Immediate Data
Type	BOOLEAN
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	false

5.7.4 FeeNumberOfWriteCycles (FeeBlockConfiguration)

Number of write cycles required for this block

Note

Not supported by the driver.

Table 5-28. 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 <=0 >=0

5.7.5 FeeClusterGroupRef (FeeBlockConfiguration)

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.

Note

Vendor specific parameter

Table 5-29. Attribute FeeClusterGroupRef (FeeBlockConfiguration) detailed description

Property	Value
Label	Fee Cluster Group Ref
Type	REFERENCE
Origin	Custom

5.7.6 FeeDeviceIndex (FeeBlockConfiguration)

Device index (handle).

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

Table 5-30. Attribute FeeDeviceIndex (FeeBlockConfiguration) detailed description

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

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
1-800-441-2447 or +1-303-675-2140
Fax: +1-303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: UM14FEEASR3.0R2.0.0
Rev. 2.6