MICROSAR Fee

Technical Reference

Small Sector

Version 2.0.0

Authors	Michael Goß, Bernhard Karl
Status	Released

Document Information

History

Author	Date	Version	Remarks
virgmi	2016-06-22	1.00.00	Initial version
virgmi	2016-08-23 2016-09-21	1.01.00	Chapter 'Requirements and Recommendations' was added. Reference to ProductInformation of SmallSectorFee was added.
virbka	2018-04-04	2.00.00	Chapter 'Incompatibility between SmallSectorFee Version 1.xx.xx and 2.xx.xx' was added. Chapter 'Overhead Calculation' was adapted New note box in chapter 'Configuring Flash API Services' Chapter 'Block Configuration' was extended

Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	AUTOSAR_SWS_FlashEEPROMEmulation.pdf	V2.0.0
[2]	AUTOSAR	AUTOSAR_SWS_DevelopmentErrorTracer.pdf	V3.2.0
[3]	AUTOSAR	AUTOSAR_BasicSoftwareModules.pdf	V1.0.0
[4]	Vector	ProductInformation_8_MICROSARSmallSectorFee.pdf	V1.0.0
[5]	Vector	AUTOSAR_SWS_NVRAMManager.pdf	V3.2.0

Caution
We have configured the programs in accordance with your specifications in the questionnaire. Whereas the programs do support other configurations than the one specified in your questionnaire, Vector´s release of the programs delivered to your company is expressly restricted to the configuration you have specified in the questionnaire.

Contents

Illustrations

Tables

1 Component History

The component history gives an overview over the important milestones that are supported in the different versions of the component.

Component Version	New Features
1.00.00	Initial version of SmallSectorFee
1.01.00	Chapter 'Requirements and Recommendations' was added Reference to SmallSectorFee's ProductInformation was added
2.00.00	Chapter 'Incompatibility between SmallSectorFee Version 1.xx.xx and 2.xx.xx' was added

Table 1-1 Component history

2 Introduction

This document describes the functionality, API and configuration of the AUTOSAR BSW module FEE as specified in [1].

Supported AUTOSAR Release*:	3, 4	
Supported Configuration Variants:	pre-compile	
Vendor ID:	FEE_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	FEE_MODULE_ID	21 decimal (according to ref. [3])

* For the detailed functional specification please also refer to the corresponding AUTOSAR SWS.

The FEE enables you to access a dedicated flash area for storing data persistently. It is intended to be used exclusively either by the NVRAM Manager or on SW instance within a Flash-Boot-Loader.

This module is especially designed for Flash devices with small sector and page sizes, e.g. RH850's internal data flash RV40F with 64 Byte sectors and 4 Byte pages. Due to these hardware properties a static addressing scheme can be applied with reasonable small overhead. Consequently, the main advantage is a significantly easier handling of all jobs due to the static addressing scheme compared to the dynamic block addressing of 'standard' FEE.

Further on, the module depends on some other modules like DET for error handling, the underlying Flash driver for hardware access and the MEMIF for consistent types.

For further information about basic SmallSectorFee mechanisms and functionality refer to [4]. This product information gives a brief overview of relevant aspects concerning flash memory and informs about SmallSectorFee implementation.

2.1 Architecture Overview

The following figure shows where the FEE is located in the AUTOSAR architecture.

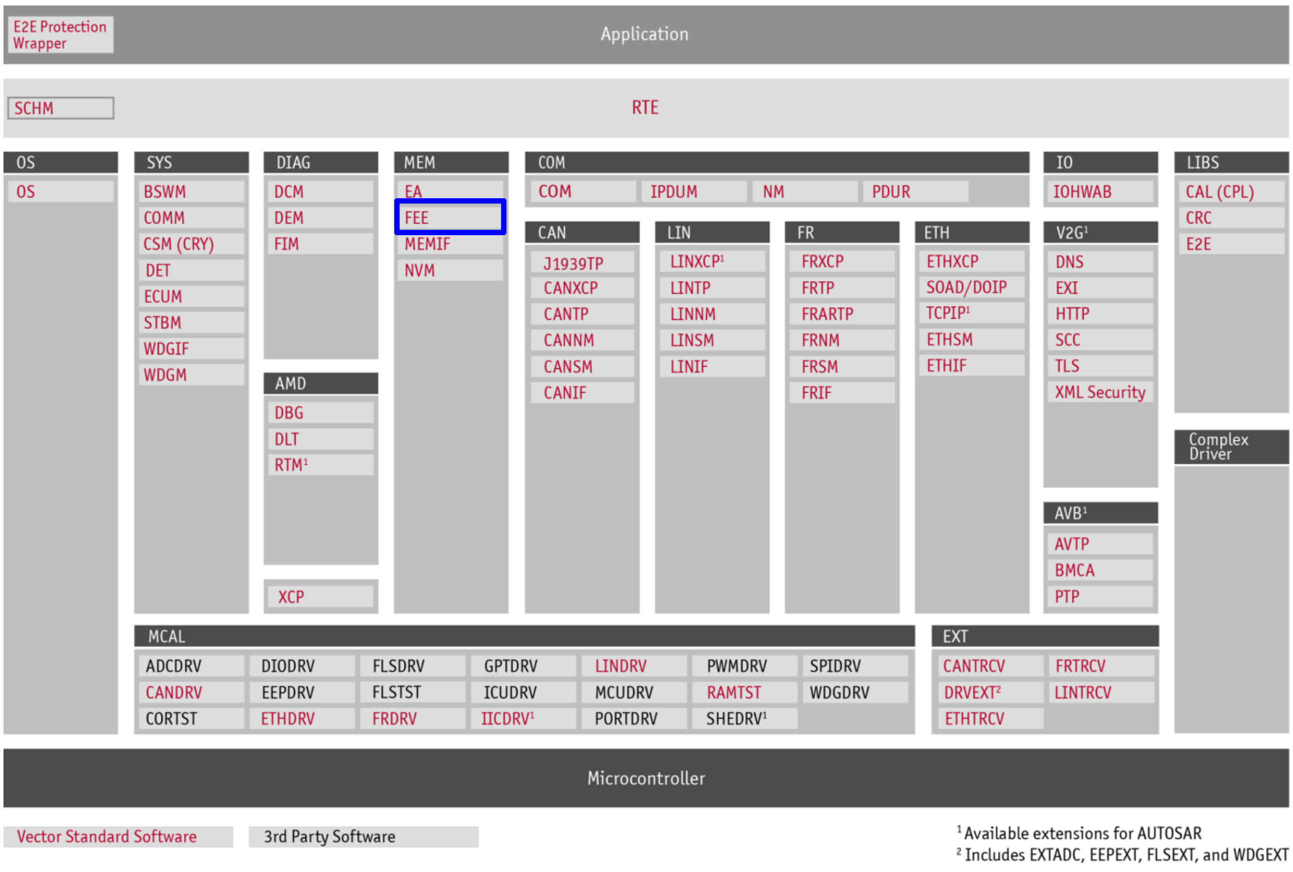


Figure 2-1 AUTOSAR 4.x Architecture Overview

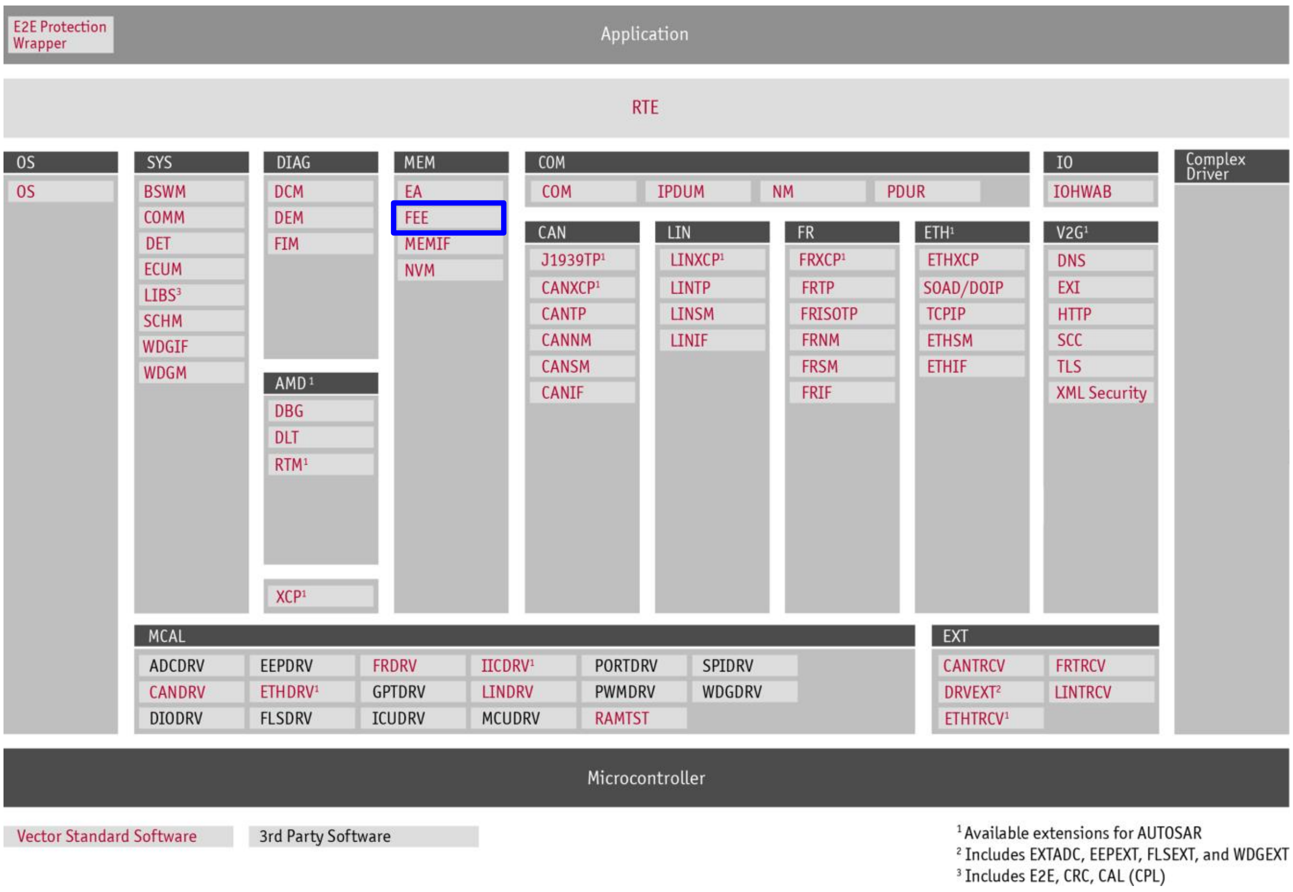


Figure 2-2 AUTOSAR 3.x Architecture Overview

The next figure shows the interfaces to adjacent modules of the FEE. These interfaces are described in chapter 5.

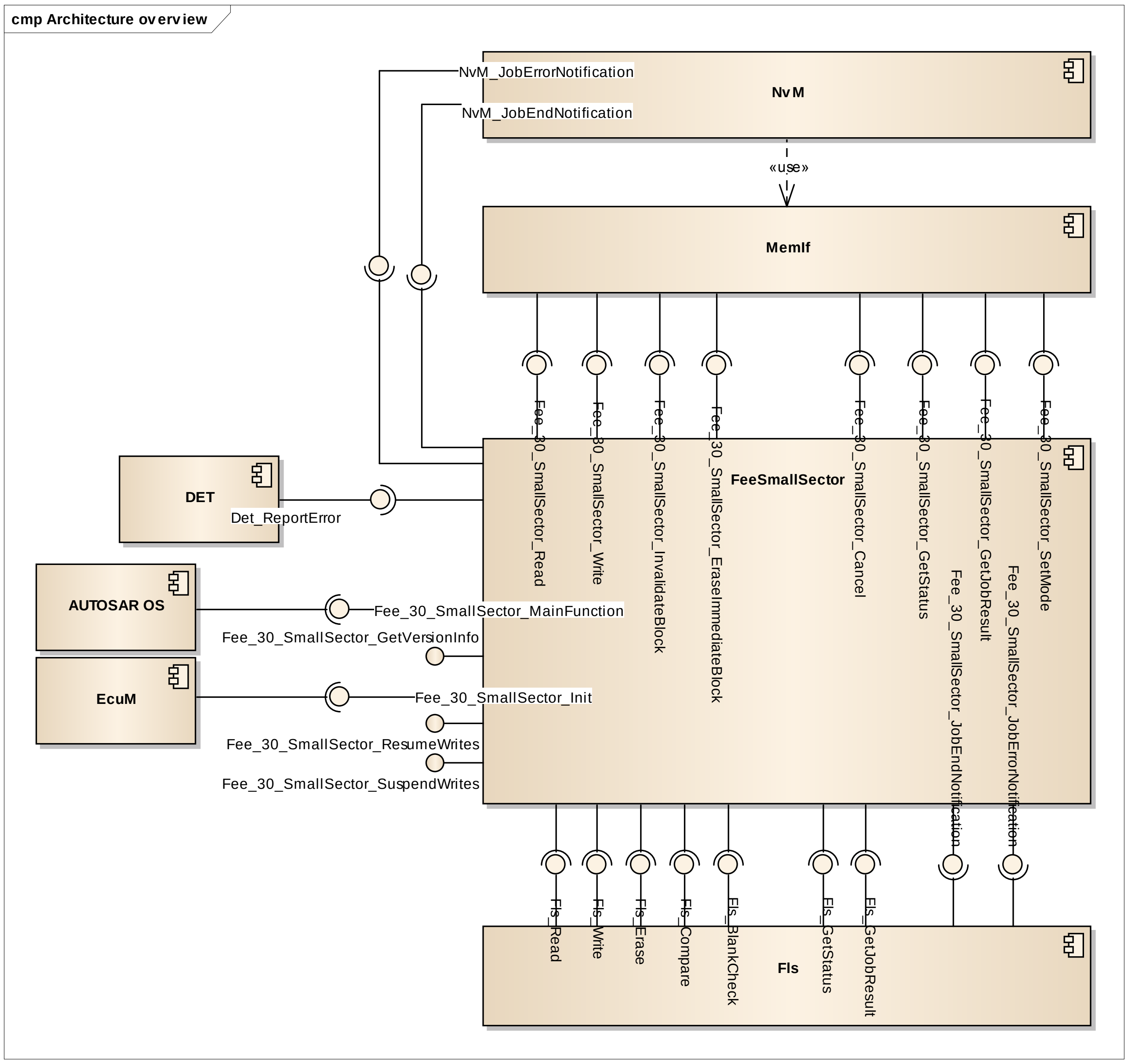

cmp Architecture overview
NvM
NvM_JobErrorNotification
NvM_JobEndNotification
«use»
MemIf
Fee_30_SmallSector_Read
Fee_30_SmallSector_Write
Fee_30_SmallSector_InvalidateBlock
Fee_30_SmallSector_EraseImmediateBlock
Fee_30_SmallSector_Cancel
Fee_30_SmallSector_GetStatus
Fee_30_SmallSector_GetJobResult
Fee_30_SmallSector_SetMode
FeeSmallSector
DET
Det_ReportError
AUTOSAR OS
Fee_30_SmallSector_MainFunction
Fee_30_SmallSector_GetVersionInfo
EcuM
Fee_30_SmallSector_Init
Fee_30_SmallSector_ResumeWrites
Fee_30_SmallSector_SuspendWrites
Fee_30_SmallSector_JobEndNotification
Fee_30_SmallSector_JobErrorNotification
Fls_Read
Fls_Write
Fls_Erase
Fls_Compare
Fls_BlankCheck
Fls
Fls_GetStatus
Fls_GetJobResult

Figure 2-3 Interfaces to adjacent modules of the FEE

3 Functional Description

3.1 Features

The features listed in the following tables cover the complete functionality specified for the FEE.

The AUTOSAR standard functionality is specified in [1], the corresponding features are listed in the tables

> Table 3-1 Supported AUTOSAR standard conform features

> Table 3-2 Not supported AUTOSAR standard conform features

Vector Informatik provides further FEE functionality beyond the AUTOSAR standard. The corresponding features are listed in the table

> Table 3-3 Features provided beyond the AUTOSAR standard

The following features specified in [1] are supported:

Supported AUTOSAR Standard Conform Features
FEE provides a service for reading blocks from Flash
FEE provides a service for writing blocks to Flash
FEE provides a service for invalidating blocks in Flash
FEE provides a service for erasing blocks which contain immediate data
FEE provides a service to initialize the module
FEE provides a service to cancel pending jobs
FEE provides a service to query the module status and job result
FEE provides a service to query its version information.
FEE provides a mechanism to spread the erase/write accesses such that the physical device is not overstressed, if the underlying Flash device does not provide at least the configured number of erase/write cycles per physical memory cell.
FEE provides a mechanism to manage each block's information, whether this block is "correct" from the point of view of the FEE.
FEE provides development error detection to check API parameters.
The Flash driver can either be polled by the FEE for its current state or the Flash driver can provide its state to the FEE module via a callback mechanism.
The FEE can be polled by the NVM or the FEE provides the result of a job to the upper layer via a callback mechanism.

Table 3-1 Supported AUTOSAR standard conform features

3.1.1 Deviations from AUTOSAR R4.0.3

The following features specified in [1] are not supported:

Not Supported AUTOSAR Standard Conform Features
Alignment of logical blocks via FeeVirtualPageSize is not supported. Instead, alignment of logical

Not Supported AUTOSAR Standard Conform Features
blocks is provided via partition specific alignments. Therefore see Table 3-3 Features provided beyond the AUTOSAR standard.
FEE does not provide debugging support.
FEE does not report any errors to DEM.
FEE does not support the usage of FeeBlockOverhead parameter, because the amount of management overhead is an internal detail, which shall not be configurable.
FEE does not support the usage of FeeMaximumBlockingTime parameter because FEE does not have any time base.
FEE does not support to set the mode of the underlying Flash driver, because handling of set mode API is not clearly specified.

Table 3-2 Not supported AUTOSAR standard conform features

3.1.2 Additions/ Extensions

The following features are provided beyond the AUTOSAR standard:

Features Provided Beyond The AUTOSAR Standard
FEE supports the usage of multiple Flash devices.
FEE supports the usage of Fls_BlankCheck API.
FEE provides the configuration of partitions, in order to separate independent contents/devices from each other.
FEE provides partition specific address alignments to which all logical blocks of a partition are aligned. Address alignment is configurable separately for each partition. Address alignment usually corresponds to the Flash device's sector size.
FEE provides partition specific write alignments which usually correspond to the Flash device's page size.
FEE provides verification of data which has been written recently to the Flash. This feature can be configured block specifically.
FEE provides detection and correction of single bit flips in a block's management information.
FEE supports two main function triggering modes. Main function of FEE can either be called cyclically with a fixed cycle time or in a background task.

Table 3-3 Features provided beyond the AUTOSAR standard

3.2 Recommendations

It's recommended to use this FEE module only with Flash devices with reasonable small sector and page sizes. This means that the size of one user block should not be significantly smaller than one physical Flash sector. Due to the fact that for every user block at least one physical Flash sector is reserved, the overhead of unused Flash memory strongly depends on the relation between block size and Flash sector size. Another aspect concerns the page size of a Flash device: In order to keep the overhead small, the Flash device's page size should be small as well.

For instance it's recommended to use this FEE module with internal data flash RV40F of Renesas' RH850 microcontroller platform. This Flash device features physical sectors with 64 Byte and physical pages with 4 Byte.

3.3 Initialization

The FEE is initialized and operational after the API service `Fee_30_SmallSector_Init()` was called.

Caution
The FEE is driven asynchronously, if a job has been requested to it. I.e. the jobs are finally processed by calls of FEE's main function.

3.4 States

FEE can change to different states during runtime which are described in the tables below.

3.4.1 Module States

Module States	Point in Time
`MEMIF_UNINIT`	The API service `Fee_30_SmallSector_Init()` has not been called yet. The service `Fee_30_SmallSector_GetStatus()` returns the value `MEMIF_UNINIT`.
`MEMIF_IDLE`	The API service `Fee_30_SmallSector_Init()` has been called and finished successfully. No job has currently been requested to the FEE. The service `Fee_30_SmallSector_GetStatus()` returns the value `MEMIF_IDLE`.
`MEMIF_BUSY`	The API services `Fee_30_SmallSector_Read()`,`Fee_30_SmallSector_Write()`,`Fee_30_SmallSector_InvalidateBlock()` or `Fee_30_SmallSector_EraseImmediateBlock()` have been called previously and the job is not yet finished. The service `Fee_30_SmallSector_GetStatus()` returns the value `MEMIF_BUSY`.

Table 3-4 Module States

3.4.2 Job States

Job State	Correlating Module State	Point in Time
`MEMIF_JOB_OK`	`MEMIF_IDLE`	After successfully finished job
`MEMIF_JOB_PENDING`	`MEMIF_BUSY`	After a job has been started
`MEMIF_JOB_CANCELLED`	`MEMIF_IDLE`	After `Fee_30_SmallSector_Cancel()`, if previous state was `MEMIF_JOB_PENDING`
`MEMIF_BLOCK_INVALID`	`MEMIF_IDLE`	After a read job has been finished and an invalidated block was found or if block was erased.

MEMIF_BLOCK_INCONSISTENT	MEMIF_IDLE	After reading a block, which wasn't finished successfully.
MEMIF_JOB_FAILED	MEMIF_IDLE	After a job wasn't finished successfully.

Table 3-5 Job States

3.5 Main Functions

All jobs (Read, Write, InvalidateBlock or EraseImmediateBlock) will be executed asynchronously and are processed by a job state machine. This means that after an asynchronous API call (e.g. `Fee_30_SmallSector_Write()`) the job and its parameters are internally stored and will be processed by the main function later on.

The function `Fee_30_SmallSector_MainFunction()` can either be called cyclically with a fixed cycle time by the OS or in a background task. The mode of main function triggering is pre-compile configurable via `FeeMainFunctionTriggering` parameter.

Only one asynchronous job (Read, Write, InvalidateBlock or EraseImmediateBlock) can be processed at a time. FEE does not provide a queue for any jobs that are requested while the module is busy processing a job. The module state needs to be `MEMIF_IDLE` before a new job can be requested. The current module state can be queried by calling the service `Fee_30_SmallSector_GetStatus()`. Usually, the upper layer module (i.e. NVM) is responsible for synchronizing and queueing the pending jobs and assigning it to the FEE consecutively, whereas in the Flash Boot Loader use-case the Flash Boot Loader application is responsible for doing this.

3.5.1 Processing of a Read Job

FEE provides the service `Fee_30_SmallSector_Read()` for reading a block. This service reads the latest data of the block specified by a block number.

This asynchronous job is initiated with the API function `Fee_30_SmallSector_Read()` and is processed by subsequent calls of `Fee_30_SmallSector_MainFunction()` until the FEE returns back to idle state, which means that the job was finished.

Caution
The FEE will always read the block, which was written last. Consequently, if writing a block fails (e.g. due to reset) a subsequent read job for this block will lead to the result `MEMIF_BLOCK_INCONSISTENT`. The FEE does not look for the most recent valid instance of this block, because it can't be assured that there is any.

3.5.2 Processing of a Write Job

FEE provides the service `Fee_30_SmallSector_Write()` for writing a block specified by a block number. The FEE searches for the next free position in the specified block's memory area and requests a job to the underlying Flash driver to write a new instance of the block.

This asynchronous job is initiated with the API function `Fee_30_SmallSector_Write()` and is processed by subsequent calls of `Fee_30_SmallSector_MainFunction()` until the FEE returns back to idle state, which means that the job was finished.

3.5.3 Processing of an InvalidateBlock Job

FEE provides the service `Fee_30_SmallSector_InvalidateBlock()` for invalidating block content specified by a block number. The FEE module marks the block as invalid and reports the job result `MEMIF_BLOCK_INVALID` if such a block will be read after the invalidation process.

This asynchronous job is initiated with the API function `Fee_30_SmallSector_InvalidateBlock()` and is processed by subsequent calls of `Fee_30_SmallSector_MainFunction()` until the FEE returns back to idle state, which means that the job was finished.

3.5.4 Processing of an EraseImmediateBlock Job

FEE provides the service `Fee_30_SmallSector_EraseImmediateBlock()` to erase a block containing immediate data. Hereupon an invalidation of the specified block is performed, because erasing the block's memory area will not accomplish the intended goal, that writing of immediate data is speeded-up by a preceding erase operation.

This asynchronous job is initiated with the API function `Fee_30_SmallSector_EraseImmediateBlock()` and is processed by subsequent calls of `Fee_30_SmallSector_MainFunction()` until the FEE returns back to idle state, which means that the job was finished.

3.6 Error Handling

3.6.1 Development Error Reporting

By default, development errors are reported to the DET using the service `Det_ReportError()` as specified in [2], if development error reporting is enabled (i.e. pre-compile parameter `FEE_DEV_ERROR_DETECT==STD_ON`).

If another module is used for development error reporting, the function prototype for reporting the error can be configured by the integrator, but must have the same signature as the service `Det_ReportError()`.

The reported FEE ID is 21.

The reported service IDs identify the services which are described in 5.2. The following table presents the service IDs and the related services:

Service ID	Service
0x00	Fee_30_SmallSector_Init
0x01	Fee_30_SmallSector_SetMode
0x02	Fee_30_SmallSector_Read
0x03	Fee_30_SmallSector_Write
0x04	Fee_30_SmallSector_Cancel
0x05	Fee_30_SmallSector_GetStatus

Service ID	Service
0x06	Fee_30_SmallSector_GetJobResult
0x07	Fee_30_SmallSector_InvalidateBlock
0x08	Fee_30_SmallSector_GetVersionInfo
0x09	Fee_30_SmallSector_EraseImmediateBlock
0x10	Fee_30_SmallSector_JobEndNotification
0x11	Fee_30_SmallSector_JobErrorNotification
0x12	Fee_30_SmallSector_MainFunction

Table 3-6 Service IDs

The errors reported to DET are described in the following table:

Error Code	Description
FEE_30_SMALL_SECTOR_E_UNINIT	API service (except for `Fee_30_SmallSector_GetStatus()`,`Fee_30_SmallSector_GetVersionInfo()` and `Fee_30_SmallSector_Init()`) called although the FEE is not yet initialized
FEE_30_SMALL_SECTOR_E_INVALID_BLOCK_NO	API service called with invalid block number
FEE_30_SMALL_SECTOR_E_INVALID_BLOCK_OFS	API service `Fee_30_SmallSector_Read()` called with invalid block offset.
FEE_30_SMALL_SECTOR_E_INVALID_DATA_POINTER	API service (`Fee_30_SmallSector_Read()` or `Fee_30_SmallSector_Write()`) called with null pointer as pointer to data buffer. API service `Fee_30_SmallSector_GetVersionInfo()` called with null pointer.
FEE_30_SMALL_SECTOR_E_INVALID_BLOCK_LEN	API service `Fee_30_SmallSector_Read()` called with invalid block length.
FEE_30_SMALL_SECTOR_E_BUSY	API service is called while currently a job is being processed.
FEE_30_SMALL_SECTOR_E_BUSY_INTERNAL	Not used.
FEE_30_SMALL_SECTOR_E_INVALID_CANCEL	API service `Fee_30_SmallSector_Cancel()` is called while module is not busy.

Table 3-7 Errors reported to DET

3.6.2 Production Code Error Reporting

FEE does not report any production errors to DEM because the AUTOSAR specification does not specify any error which shall be reported in production mode.

3.7 Partitions

FEE employs a concept of partitions. A partition can be thought of an emulation space that is managed separately from other ones:

- Multiple Flash devices / Flash drivers are supported by using separate partitions in FEE module
- Alignments, such as write, read and address alignment, can be configured for each partition specifically
- Errors in one partition do not affect data in other ones
- Each partition has its own start address and size. Next partition does not have to start at the end of the previous partition. There can be gaps between the partitions.
- Partitions do not overlap each other
- Block with smallest Block ID has the smallest address in the partition. Each block has to be assigned to a partition
- A partition's address alignment must be identical or an integer multiple of its referenced Flash's sector size
- A partition's write alignment must be identical or an integer multiple of its referenced Flash's page size

Note
FEE configurations for FBL and Application do not need to share all partitions. E.g. a partition containing only application data may remain unknown to the FBL. However, shared partitions must refer to identical Fls configurations (FlsConfigSet container), and they must match in address and size, as well as in alignment settings.

3.8 Service for handling under-voltage situations

The FEE provides a set of functions to handle under-voltage situations properly.

`Fee_30_SmallSector_SuspendWrites()` is intended to react on actual under-voltage situation detected via dedicated monitoring circuitry. Usually there is some amount of time (few milliseconds) to react on such conditions until a low voltage reset occurs. Its counterpart, `Fee_30_SmallSector_ResumeWrites()` was introduced in order to prevent from stalling, if voltage reaches normal range, without any reset.

As long as writes are suspended the FEE no longer requests any write-class job from underlying Flash driver (e.g. `Fls_Write`, `Fls_Erase`) due to high risk of a reset. A currently pending write job of FEE will be paused until writes are resumed again. If FEE is currently idle when writes are suspended, FEE write jobs are accepted but will not be processed until writes are resumed again.

3.9 MainFunction Triggering

In AUTOSAR release 4.x an additional option is introduced to be able to call the `Fee_30_SmallSector_MainFunction` in a cyclic task or in a background task.

The cyclic task (default configuration) is used when the main function shall be triggered periodically. Typically the cycle time needs to be defined, for example 10ms.

If the `Fee_30_SmallSector_MainFunction` shall be accessed quicker without periodic time base, the function can also be called in a background task. The background task runs when the system has nothing else to do. The `Fee_30_SmallSector_MainFunction` is called as often as the available CPU load allows.

Caution
If the system is overloaded, it may happen that the background task is no longer called.

Note
The `Fee_30_SmallSector_MainFunction` should not be triggered faster than `Fls_MainFunction` because the FEE must wait for the FLS.

4 Integration

This chapter gives necessary information for the integration of the MICROSAR FEE into an application environment of an ECU.

4.1 Scope of Delivery

The delivery of the FEE contains the files which are described in the chapters 4.1.1 and 4.1.2:

4.1.1 Static Files

File Name	Description
Fee_30_SmallSector.c	Contains the implementation of the FEE interfaces.
Fee_30_SmallSector.h	Declares the interfaces of the FEE.
Fee_30_SmallSector_Cbk.h	Declares the callback functions of the FEE.
Fee_30_SmallSector_PartitionHandler.c	Responsible for partition relevant data.
Fee_30_SmallSector_PartitionHandler.h	Declares the interface of PartitionHandler.
Fee_30_SmallSector_BlockHandler.c	Responsible for block relevant data.
Fee_30_SmallSector_BlockHandler.h	Declares the interface of BlockHandler.
Fee_30_SmallSector_DatasetHandler.c	Responsible for dataset relevant data.
Fee_30_SmallSector_DatasetHandler.h	Declares the interface of DatasetHandler.
Fee_30_SmallSector_InstanceHandler.c	Responsible for instance relevant data.
Fee_30_SmallSector_InstanceHandler.h	Declares the interface of InstanceHandler.
Fee_30_SmallSector_TaskManager.c	Responsible for coordinating internal sub-components.
Fee_30_SmallSector_TaskManager.h	Declares the interface of TaskManager.
Fee_30_SmallSector_FlsCoordinator.c	Provides access to Flash driver's services.
Fee_30_SmallSector_FlsCoordinator.h	Declares the interface of FlsCoordinator.
Fee_30_SmallSector_Layer1_Read.c	Internal layer 1 sub-component for read jobs.
Fee_30_SmallSector_Layer1_Read.h	Declares the interface of layer 1 read sub-component.
Fee_30_SmallSector_Layer1_Write.c	Internal layer 1 sub-component for write, invalidate and erase jobs.
Fee_30_SmallSector_Layer1_Write.h	Declares the interface of layer 1 write sub-component.
Fee_30_SmallSector_Layer2_WriteInstance.c	Internal layer 2 sub-component for writing instances.
Fee_30_SmallSector_Layer2_WriteInstance.h	Declares the interface of layer 2 write instance sub-component.
Fee_30_SmallSector_Layer2_InstanceFinder.c	Internal layer 2 sub-component for finding instances.
Fee_30_SmallSector_Layer2_InstanceFinder.h	Declares the interface of layer 2 instance finder sub-component.

File Name	Description
Fee_30_SmallSector_Layer2_DatasetEraser.c	Internal layer 2 sub-component for erasing datasets.
Fee_30_SmallSector_Layer2_DatasetEraser.h	Declares the interface of layer 2 dataset eraser sub-component.
Fee_30_SmallSector_Layer3_ReadManagementBytes.c	Internal layer 3 sub-component for reading management information of instances.
Fee_30_SmallSector_Layer3_ReadManagementBytes.h	Declares the interface of layer 3 read management bytes sub-component.

Table 4-1 Static files

4.1.2 Dynamic Files

The dynamic files are generated by the configuration tool DaVinci Configurator.

File Name	Description
Fee_30_SmallSector_Cfg.c	Contains definitions of the configuration, e.g. partition configuration and block configuration
Fee_30_SmallSector_Cfg.h	Contains declarations of the configuration and macro definitions.

Table 4-2 Generated files

4.2 Incompatibility between SmallSectorFee Version 1.xx.xx and 2.xx.xx

This incompatibility affects only projects that shall be able to access flash data, which is written with the previous version of the SmallSectorFee.

In the new major version 2.xx.xx of the SmallSectorFee an additional pattern was added to the memory layout, which increases the reset robustness. Therefore, the old SmallSectorFEE flash data is no longer compatible with the required flash format of the new one. The new SmallSectorFEE cannot read data written by the old one and reports the existing data to be inconsistent.

If the already stored data have to be used, a manual update strategy is required like:

> Use the old SmallSectorFEE to retrieve stored data from the ECU and store that data externally.

> Update the SIP with the new SmallSectorFEE.

> Write the stored data with the new SmallSectorFEE.

If the manual update strategy is not applicable, it is recommended to continue using the older SmallSectorFee Version.

If two SmallSectorFee modules are used for the Flash Bootloader and Application use-case with shared data, both modules must have the same Version.

4.3 Migration from FEE to SmallSectorFEE

When it comes to an update of a project, where MICROSAR FEE (“Standard” FEE) is replaced by MICROSAR SmallSectorFEE, this section shows how to migrate the consisting FEE configuration to SmallSectorFEE configuration.

1. **Update SIP with SmallSectorFEE description and generator**

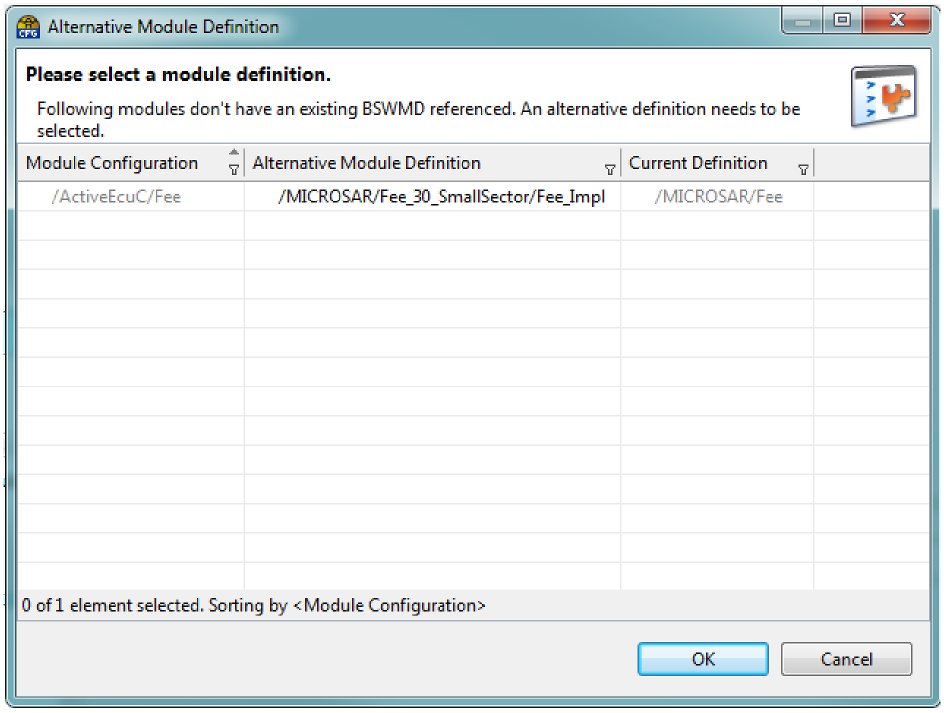
The SIP of the updated project should only contain SmallSectorFEE parts. Description and Generator of “Standard” FEE shall be removed from the project.

2. **Update BSWMD references**

Upon next start of Configurator it will be detected that the BSWMD file of “Standard” FEE is missing and SmallSectorFEE’s BSWMD is found instead, according to Figure 4-1 Update references of BSWMD file.

Select OK in order to choose the alternative module definition /MICROSAR/Fee_30_SmallSector/Fee_Impl for the current definition /MICROSAR/Fee.

The DaVinci Configurator 5 then takes care of replacing the references within the project, so that as much information as possible is retained. Parameters which are identical in both the previous and the new definition are matched easily, which is applicable to all AUTOSAR parameters. Due to the fact that even most of the vendor specific parameters, e.g. FeePartitionConfiguration parameters, haven’t changed significantly there’s no real data loss.


Alternative Module Definition

Please select a module definition.

Following modules don't have an existing BSWMD referenced. An alternative definition needs to be selected.

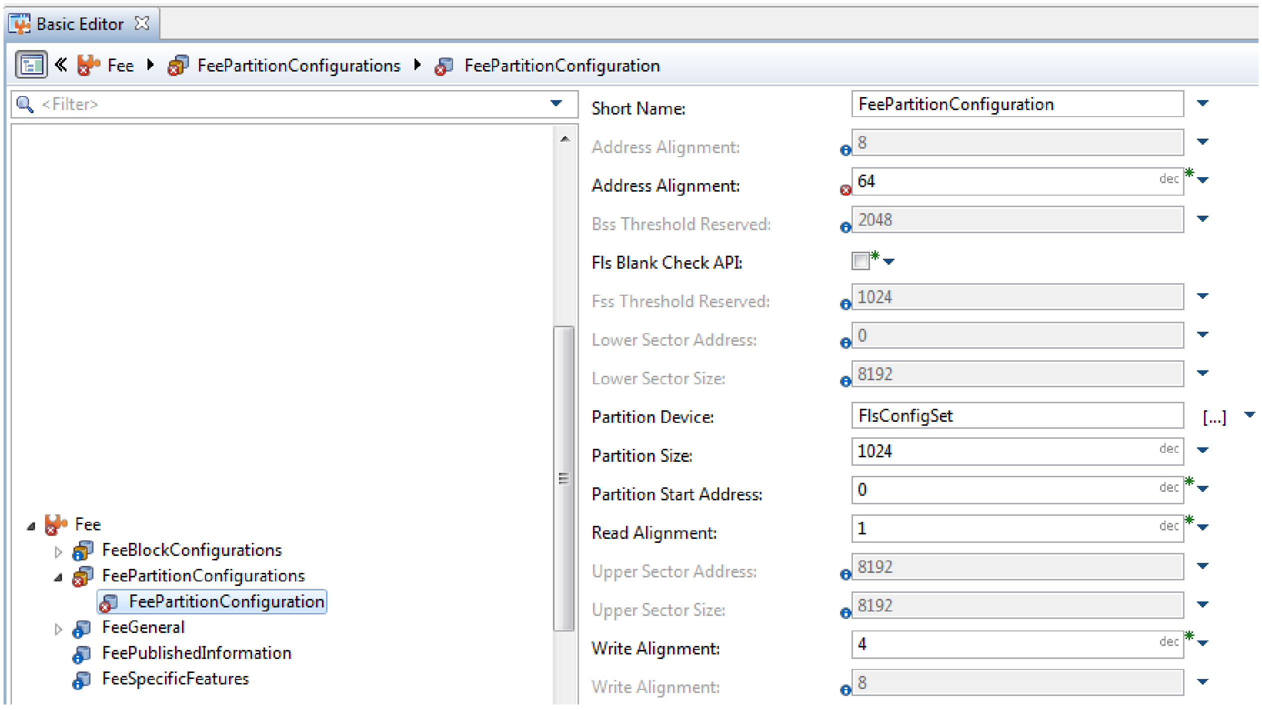
Module Configuration	Alternative Module Definition	Current Definition
/ActiveEcuC/Fee	/MICROSAR/Fee_30_SmallSector/Fee_Impl	/MICROSAR/Fee

0 of 1 element selected. Sorting by <Module Configuration>

Figure 4-1 Update references of BSWMD file

3. **Remove incorrect definitions**

After updating the BSWMD file and after starting the DaVinci Configurator all the parameters are marked with a small info icon which could not be assigned to the new definition.



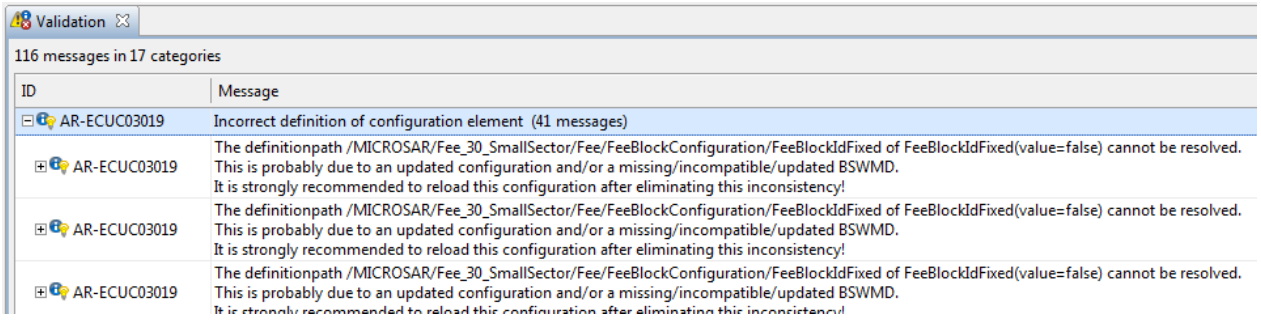
Basic Editor

Fee ▸ FeePartitionConfigurations ▸ FeePartitionConfiguration

Short Name:	FeePartitionConfiguration
Address Alignment:	8
Address Alignment:	64
Bss Threshold Reserved:	2048
Fls Blank Check API:	☐
Fss Threshold Reserved:	1024
Lower Sector Address:	0
Lower Sector Size:	8192
Partition Device:	FlsConfigSet
Partition Size:	1024
Partition Start Address:	0
Read Alignment:	1
Upper Sector Address:	8192
Upper Sector Size:	8192
Write Alignment:	4
Write Alignment:	8

Figure 4-2 Example PartitionConfiguration after updating BSWMD references

Figure 4-2 Example PartitionConfiguration after updating BSWMD references depicts an example PartitionConfiguration with some parameters greyed out, which are no longer available after updating the FEE module definition. The DaVinci Configurator 5 shows appropriate information in the Validation view, as depicted in Figure 4-3 DaVinci Configurator signals incorrect definition of configuration elements.



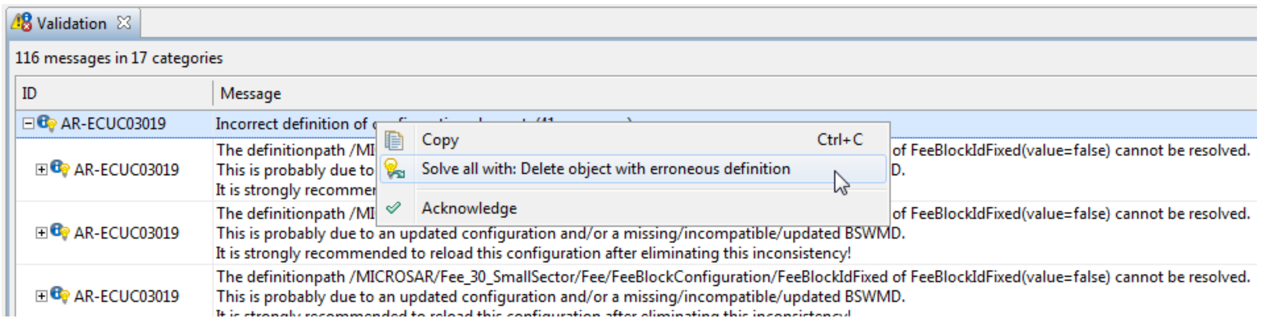
Validation

116 messages in 17 categories

ID	Message
AR-ECUC03019	Incorrect definition of configuration element (41 messages)
AR-ECUC03019	The definitionpath /MICROSAR/Fee_30_SmallSector/Fee/FeeBlockConfiguration/FeeBlockIdFixed of FeeBlockIdFixed(value=false) cannot be resolved. This is probably due to an updated configuration and/or a missing/incompatible/updated BSWMD. It is strongly recommended to reload this configuration after eliminating this inconsistency!
AR-ECUC03019	The definitionpath /MICROSAR/Fee_30_SmallSector/Fee/FeeBlockConfiguration/FeeBlockIdFixed of FeeBlockIdFixed(value=false) cannot be resolved. This is probably due to an updated configuration and/or a missing/incompatible/updated BSWMD. It is strongly recommended to reload this configuration after eliminating this inconsistency!
AR-ECUC03019	The definitionpath /MICROSAR/Fee_30_SmallSector/Fee/FeeBlockConfiguration/FeeBlockIdFixed of FeeBlockIdFixed(value=false) cannot be resolved. This is probably due to an updated configuration and/or a missing/incompatible/updated BSWMD. It is strongly recommended to reload this configuration after eliminating this inconsistency!

Figure 4-3 DaVinci Configurator signals incorrect definition of configuration elements

To remove all incorrect definitions a solving action can be processed from the context menu of the top element as shown in Figure 4-4 Choose solving action to delete all erroneous definitions.



Validation

116 messages in 17 categories

ID	Message
AR-ECUC03019	Incorrect definition of c
AR-ECUC03019	The definitionpath /MI … of FeeBlockIdFixed(value=false) cannot be resolved. This is probably due to … D. It is strongly recommer
AR-ECUC03019	The definitionpath /MI … of FeeBlockIdFixed(value=false) cannot be resolved. This is probably due to an updated configuration and/or a missing/incompatible/updated BSWMD. It is strongly recommended to reload this configuration after eliminating this inconsistency!
AR-ECUC03019	The definitionpath /MICROSAR/Fee_30_SmallSector/Fee/FeeBlockConfiguration/FeeBlockIdFixed of FeeBlockIdFixed(value=false) cannot be resolved. This is probably due to an updated configuration and/or a missing/incompatible/updated BSWMD.

Figure 4-4 Choose solving action to delete all erroneous definitions

4. Solve appearing errors

After the erroneous definitions were removed, still some errors may be present because of incorrect parameter configuration. In order to solve these configuration issues please refer to error descriptions in Validation view of DaVinci Configurator and to section 6.2 Configuration with DaVinci Configurator in order to get some hints for configuration.

5 API Description

For an interfaces overview please see Figure 2-3.

5.1 Type Definitions

The FEE does not specify any API data types.

5.2 Services provided by FEE

5.2.1 Fee_30_SmallSector_Init

Prototype	
`void` **`Fee_30_SmallSector_Init`** `( void )`	
Parameter	
`--`	--
Return code	
`void`	--
Functional Description	
This service initializes the FEE module and all necessary internal variables. The FEE module doesn’t support any runtime configuration. Hence, a pointer to the configuration structure is not needed by this service. The FEE does not initialize the underlying Flash driver. This shall be done separately, e.g. by the ECUM module.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is non-reentrant. > This service shall not be called during a pending job.	
Expected Caller Context	
> Expected to be called in application context.	

Table 5-1 Fee_30_SmallSector_Init

5.2.2 Fee_30_SmallSector_SetMode

Prototype	
`void` **`Fee_30_SmallSector_SetMode`** `( void )`	
Parameter	
`Mode`	Parameter is not used.
Return code	
`void`	--

Functional Description
This service will not be supported by current implementation of FEE module because SetMode handling is not clearly specified by AUTOSAR.
Particularities and Limitations
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is non-reentrant. > This service has no effect at all
Expected Caller Context
> Expected to be called in application context.

Table 5-2 Fee_30_SmallSector_SetMode

5.2.3 Fee_30_SmallSector_Read

Prototype	

`Std_ReturnType` **`Fee_30_SmallSector_Read`** `(`
`uint16 BlockNumber, uint16 BlockOffset, uint8* DataBufferPtr, uint16 Length)`	

Parameter	
`BlockNumber`	Handle of a logical block (depending on block configuration)
`BlockOffset`	Read address offset inside the block
`DataBufferPtr`	Pointer to data buffer
`Length`	Number of bytes to read
Return code	
`E_OK`	Read job has been accepted
`E_NOT_OK`	Read job has been rejected
Functional Description	
This service invokes the read processing of the specified block. After the job has been processed by the `Fee_30_SmallSector_MainFunction` the requested data has been read and passed to the caller. **Note** The job processing is asynchronous. Hence, it is necessary to poll the FEE about its current status by calling the function `Fee_30_SmallSector_GetStatus()` to check if the job was completed. Finally, the result of the finished job can be queried by calling the function `Fee_30_SmallSector_GetJobResult()`. Accordingly, it is possible to notify the upper layer (usually the NVM) via the callback mechanism. Additionally, if development mode is configured, parameter checks are performed and in case of failure they are reported to the DET with according service ID and the reason of occurrence (refer to 3.6.1).	

Particularities and Limitations
> Service ID: see table 'Service IDs' > This function is asynchronous. > This function is non-reentrant.
Expected Caller Context
> Expected to be called in application context.

Table 5-3 Fee_30_SmallSector_Read

5.2.4 Fee_30_SmallSector_Write

Prototype	

`Std_ReturnType` **`Fee_30_SmallSector_Write`** `(`
`uint16 BlockNumber, uint8* DataBufferPtr)`	

Parameter	
`BlockNumber`	Handle of a logical block (depending on block configuration)
`DataBufferPtr`	Pointer to data buffer
Return code	
`E_OK`	Write job has been accepted
`E_NOT_OK`	Write job has been rejected
Functional Description	
This service invokes the write processing of the specified block. After the job has been processed by the `Fee_30_SmallSector_MainFunction` the requested data has been written to the non-volatile memory. **Note** The job processing is asynchronous. Hence, it is necessary to poll the FEE about its current status by calling the function `Fee_30_SmallSector_GetStatus()` to check if the job was completed. Finally, the result of the finished job can be queried by calling the function `Fee_30_SmallSector_GetJobResult()`. Accordingly, it is possible to notify the upper layer (usually the NVM) via the callback mechanism. Additionally, if development mode is configured, parameter checks are performed and in case of failure they are reported to the DET with according service ID and the reason of occurrence (refer to 3.6.1).	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is asynchronous. > This function is non-reentrant.	
Expected Caller Context	
> Expected to be called in application context.	

Table 5-4 Fee_30_SmallSector_Write

5.2.5 Fee_30_SmallSector_Cancel

Prototype	
`void` **`Fee_30_SmallSector_Cancel`** `( void )`	
Parameter	
`--`	--
Return code	
`void`	--
Functional Description	
This service cancels a currently pending job. **Note** This service is a synchronous call and does not have to be triggered by the `Fee_30_SmallSector_MainFunction()`. The status of the FEE will be set to `MEMIF_IDLE` and the job result will be set to `MEMIF_JOB_CANCELLED`, if a job was currently pending. If the FEE is currently `MEMIF_IDLE`, calling this service is without any effect. A development error will be reported to the DET module in this case, if development error detection is enabled.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is non-reentrant.	
Expected Caller Context	
> Expected to be called in application context.	

Table 5-5 Fee_30_SmallSector_Cancel

5.2.6 Fee_30_SmallSector_GetStatus

Prototype	
`MemIf_StatusType` **`Fee_30_SmallSector_GetStatus`** `( void )`	
Parameter	
`--`	--
Return code	
`MEMIF_UNINIT`	The FEE is currently not initialized. `Fee_30_SmallSector_Init()` must be called to use the functionality of FEE.
`MEMIF_IDLE`	The FEE is currently idle. No asynchronous job is pending.
`MEMIF_BUSY`	The FEE is currently busy. An asynchronous job is currently being processed by the FEE.

Functional Description
This service returns synchronously the current module state of the FEE.
Particularities and Limitations
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant.
Expected Caller Context
> Expected to be called in application context.

Table 5-6 Fee_30_SmallSector_GetStatus

5.2.7 Fee_30_SmallSector_GetJobResult

Prototype	
`MemIf_JobResultType` **`Fee_30_SmallSector_GetJobResult`** `( void )`	
Parameter	
--	--
Return code	
`MEMIF_JOB_OK`	The last job has been finished successfully.
`MEMIF_JOB_PENDING`	The last requested job is waiting for execution or is currently being executed.
`MEMIF_JOB_CANCELLED`	The last job has been canceled via the `Fee_30_SmallSector_Cancel()` service.
`MEMIF_JOB_FAILED`	The Flash driver reported an error or the FEE could not achieve the requested job due to hardware errors (e.g. memory cells defect)
`MEMIF_BLOCK_INCONSISTENT`	The requested block's management information is inconsistent; hence it may contain corrupt data. This result happens if a write job has not been completed (e.g. due to voltage drop) or if bit flips in the management information occurred and can't be corrected. Furthermore, if write-verify for a specific block is enabled and the verification of the written data fails, the job result is set to `INCONSISTENT`.
`MEMIF_BLOCK_INVALID`	The requested block has been invalidated previously via the service `Fee_30_SmallSector_InvalidateBlock()` or it is erased (e.g. by calling `Fee_30_SmallSector_EraseImmediateBlock()`).
Functional Description	
This service returns the job result of the last job which was executed.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant.	
Expected Caller Context	
> Expected to be called in application context.	

Table 5-7 Fee_30_SmallSector_GetJobResult

5.2.8 Fee_30_SmallSector_InvalidateBlock

Prototype	
`Std_ReturnType` **`Fee_30_SmallSector_InvalidateBlock`** `( uint16 BlockNumber )`	
Parameter	
`BlockNumber`	Handle of a logical block (depending on block configuration)
Return code	
`E_OK`	Invalidate job has been accepted
`E_NOT_OK`	Invalidate job has been rejected
Functional Description	
This service invokes the invalidate processing of the specified block. After the job has been processed by the `Fee_30_SmallSector_MainFunction` the requested block is marked as `INVALID`. **Note** The job processing is asynchronous. Hence, it is necessary to poll the FEE about its current status by calling the function `Fee_30_SmallSector_GetStatus()` to check if the job was completed. Finally, the result of the finished job can be queried by calling the function `Fee_30_SmallSector_GetJobResult()`. Accordingly, it is possible to notify the upper layer (usually the NVM) via the callback mechanism. Additionally, if development mode is configured, parameter checks are performed and in case of failure they are reported to the DET with according service ID and the reason of occurrence (refer to 3.6.1).	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is asynchronous. > This function is non-reentrant.	
Expected Caller Context	
> Expected to be called in application context.	

Table 5-8 Fee_30_SmallSector_InvalidateBlock

5.2.9 Fee_30_SmallSector_GetVersionInfo

Prototype	
`void` **`Fee_30_SmallSector_GetVersionInfo`** `( Std_VersionInfoType *VersionInfoPtr )`	
Parameter	
`VersionInfoPtr`	Pointer to where to store the version information of this module.
Return code	
`void`	--

Functional Description
This service returns the version information of this module. The version information includes: > Module ID > Vendor ID > Vendor specific version numbers Additionally, if development mode is configured, parameter checks are performed and in case of failure they are reported to the DET with according service ID and the reason of occurrence (refer to 3.6.1).
Particularities and Limitations
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is non-reentrant. > This service is can be en-/disabled via configuration parameter `FeeVersionInfoApi`
Expected Caller Context
> Expected to be called in application context.

Table 5-9 Fee_30_SmallSector_GetVersionInfo

5.2.10 Fee_30_SmallSector_EraseImmediateBlock

Prototype	
`Std_ReturnType` **`Fee_30_SmallSector_EraseImmediateBlock`** `( uint16 BlockNumber )`	
Parameter	
`BlockNumber`	Handle of a logical block (depending on block configuration)
Return code	
`E_OK`	Erase job has been accepted
`E_NOT_OK`	Erase job has been rejected
Functional Description	
This function doesn't erase flash memory. The addressed block is marked as invalid, thus a subsequent read request on the invalidated block completes with `MEMIF_BLOCK_INVALID`. **Note** The job processing is asynchronous. Hence, it is necessary to poll the FEE about its current status by calling the function `Fee_30_SmallSector_GetStatus()` to check if the job was completed. Finally, the result of the finished job can be queried by calling the function `Fee_30_SmallSector_GetJobResult()`. Accordingly, it is possible to notify the upper layer (usually the NVM) via the callback mechanism. Additionally, if development mode is configured, parameter checks are performed and in case of failure they are reported to the DET with according service ID and the reason of occurrence (refer to 3.6.1).	

Particularities and Limitations
> Service ID: see table 'Service IDs' > This function is asynchronous. > This function is non-reentrant.
Expected Caller Context
> Expected to be called in application context.

Table 5-10 Fee_30_SmallSector_EraseImmediateBlock

5.2.11 Fee_30_SmallSector_MainFunction

Prototype	
`void` **`Fee_30_SmallSector_MainFunction`** `( void )`	
Parameter	
--	--
Return code	
`void`	--
Functional Description	
This service triggers the processing of internal state machine and handles the asynchronous job and management operations. The complete handling of the job and the detection of invalidated or inconsistent blocks will be done in the internal job state machine. Additionally, if development mode is configured, parameter checks are performed and in case of failure they are reported to the DET with according service ID and the reason of occurrence (refer to 3.6.1).	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is non-reentrant.	
Expected Caller Context	
> May be called at interrupt level	

Table 5-11 Fee_30_SmallSector_MainFunction

5.2.12 Fee_30_SmallSector_SuspendWrites

Prototype	
`void` **`Fee_30_SmallSector_SuspendWrites`** `( void )`	
Parameter	
--	--
Return code	
`void`	--

Functional Description
This service instructs FEE to block all write class jobs (writing, invalidating and erasing). Pending jobs will be paused, i.e. they won't be finished. FEE will enter a safe state (by means of Flash content). Once such state was reached, FEE does not issue new write requests to FLS. Multiple subsequent calls of this service don't have additional effects, i.e. to re-enable write accesses only one call of `Fee_30_SmallSector_ResumeWrites` is necessary.
Particularities and Limitations
> This function is synchronous. > This function is non-reentrant.
Expected Caller Context
> Expected to be called in application context.

Table 5-12 Fee_30_SmallSector_SuspendWrites

5.2.13 Fee_30_SmallSector_ResumeWrites

Prototype	
`void` **`Fee_30_SmallSector_ResumeWrites`** `( void )`	
Parameter	
--	--
Return code	
`void`	--
Functional Description	
This service instructs FEE to allow all write class jobs (writing, invalidating and erasing), after being suspended by `Fee_30_SmallSector_SuspendWrites`. Multiple calls of this service don't have any additional effects, i.e. to disable write class jobs only `Fee_30_SmallSector_SuspendWrites` needs to be called only once.	
Particularities and Limitations	
> This function is synchronous. > This function is non-reentrant.	
Expected Caller Context	
> May be called at interrupt level	

Table 5-12 Fee_30_SmallSector_ResumeWrites

5.3 Services used by FEE

In the following table services provided by other components, which are used by the FEE are listed. For details about prototype and functionality refer to the documentation of the providing component.

Component	API
DET	Det_ReportError
FLS	Fls_Read Fls_Write Fls_Compare Fls_Erase Fls_BlankCheck (if configured) Fls_GetJobResult Fls_GetStatus
NVM	NvM_JobEndNotification (if configured) NvM_JobErrorNotification (if configured)

Table 5-13 Services used by the FEE

5.4 Callback Functions

This chapter describes the callback functions that are implemented by the FEE and can be invoked by other modules. The prototypes of the callback functions are provided in the header file `Fee_30_SmallSector_Cbk.h` by the FEE.

5.4.1 Fee_30_SmallSector_JobEndNotification

Prototype	
`void` **`Fee_30_SmallSector_JobEndNotification`** `( void )`	
Parameter	
--	--
Return code	
`void`	--
Functional Description	
This routine shall be called by the underlying flash driver to report the successful end of an asynchronous operation. **Note** This function is configurable at pre-compile time using the parameter `FeePollingMode.`	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is non-reentrant.	
Expected Caller Context	
> This routine might be called on interrupt level, depending on the calling function	

Table 5-14 Fee_30_SmallSector_JobEndNotification

5.4.2 Fee_30_SmallSector_JobErrorNotification

Prototype	
`void` **`Fee_30_SmallSector_JobErrorNotification`** `( void )`	
Parameter	
`--`	--
Return code	
`void`	--
Functional Description	
This routine shall be called by the underlying flash driver to report the failure of an asynchronous operation. **Note** This function is configurable at pre-compile time using the parameter `FeePollingMode.`	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is non-reentrant.	
Expected Caller Context	
> This routine might be called on interrupt level, depending on the calling function	

Table 5-15 Fee_30_SmallSector_JobErrorNotification

5.5 Configurable Interfaces

API	Description
Function `Fee_30_SmallSector_GetVersionInfo`	This function can be enabled/disabled by the configuration switch ‘Version Information API’

Table 5-16 Configurable Interfaces

6 Configuration

In the FEE the attributes can be configured according with the following methods:

> DaVinci Configurator 5, domain “Memory” (AUTOSAR 4 packages only). Parameters are explained within the tool. Parameters described in this chapter might not directly correspond to parameters visible in Configurator 5’s GUI.

> DaVinci Configurator 4 (AUTOSAR 3 packages only; for a detailed description see this chapter)

6.1 Configuration Variants

The FEE supports the configuration variants

> `VARIANT-PRE-COMPILE`

The configuration classes of the FEE parameters depend on the supported configuration variants. For their definitions please see the

> Fee_30_SmallSector_bswmd_Asr3.2.1.arxml (AUTOSAR 3 use-case)

> Fee_30_SmallSector_bswmd_Asr4.0.3.arxml (AUTOSAR 4 use-case)

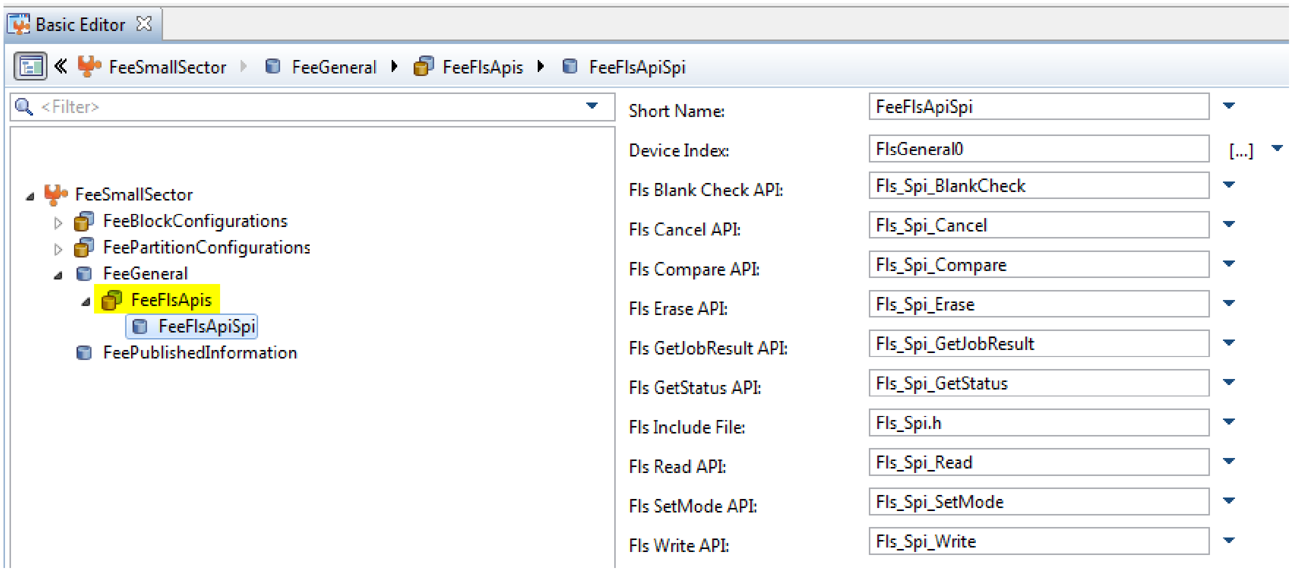
6.2 Configuration with DaVinci Configurator

Configuration parameters are well described within the tool, thus this section should point out the key elements of configuration.

6.2.1 Configuring Flash API Services

By default SmallSectorFee assumes the Flash’s API services declared like Fls_<Operation> (i.e. Fls_Read, Fls_Write, Fls_Erase, etc.).

If declaration of Flash’s API services differs from this default, e.g. Fls_<VendorInfix>_Read an according FeeGeneral/FeeFlsApi container shall be instantiated and the service names shall be entered manually as shown in Figure 6-1 Configuring FeeFlsApi container.



Short Name:	FeeFlsApiSpi
Device Index:	FlsGeneral0
Fls Blank Check API:	Fls_Spi_BlankCheck
Fls Cancel API:	Fls_Spi_Cancel
Fls Compare API:	Fls_Spi_Compare
Fls Erase API:	Fls_Spi_Erase
Fls GetJobResult API:	Fls_Spi_GetJobResult
Fls GetStatus API:	Fls_Spi_GetStatus
Fls Include File:	Fls_Spi.h
Fls Read API:	Fls_Spi_Read
Fls SetMode API:	Fls_Spi_SetMode
Fls Write API:	Fls_Spi_Write

Figure 6-1 Configuring FeeFlsApi container

Figure 6-1 Configuring FeeFlsApi containerdepicts an example with Flash API services (here: Fls_Spi_<operation>) differing from default. The parameter 'Device Index' shall be set to FlsGeneral container of the corresponding FLS module in configuration.

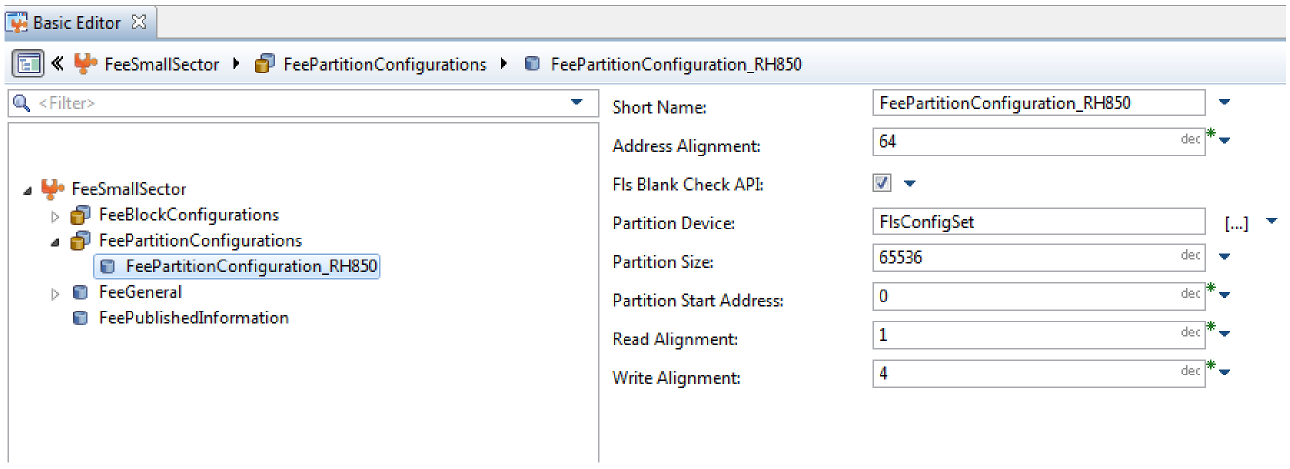
If the FLS driver does not provide a BlankCheck-API, the content of this configuration parameter is not relevant and will be ignored. Nevertheless it shouldn't be left empty.

Note
When using the RH850 with the Renesas FLS Driver there is also the possibility to use the command Fls_ReadImmediate, which is faster than the standard Fls_Read.
In this case the BlankCheck-API has to be enabled.

6.2.2 Partition Configuration

For each partition three different alignment parameters can be configured. This is due to the fact that theoretically each partition could reference a separate FLS module with different hardware properties.

> AddressAlignment shall be configured to the same size of referenced FLS module's physical sector
> WriteAlignment shall be configured to the same size of referenced FLS module's physical page
> ReadAlignment shall be configured as small as possible in order to avoid reading overhead. For example RH850's RV40F data flash allows 1 byte read accesses.


Basic Editor
FeeSmallSector
FeePartitionConfigurations
FeePartitionConfiguration_RH850
<Filter>
FeeSmallSector
FeeBlockConfigurations
FeePartitionConfigurations
FeePartitionConfiguration_RH850
FeeGeneral
FeePublishedInformation

Short Name:	FeePartitionConfiguration_RH850
Address Alignment:	64
Fls Blank Check API:	✓
Partition Device:	FlsConfigSet
Partition Size:	65536
Partition Start Address:	0
Read Alignment:	1
Write Alignment:	4

Figure 6-2 SmallSectorFee Partition Configuration example (RH850)

The connection between a FeePartition and a FLS module shall be established via 'Partition Device' parameter, which holds the reference to FLS' FlsConfigSet container.

Both 'Partition Start Address' and 'Partition Size' needs to be aligned to the partition's AddressAlignment.

In case the referenced FLS driver provides a Fls_BlankCheck API, it's recommended to enable the configuration parameter 'Fls Blank Check API', so that FEE can make use of this API service.

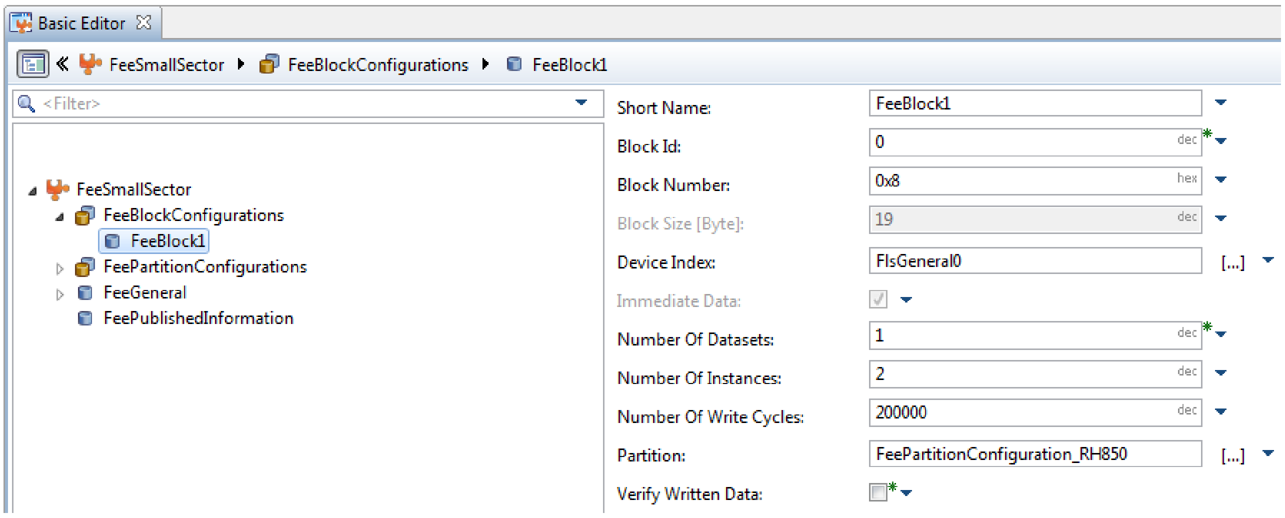
Note

If Renesas RH850 is used with internal data flash RV40F, it is strongly recommended to enable the 'Fls Blank Check API'.

6.2.3 Block Configuration

In BlockConfiguration the number of instances is mentionable. It's calculated automatically using the parameters 'Number Of Write Cycles' and FlsSpecifiedEraseCycles in FlsPublishedInformation. FEE uses a walking block mechanism to spread write jobs over several instances in order to prevent the Flash cells from getting overstressed.

The 'Number Of Instances' is equal to 'Number Of Write Cycles' divided by FlsSpecifiedEraseCycles rounded up to the next integer value.



Short Name:	FeeBlock1
Block Id:	0
Block Number:	0x8
Block Size [Byte]:	19
Device Index:	FlsGeneral0
Immediate Data:	✓
Number Of Datasets:	1
Number Of Instances:	2
Number Of Write Cycles:	200000
Partition:	FeePartitionConfiguration_RH850
Verify Written Data:	

Figure 6-3 Block Configuration

Additionally, in BlockConfiguration the number of Dataset is mentionable. This parameter is related to the different types of NV blocks (specified in [5]). These types are realized by the parameter 'Number Of Datasets':

> Native Block as standard block type means the data is stored once in the NV area (Number Of Datasets = 1).

> Redundant Blocks means the data is stored twice in the NV area, which increases the availability in case of errors (Number Of Datasets = 2).

> Dataset Blocks means the data is stored like an array in the NV area, the maximum value depends on parameter NvMDatasetSelectionBits (1 < Number Of Datasets < n).

6.3 Overhead Calculation

Since addressing scheme of SmallSectorFEE is statically, the overhead for each block in configuration can be calculated. Overhead can therefore be split into two groups:

> Fixed overhead due to management information of a block

> Overhead due to page and physical sector alignments

The user of SmallSectorFEE can only influence alignment-related overhead, because overhead for management information is unavoidable.

A FEE block consists of at least one FEE dataset. A FEE dataset consists of at least one FEE instance. In the following the algorithms are introduced to calculate the size of these entities depending on configured data length and expected number of writes.

Explanation of terms:

Name	Description
FeeMgmtSize	1 Byte (Fixed value in implementation)
FeeInstanceSize	Size of one instance for a configured FEE block
FeeNrInstances	Configured number of instances for FEE block. Depends on expected number of writes of FEE block and flash device's specified erase cycles
FeeNrDatasets	Configured number of datasets for FEE block.
DataLength	Configured size of user data for FEE block
FlsPageSize	Size of smallest writeable unit of flash device. Should be identical to FEE's write alignment.
FlsSectorSize	Size of smallest erasable unit of flash device Should be identical to FEE's address alignment.
align(value, alignment)	If 'value' is already aligned to 'alignment', 'value' is returned. Otherwise the next greater value aligned to 'alignment' is returned.

Table 6-1 Explanation of terms used in algorithms

1. Calculation of size for one instance of FEE block:

FeeInstanceSize = *align*(2 * FlsPageSize + FeeMgmtSize + DataLength, FlsPageSize)

2. Calculation of size for one dataset of FEE block:

FeeDatasetSize = *align* (2 * FlsPageSize + FeeInstanceSize * FeeNrInstances, FlsSectorSize)

3. Calculation of size for one FEE block:

FeeBlockSize = FeeDatasetSize * FeeNrDatasets

It becomes clear that overhead due to alignment appears two times. FEE instances are aligned to flash device's page boundaries and FEE datasets are aligned to flash device's sector boundaries. By adjusting the parameters DataLength and FeeNrInstances it is possible to reduce alignment-related overhead. Both parameters should be chosen in a way that the number of padding bytes is minimalized to fill the entity up to the next alignment boundary. It should be considered to reduce the size of DataLength in order to decrease the total size of FeeDataSize.

7 Requirements and Recommendations

This section shall point out requirements the user system needs to comply with when using this FEE module. Additionally, recommendations are made for use-case of this module and its configuration with regard to limitations of FEE.

7.1 Requirements for the User System

The following requirements are focused on FEE's underlying layers: Flash driver and Flash memory hardware.

7.1.1 General Requirements

> Only addressed flash pages and sectors shall be affected during a job

7.1.2 Write-Related

> Only if intended data was written successfully and persistently to flash memory, FLS driver shall return with MEMIF_JOB_OK

> Only erased flash cells shall be writeable

> Successfully written flash cells shall not alter their values over time

> Write aborts (e.g. due to reset) shall only affect the flash page which is currently being written

7.1.3 Read/Compare-Related

> Only if intended data was read/compared successfully from flash memory, FLS driver shall return with MEMIF_JOB_OK

> If FLS driver does not support a BlankCheck API, erased flash cells shall contain FLS' erased value

7.1.4 Erase-Related

> Only if intended flash area was erased successfully and persistently, FLS driver shall return with MEMIF_JOB_OK

> Erased flash cells shall remain erased over time until a write job is performed

> Erase aborts (e.g. due to reset) shall only affect the flash sector which is currently being erased

7.1.5 BlankCheck-Related

> Only if all flash cells in intended flash area are erased, FLS driver shall return with MEMIF_JOB_OK

7.2 Recommendations

The following items shall serve as basis for general considerations regarding use-case and FEE configuration.

> The SmallSectorFEE shall be used with flash devices with reasonable small physical sectors. This means that sector size shall not exceed the size of single configured

blocks by far. SmallSectorFEE assigns every configured block to separate flash sectors, thus overhead is kept small if block size is about the size of a flash sector or a multiple of it.

> It's recommended to use this FEE module with internal data flash RV40F of Renesas' RH850 platform because of its 64 byte physical flash sectors.
> If erase state of flash cells can't be determined via Read service but BlankCheck service, using the FLS' BlankCheck service shall be enabled in FeePartitionConfiguration.
> 'AddressAlignment' parameter in FeePartitionConfiguration shall be identical to referenced FlsSector's sector size.
> 'WriteAlignment' parameter in FeePartitionConfiguration shall be identical to referenced FlsSector's page size.
> 'ReadAlignment' parameter in FeePartitionConfiguration shall be configured as small as FLS driver and flash hardware allows in order to keep reading overhead small.
> In FeeBlockConfiguration the number of instances for a block is calculated by 'Number Of Write Cycles' and FLS module's 'Specified Erase Cycles'. If the expected 'Number Of Write Cycles' for a block is nearly as large as a multiple of FLS module's 'Specified Erase Cycles' it should be considered to increase the expected 'Number Of Write Cycles' so that one additional instance is allocated for this block. This way enough reserve is established to prevent the flash area for this block from getting overstressed.

8 Abbreviations

Abbreviation	Description
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSW	Basis Software
DEM	Diagnostic Event Manager
DET	Development Error Tracer
EAD	Embedded Architecture Designer
ECU	Electronic Control Unit
EEPROM	Electrically Erasable Programmable Read-Only Memory
FEE	Flash EEPROM Emulation
FLS	Flash module
HIS	Hersteller Initiative Software
ISR	Interrupt Service Routine
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
NVM	Non-Volatile RAM Manager
RAM	Random Access Memory
SIP	Software Integration Package
SWS	Software Specification

Table 8-1 Abbreviations

9 Contact

Visit our website for more information on

> News
> Products
> Demo software
> Support
> Training data
> Addresses

www.vector.com