Flash EEPROM Emulation user guide

TRAVEO™ T2G family

About this document

Scope and purpose

This guide describes the architecture, configuration and usage of Flash EEPROM Emulation (FEE). This guide also explains the functionality of the driver and provides references to the module's API.

The installation, build process, and general information about the use of the EB tresos Studio are not within the scope of this document. See the *EB tresos Studio for ACG8 user's guide* [9] for detailed information of these topics.

Intended audience

This document is intended for anyone using the Flash EEPROM Emulation software.

Document structure

The 1 General overview chapter gives a brief introduction to FEE, explains the embedding in the AUTOSAR environment, and describes the supported hardware and development environment.

The 2 Using the Flash EEPROM Emulation chapter details the steps required to use the ADC driver in your application.

The 3 Structure and dependencies chapter describes the file structure and the dependencies for the ADC driver.

The 4 EB tresos Studio configuration interface chapter describes the driver's configuration with the EB tresos Studio software.

The 5 Functional description chapter gives a functional description of all services offered by the ADC driver.

The 6 Hardware resources chapter gives a description of all hardware resources used.

The Appendix A and Appendix B chapters provides a complete API reference and access register table.

Abbreviations and definitions

Table 1 Abbreviations

Abbreviations	Description
API	Application Programming Interface
ASIL	Automotive Safety Integrity Level
AUTOSAR	Automotive Open System Architecture
Basic Software	Standardized part of software which does not fulfill a vehicle functional job.
BSW	Basic Software. Standardized part of software which does not fulfill a vehicle functional job.
DET	Default Error Tracer
DMA	Direct Memory Access
EB tresos Studio	Elektrobit Automotive configuration framework
ECC	Error Checking Code
ECU	Engine Control Unit
EEPROM	Electrically erasable programmable ROM
FEE	Flash EEPROM Emulation
FLS	Flash driver module
Flash sector	A flash sector is the smallest amount of flash memory that can be erased in one pass. The size of the flash sector depends upon the flash technology and is therefore, hardware dependent.
Flash page	A flash page is the smallest amount of flash memory that can be programmed in one pass. The size of the flash page depends upon the flash technology and is therefore, hardware dependent.
GCE	Generic Configuration Editor
GP RAM	General purpose RAM
MPU	Memory Protection Unit
µC	Microcontroller
OS	Operating System
SchM	BSW Scheduler
SW	Software

Table of contents

1 General overview

1.1 Introduction to the Flash EEPROM Emulation

The Flash EEPROM Emulation abstract from the device-specific addressing scheme and segmentation provides the upper layers with a virtual addressing scheme and segmentation as well as a "virtually" unlimited number of erase cycles.

1.1.1 Features of FEE

- Automatic recycling
  If the sector is full, it is recycled automatically. FEE moves valid data to a new sector and creates free space.
- Robust against power down
  If power goes down during writing (or erasing), data on the sector may be undefined. FEE checks the data state (marker) of the sector during initialization. If indefinite data exists (due to power off, reset, and so on), data is automatically recovered and only correct data is moved to a new sector. TRAVEO™ T2G family FEE's robustness against power down is based on a protocol with markers in flash memory that identify the last valid entry.
- Availability of multiple configurations
  FEE can manage sectors separately by two configurations. For example, you can manage boot data and Log data in separate sectors.

1.2 User profile

This guide is intended for users with a basic knowledge of the following domains:

- Flash memory
- Embedded systems
- The AUTOSAR terminology
- The C programming language

1.3 Embedding in the AUTOSAR environment

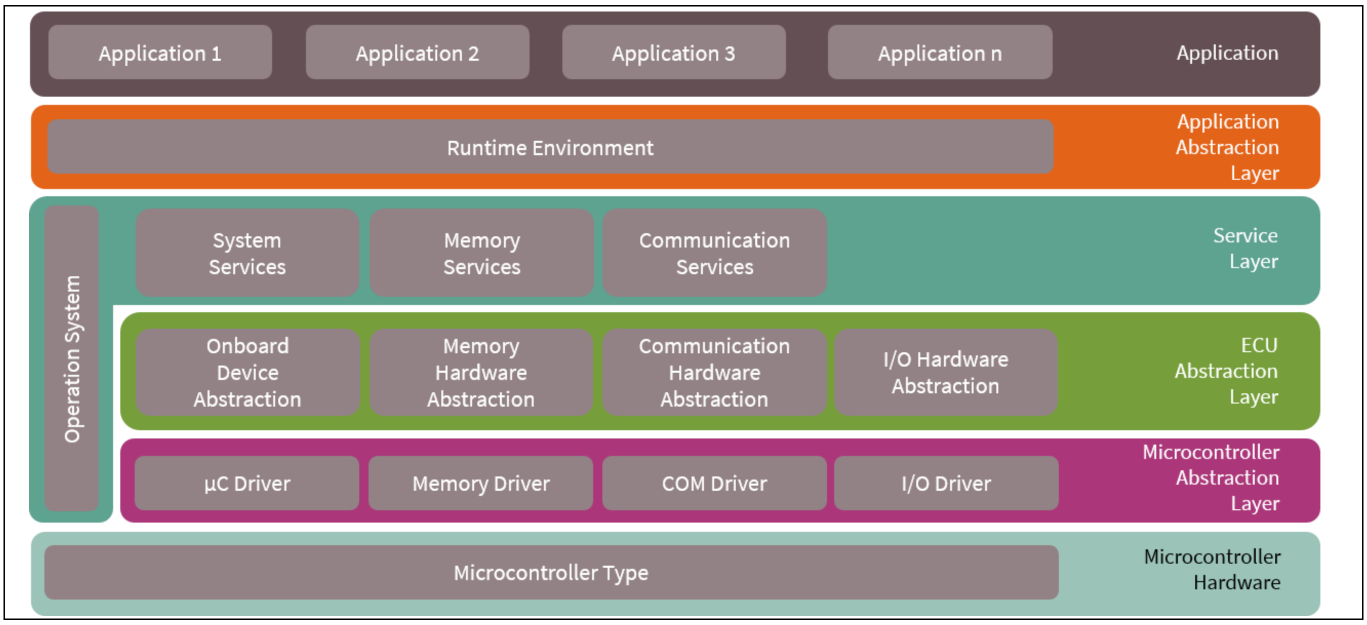


Figure 1 Overview of AUTOSAR software layers

Figure 1 depicts the layered AUTOSAR software architecture. FEE (Figure 2) is one of the memory hardware abstraction layer.

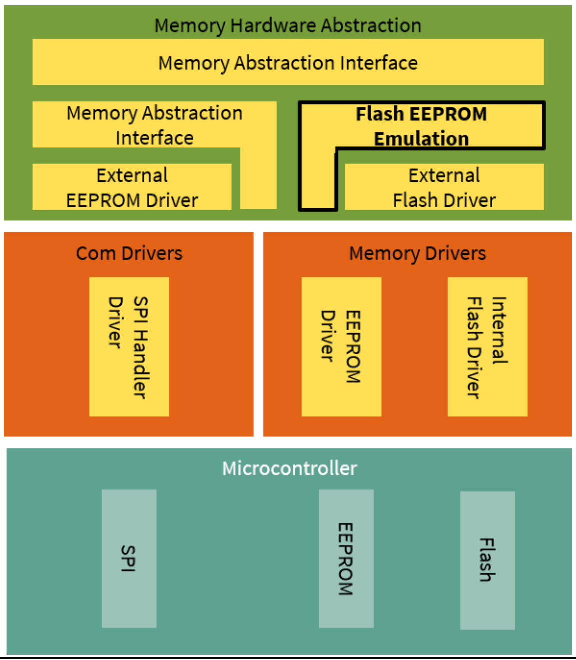


Figure 2 Flash EEPROM Emulation

1.4 Supported hardware

This version of the Flash EERPOM Emulation supports the TRAVEO™ T2G microcontroller. No further special external hardware devices are required. See the resource module user guide for supported subderivative.

1.5 Development environment

The development environment corresponds to AUTOSAR release 4.2.2. The modules Base, Make, Resource, and FLS are needed for the proper functionality of FEE.

1.6 Character set and encoding

All source code files of FEE are restricted to the ASCII character set. The files are encoded in UTF-8 format, with only the 7-bit subset (values 0x00 # 0x7F) being used.

2 Using the Flash EEPROM Emulation

This chapter describes the necessary steps to incorporate FEE into your application.

2.1 Installation and prerequisites

Note: *Before continuing with this chapter, see the EB tresos Studio for ACG8 user's guide [9] first. This provides required basic information on the installation procedure of EB tresos AUTOSAR components and the use of the EB tresos Studio and the EB tresos AUTOSAR build environment. In particular, you will find an explanation on how to set up and integrate your own application within the EB tresos AUTOSAR build environment.*

The installation of FEE complies with the general installation procedure for EB tresos AUTOSAR components given in the *EB tresos Studio for ACG8 user's guide* [9]. If the driver has been successfully installed, the driver will appear in the module list of the EB tresos Studio (see 4 EB tresos Studio configuration interface).

This guide assumes that the project is properly set up and is using the application template as described in the *EB tresos Studio for ACG8 user's guide [9]*. This template provides the necessary folder structure, project and makefiles needed to configure and compile an application within the build environment. You must be familiar with the usage of the command line shell.

2.2 Configuring the FEE

This section provides an overview of the configuration structure defined by AUTOSAR to use the FEE.

The following containers are used to configure common behavior.

- `FeeBlockConfiguration`: Configuration of block-specific parameters for the FEE.
- `FeeBlockConfigurationEx`: Extra configuration of block specific parameters for the FEE.
- `FeeGeneral`: Container for general parameters. These parameters are not specific to a block.
- `FeePublishedInfomation`: Additional published parameters not covered by the `CommonPublishedInfomation` container.

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

See 4.1 General configuration and 4.2 Vendor-specific configuration for the details of a configuration to be set up.

Note: *FEE can manage sectors separately by two configurations (`FeeBlockConfiguration` and `FeeBlockConfigurationEx`) each configuration has the number of sectors and block information used. In user configuration, each block is configured to `FeeBlockConfiguration` or `FeeBlockConfigurationEx`. Blocks in `FeeBlockConfiguration` will never shift to `FeeBlockConfigurationEx`. Figure 3 depicts the data structure.*

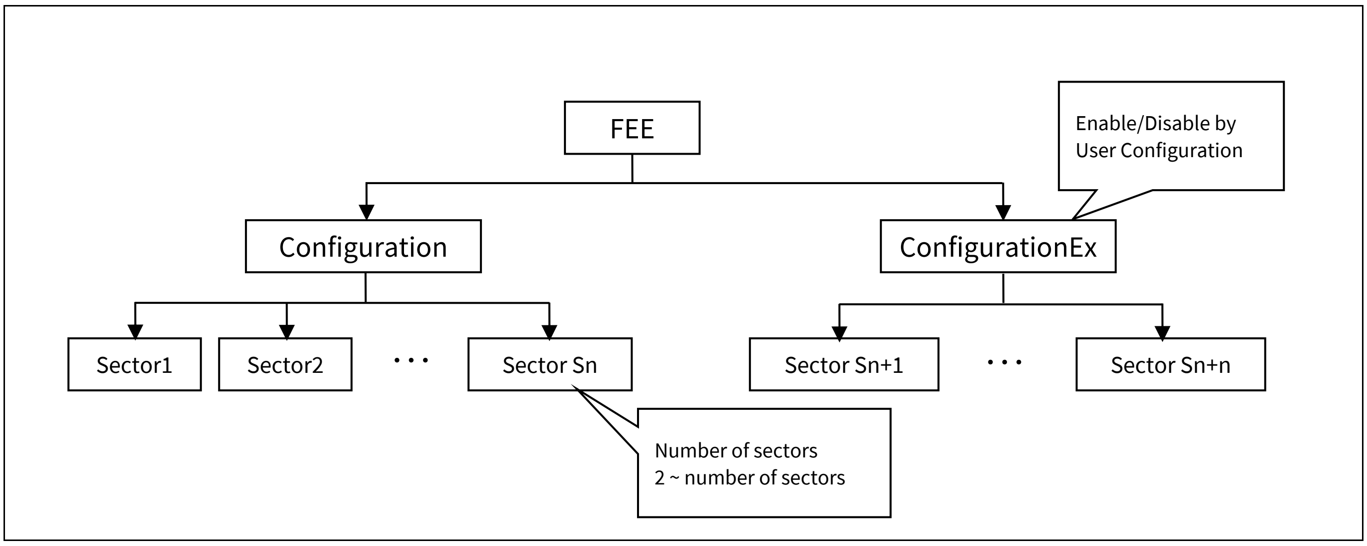


Figure 3 Relationship between FeeBlockConfiguration and FeeBlockConfigurationEx

2.2.1 Architecture specifics

- `FeeErrorCalloutFunction`: Specifies an error callout handler, which is called when any errors are detected during runtime.
- `FeeIncludeFile`: Iincludes some definitions (declaration for error callout handler).

The name of job end notification is fixed by the configuration parameters `FeeNvmJobEndNotification`.

The name of job error notification is fixed by the configuration parameters `FeeNvmJobErrorNotification`.

2.3 Adapting an application

To use FEE in your application, include the FEE header file by adding the following line of code in your source file:

```
#include "Fee.h" /* Fee Header */
```

This publishes all needed function/data prototypes and symbolic names of the configuration to the application. You must also implement the error callout function for ASIL safety extension.

Declare the error callout function in the file specified by the `FeeIncludeFile parameter` and implement the error callout function in your application (see Error Callout API in Required callback functions).

Then, add the source files *Fee.c* and *Fee_Pub.c* to your project.

In the next step, FEE needs to be initialized and configured. The FEE module will automatically be enabled if an appropriate parameter configuration of the FEE module is available in the application.

The FEE module initialization can be done with the following function call and parameter:

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

All other API calls might be used after successful initialization of the FEE whenever necessary.

2.4 Starting the build process

Do the following to build your application:

Note: *For a clean build, use the build command with target `clean_all`. before (`make clean_all`). On the command shell, type the following command to generate the necessary configuration-dependent files. See 3.2 Configuration files.*

```
> make generate
```

1. Type the following command to resolve required file dependencies:

   ```
   > make depend
   ```

2. Type the following command to compile and link the application

   ```
   > make (optional target: all)
   ```

The application is now built. All files are compiled and linked to a binary file which can be downloaded to the target hardware.

2.5 Memory mapping

Fee_MemMap.h in the *$(TRESOS_BASE)/plugins/MemMap_TS_T40D13M0I0R0/include* directory is a sample. This sample file is replaced by the file generated by the MEMMAP module. Input to the MEMMAP module is generated as *Fee_Bswmd.arxml* in the *$(PROJECT_ROOT)/output/generated/swcd* directory of your project folder.

2.5.1 Memory allocation keyword

- FEE_START_SEC_CODE_ASIL_B / FEE_STOP_SEC_CODE_ASIL_B
  The memory section type is CODE. All executable code is allocated in this section.
- FEE_START_SEC_CONST_ASIL_B_UNSPECIFIED / FEE_STOP_SEC_CONST_ASIL_B_UNSPECIFIED
  The memory section type is CONST. All constants are allocated in this section.
- FEE_START_SEC_VAR_INIT_ASIL_B_UNSPECIFIED / FEE_ STOP_SEC_VAR_INIT_ASIL_B_UNSPECIFIED
  The memory section type is VAR. All initialized variables are allocated in this section.
- FEE_START_SEC_VAR_NO_INIT_ASIL_B_UNSPECIFIED /
  FEE_STOP_SEC_VAR_NO_INIT_ASIL_B_UNSPECIFIED
  The memory section type is VAR. All uninitialized variables are allocated in this section.

3 Structure and dependencies

FEE consists of static, configuration, and generated files.

3.1 Static files

Static files of the FEE module are in the directory *$(TRESOS_BASE)/plugins/Fee_TS_**. These files contain the functionality of the driver, which does not depend on the current configuration.

All necessary source files will automatically be compiled and linked during the build process and all include paths will be set.

3.2 Configuration files

The configuration of the FEE module is done using the EB tresos Studio. When saving a project, the configuration description is written in the file *Fee.epc*, which is in *$(PROJECT_ROOT)/config* of your project folder. This file serves as input for the generation of the configuration dependent source and header files during the build process.

3.3 Generated files

During the build process, the following files are generated based on the current configuration description. These files are in the folder *output/generated* of your project folder.

- *include/Fee_Cfg.h* contains the configuration declarations for FEE that are target independent.

Note: *Generated source files must not be added to your application make file. They will be compiled and linked automatically during the build process.*

3.4 Dependencies

Figure 4 depicts how the Flash driver is embedded in the memory stack.

Note: *To use the Flash EEPROM Emulation, the Flash driver (see Specification of Flash driver [2]), the memory abstraction interface (see requirements of memory hardware abstraction layer [8]) and the BSW scheduler Module (see specification of RTE [5]) have to be enabled and configured. optionally the DET (see specification ofdDefault error tracer [4]) can be enabled and configured, too.*

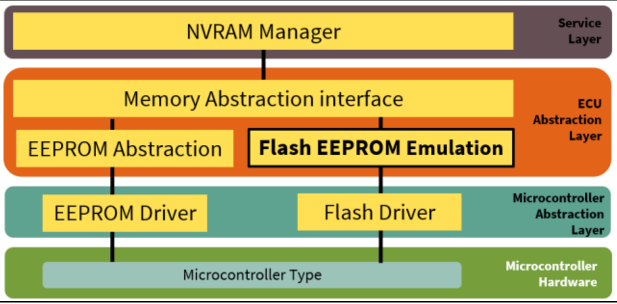


Figure 4 Relationship between the Flash EEPROM Emulation and other AUTOSAR modules

3.4.1 Memory abstraction interface

The memory abstraction interface is part of the ECU abstraction layer which is located above FEE. The memory abstraction interface is the only module that calls the FEE module functions and provides virtual memory mapping.

3.4.2 Flash driver

The Flash driver is part of the microcontroller abstraction layer which is located below FEE and provides services for reading, writing, and erasing flash memory and a configuration interface for setting/resetting the write/erase protection if supported by the underlying hardware.

3.4.3 DET

The DET is optional and handles all default errors.

3.4.4 BSW scheduler

The BSW scheduler calls the main function and handles the critical sections that are used within the FEE module.

3.4.5 Error callout handler

The error callout handler is called on every error that is detected, regardless of whether default error detection is enabled or disabled. The error callout handler is an ASIL safety extension that is not specified by AUTOSAR. It is configured via configuration `FeeErrorCalloutFunction` parameter.

4 EB tresos Studio configuration interface

The GUI is not part of the current delivery. For further information, see *EB tresos Studio for ACG8 user's guide* [9].

Note: *The ECU parameter description of the Elektrobit automotive FEE module basically corresponds to the description defined by the AUTOSAR specification of the specification of ECU configuration [6]. However, as there are some vendor-specific extensions, use the ECU parameter description file that is delivered with the FEE module, which is in $(TRESOS_BASE)/plugins/Fee_TS_*/config/Fee.xdm.*

4.1 General configuration

The FEE configuration, including different parameters and their meaning, is described in the *AUTOSAR specification of the Flash EEPROM Emulation* [3] and the *AUTOSAR specification of the ECU configuration* [6].

4.2 Vendor-specific configuration

This section summarizes the differences between the configuration given in the AUTOSAR documentations and the configuration necessary for this module.

4.2.1 Parameter constraints

The range of several parameters of the general FEE configuration was reduced to the meaningful value of FLS (hardware specific values for the TRAVEO™ T2G microcontroller). These parameters are listed here. The parameters are preconfigured by using default values for the selected derivate (when changing the derivate, a manual update is possible by clicking **Calc** in EB tresos Studio). If a parameter is not used by the module or if the parameter is not configurable, the field cannot be edited.

4.2.1.1 Container FeeGeneral

FeeDevErrorDetect

Range

TRUE, FALSE

Annotation

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

- TRUE: Default error detection is enabled.
- FALSE: Default error detection is disabled.

FeeNvmJobEndNotification

Range

TRUE (function name is fixed to `NvM_JobEndNotification`.)

Annotation

Pre-processor switch to enable and disable callback function.

- `JobEndNotification` – This switch is fixed to TRUE.

FeeNvmJobErrorNotification

Range

TRUE (function name is fixed to `NvM_JobErrorNotification`.)

Annotation

Pre-processor switch to enable and disable callback function.

- `JobErrorNotification` - This switch is fixed to TRUE.

FeePollingMode

Range

TRUE

Annotation

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

This module uses only the polling mode. Therefore, this switch is fixed to TRUE.

FeeSetModeSupported

Range

TRUE, FALSE

Annotation

Pre-processor switch to enable and disable the `SetMode` functionality of this module.

- TRUE: `SetMode` functionality is enabled.
- FALSE: `SetMode` functionality is disabled.

FeeVersionInfoApi

Range

TRUE, FALSE

Annotation

Pre-processor switch to enable and disable the API to read the version information of modules.

- TRUE: Version information API is enabled.
- FALSE: Version information API is disabled.

FeeVirtualPageSize

Range

4

Annotation

Size of logical blocks aligned in this module specified in bytes. The value is fixed to 4.

FeeMainFunctionPeriod

Range

0.01 to 1000

Annotation

The call cycle of the main function of this module. This parameter is given in milliseconds.

Note: *Set values greater than 1.*

FeeDelayRecycleOperation

Range

TRUE, FALSE

Annotation

Pre-processor switch to enable or disable the delay for the recycle operation in FEE initialization.

The current FEE can be read after the initialization process. If a part of the data is corrupted, a recycle will occur to fix the data block in the initialization process. During the recycle operation, sector erasing and data moving are performed; normally these processes take longer to complete.

- TRUE: Recycle is not performed during initialization. Therefore, data read will start shortly. However, recycle occurs in the write operation (Fee_MainFunction after Fee_Write). Thus, the first write operation may take longer.
- FALSE: Recycle is performed during initialization. (default)

4.2.1.2 Container FeeBlockConfiguration

FeeBlockNumber

Range

1 to 65534

Annotation

Block identifier.

FeeBlockSize

Range

For the `Config` block: 1 - `FeeBlockMaxsize`

For the `ConfigEx` block: 1 - `FeeBlockMaxsizeEx`

Annotation

Size of logical block in bytes.

FeeImmediateData

Range

TRUE, FALSE

Annotation

Marker for high priority data.

- Handling of “immediate” data

  FEE module ensures that it can write “immediate” blocks without the need to erase the corresponding memory area.

  In other words, the data of the block defined as immediate can be reserved and written without recycling.

  However, when rewriting the immediate block, it is necessary to invalidate the target block using the `Fee_EraseImmediateBlock()` API.

  `Fee_EraseImmediateBlock()` invalidates the block and also calculates the free space in the sector to determine whether the target block is writable.

  If writing is not possible, recycling will occur.
- Handling of normal "not immediate" data

  Calculates the free space in the sector when writing.

  If there is a free space, data is written to the same sector. If not, recycling will occur.

  Also, when rewriting, it is not necessary to invalidate the target block using the `Fee_EraseImmediateBlock()` API.

  You can rewrite the target block with Fee_Write. Therefore, recycling may occur when writing.

TRUE: Block contains immediate data.

FALSE: Block does not contain immediate data.

FeeNumberOfWriteCycles

Range

0 to 0xFFFFFFFF

Annotation

Expected number of erase/write cycles for each logical block.

(`FeeNumberOfWriteCycles` is unused in the FEE)

FeeDeviceIndex

Range

0

Annotation

Identifier of the flash device where the block data is stored. This module uses only one device. Therefore, this index is fixed to 0.

FeeSelectConfigEx

Range

TRUE, FALSE

Annotation

Marker to indicate the location of the block.

- TRUE: Located on `ConfigEx`.
- FALSE: Located on `Config`.

4.2.1.3 Container FeePublishedInformation

FeeBlockOverhead

Range

20

Annotation

The size of block management area, which is fixed to 20.

FeePageOverhead

Range

16

Annotation

The size of page management area, which is fixed to 16.

4.2.2 Vendor and module specific parameters

4.2.2.1 Container FeeGeneral

FeeInitiallyEraseEmptySectors

Range

TRUE, FALSE

Annotation

Pre-processor switch to enable and disable the function to erase all sectors for brand new chip.

- TRUE: All sectors for brand new chip are erased when `Fee_init()` is executed.
- FALSE: The sectors for brand new chip are not erased when `Fee_init()` is executed.

Note: *If you are using this value, erase the work flash area of FEE.*

FeeUnmatchedBlockCheck

Range

TRUE, FALSE

Annotation

Consistency check is done in Fee_Init.

Pre-processor switch to enable and disable the consistency check of a block.

- TRUE: The consistency check of block ID and data length is enabled.

Note: *When installing FEE in the product version, "TRUE" should be selected.*

- FALSE: The consistency check of block ID and data length is disabled.

When FEE found the "undefined block ID" or "block ID with unmatched data length" on work flash, FEE performs the following processing.

- If BlockID not defined in config exists on work flash.
  - Delete the target BlockID on work flash.
- If data length is different between config and work flash
  - [data length in config > data length in work flash]:
    - Extend the data length of work flash according to config.
    - Copy the original data to the above area.
    - The FF is set in the increased area.
  - [data length in config < data length in work flash]:
    - Delete the target BlockID on work flash.

FeeSetCycleModeApi

Range

TRUE, FALSE

Annotation

Pre-processor switch to enable and disable the API to switch the cycle mode.

- TRUE: The API to set the cycle mode is enabled.

Note: *While using the Setting the call cycle mode function, this value should be "TRUE".*

- FALSE: The API to set the cycle mode is disabled.

FeeClearApi

Range

TRUE, FALSE

Annotation

Pre-processor switch to enable and disable the API to erase all sectors.

- TRUE: The API to erase all sectors is enabled.
- FALSE: The API to erase all sectors is disabled.

FeeCleanupAndEraseApi

Range

TRUE, FALSE

Annotation

Pre-processor switch to enable and disable the API to recycle sector.

- TRUE: Recycle API is enabled.
- FALSE: Recycle API is disabled.

FeeGetRemainingPagesApi

Range

TRUE, FALSE

Annotation

Pre-processor switch to enable and disable the API to read out the remaining number of pages. For more detail, see 7.3.12 Fee_GetRemainingPages/Fee_GetRemainingPagesEx.

- TRUE: The API to read out the remaining number of pages is enabled.
- FALSE: The API to read out the remaining number of pages API disabled.

FeeWorkFlashRelativeEndAddress

Range

If Extra configuration is used:

0x4000 – final address of work flash

If Extra configuration is not used:

0x2000 – final address of work flash

Annotation

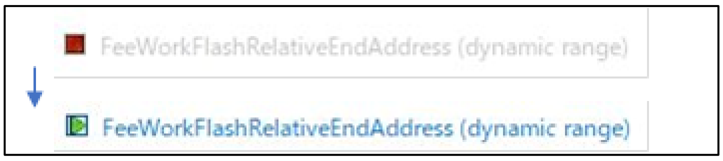
Specify the final address of the work flash used by FEE.

Default value: Final maximum address of work flash.

When this parameter is not specified, this address is calculated automatically using FLS memory mapping.

However, in this case, the memory map of FLS setting must be the same as FEE target area.

If the memory map is the difference between FLS and FEE, then in the tresos GUI, click the toggle button of the FEE setting and specify the address. The address is specified as multiples of 0x1000. (For example: 0x2000, 0x3000, 0x4000, …)


FeeWorkFlashRelativeEndAddress (dynamic range)
FeeWorkFlashRelativeEndAddress (dynamic range)

FeeBlockMaxSize

Range

1 to 3072

Annotation

Maximum size of logical blocks specified in bytes.

FeeBlockMaxSize is vendor-specific parameter which is configured.

FeeSectorStartAddress

Range

0 – [depends on work flash size]

Annotation

Start address of sectors defined by `Config`.

The address is specified as multiples of 0x1000. (For example: 0x0000, 0x1000, 0x2000, …)

Note: *The following addresses are the areas used by FEE.*
`FeeSectorStartAddress - FeeWorkFlashRelativeEndAddress`

The number of sectors is automatically calculated by FEE based the total block length.

FeeDefaultCycleMode

Range

MEMIF_MODE_SLOW, MEMIF_MODE_FAST

Annotation

MEMIF_MODE_SLOW: Wait cycle time of MainFunction

MEMIF_MODE_FAST: No wait cycle time for MainFunction

For details on `FeeDefaultCycleMode`, see 5.1.15 Timeout monitoring.

ConfigIfUseThresholdPageSize

Range

TRUE, FALSE

Annotation

- TRUE: If recycle occurs during “immediate block” writing in Config area, “immediate block” is written to the current sector as “Theshold area” before recycle and then recycle is started.
  The following notes apply if this configuration is set to "TRUE".
  - The size of one sector should be set manually using `FeeNormalPageSize`.
  - The reserve area size for writing “immediate block” to the current sector before recycle must be manually set using `FeeThresholdPageSize`.
  - When an “immediate block” with the same number is written multiple times, it should be invalidated using the `Fee_EraseImmediateBlock()` API, but this invalidation process is no longer necessary. (Note that it is possible to write the same BlockID multiple times, and only the last written BlockID is valid).
- FALSE: In conventional function, if recycle occurs when data is written, then the data is written to a new sector.

FeeNormalPageSize

Range

0x1000 – [depends on work flash size]

Annotation

Specify one sector size of `Config` area.

The one sector size is specified as multiples of 0x1000. (For example: 0x1000, 0x2000, …)

This configuration must be specified if `ConfigIfUseThresholdPageSize` is set to "TRUE".

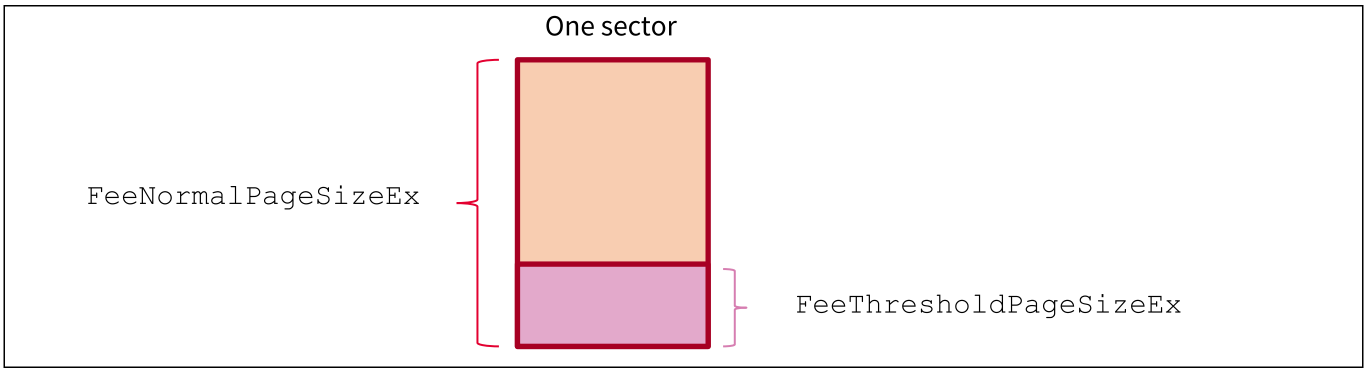
FeeThresholdPageSize

Range

[Max block datasize] - [depends on `FeeNormalPageSize`]

Annotation

`ThresholdPageSize` is the reserve space area that is used for “immediate block”. If recycle occurs due to writing “immediate block”, it will be written on threshold area first and then recycle will happen.



This configuration must be specified if `ConfigIfUseThresholdPageSize` is set to "TRUE".

[Max block datasize]:
A value greater than or equal to the maximum data size defined in all BlockIDs should be specified.
[depends on `FeeNormalPageSize`]
It should be specified to satisfy the following formula.

```
  (FeeNormalPageSize - FeeThresholdPageSize)
                                    >=    total block data size + total block id count × 16 + 40
```

4.2.2.2 Container ConfigEx

ConfigEx

Range

TRUE, FALSE

Annotation

- TRUE: Extra configuration is used.
- FALSE: Extra configuration is not used.

FeeSectorStartAddressEx

Range

0 – [depends on work flash size]

Annotation

Start address of sectors defined by `ConfigEx`.

The address is specified as multiples of 0x1000. (For example: 0x0000, 0x1000, 0x2000, …)

Note: The interval between FeeSectorStartAddress and `FeeSectorStartAddressEx` must be 0x2000 or more.

The number of sectors is automatically calculated by FEE based the total block length.

Example 1:

- FeeSectorStartAddress = 0x0000
- `FeeSectorStartAddressEx` = 0x4000
- `FeeWorkFlashRelativeEndAddress` = 0x10000

0x0000 to 0x3FFF: Config area

0x4000 to 0xFFFF: ConfigEx area

Example 2:

- FeeSectorStartAddress = 0x6000
- `FeeSectorStartAddressEx` = 0x0000
- `FeeWorkFlashRelativeEndAddress` = 0x8000

0x6000 to 0x7FFF: Config area

0x0000 to 0x5FFF: ConfigEx area

FeeBlockMaxsizeEx

Range

1 to 3072

Annotation

Maximum size of logical blocks specified in bytes.

`FeeBlockMaxSizeEx` is vendor-specific parameter which is configured.

ConfigExIfUseThresholdPageSize

Range

TRUE, FALSE

Annotation

- TRUE: If recycle occurs during "immediate block" writing in `ConfigEx` area, "immediate block" is written to the current sector as "Theshold area" before recycle and then recycle is started.
  The following notes apply if this configuration is set to "TRUE".
  - The size of one sector should be set manually using `FeeNormalPageSizeEx.`
  - The reserve area size for writing "immediate block" to the current sector before recycle must be manually set using `FeeThresholdPageSizeEx`.
  - When an "immediate block" with the same number is written multiple times, it should be invalidated using the `Fee_EraseImmediateBlock()` API, but this invalidation process is no longer necessary. (Note that it is possible to write the same BlockID multiple times, and only the last written BlockID is valid).
- FALSE: In conventional function, if recycle occurs when data is written, then the data is written to a new sector.

FeeNormalPageSizeEx

Range

0x1000 – [depends on work flash size]

Annotation

Specify one sector size of `ConfigEx` area.

The one sector size is specified as multiples of 0x1000. (For example: 0x1000, 0x2000, …)

This configuration must be specified if `ConfigExIfUseThresholdPageSize` is set to "TRUE".

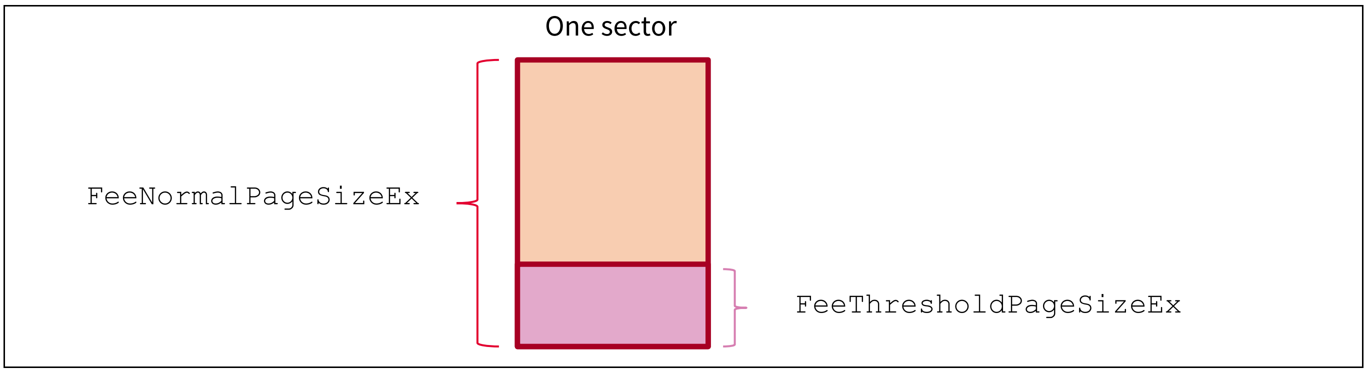
FeeThresholdPageSizeEx

Range

[Max block datasize] – [depends on `FeeNormalPageSizeEx`]

Annotation

`ThresholdPageSizeEx` is the reserve space area that is used for "immediate block". If recycle occurs due to writing "immediate block", it will be written on threshold area first and then recycle will happen.



This configuration must be specified if `ConfigExIfUseThresholdPageSize` is set to "TRUE".

[Max block datasize]:
A value greater than or equal to the maximum data size defined in all BlockIDs should be specified.

[depends on `FeeNormalPageSize`]
It should be specified to satisfy the following formula.
(`FeeNormalPageSizeEx - FeeThresholdPageSizeEx`)
>= total block data size + total block id count × 16 + 40

4.2.3 Other modules

4.2.3.1 FLS module

When using FEE, set the following for FLS. For details on FLS settings, see the FLS user guide.

General

Fee_Cfg.h is defined in the FLS module configuration "FlsIncludeFile".

Always set the following functions to TRUE:

- `FlsBlankCheckApi`
- `FlsCompareApi`
- `FlsGetJobResultApi`
- `FlsCancelApi`
- `FlsGetStatusApi`
- `FlsSetModeApi`
- `FlsReadImmediateApi`
- `FlsWriteVerification`

Always set the following functions to FALSE:

- `FlsBeforeWriteVerificaion`

FlsConfigSet->General

Call cycle

The value of `FeeMainFunctionPeriod` for FEE and `FlsCallCycle` for FLS should be set to the same value. If not, the time out function will not operate correctly.

Example:

```
FeeMainFunctionPeriod = 1 (msec)          FlsCallCycle = 0.001 (sec)
```

Note: *Set the value of `FeeMainFunctionPeirod` to 1 or higher. If you want to set the value to less than 1, set the value of `FlsCallCycle` to "0".*

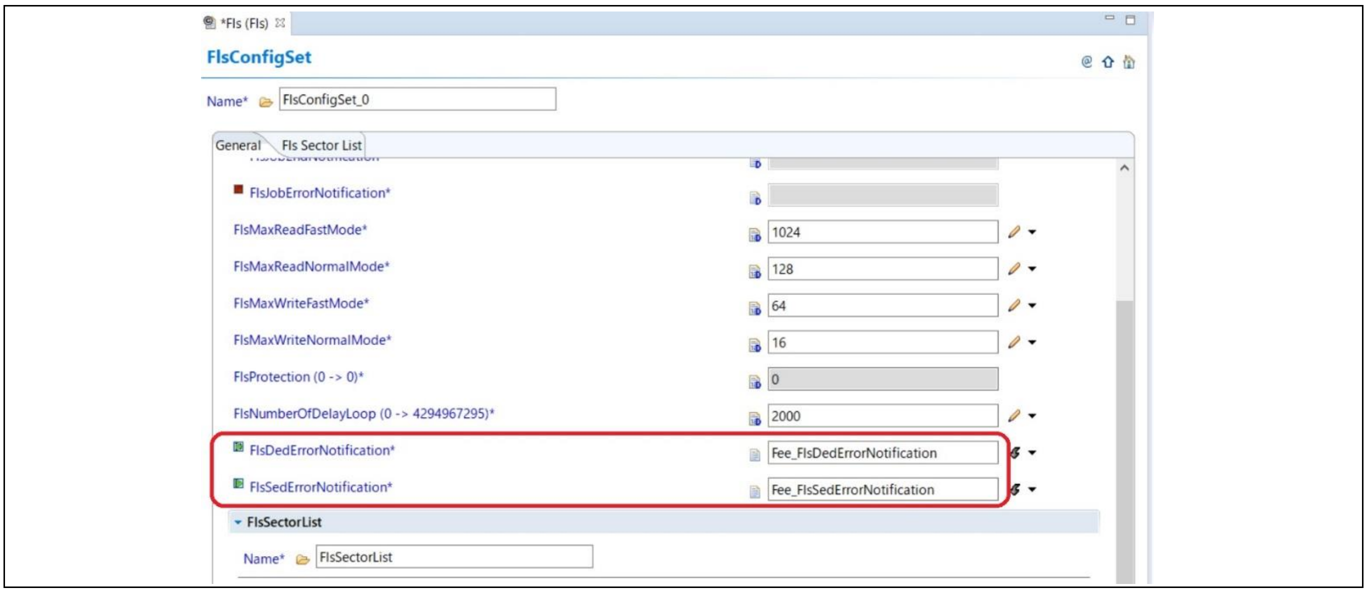
Error notification

Define this item only when operating FEE on Core® M4/M7. Definition is not required when operating FEE on Core M0.

For `FlsDedErrorNotification` and `FlsSedErrorNotification`, specify the FEE callback function as follows.

Figure 5 shows the description on tresos

[tresos item]	[FEE callback function name]
– `FlsDedErrorNotification` =>	Fee_FlsDedErrorNotification
– `FlsSedErrorNotification` =>	Fee_FlsSedErrorNotification



*Fls (Fls)

FlsConfigSet

Name* FlsConfigSet_0

General | Fls Sector List

FlsJobErrorNotification*	
FlsMaxReadFastMode*	1024
FlsMaxReadNormalMode*	128
FlsMaxWriteFastMode*	64
FlsMaxWriteNormalMode*	16
FlsProtection (0 -> 0)*	0
FlsNumberOfDelayLoop (0 -> 4294967295)*	2000
FlsDedErrorNotification*	Fee_FlsDedErrorNotification
FlsSedErrorNotification*	Fee_FlsSedErrorNotification

FlsSectorList

Name* FlsSectorList

Figure 5 **Description on Tresos**

FlsConfigSet->Fls Sector List

Sector

The FLS sectors used in FEE should be set continuously.

The total size of the sector required by FEE is:

- 8192 bytes or more (if ConfigEx is not used)
- 16,384 bytes or more (if ConfigEx is used)

Note: *When the configuration variant is VARIANT-POST-BUILD (postbuild), Fls_Init must be called with a parameter* `&Fls_Config_0` *points first* `FlsConfigSet`*:*

```
Fls_Init(&Fls_Config_0);
```

When the configuration variant is VARIANT-PRE-COMPILE (precompile), Fls_Init must be called i with a parameter `NULL_PTR` *points first* `FlsConfigSet`*:*

```
Fls_Init(NULL_PTR);
```

4.2.3.2 DET

The DET must be configured according to DET user guide, if default error is activated.

5 Functional description

5.1 Function of the FEE

FEE provides a hardware independent interface for the NVRAM manager.

5.1.1 FEE state machine

Figure 6 shows the Flash driver's state machine.

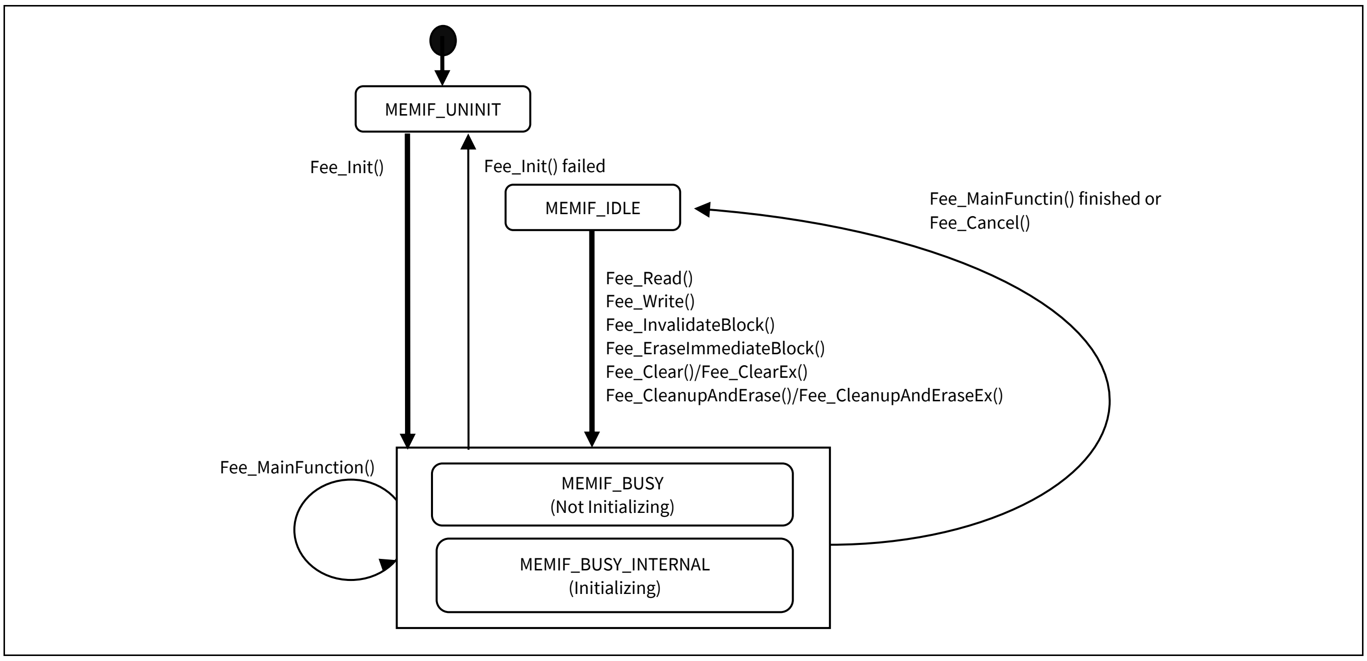


Figure 6 State machine of FEE

State MEMIF_UNINIT

After power on, FEE is in the `MEMIF_UNINIT` state in which it has not yet been initialized.

State MEMIF_IDLE

After successful initialization, the `MEMIF_IDLE` state is reached and FEE is ready.

If an ongoing request (Read, Write, and so on) is finished or canceled, the driver is also in in the MEMIF_IDLE state and is ready for the next request.

State MEMIF_BUSY

The `MEMIF_BUSY` state indicates that FEE has accepted some request except the initialization. FEE will change the status to `MEMIF_BUSY to` execute this request during the next call of the function `Fee_MainFunction()`. Also, FEE will remain in this state until the request is finished or canceled by the user.

State MEMIF_BUSY_INTERNAL

The state `MEMIF_BUSY_INTERNAL` indicates that the FEE has accepted the initialization request, and this request will be executed during the next call of the function `Fee_MainFunction()`. The FEE will remain in this state until the request is finished or canceled by the user.

5.1.2 FEE job result state

Figure 7 shows the Flash driver's job result state machine.

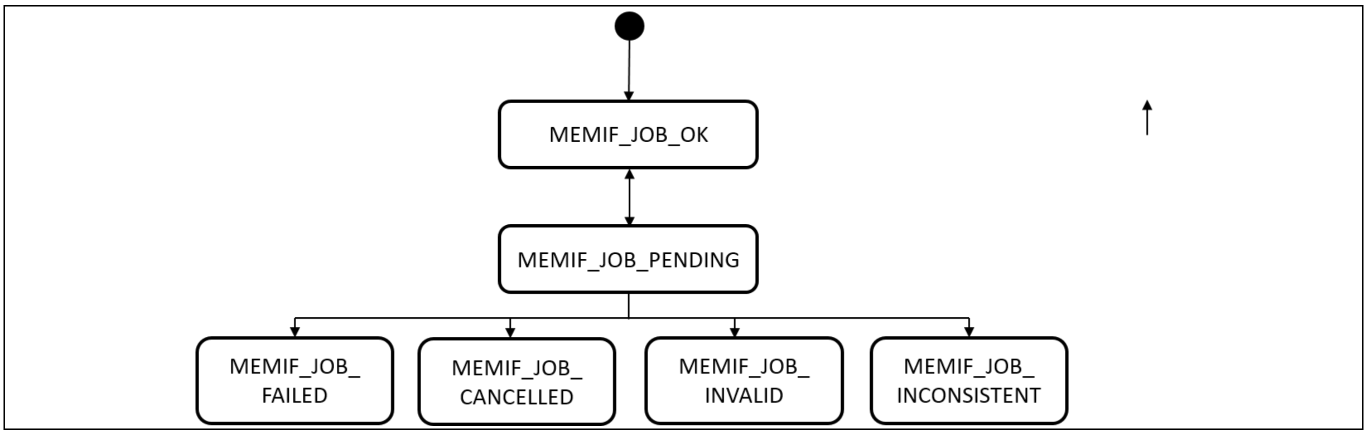


Figure 7 **State machine of the job result**

MEMIF_JOB_OK

The last job finished successfully. This state is also used after initialization.

MEMIF_JOB_PENDING

Some requested job are pending and will be executed on the next call of `Fee_MainFunction()`.

MEMIF_JOB_FAILED

The last job failed due to hardware error, timeout, and so on.

MEMIF_JOB_CANCELED

User cancelled the last job by calling the function `Fee_Cancel()`.

MEMIF_JOB_INVALID

The requested block was already invalidated, or there is no data with requested ID.

MEMIF_ JOB_INCONSISTENT

The requested block was inconsistent. The compared data is not corresponding.

5.1.3 Initialization

The initialization is done via the function call:

```
Fee_Init(ConfigPtr);
```

The initialization will be executed on the next call of `Fee_MainFunction()`.FEE is now in the `MEMIF_BUSY_INTERNAL` state. On each call of `Fee_MainFunction()`, perform the specified processes (see Behavior). After processing is successfully finished, the FEE module state is changed to `MEMIF_IDLE` and the job result is set to `MEMIF_JOB_OK`. If a job end notification function was configured, the function will also be called. If any error occurred during the process, the FEE module state is set back to `MEMIF_UNINIT` and the job result is set to `MEMIF_JOB_FAILED`. If a job error notification function was configured, the function will also be called.

Behavior:

The following are the behaviors of `Fee_Init()`:

1. `Fee_Init()` is called for band new work flash.

   In this case, whole sectors specified by configuration are initialized (erased) during the first operation.

   Example: If 28 sectors are specified, the `Fee_Init()` time takes the erase time of 28 sectors.

2. `Fee_Init()` is called after power shutdown.

   If the data of work flash is corrupted, recycling might occur for data correction. In this case, `Fee_Init()` erases the corrupted sector. The `Fee_Init()` time takes the erase time of 1 sector.

3. All other scenarios:

   `Fee_Init()` only creates index for data on work flash.

After initialization, the FEE module accepts a read, write, or erase job for the flash memory.

5.1.4 Reading data from flash memory

FEE supports reading data from the flash memory. A read job is set up via the command.

```
ReturnValue = Fee_Read (BlockNumber, BlockOffset, DataBufferPtr, Length);
```

If the function returns E_OK, the job was accepted and will be executed on the next call of `Fee_MainFunction()`. FEE is now in the `MEMIF_BUSY` state and will not accept other commands. The job result is set to `MEMIF_JOB_PENDING`. Then, the FEE module executes the physical address of flash memory by the parameters `BlockNumber` and `BlockOffset`. On each call of `Fee_MainFunction()`, data is read from the executed physical address of flash memory and copied to `DataBufferPtr`. After the number of bytes corresponding to the e data length is successfully copied from flash memory, the FEE module state is set back to `MEMIF_IDLE` and the job result is set to `MEMIF_JOB_OK`. If a job end notification function was configured, the function will also be called. If any error occurred during the read process, the driver will set the job result to `MEMIF_JOB_FAILED`. If a job error notification function was configured, the function will also be called.

5.1.5 Writing data to the flash memory

FEE supports writing data to the flash memory. A write job is set up via the following command:

```
ReturnValue = Fee_Write (BlockNumber,DataBufferPtr);
```

If the function returns E_OK, it indicates that the job was accepted and will be executed on the next call of `Fee_MainFunction()`. FEE is now in the `MEMIF_BUSY` state and will not accept other commands. The job result is set to `MEMIF_JOB_PENDING`. Then, the FEE module executes the physical address of flash memory by the parameter `BlockNumber`. On each call of Fee_MainFunction(),data from `DataBufferPtr` is written to execute the physical address of flash memory. After the configured block's size of data is successfully written to flash memory, the FEE module state is set back to `MEMIF_IDLE` and the job result is set to `MEMIF_JOB_OK`. If a job end notification function was configured, the function will also be called. If any error occurred during the write process, the driver will set the job result to `MEMIF_JOB_FAILED`. If a job error notification function was configured, the function will also be called.

5.1.6 Invalidate data of flash memory

FEE supports invalidating data of the flash memory. An invalidate job is set up via the command:

```
ReturnValue = Fee_InvalidateBlock (BlockNumber);
```

If the function returns E_OK, it indicates that the job was accepted and will be executed on the next call of `Fee_MainFunction()`. The FEE now in the `MEMIF_BUSY` state and will not accept other commands. The job result is set to `MEMIF_JOB_PENDING`. Then, the FEE module executes the physical address of flash memory by the parameter `BlockNumber`. C, data on the specified physical address of flash memory is invalidated. After the successful invalidation of data, the FEE module state is set back to `MEMIF_IDLE` and the job result is set to `MEMIF_JOB_OK`. If a job end notification function was configured, the function will also be called. If any error occurred during the write process, the driver will set the job result to `MEMIF_JOB_FAILED`. If a job error notification function was configured, the function will also be called.

5.1.7 Erase immediate data of flash memory

FEE can erase the immediate data of the flash memory. The `Fee_EraseImmediateBlock` is internally mapped to the `Fee_InvalidateBlock()` function. An invalidate job is set up via the command:

```
ReturnValue = Fee_EraseImmediateBlock (BlockNumber);
```

If the function returns E_OK, it indicates that the job was accepted and will be executed on the next call of `Fee_MainFunction()`.FEE is now in the `MEMIF_BUSY` state and will not accept other commands. The job result is set to `MEMIF_JOB_PENDING`. Then, the FEE module executes the physical address of flash memory by the parameter `BlockNumber`. On each call of `Fee_MainFunction()`,data on the specified physical address of flash memory is invalidated. After the invalidation of data successfully finishes, the FEE module state is set back to `MEMIF_IDLE` and the job result is set to `MEMIF_JOB_OK`. If a job end notification function was configured, the function will also be called. If any error occurred during the write process, the driver will set the job result to `MEMIF_JOB_FAILED`. If a job error notification function was configured, the function will also be called.

5.1.8 Erasing all data from flash memory

The FEE supports erase all data from the flash memory. An erase job is set up via either of the following commands:

```
ReturnValue = Fee_Clear ( ); //To erase data from "Config" area
ReturnValue = Fee_ClearEx ( ); //To erase data from "ConfigEx" area
```

Since there are two configurations, `Config` and `ConfigEx`, can be set in FEE, both configuration can be erased separately by `Fee_Clear` and `Fee_ClearEx`, respectively. If the function returns E_OK, it indicates that the the job was accepted and will be executed on the next call of `Fee_MainFunction()`.FEE is now in the `MEMIF_BUSY` state and will not accept other commands. The job result is set to `MEMIF_JOB_PENDING`. On each call of `Fee_MainFunction` data on flash memory is erased. After data is successfully erased, the FEE module state is set back to `MEMIF_IDLE` and the job result is set to `MEMIF_JOB_OK`. If a job end notification function was configured, the function will also be called. If any error occurred during the erase process, the driver will set the job result to `MEMIF_JOB_FAILED`. If a job error notification function was configured, the function will also be called.

5.1.9 Canceling job prior to maturity

Any ongoing flash job can be canceled by calling the function:

```
Fee_Cancel ( );
```

The function always cancels the ongoing job, sets the pending job result to `MEMIF_JOB_CANCELLED,` and sets the driver back to idle mode. If an error notification function was configured, the function will be called. FEE is ready for the next job immediately after returning from this function call.

5.1.10 Getting a remaining page

The writable remaining pages are returned by calling the following functions:

```
ReturnValue = Fee_GetRemainingPages ( ); //To get pages from "Config" area
ReturnValue = Fee_GetRemainingPagesEx ( ); //To get pages from "ConfigEx" area
```

The function can be executed when the driver is in idle mode. If data greater than the remaining pages is written, sector is erased after recycling.

5.1.11 Recycling a sector

The recycling of a sector occurs by calling the functions:

```
ReturnValue = Fee_CleanupAndErase ( ); //To recycle a sector for "Config" area
ReturnValue = Fee_CleanupAndEraseEx ( ); //To recyle a sector for "ConfigEx" area
```

Recycling of a sector is usually automatic. By calling the functions, the sectors can be forcefully recycled.

The function can be executed when the driver is in idle mode.

5.1.12 Retrieving status information

Two API functions are offered to get the current state of the driver and the current state of the job result.

```
    ModuleState = Fee_GetStatus ( );
  JobResult = Fee_GetJobResult ( );
```

For more information on the module's state and job result, see FEE state machine and FEE job result state, respectively.

Note: *While the flash memory cells are being programmed or erased, the microcontroller will not be transited to low power consumption modes. To know whether the flash memory processing is ongoing, the function `Fee_GetStatus` is called and the microcontroller can be transited to the modes only if `Fee_GetStatus` returns `MEMIF_IDLE`.*

5.1.13 Periodic_API_Implementation

`Fee_MainFunction()/Fls_MainFunction` should be called periodically from the application to perform an operation in progress[1]. When the operation is finished, FEE/FLS changes the job status from BUSY to IDLE and calls the callback functions. If the operation is not completed until after the function is called as many times as specified, a timeout process is performed. (For detailed information about the timeout process, see 5.1.15 Timeout monitoring).

Note: *`Fee_MainFunction()/Fls_MainFunction` should be called periodically. The periodical time is controlled by the API `Fee_SetCycleMode`. (For detailed information, see 5.1.16 Setting the call cycle mode).*

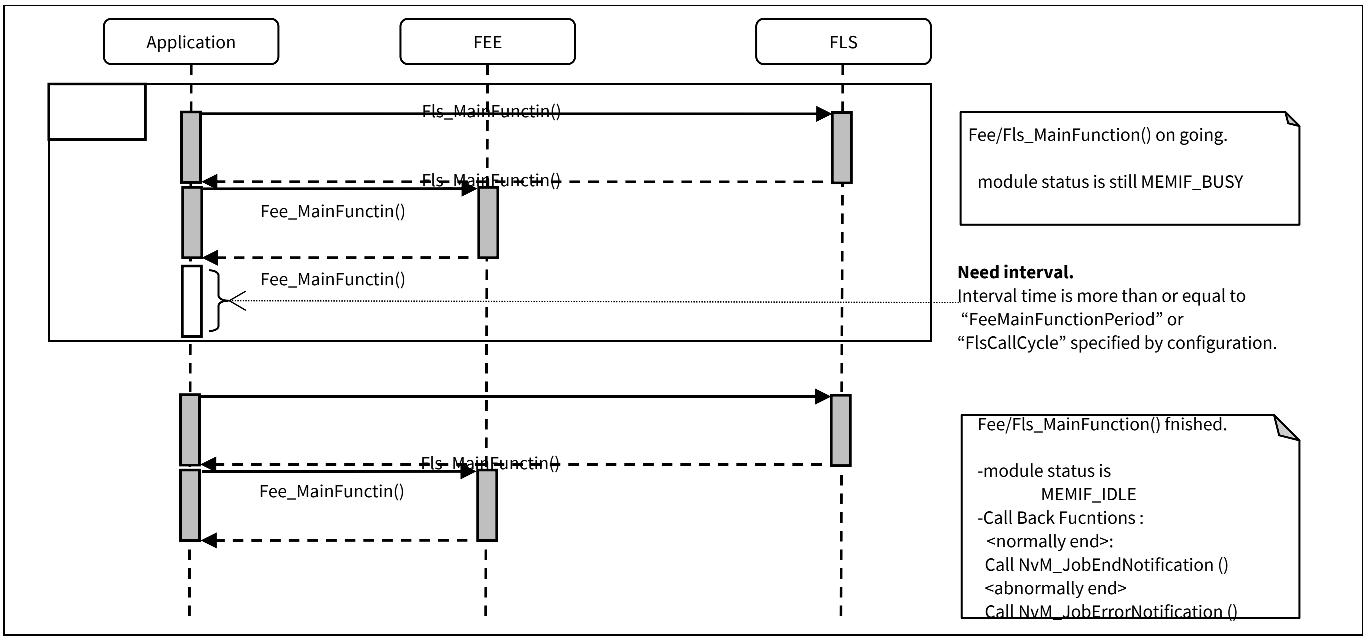

Application
FEE
FLS
Fls_MainFunctin()
Fls_MainFunctin()
Fee_MainFunctin()
Fee_MainFunctin()
Fee/Fls_MainFunction() on going.
module status is still MEMIF_BUSY
Need interval.
Interval time is more than or equal to "FeeMainFunctionPeriod" or "FlsCallCycle" specified by configuration.
Fls_MainFunctin()
Fee_MainFunctin()
Fee/Fls_MainFunction() fnished.
-module status is MEMIF_IDLE
-Call Back Fucntions :
<normally end>:
Call NvM_JobEndNotification ()
<abnormally end>
Call NvM_JobErrorNotification ()

Figure 8 Operation sequence

[1] For detailed information on operations performed by these functions, see 5.1.3 Initialization, 5.1.4 Reading data from flash memory, 5.1.5 Writing data to the flash memory, 5.1.6 Invalidate data of flash memory, 5.1.7 Erase immediate data of flash memory, 5.1.8 Erasing all data from flash memory, 5.1.9 Canceling job prior to maturity, 5.1.11Recycling a sector.

5.1.14 Setting the driver operation mode

The module can be switched between a slow and a fast operation mode. The default mode "slow" is used right after initialization. To switch to the fast mode, the following function must be called.

```
Fee_SetMode (MEMIF_MODE_FAST);
```

To return to the slow mode, the `Fee_SetMode` function must be called with the parameter `MEMIF_MODE_SLOW` while the FEE module is in idle mode. The function can be executed when the driver is in idle mode.

5.1.15 Timeout monitoring

The driver provides a timeout monitoring for the deadline of initialize, read, write, invalid and erase (invalidate) functions.

The maximum timeout value is calculated based on `FeeMainFunctionPeriod` specified by configuration[1] . `FeeMainFunctionPeriod` means the periodic time at which `Fee_MainFunction` is called. If time out occurs, FEE calls `ErrorCallBackFunction`[2] and abnormally ends.

Example of `Fee_Init() : FeeMainFunctionPeriod = 1ms`

```
Fee_Init(NULL);

do

  {

    Fls_MainFunction();

    Fee_MainFunction();  ----------->   Timeout monitoring

    wait_cycle(1ms);                        MaxIntCallCycle -=
FeeMainFunctionPeriod;

  }                                         If(MaxIntCallCycle < 0){

while( (Fee_GetStatus() != MEMIF_IDLE)         Call ErrorCallBackFunction;

&& (Fee_GetStatus()!=MEMIF_UNINIT) );       }
```

[1] For detailed information, see 5.1.16 Setting the call cycle mode.

[2] For detailed information, see 7.6 Configurable interfaces

5.1.16 Setting the call cycle mode

The module can be switched between a slow and a fast operation mode during the cycle time of `Fee_MainFunction/Fls_MainFunction`. The default mode is set by `FeeDefaultCycleMode`. To switch modes (slow <> fast), the following function should be called.

```
ReturnValue = Fee_SetCycleMode ([MEMIF_MODE_SLOW | MEMIF_MODE_FAST]);
```

- `MEMIF_MODE_SLOW`:

  This mode is used when `Fee_MainFunction/Fls_MainFunction` is performed with wait cycle time.

  The periodical time should be more than or equal to `FeeMainFunctionPeriod` or `FlsCallCycle` specified by user configuration.

  Consider the following example, where `FeeMainFunctionPeriod` and `FlsCallCycle` are set to 1 msec:

```
do
  {
    Fls_MainFunction();
    Fee_MainFunction();
    wait_cycle(1msec);               <-  Wait cycle time is 1ms
  }
while(Fee_GetStatus() != MEMIF_IDLE);
```

- `MEMIF_MODE_FAST`:

  This mode is used when `Fee_MainFucntion/Fls_MainFunction` is performed with no wait cycle.

  Consider the following example:

```
do
  {
    Fls_MainFunction();
    Fee_MainFunction();
                             <- no wait cycle time
  }
while(Fee_GetStatus() != MEMIF_IDLE);
```

To switch to slow mode (fast mode), the `Fee_SetCycleMode` function must be called with the parameter `MEMIF_MODE_SLOW` (`MEMIF_MODE_FAST`) while the FEE module is in idle state. The function can be executed when the driver is in idle state.

If the function returns E_OK, the job was accepted and normally finished. If the function returns E_NOT_OK, the job was not accepted and the mode was not switched. If it is E_NOT_OK, confirm whether the job state is idle.

5.2 Virtual flash memory layout

FEE provides upper layers with a 32-bit virtual linear address space and uniform segmentation scheme. This virtual 32-bit address will consist of 16-bit block number and 16-bit block offset. The FEE always maps the physical flash memory address to virtual linear address space. FEE uses the work flash memory only. For more information of physical flash address space, see the FLS module User Guide. For more information on virtual linear address space, see the AUTOSAR SWS (FEE).

5.3 Default error detection

The module's services perform regular error checks.

If default error detection is enabled, all errors are reported to DET, a central error hook function within the AUTOSAR environment. The error hook routine is called and the error code, service ID, module ID, and instance ID are passed as parameters. The checking itself cannot be deactivated for safety reasons.

Table 2 shows the default error checks that are performed by the service of FEE. 7.3 Functions explains which error codes are reported by each API function.

Table 2 Default error codes occurring during development[1]

Related error code	Value	Type of error
`FEE_E_UNINIT`	0x01	FEE has not been initialized. [How to handle] Execute initialization (`Fee_Init`) before executing each API (`Fee_write`, `Fee_Read`, and so on).
`FEE_E_INVALID_BLOCK_NO`	0x02	Block number is different from the value that has been set in the configuration. [How to handle] Check the block number specified for each API (`Fee_write`, `Fee_Read`, and so on).
`FEE_E_INVALID_BLOCK_OFS`	0x03	Offset is out of the block size set in the configuration. [How to handle] Check the block offset specified for API (`Fee_Read`).
`FEE_E_PARAM_POINTER`	0x04	Top address of the storage destination of the data is NULL. [How to handle] Check the data buffer pointer specified for API (`Fee_write`, `Fee_Read`, and so on).
`FEE_E_INVALID_BLOCK_LEN`	0x05	The result of `Length` + `BlockOffset` is greater than the configured `BlockSize` or the parameter `Length` is 0 or less. [How to handle] Check the block length specified for `Fee_Read`.

[1] Errors that would typically only occur during development.

Related error code	Value	Type of error
`FEE_E_IMMEDIATEDATASPACE_UNAVAILABLE`	0x12	Lack of immediate data reserve area error information. [How to handle] The immediate target block has already been written. Invalidate it using the `Fee_EraseImmediateBlock`, API and try to write again.

Table 3 Default error codes occurring during development and in the field[1]

Related error code	Value	Type of error
`FEE_E_BUSY`	0x06	FEE state is MEMIF_BUSY. [How to handle] Execute the API again after the status changes to MEMIF_IDLE.
`FEE_E_BUSY_INTERNAL`	0x07	FEE state is MEMIF_BUSY_INTERNAL (Under Initializing). [How to handle] Execute the API again after the status changes to MEMIF_IDLE.
`FEE_E_INVALID_CANCEL`	0x08	When processing cancel, FEE state is `MEMIF_IDLE`. [How to handle] Check the execution point of `Fee_Cancel`. Processing can continue normally.
`FEE_E_TIMEOUT_ERROR_OCCURRED`	0x11	Time-out error information. [How to handle] Check the FEE CONFIGURATION in: 5.1.15 Timeout monitoring 5.1.16 Setting the call cycle mode If the above does not work, try the following: (1) Execute the API again. (2) Initialize (execute `Fee_Init`) and execute the API again. (3) Erase all sectors.
`FEE_E_1BIT_ECC_ERROR_OCCURRED`	0x16	1-bit ECC error occurred during reading. [only for core M4/M7, not for core M0] [How to handle] MEMIF_JOB_OK is returned for `Fee_GetJobResult`. However, it may change to 2bit ECC error. Recycle using `Fee_CleanupAndErase`. If you are using ConfigEx, also execute `Fee_CleanupAndEraseEx`.
`FEE_E_2BIT_ECC_ERROR_OCCURRED`	0x17	2-bit ECC error occurred during reading. [only for core M4/M7, not for core M0] [How to handle]

[1] Errors that may occur both during development and in the field.

Related error code	Value	Type of error
		MEMIF_JOB_FAILED is returned for `Fee_GetJobResult`. However, when 2bit ECC error is detected during `Fee_Init`, MEMIF_IDLE is returned for `Fee_GetStatus`. The status of Block ID with 2bit ECC error is changed to invalid. (Unreadable). - The block ID with 2bit ECC error which is detected during `Fee_Init` cannot be notified to the upper applications, so it is recommended to rewrite the information of all blocks. - For the block ID with 2bit ECC error which is detected during Fee_Read, rewrite the data.

Table 4 Default error codes occurring in the field[1]

Related error code	Value	Type of error
`FEE_E_HARDWARE_ERROR_OCCURRED`	0x10	Hardware error information [How to handle] Try the following: (1) Execute the API again. (2) Initialize (Execute `Fee_Init`) and execute the API again. (3) Erase all sectors.
`FEE_E_BLOCKID_UNMATCHED_ERROR_OCCURRED`	0x14	Block ID does not match the configuration when Fee is initializing. [How to handle] (1) Use the original configuration file. Set "FeeUnmatchedBlockCheck" to "False". Execute `Fee_Init` again. (2) Erase all sectors.
`FEE_E_BLOCKSIZE_UNMATCHED_ERROR_OCCURRED`	0x15	Block size is not matched with configuration when Fee is initializing. [How to handle] (1) Use the original configuration file. Set "FeeUnmatchedBlockCheck" to "False". Execute `Fee_Init` again. (2) Erase all sectors.

5.4 Reentrancy

`Fee_GetVersionInfo()` is reentrant. All other API functions of FEE are non-reentrants.

5.5 Debugging support

The FEE does not support debugging.

[1] Errors that may occur in the field.

5.6 Note

None.

6 Hardware resources

6.1 Interrupts

The FEE does not use any interrupt.

7 Appendix A – API reference

7.1 Data types

7.1.1 External data types

Description

The FEE imports data types from the module MemIf and AUTOSAR standard data types.

7.1.2 Std_ReturnType

Description

AUTOSAR standard API return type.

7.1.3 Std_VersionInfoType

Description

This type is used to request the version of Fee using the `Fee_GetVersionInfo()` function.

7.1.4 MemIf_ModeType

Description

This type denotes the module operation mode. It is used as the parameter value of the `Fee_SetMode()` function.

7.1.5 MemIf_StatusType

Description

This type denotes the current status of the underlying abstraction module and device driver. It is used as the return value of the `Fee_GetStatus()` function.

7.1.6 MemIf_JobResultType

Description

This type denotes the result of the last job.

7.2 Macros

7.2.1 Error codes

See Table 5.

7.2.2 Version information

The version information listed in Table 5 is published in the module's header file.

Table 5 Version information

Name	Value	Description
`FEE_SW_MAJOR_VERSION`	See release notes	Vendor-specific major version number
`FEE_SW_MINOR_VERSION`	See release notes	Vendor-specific minor version number
`FEE_SW_PATCH_VERSION`	See release notes	Vendor-specific patch version number

7.2.3 Module information

Table 6 Module information

Name	Value	Description
`FEE_MODULE_ID`	21	Module ID
`FEE_VENDOR_ID`	66	Vendor ID

7.2.4 API service IDs

Table 7 API service IDs

API name	Value
`Fee_Init()`	0x00
`Fee_SetMode()`	0x01
`Fee_Read()`	0x02
`Fee_Write()`	0x03
`Fee_Cancel()`	0x04
`Fee_GetStatus()`	0x05
`Fee_GetJobResult()`	0x06
`Fee_InvalidateBlock()`	0x07
`Fee_GetVersionInfo()`	0x08
`Fee_EraseImmediateBlock()`	0x09
`Fee_MainFunction()`	0x12
`Fee_Clear()/Fee_ClearEx()`	0x30
`Fee_GetRemainingPages()/Fee_GetRemainingPagesEx()`	0x31
`Fee_CleanupAndErase()/Fee_CleanupAndEraseEx()`	0x33
`Fee_SetCycleMode()`	0x34

7.3 Functions

7.3.1 Fee_Init

Syntax

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

Service ID

0x00

Sync/Async

Asynchronous

Reentrancy

Non-reentrant

Parameters (in)

NULL_PTR only

Parameters (out)

None

Return value

None

DET errors

None

Description

None

Caveats

None

7.3.2 Fee_SetMode

Syntax

```
void Fee_SetMode(MemIf_ModeType Mode)
```

Service ID

0x01

Sync/Async

Synchronous

Reentrancy

Non-reentrant

Parameters (in)

Mode:

Underlying FLS module operation mode (`MEMIF_MODE_SLOW`, `MEMIF_MODE_FAST`).

Parameters (out)

None

Return value

None

DET errors

`FEE_E_UNINIT`: Module is not yet initialized.

`FFE_E_BUSY`: Module is currently busy.

Description

Setting the FEE and the FLS operation mode

Caveats

FEE must be initialized before this function is called. FEE must be in IDLE state when this function is called.

7.3.3 Fee_Read

Syntax

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

Service ID

0x02

Sync/Async

Asynchronous

Reentrancy

Non-reentrant

Parameters (in)

`BlockNumber`: Target block's number.

`BlockOffset`: Target block's offset.

`Length`: Number of bytes to read.

Parameters (out)

`DataBufferPtr`: The pointer to read data buffer.

Return value

`E_OK`: Read request was accepted.

`E_NOT_OK`: Read request was not accepted.

DET errors

`FEE_E_INVALID_BLOCK_NO`: The parameter `BlockNumber` is an invalid number.

`FEE_E_INVALID_BLOCK_OFS`: The parameter `BlockOffset` is an invalid offset.

`FEE_E_PARAM_POINTER`: The parameter `DataBufferPtr` is NULL.

`FEE_E_INVALID_BLOCK_LEN`: The result of `Length` + `BlockOffset` is greater than the configured `BlockSize` or the parameter `Length` is 0 or less

`FEE_E_UNINIT`: Module is not yet initialized.

`FEE_E_BUSY`: Module is currently busy.

Description

Sets up a read job for FEE. FEE executes physical address by `BlockNumber` and `BlockOffset`. FEE reads `Length` bytes data from executed address, and copies it to `DataBufferPtr`.

Caveats

FEE must be initialized before this function is called.

Only one job can be accepted at the same time.

7.3.4 Fee_Write

Syntax

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

Service ID

0x03

Sync/Async

Asynchronous

Reentrancy

Non-reentrant

Parameters (in)

`BlockNumber`: Target block's number.

`DataBufferPtr`: The pointer to write data buffer.

Parameters (out)

None

Return value

`E_OK`: Write request was accepted.

`E_NOT_OK`: Write request was not accepted.

DET errors

`FEE_E_INVALID_BLOCK_NO`: The parameter `BlockNumber` is an invalid number.

FEE_E_PARAM_POINTER: The parameter DataBufferPtr is NULL.

`FEE_E_UNINIT`: Module is not yet initialized.

`FEE_E_BUSY`: Module is currently busy.

Description

Sets up a write job for FEE. FEE executes physical address by `BlockNumber`. FEE writes the configured length of data from `DataBufferPtr` to executed address of flash memory.

Caveats

FEE must be initialized before this function is called.

Only one job can be accepted at the same time.

7.3.5 Fee_Cancel

Syntax

```
void Fee_Cancel(void)
```

Service ID

0x04

Sync/Async

Synchronous

Reentrancy

Non-reentrant

Parameters (in)

None

Parameters (out)

None

Return value

None

DET errors

`FEE_E_UNINIT`: Module is not yet initialized.

`FEE_E_INVALID_CANCEL`: Module has no jobs to cancel.

Description

This function cancels an ongoing job immediately. If there is no ongoing job, the function refreshes internal data.

Caveats

FEE must be initialized before this function is called.

FEE must have jobs before this function is called.

7.3.6 Fee_GetStatus

Syntax

`MemIf_StatusType Fee_GetStatus(void)`

Service ID

0x05

Sync/Async

Synchronous

Reentrancy

Non-reentrant

Parameters (in)

None

Parameters (out)

None

Return value

`MEMIF_UNINIT`: FEE has not been initialized.

`MEMIF_IDLE`: FEE is currently idle.

`MEMIF_BUSY`: FEE is currently busy.

DET errors

None

Description

This function returns the current state of FEE.

Caveats

None

7.3.7 Fee_GetJobResult

Syntax

`MemIf_JobResultType Fee_GetJobResult(void)`

Service ID

0x06

Sync/Async

Synchronous

Reentrancy

Non-reentrant

Parameters (in)

None

Parameters (out)

None

Return value

`MEMIF_JOB_OK`: The last job has been finished successfully.

`MEMIF_JOB_PENDING`: The last job is waiting for execution or is currently being executed.

`MEMIF_JOB_CANCELED`: The last job has been canceled (the job failed).

`MEMIF_JOB_FAILED`: The last job has not been finished successfully (the job failed).

`MEMIF_BLOCK_INVALID`: The requested block has been invalidated; the requested read operation cannot be performed.

DET errors

`FEE_E_UNINIT`: Module is not yet initialized.

Description

This function returns the last job result.

Caveats

FEE must be initialized before this function is called.

7.3.8 Fee_InvalidateBlock

Syntax

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

Service ID

0x07

Sync/Async

Asynchronous

Reentrancy

Non-reentrant

Parameters (in)

`BlockNumber`: Target block's number

Parameters (out)

None

Return value

`E_OK`: Invalidate request was accepted.

`E_NOT_OK`: Invalidate request was not accepted.

DET errors

`FEE_E_INVALID_BLOCK_NO`: The parameter `BlockNumber` is invalid number.

`FEE_E_UNINIT`: Module is not yet initialized.

`FEE_E_BUSY`: Module is currently busy.

Description

Sets up an invalidate job for FEE. FEE executes physical address by `BlockNumber`. FEE invalidates the flash memory data.

Caveats

FEE must be initialized before this function is called.

Only one job can be accepted at the same time.

7.3.9 Fee_GetVersionInfo

Syntax

`void Fee_GetVersionInfo(Std_VersionInfoType* VersionInfoPtr)`

Service ID

0x08

Sync/Async

Synchronous

Reentrancy

Reentrant

Parameters (in)

None

Parameters (out)

`VersionInfoPtr`: The pointer of the version information copy.

Return value

None

DET errors

FEE_E_PARAM_POINTER: VersionInfoPtr is NULL

Description

This function returns the version information of the module.

Caveats

None

7.3.10 Fee_EraseImmediateBlock

Syntax

`Std_ReturnType Fee_EraseImmediateBlock(uint16 BlockNumber)`

Service ID

0x09

Sync/Async

Asynchronous

Reentrancy

Non-reentrant

Parameters (in)

`BlockNumber`: Target block's number

Parameters (out)

None

Return value

`E_OK`: Erase the block data request was accepted.

`E_NOT_OK`: Erase the block data request was not accepted.

DET errors

`FEE_E_INVALID_BLOCK_NO`: The parameter `BlockNumber` is an invalid number, or the block was not configured to immediate block.

`FEE_E_UNINIT`: Module is not yet initialized.

`FEE_E_BUSY`: Module is currently busy.

Description

Sets up an erase (invalidate) immediate data job for FEE. It is internally mapped to the `FEE_InvalidateBlock()` function. FEE executes physical address by `BlockNumber`. FEE invalidates the target block of flash memory.

Caveats

FEE must be initialized before this function is called.

Only one job can be accepted at the same time.

The target block must be configured to the immediate block.

7.3.11 Fee_Clear / Fee_ClearEx

Syntax

`Std_ReturnType Fee_Clear(void) / Std_ReturnType Fee_ClearEx(void)`

Service ID

0x30

Sync/Async

Asynchronous

Reentrancy

Non-reentrant

Parameters (in)

None

Parameters (out)

None

Return value

`E_OK`: Erase the all data request was accepted

`E_NOT_OK`: Erase the all data request was not accepted

DET errors

`FEE_E_UNINIT`: Module is not yet initialized

`FEE_E_BUSY`: Module is currently busy

Description

Sets up an all data erase job for the FEE module. The FEE erases all blocks of flash memory. The `Config` / `ConfigEx` areas will be erased separately by `Fee_Clear` / `Fee_ClearEx`.

Caveats

The FEE must be initialized before this function is called.

7.3.12 Fee_GetRemainingPages/Fee_GetRemainingPagesEx

Syntax

`uint32 Fee_GetRemainingPages(void)/uint32 Fee_GetRemainingPagesEx(void)`

Service ID

0x31

Sync/Async

Synchronous

Reentrancy

Non-reentrant

Parameters (in)

None

Parameters (out)

None

Return value

In case of `Fee_GetRemainingPages`

Number of Remaining page for Config:

((empty area size - Immediate Data area size - block management area size) / `FeeVirtualPageSize`).

In case of `Fee_GetRemainingPagesEx`

Number of Remaining page for `ConfigEx`:

((empty area size - Immediate Data area size - block management area size) / `FeeVirtualPageSize`).

DET errors

`FEE_E_UNINIT`: Module is not yet initialized

`FEE_E_BUSY`: Module is currently busy

Description

Acquire remainder pages of the FEE domain (`Config` / `ConfigEx`).

Caveats

The FEE has to be initialized before this function is called.

7.3.13 Fee_CleanupAndErase / Fee_CleanupAndEraseEx

Syntax

```
Std_ReturnType Fee_CleanupAndErase(void) /
Std_ReturnType Fee_CleanupAndEraseEx(void)
```

Service ID

0x33

Sync/Async

Asynchronous

Reentrancy

Non-reentrant

Parameters (in)

None

Parameters (out)

None

Return value

`E_OK`: Recycle request was accepted

`E_NOT_OK`: Recycle request was not accepted

DET errors

`FEE_E_UNINIT`: Module is not yet initialized

`FEE_E_BUSY`: Module is currently busy

Description

The recycling a sector is usually automatically carried out. By calling the function, the recycling a sector can occur forcibly.

Caveats

The FEE has to be initialized before this function is called.

7.3.14 Fee_SetCycleMode

Syntax

`Std_ReturnType Fee_SetCycleMode(MemIf_ModeType Mode)`

Service ID

0x34

Sync/Async

Synchronous

Reentrancy

Non-reentrant

Parameters (in)

Mode:

Timeout monitoring operation mode

(`MEMIF_MODE_SLOW`, `MEMIF_MODE_FAST`).

Parameters (out)

None

Return value

`E_OK`: Timeout monitoring operation mode request was accepted

`E_NOT_OK`: Timeout monitoring operation mode request was not accepted

DET errors

`FEE_E_UNINIT`: Module is not yet initialized

`FFE_E_BUSY`: Module is currently busy

Description

Sets FEE and FLS timeout monitoring operation modes.

Caveats

Fee must be initialized before this function is called.

Fee must be in IDLE state when this function is called.

7.4 Scheduled functions

7.4.1 Fee_MainFunction

Syntax

```
void Fee_MainFunction(void)
```

Service ID

0x12

Timing

`FIXED_CYCLIC`

Reentrancy

Non-reentrant

Parameters (in)

None

Parameters (out)

None

Return value

None

DET errors

None

Description

This function performs the asynchronous processing of the jobs.

Caveats

FEE must be initialized before this function is called.

7.5 Expected interfaces (optional)

If default error detection is enabled FEE uses the following callback function provided by the DET.

7.5.1 Det_ReportError

Syntax

```
Std_ReturnType Det_ReportError(uint16 ModuleId, uint8 InstanceId, uint8 ApiId, uint8
ErrorId)
```

Sync/Async

Synchronous

Reentrancy

Reentrant

Parameters (in)

`ModuleId`: Module ID of FEE.

`InstanceId`: Instance ID of FEE.

`ApiId`: ID of the API function that calls this function.

`ErrorId`: ID of the detected error.

Return value

Always returns `E_OK`

Description

Function for reporting default errors.

7.6 Configurable interfaces

The following callback functions are configurable and usually provided by the NVRAM manager.

7.6.1 NvM_JobEndNotification

Syntax

```
void NvM_JobEndNotification(void)
```

Reentrancy

Don't care

Parameters (in)

None

Return value

None

Description

This callback function will be called when a job has been completed with a positive result.

Configurable

In this module, there are no configuration jobs.

7.6.2 NvM_JobErrorNotification

Syntax

```
void NvM_JobErrorNotification(void)
```

Reentrancy

Don't care

Parameters (in)

None

Return value

None

Description

This callback function will be called when a job has been completed with a negative result.

Configurable

In this module, there are no configuration jobs.

7.7 Required callback functions

Callout functions

7.7.1 Error callout API

FEE requires an error callout handler. Each error is reported to this handler, and error checking cannot be switched OFF. The name of the function to be called can be configured by the parameter `FeeErrorCalloutFunction`.

Syntax

```
void Error_Handler_Name(uint16 ModuleId, uint8 InstanceId,uint8 ApiId, uint8 ErrorId)
```

Reentrancy

Reentrant

Parameters (in)

`ModuleId`: Module ID of calling module.

`InstanceId`: Instance ID of calling module.

`ApiId`: ID of the API function that calls this function.

`ErrorId`: ID of the detected error.

Return value

None

Description

Function for reporting errors.

8 Appendix B – Access register table

The FEE does not use any register.

Note: The underlying FLS uses hardware registers. (see FLS's user guide)

References

AUTOSAR requirements and specifications

Bibliography

[1] General specification of basic software modules, AUTOSAR release 4.2.2.

[2] Specification of flash driver, AUTOSAR release 4.2.2.

[3] Specification of flash EEPROM emulation, AUTOSAR release 4.2.2.

[4] Specification of default error tracer, AUTOSAR release 4.2.2.

[5] Specification of RTE, AUTOSAR release 4.2.2.

[6] Specification of ECU configuration, AUTOSAR release 4.2.2.

[7] Specification of NVRAM manager, AUTOSAR release 4.2.2.

[8] Requirements of memory hardware abstraction layer, AUTOSAR release 4.2.2.

Elektrobit automotive documentation

Bibliography

[9] EB tresos Studio for ACG8 user's guide.

Hardware documentation

The hardware documents are listed in the delivery notes.

Related standards and norms

Bibliography

Layered software architecture, AUTOSAR release 4.2.2.

Revision history

Document revision	Date	Description of changes
**	2018-01-23	Initial release.
*A	2019-01-16	Updated hardware documentation Updated vendor and module specific parameters - Added FeeUnmatchedBlockCheck Removed FEE_E_BUSY_INTERNAL and MEMIF_BUSY_INTERNAL_state - 5.1.1 FEE state machine - 5.3 Default error detection - A.3.2 Fee_SetMode - A.3.3 Fee_Read - A.3.4 Fee_Write - A.3.5 Fee_Cancel - A.3.6 Fee_GetStatus - A.3.8 Fee_InvalidateBlock - A.3.10 Fee_EraseImmediateBlock - A.3.11 Fee_Clear / Fee_ClearEx - A.3.12 Fee_GetRemainingPages / Fee_GetRemainingPageEx - A.3.13 Fee_CleanupAndErase / Fee_CleanupAndEraseEx - A.3.14 Fee_SetCycleMode Updated the explanation of State MEMIF_BUSY of 5.1.1 FEE state machine Change the document name from users guide to user guide.
*B	2019-06-20	Updated hardware documentation information. Updated 4.2.2 Vendor and module specific parameters - Updated `FeeUnmatchedBlockCheck` - Added `FeeWorkFlashRelativeEndAddress` - Updated `FeeSectorStartAddress` - Updated `FeeSectorStartAddressEx` - Deleted `FeeSectorSize` - Deleted `FeeSectorSizeEx` Updated 4.3.2 Sector number and sector address Updated 5.1.13 Periodic_API_Implementation Updated 5.1.15 Timeout monitoring
*C	2020-01-09	Updated 1.1 Introduction to the Flash EEPROM Emulation. - Added 1.1.1 Feature of FEE
*D	2020-06-25	Updated 4.2.2 Vendor and module specific parameters - Updated FeeWorkFlashRelativeEndAddress Added 4.2.3.1 FLS module Updated 5.1.1 FEE state machine Updated 5.3 Default error detection

Document revision	Date	Description of changes
		Updated A.3.7 Fee_GetJobResult
*E	2020-09-05	Added 2.5 Memory mapping Updated 4.2.1 Parameter constraints - Updated `FeeImmediateData` Updated 5.3 Default error detection - Added `FEE_E_1BIT_ECC_ERROR_OCCURRED` - Added `FEE_E_2BIT_ECC_ERROR_OCCURRED`
*F	2020-11-16	Updated to Infineon template.
*G	2021-04-15	Updated `FeeUnmatchedBlockCheck` Updated General in FLS module Updated 5.1.3 Initialization Updated 5.1.8 Erasing all data from flash memory Updated 5.1.10 Getting a remaining page Updated 5.4 Reentrancy
*H	2021-12-23	Updated to Infineon style.
*I	2022-02-18	Updated 4.2.1 Parameter constraints Added `FeeDelayRecycleOperation`
*J	2022-10-27	Update 4.2.2 Vendor and module specific parameters Updated FeeUnmatchedBlockCheck Added ConfigIfUseThresholdPageSize Added FeeNormalPageSize Added FeeThresholdPageSize Added ConfigExIfUseThresholdPageSize Added FeeNormalPageSizeEx Added FeeThresholdPageSizeEx
*K	2023-06-01	Updated 5.4 Reentrancy
*L	2023-12-08	Web release. No content updates.

Edition 2023-12-08

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about this document?

Email: erratum@infineon.com

Document reference

002-22364 Rev. *L