VECTOR

MICROSAR Classic FiM

Technical Reference

Function Inhibition Manager

Version 10.0.1

Authors	vistne
Status	Released

Document Information

History

Author	Date	Version	Remarks
vistne	2012-10-19	1.0.0	first version of FiM according AR4
vistne	2013-03-15	1.1.0	added calibration section, added OBD support, removed restriction regarding cyclic event evaluation
vistne	2013-06-28	1.2.0	changed include structure
vistne	2013-10-18	2.0.0	added info to FiM_DemTriggerOnEventStatus
vistne	2014-03-07	2.1.0	added Post-Build Loadable description, some smaller changes
vistne	2014-10-31	3.0.0	described format of version info numbers added section for integration in AR 3 stack
vistne	2015-03-20	3.1.0	described new 3.1.0 features
vistne	2016-01-08	4.0.0	removed calibration section and descriptions related to cyclic event evaluation
vistne	2016-11-18	4.2.0	added …VAR_INIT… for compiler abstraction / memory mapping
vistne	2017-10-27	5.0.0	rework for AR4.3
vistne	2017-11-24	5.1.0	small changes
vistne	2018-03-16	5.2.0	added description for calibration using vPblCalib
vistne	2018-08-24	6.0.0	changes for Multicore
vistne	2018-09-28	6.1.0	added FiM_PostInit
vistne	2018-11-23	6.2.0	small changes
vistne	2019-03-15	6.4.0	production version
vistne	2020-04-09	7.0.0	fixes regarding inter core synchronization and communication (ESCAN00103546 and ESCAN00103578), fix regarding memory sections (ESCAN00103001), new Exclusive Area, changes for memory sections
vistne	2021-02-12	9.0.0	added affected FIDs identification several smaller improvements
vistne	2022-12-16	9.1.0	fix of ESCAN00112480 required changes in Memory Mapping and FiM_DemInitSatellite for single partition use cases without explicitly assigned OS application, see sections 3.2.5, 3.2.6 and 4.2.3, fix for description of FiM_GetFunctionPermission and FiM_GetFunctionPendingStatus (ESCAN00112020), fix for description of ApplFiM_SyncCompareAndSwap (ESCAN00112641), product name updated to MICROSAR Classic
vsgeei vistne	2023-06-30	10.0.0	Added Generation of A2L Measurement section usage of MemMap generator changed file structure

vistne	2023-08-25	10.0.1	removed section measurable objects (via Diag_A2lGen)

Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	Specification of Function Inhibition Manager	4.3.0
[2]	AUTOSAR	Specification of Default Error Tracer	4.3.0
[3]	AUTOSAR	List of Basic Software Modules	4.3.0
[4]	Vector	MICROSAR Diagnostic Event Manager (DEM) for OBD, Technical Reference Addendum	see delivery
[5]	Vector	MICROSAR Post-Build Loadable, Technical Reference	see delivery
[6]	Vector	vPblCalib, Technical Reference	see delivery
[7]	Vector	AUTOSAR Measurement & Calibration Manual	see delivery
[8]	Vector	MICOSAR MemMap, Technical Reference	see delivery

Scope of the Document

This technical reference describes the general use of the Function Inhibition Manager Basic Software Module.

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.

Caution
This symbol calls your attention to warnings.

Contents

Illustrations

Tables

1 Introduction

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

Supported AUTOSAR Release:	4.3	
Supported Configuration Variants:	pre-compile, post-build	
Vendor ID:	FiM_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	FiM_MODULE_ID	11 decimal (according to ref. [3])

The Function Inhibition Manager is responsible for providing a control mechanism for software components and the functionality therein. In this context, functionality can comprise one, several or parts of runnable entities with the same set of permission / inhibit conditions.

1.1 Architecture Overview

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

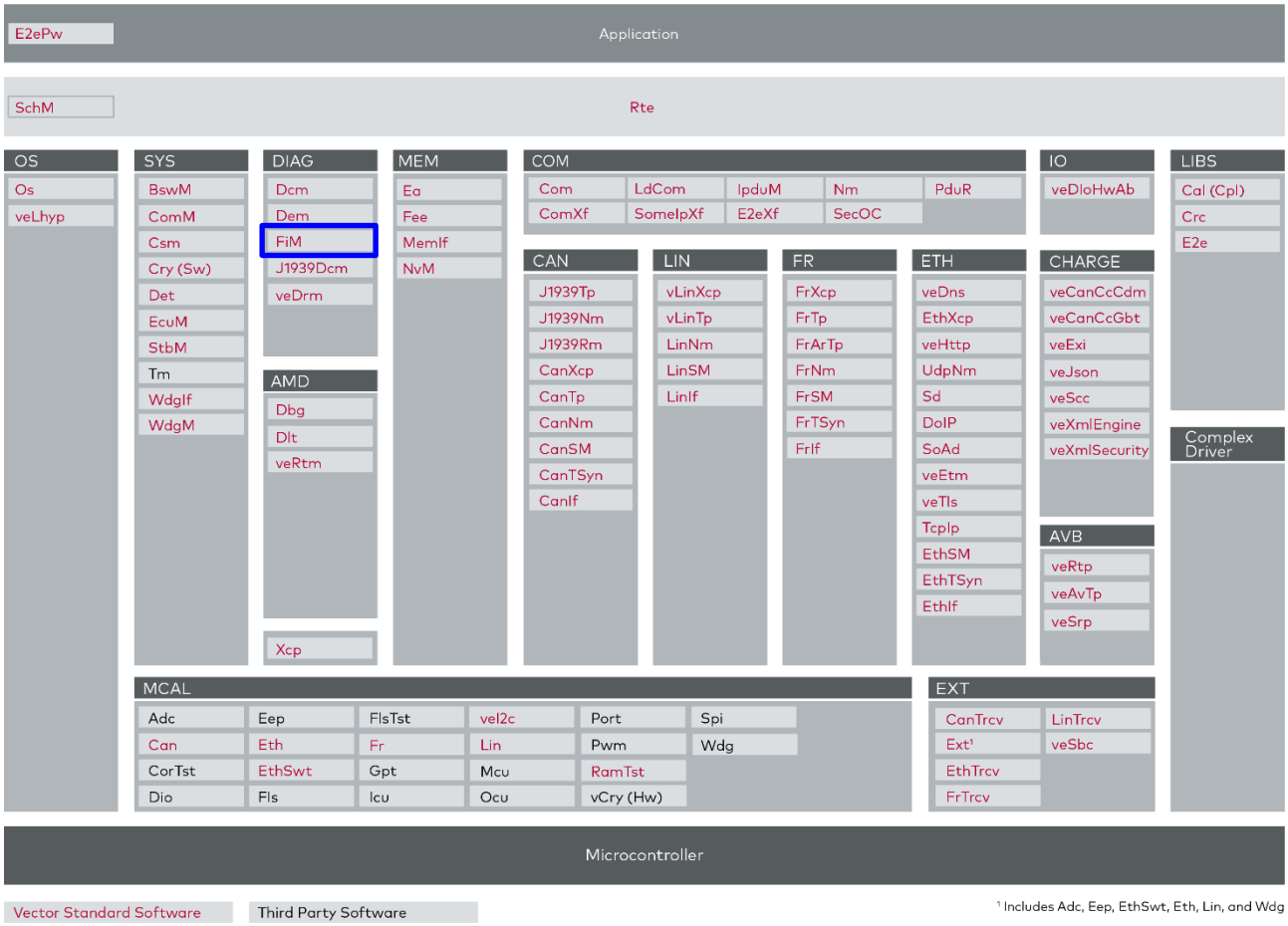

Figure 1-1 AUTOSAR 4.2 Architecture Overview

The next figure shows the interfaces to adjacent modules of the FiM. These interfaces are described in chapter 4.

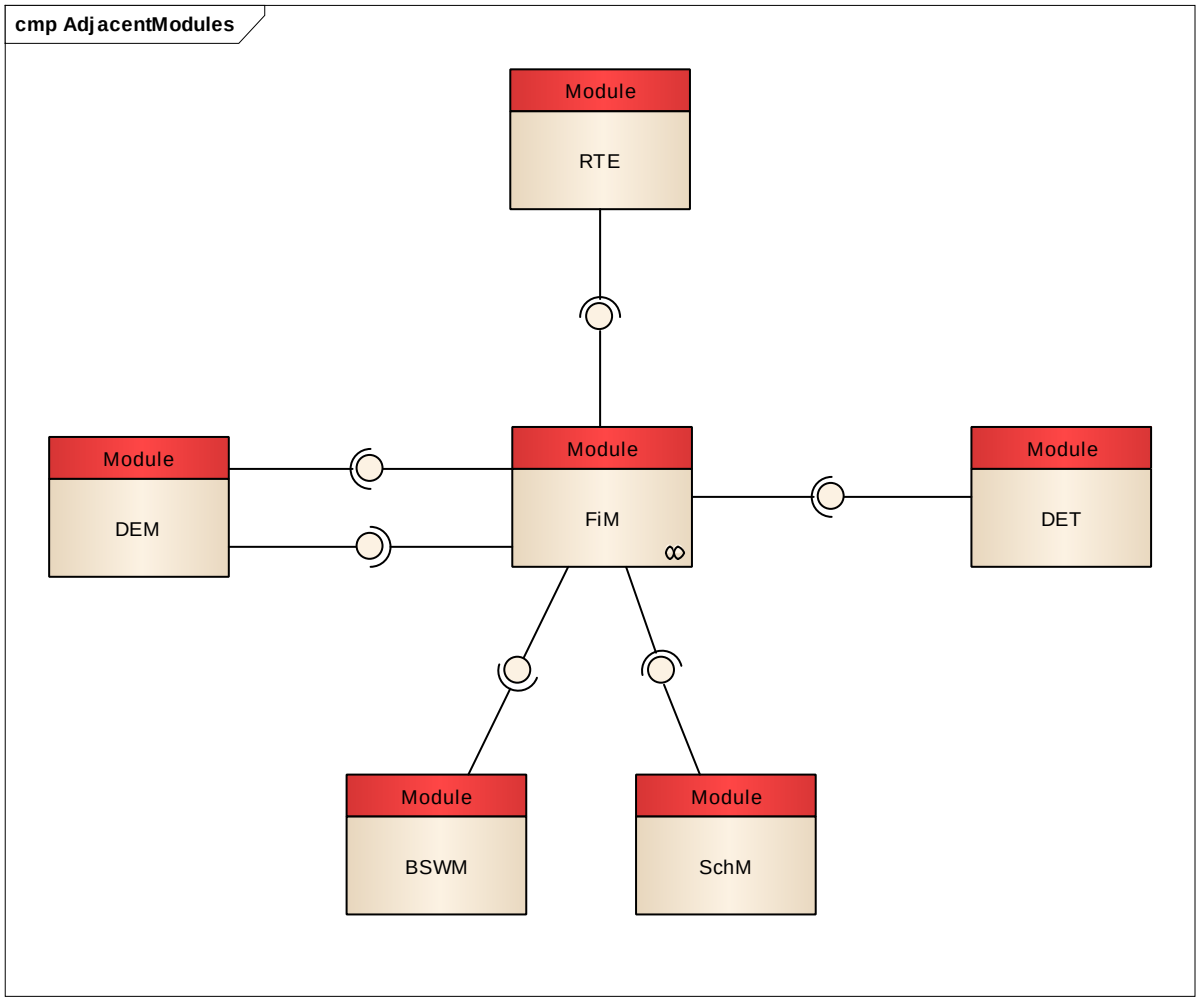

Figure 1-2 Interfaces to adjacent modules of the FiM

Applications do not access the services of the BSW modules directly. They use the service ports provided by the BSW modules via the RTE. The service ports provided by the FiM are listed in chapter 4.5 and are defined in [1].

2 Functional Description

2.1 Features

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

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

- Table 2-1 Supported AUTOSAR standard conform features
- Table 2-2 Not supported AUTOSAR standard conform features

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

- Table 2-3 Features provided beyond the AUTOSAR standard

The following features specified in [1] are supported:

Supported AUTOSAR Standard Conform Features
Get function permission by FID
Use DEM monitor status as inhibition condition
DEM notify FIM on monitor status change
Pre-compile configuration
AUTOSAR service component template generation
Development error detection over DET
Post-Build Loadable (certain configurations, see section 5.4.1)
Status variable tracking via measurement
Module individual post-build loadable update

Table 2-1 Supported AUTOSAR standard conform features

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

Not Supported AUTOSAR Standard Conform Features
Cyclic evaluation of DEM Events
Link time configuration
Summarized events
Monitored Components
Direct Configuration via calibration tool
Set function availability
Debugging
MICROSAR Classic Identity Manager using Post-Build Selectable

Table 2-2 Not supported AUTOSAR standard conform features

The following features are provided beyond the AUTOSAR standard:

Features Provided Beyond the AUTOSAR Standard
Usage on multiple OS partitions
FiM_InitMemory, see section 4.2.11
FiM_GetFunctionPendingStatus, see section 4.2.9
FiM_DemTriggerOnEventStatus, see section 4.2.7
Cyclic FID calculation, see section 5.2
Usage without RTE
Configuration via calibration tool using vPbICalib (see section 5.3.1)

Table 2-3 Features provided beyond the AUTOSAR standard

2.2 Major Changes between FiM versions

Note
The next sections contain information about changes between different versions of the FiM. These are only overviews and may not be complete. See remainder of Technical Reference for details.

2.2.1 Major Changes in FiM version 10.x compared to FiM version 9.1

This section provides an overview of major changes in the 10.x version of the FiM compared to the previous 9.1 version.

FiM version 10.x

- uses the MemMap component for memory mapping
- has changed memory section names (see section 3.2)
- has a changed file structure (see section 3.1)
- supports usage of MC Support Data and A2L file generation for measurement (see section 5.3.1)

Caution
Check and adapt your memory mapping.

2.2.2 Major Changes in FiM version 9.1.x compared to FiM version 9.0

This section provides an overview of major changes in the 9.1.x version of the FiM compared to the previous 9.0 version.

FiM version 9.1.x changes the default memory section names in single partition use cases if no explicit application id is configured:

- see sections 3.2.5 and 3.2.6: from …_0_... to …_INVALID_OSAPPLICATION_...

Caution
Check and adapt your memory mapping files if you have such a configuration.

2.2.3 Major Changes in FiM version 9.x compared to FiM version 8.x

This section provides an overview of major changes in the 9.x version of the FiM compared to the previous 8.x version.

FiM version 9.0 introduces

- added option that cyclic FID calculation only considers FIDs that may have changed since last evaluation (“Affected FIDs Identification”).

2.2.4 Major Changes in FiM version 8.x compared to FiM version 7.x

This section provides an overview of major changes in the 8.x version of the FiM compared to the previous 7.x versions.

FiM version 8.0 introduces

- changes for new configuration structure according to AR 4.3 – see BSWMD file.

2.2.5 Major Changes in FiM version 7.x compared to FiM version 6.x

This section provides an overview of major changes in the 7.x version of the FiM compared to the previous 6.x version.

2.2.5.1 Changes in Support of Multiple Partitions

FiM version 7.0 introduces

- support of satellites running on trusted partitions while master is running on an untrusted partition,
- new memory sections and changes to previously existing ones,
- a new exclusive area FIM_EXCLUSIVE_AREA_1 for separate implementation of inter- and intra-core synchronization.

Caution
Re-evaluate your mapping of Exclusive Areas:

Both the existing FIM_EXCLUSIVE_AREA_0 and the new exclusive area FIM_EXCLUSIVE_AREA_1 must be mapped to appropriate synchronization mechanisms.

2.2.6 Major Changes in FiM version 6.x compared to FiM version 5.x

This section provides an overview of major changes in the 6.x version of the FiM compared to the previous 5.x version.

2.2.6.1 Support of Multiple Partitions

- FiM supports distribution onto multiple OS partitions, thus integrating with Vector's multi partition DEM solution.
- FiM does not use post-build RAM anymore. RAM that is relevant for later post-build updates is now configured via the new configuration parameter FiMConfigSet/FiMSatellite/FiMMaxNrInhStates on a per satellite basis at pre-compile time.
- FiM version 6.1 introduces FiM_PostInit; compared to version 6.0 this avoids that satellites need to have write permission on FiM's master partition.

2.2.7 Major Changes in FiM version 5.x compared to FiM version 4.x

This section provides an overview of major changes in the 5.x version of the FiM compared to the 4.x version.

Note
Please also refer to the Technical Reference of FiM version 5.x.

2.2.7.1 Usage of Monitor Status

From version 5.x on FiM uses DEM's AUTOSAR 4.3 monitor status bits instead of UDS status bits for calculating FID states. Monitor status changes are communicated to FiM using the API FiM_DemTriggerOnMonitorStatus while event status changes before were communicated via FiM_DemTriggerOnEventStatus.

FiM_DemTriggerOnEventStatus is still available for OBD use cases where changes of the Pending status bits need to be passed from DEM to FiM.

2.2.7.2 FiM_GetPendingStatus

FIDs that should be blocked depending on an event's pending status need to be configured separately from version 5.x on. Previously, the pending status of any event that was connected via a normal inhibition configuration had been considered.

2.2.7.3 Initialization

Compared to version 4.x FiM_Init in version 5.x does only a pre-initialization of the module. FiM is not ready for operation afterwards. DEM needs to call FiM_DemInit to complete FiM's initialization.

2.2.7.4 Runtime

The calculation of FID states in version 4.x was mainly done in the context of DEM's event status update trigger (FiM_DemTriggerOnEventStatus). While this is still true for updates caused by pending status changes, updates caused by monitor status changes are only handled partially during runtime of FiM_DemTriggerOnMonitorStatus from version 5.x on. The rest is done during the call of FiM_GetFunctionPermission.

Note
From version 6.x on the configuration parameter FiMGeneral/FiMCyclicFidCalculation can be used to defer the calculation part done in FiM_GetFunctionPermission to FiM_MainFunction.

2.3 FiM Module Architecture

FiM is logically separated into multiple interacting software components:

- A dedicated FiM satellite per OS partition
  - provides as service SWC the interface FunctionInhibition (section 4.5.1.1.1) if a FiM access is configured for the partition and
  - calculates the inhibition status of configured events located on the DEM satellite in the same partition.
- For one particular OS partition, a FiM master SWC provides functionality for the DEM master.

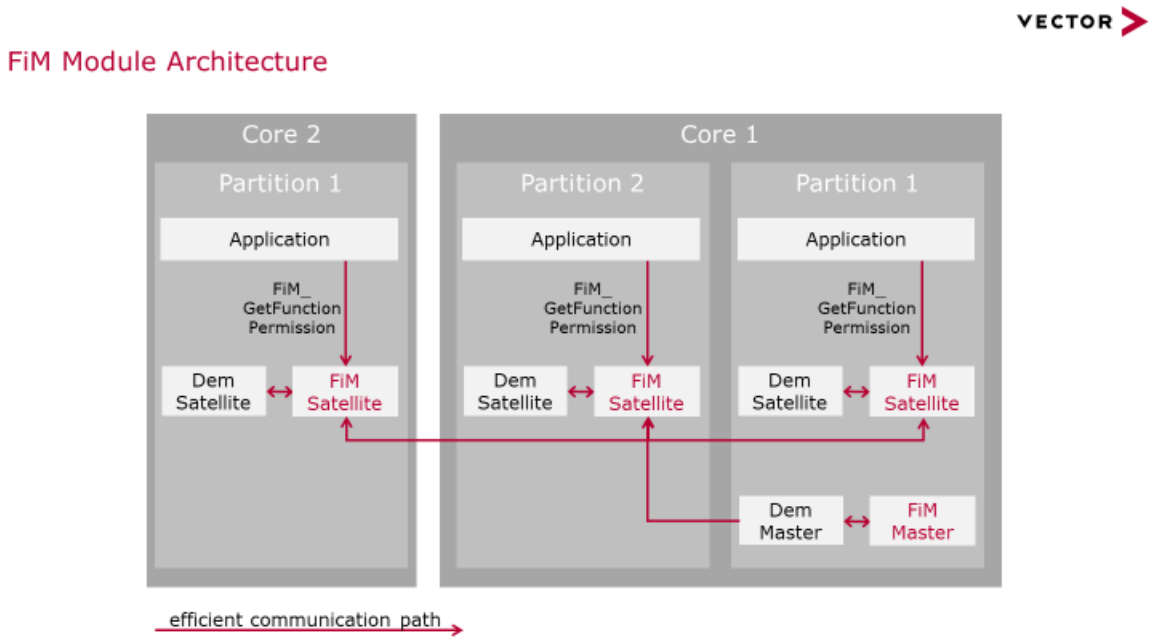

Figure 2-1 FiM Module Architecture

Note

It's **mandatory** to have a FiM satellite on the OS partition where the FiM master resides.

It's **mandatory** that FiM master and DEM master are mapped to the same partition.

It's **recommended** to have a FiM satellite on each partition where a DEM satellite resides.

It's **recommended** to have a DEM satellite on each partition where a FiM satellite resides.

2.3.1 FiM Satellite(s)

A FiM satellite performs processing of monitor status changes passed by an adjacent DEM satellite or the DEM master. Since events are assigned to exactly one DEM satellite, the FiM satellite on the same partition handles the monitor status changes of this DEM satellite's events.

Note
Inhibition configurations can only reference events if there is a FiM satellite configured on the OS partition onto which the event is mapped in the DEM configuration.

Since FiM satellites cannot be changed during the post-build phase, no reference to an event can be added to the FiM configuration during post-build phase if an appropriate FiM satellite had not been configured at pre-compile time.

It is therefore recommended to configure a FiM satellite on each OS partition where a DEM satellite is configured.

Also, a FiM satellite provides access to the permission status of the FIDs that are mapped to it.

Note
It is recommended that FIDs are mapped to the OS partition of the application that needs its permission status.

While it is possible that an application accesses the permission status of a FID mapped to a different OS application, this access is routed via the RTE and less runtime efficient than access to local FIDs.

2.3.2 FiM Master

FiM master provides IUMPR lock processing for the DEM master. The FiM master needs to be mapped onto the same OS partition as the DEM master.

2.3.3 Communication Constraints

To circumvent expensive general cross-partition communication implemented by the RTE, FiM uses internal data exchange based on shared memory and atomic compare/exchange instructions.

Instructions for efficient synchronization are provided by all multi-core platforms. However, due to the lack of a common library an implementation needs to be provided during integration. For details, please refer to section 3.3.1.

The memory used by FiM is organized using Memory Sections. The Memory Sections are grouped according to necessary access permissions. Within this document these groups are called Memory Section Groups. See section 3.2 for further information on memory mapping.

All partitions where FiM master and satellites are running on require read access to constant configuration data.

The partitions that FiM master and satellites are running on can be of different trust levels. Depending on these the following scenarios are supported.

2.3.3.1 FiM Master and All Satellites Running on Untrusted Partitions

In this scenario, FiM does not fulfill any requirements for freedom of interference. All FiM memory section groups for variable data may be writable from untrusted partitions. See Figure 2-2.

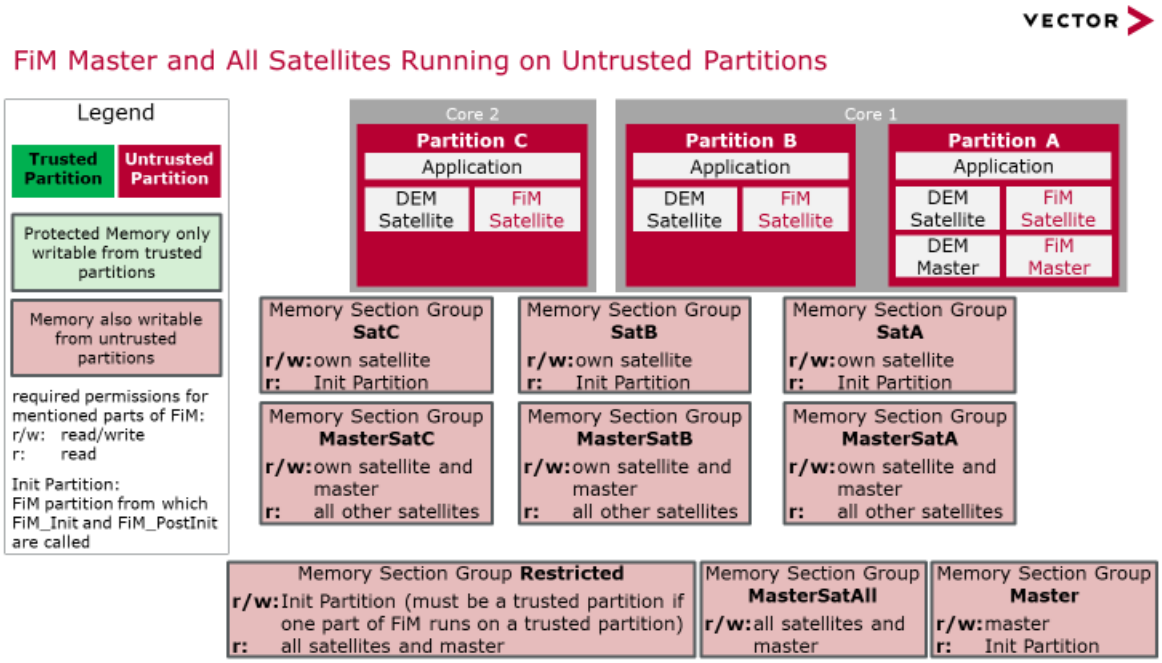

Figure 2-2 Memory Section Access: FiM Master and All Satellites Running on Untrusted Partitions

2.3.3.2 FiM Master and All Satellites Running on Trusted Partitions

In this scenario the complete FiM, i.e. the master and all satellites, fulfill the requirement for freedom of interference. All FiM memory section groups for variable data may only be writable from trusted partitions. Figure 2-3 depicts this situation.

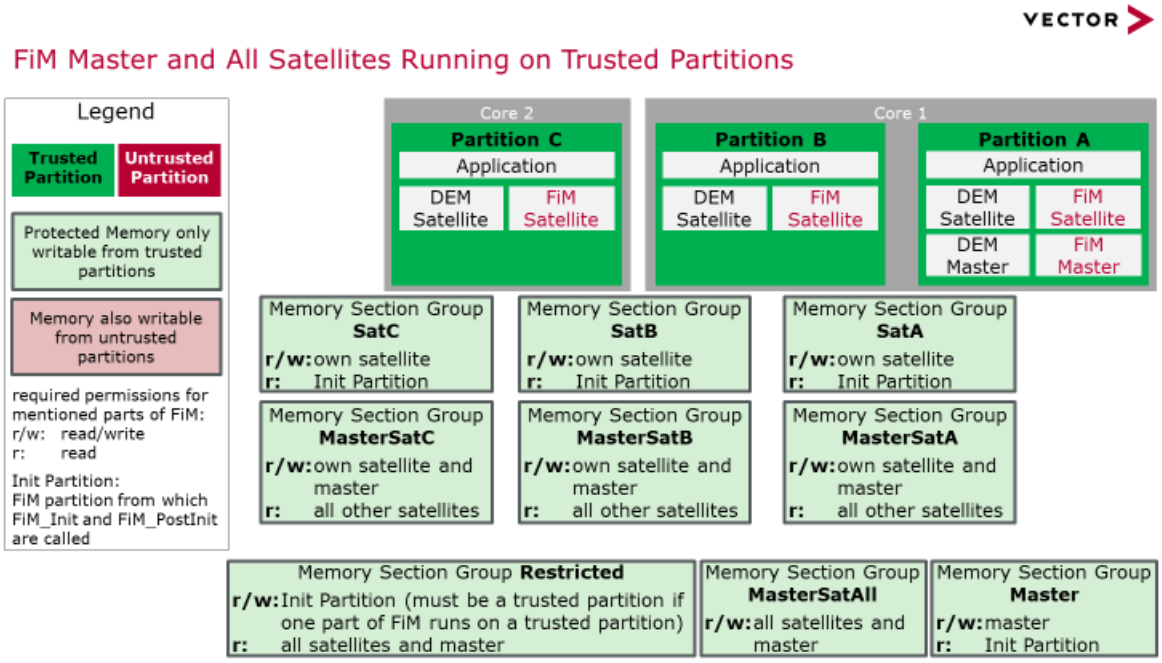

Figure 2-3 Memory Section Access: FiM Master and All Satellites Running on Trusted Partitions

2.3.3.3 FiM Master and Some Satellites Running on Trusted Partitions, Some Satellites Running on Untrusted Partitions

In this scenario the FiM master and at least the satellite running on the master's partition fulfill the requirement for freedom of interference. At least one other FiM satellite is running on an untrusted partition and does not fulfill the requirement for freedom of interference. Only specific memory section groups for variable data may have write access from the untrusted partition(s). Figure 2-4 depicts this situation.

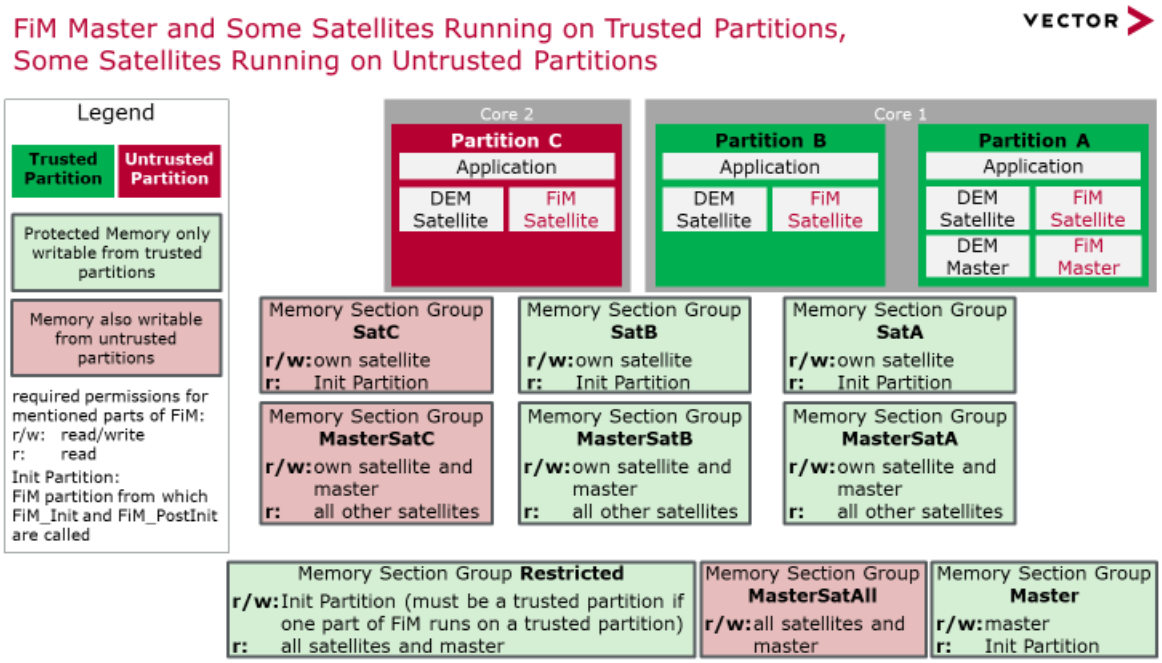

Figure 2-4 Memory Section Access: FiM Master and Some Satellites Running on Trusted Partitions, Some Satellites Running on Untrusted Partitions

2.3.3.4 FiM Master and Some Satellites Running on Untrusted Partitions, Some Satellites Running on Trusted Partitions

In this scenario the FiM master and at least the satellite running on the master's partition do not fulfill the requirement for freedom of interference. At least one other FiM satellite is running on a trusted partition and does fulfill the requirement for freedom of interference. Only specific memory section groups for variable data may have write access from the untrusted partition(s). Figure 2-5 depicts this situation.

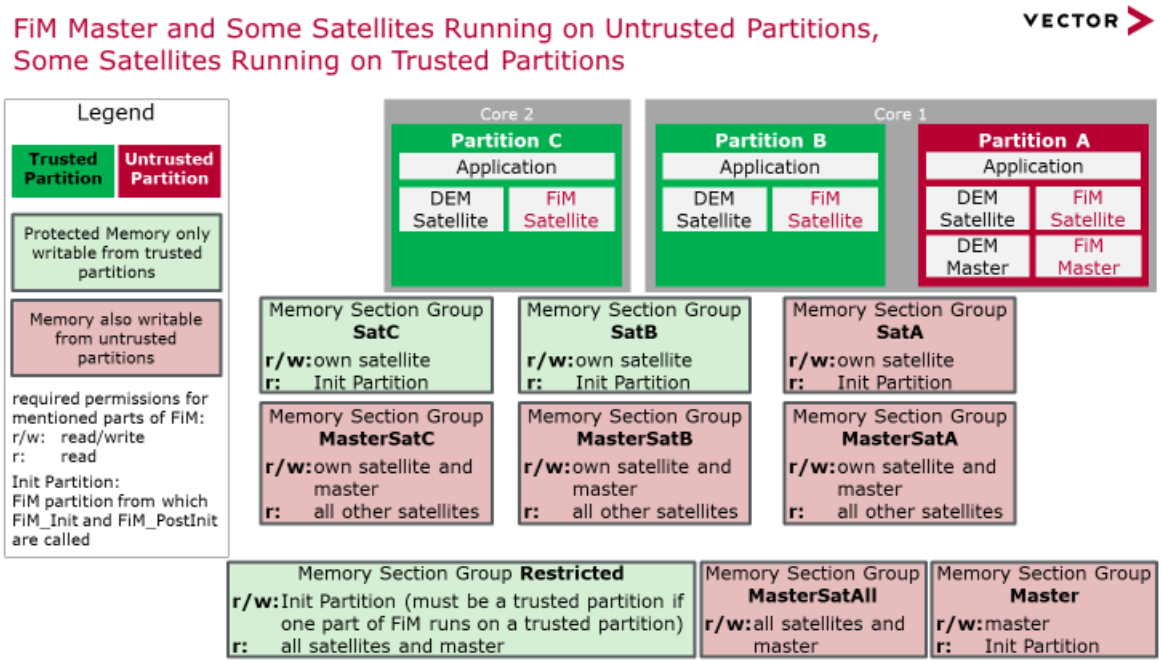

Figure 2-5 Memory Section Access: FiM Master and Some Satellites Running on Untrusted Partitions, Some Satellites Running on Trusted Partitions

Caution

For this use case the BSWM auto configuration is invalid and must be adapted manually.

Otherwise, FiM_Init() would be called from BSWM Main Partition (which in this use case is an untrusted partition).

That's because FiM automatically configures FiM_Init() independently from the use case in the EcuC Module in DaVinci Configurator Pro 5 and the initialization sequence created with the BSWM auto configuration is based on the information given in the EcuC Module.

2.4 Initialization

2.4.1 Multi Partition Use Case

Initialization of the FiM module is a multiple-step process.

1. (Optional step) Use FiM_InitMemory() to initialize static RAM variables in case the start-up code is not used to initialize RAM.
2. FiM_Init() must be called, see section 4.2.1 from which partition this must be done. Within this function the FiM does some basic pre-initialization for the FiM master and all satellites.
3. Use FiM_DemInit<…>() to initialize the FiM:
   1. DEM master has to call FiM_DemInitMaster() on the master partition.
   2. Each DEM satellite (if available, see caution box below) has to call FiM_DemInitSatellite() on each satellite partition.
4. Call FiM_PostInit() after master and all satellites have completed initialization. (see section 4.2.5 from which partition this must be done).

Caution
To allow for proper initialization of all FiM satellites, it is recommended that there is a DEM satellite configured on each OS partition where a FiM satellite is configured.

If a FiM satellite is configured on a partition without a DEM satellite FiM_DemInitSatellite() must be called from some other module on the same OS partition as the FiM satellite passing the OS application id of this partition.

Afterwards, FiM is ready for operation.

Caution
The FiM is not fully operational before

- FiM_Init() and
- FiM_DemInitMaster() and
- FiM_DemInitSatellite() for **all** FiM satellites and
- FiM_PostInit()

have been called.

See also section 2.5.1 Initialization States.

Note
As the OS might not be fully initialized during initialization of the FiM, it is not possible to check the validity of the call context.
Special care is needed to call the initialization functions from the correct partition:
- FiM_Init() and FiM_PostInit() only from the partition described in 4.2.1 and 4.2.5
- FiM_DemInitMaster() only from the master partition
- FiM_DemInitSatellite() only from the respective partition.

FiM_DemInitSatellite() is called by each DEM satellite (if available) for the FiM satellite on its own partition.
FiM_DemInitMaster() is called by DEM Master for FiM master that needs to reside on the same partition.

2.4.2 Single Partition Use Case

For configurations using a single OS partition, the standard AUTOSAR initialization sequence is still supported:

1. (Optional step) Use FiM_InitMemory() to initialize static RAM variables in case the start-up code is not used to initialize RAM.
2. FiM_Init() must be called. Within this function the FiM does some basic pre-initialization for the FiM master and the (single) satellite.
3. DEM master must initialize FiM master and the (single) FiM satellite using FiM_DemInit().

Note
The API FiM_DemInit() can be used to combine the initialization of master and satellite, but only in configurations with a single partition and thus with a single satellite.

Note
Instead of using the standard AUTOSAR initialization sequence by calling FiM_DemInit(), DEM master can instead call FiM_DemInitMaster(), FiM_DemInitSatellite() for the single FiM satellite and FiM_PostInit(), thus treating the single partition use case as a special form of the multi partition use case.

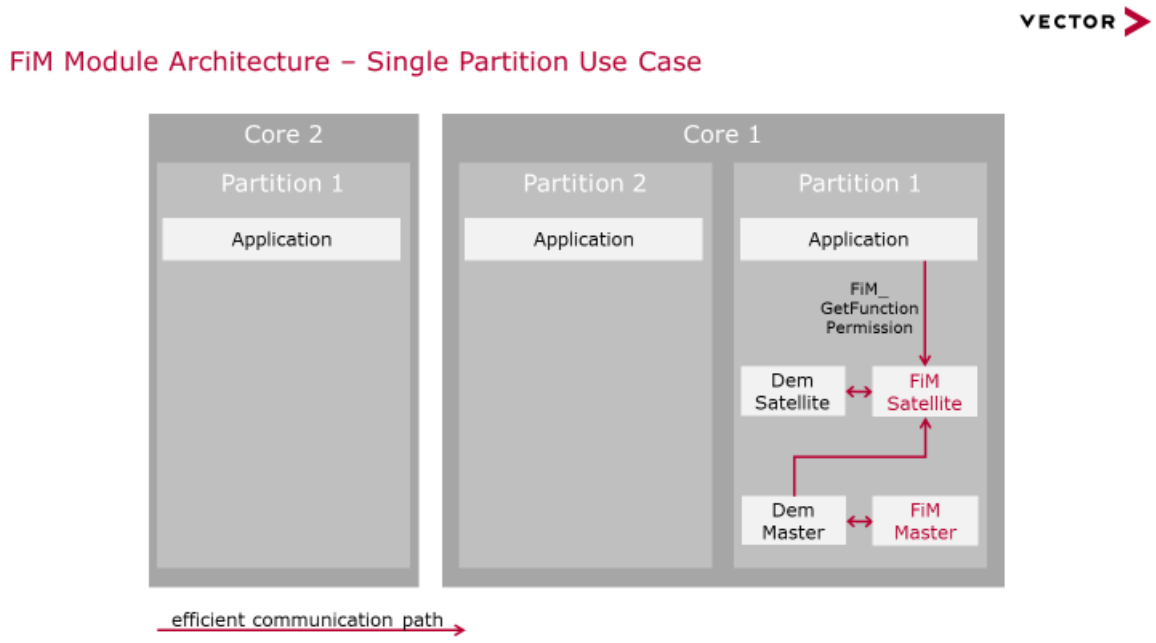

Figure 2-6 FiM Module Architecture (Single Partition Use Case)

Note

In the single partition use case, no OS application ID needs to be configured. Therefore, it is not possible to check the validity of the call context.

Special care is needed to call all functions from the master partition:

- FiM_DemTriggerOnMonitorStatus() and FiM_DemTriggerOnEventStatus()
- FiM_MainFunction()
- FiM_DemInit() (or FiM_DemInitSatellite() and FiM_DemInitMaster())
- FiM_Init() (and FiM_PostInit() if FiM_DemInit() is not used)

2.5 States

The FiM has the following internal states:

- inhibition state (one per event/inhibition combination)
- function pending state (one per FID that is connected to a pending configuration if OBD is enabled)
- initialization state of FiM master and of each FiM satellite

2.5.1 Initialization States

FiM master and each satellite have an initialization state of its own. It can be one of

- uninitialized
- pre-initialized
- initialized

At startup (or after calling FiM_InitMemory()) the initialization states of FiM master and all satellites are uninitialized.

After the call of FiM_Init() the initialization states of FiM master and all satellites are pre-initialized.

After the call of FiM_DemInitMaster() the initialization state of FiM master is initialized.

After the call of FiM_DemInitSatellite() the initialization state of the FiM satellite that this function was called for is initialized.

Note
FiM accepts triggers for monitor status changes if the corresponding satellite is initialized.

FiM accepts requests for function permissions if **all** satellites are initialized.

FiM accepts triggers for event status changes if the master is initialized.

For an overview see Figure 2-7.

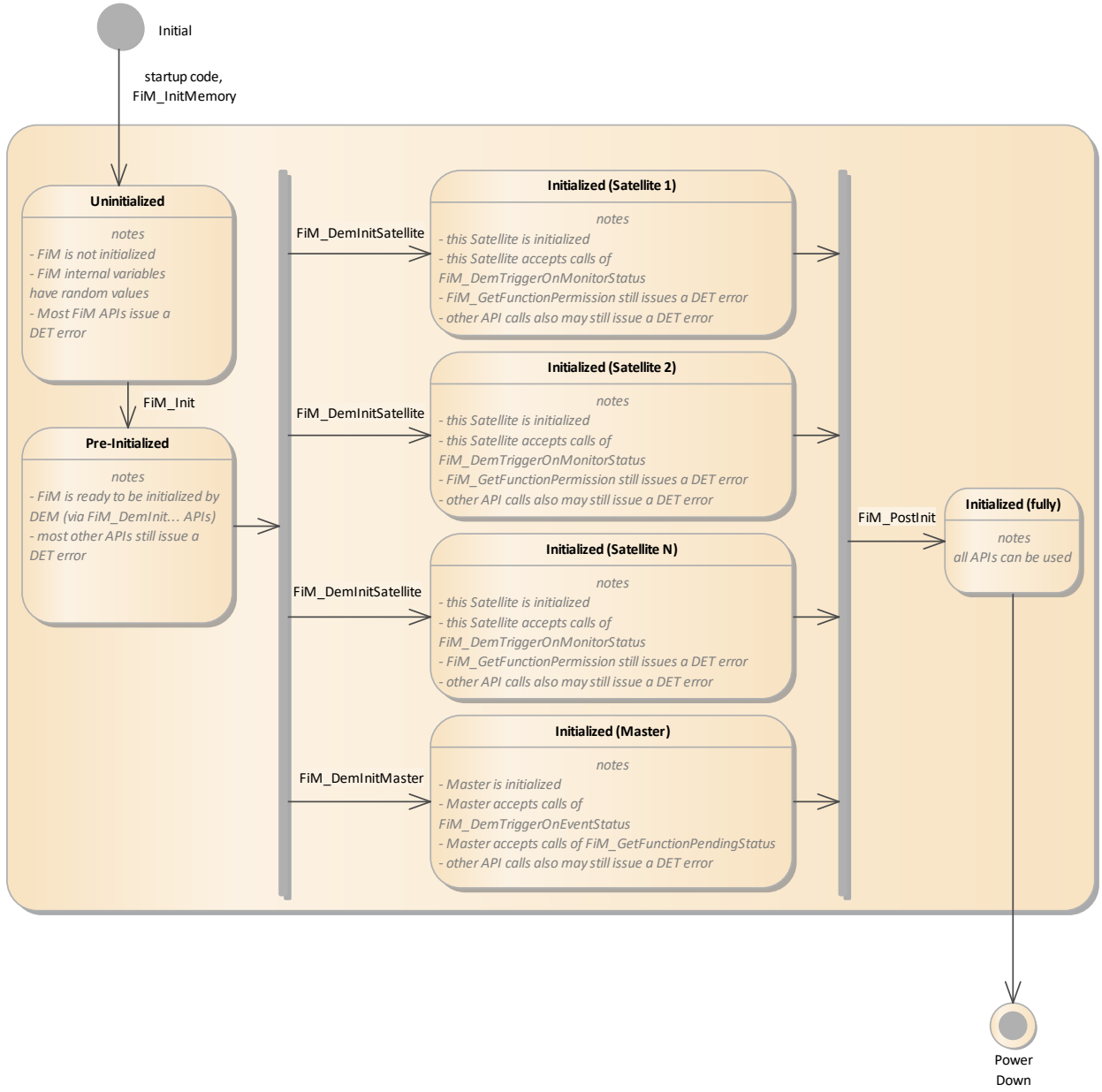

Figure 2-7 FiM Initialization States

2.6 Main Function

If cyclic FID calculation is configured the following applies:

- The main function calculates the FID permission status from the inhibition status of the event / inhibition code combinations that are linked to the FID.
- The inhibition status of the event / inhibition code combinations is calculated during calls of FiM_DemTriggerOnMonitorStatus() and FiM_DemInitSatellite().
- The number of FID permission states calculated during one call of FiM_MainFunction() can be limited by configuration.
- The calculation of FID permission states can be limited by configuration to those FIDs that are potentially affected by status changes of connected events since the last calculation.
- FiM_GetFunctionPermission() does not calculate the function permission itself but accesses the stored permissions calculated during calls of FiM_MainFunction().

Note
In general FiM_GetFunctionPermission() will need less runtime when cyclic FID calculation is enabled.

Note
The storage of FID permission states results in higher RAM usage.

Note
The option to calculate only potentially affected FIDs results in a higher RAM and a – potentially significantly – higher ROM usage.

Caution
If cyclic FID calculation is configured, the effects of a reported monitor status change to the function permissions are delayed up to several FiM_MainFunction() cycles.

If cyclic FID calculation is not configured the main function is empty and does not need to be scheduled.

See also section 4.2.12 for further information.

2.7 Error Handling

2.7.1 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 `FIM_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 FiM ID is 11 decimal.

The reported service IDs identify the services which are described in section 4.2. The following tables present service IDs, the related services and the errors reported to DET.

Service ID	Service
0x00	FiM_Init
0x01	FiM_GetFunctionPermission
0x02	FiM_DemTriggerOnMonitorStatus
0x03	FiM_DemInit
0x04	FiM_GetVersionInfo
0x05	FiM_MainFunction
0x80	FiM_GetFunctionPendingStatus
0x82	FiM_DemTriggerOnEventStatus
0x83	FiM_DemInitMaster
0x84	FiM_DemInitSatellite
0x85	FiM_PostInit

Table 2-4 Service IDs

Error Code		Description
0x01	FIM_E_UNINIT	API function is called before the FIM module has reached the necessary initialization status
0x02	FIM_E_FID_OUT_OF_RANGE	FiM_GetFunctionPermission/FiM_Get FunctionPendingStatus is called with wrong FID
0x03	FIM_E_EVENTID_OUT_OF_RANGE	Dem calls FiM with invalid/unconfigured EventId **Note:** This error code is **not used** by FiM.
0x04	FIM_E_PARAM_POINTER	API function is invoked with NULL Pointer
0x05	FIM_E_INIT_FAILED	initialization of FiM could not be completed, i.e. FiM_Init FiM_DemInit<…> or FiM_PostInit failed

Table 2-5 Errors reported to DET

2.7.2 Production Code Error Reporting

Production code related errors are not defined for the FiM.

3 Integration

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

3.1 Scope of Delivery

The delivery of the FiM consists of the files described in the following sections.

3.1.1 Static Files

File Name	Description
FiM.c	Main source code file.
FiM.h	Main header file.
FiM_Types.h	Header file containing FiM types.
FiM_<*>.c	Unit source code files.
FiM_<*>.h	Unit headers, do not include directly.
FiM_bswmd.arxml	Definition of FiM configuration parameters.

Table 3-1 Static files

3.1.2 Dynamic Files

File Name	Description
FiM_Cfg.h	Generated file that contains the configuration switches of the FiM.
FiM_Cfg_<*>.h	Generated files that contain unit specific access macros to configuration data.
FiM_Lcfg.h	Generated file that contains forward declarations for the configuration values/tables of the FiM.
FiM_Lcfg.c	Generated file that contains configuration values/tables of the FiM.
FiM_PBcfg.h	Generated file that contains forward declarations for the postbuild configuration values/tables of the FiM.
FiM_PBcfg.c	Generated file that contains post-buildable configuration values/tables of the FiM. For easier handling, this file is created in pre-compile configurations as well. If your build environment produces error messages due to this file not defining any symbols, please exclude it from the build.
FiM_MemMap.h	Generated file that provides FiM specific memory section mapping.
FiM_swc.arxml	Generated file that is used for the configuration of the RTE. It contains the information to get prototypes of callback functions offered by other components.

Table 3-2 Dynamic files

3.2 Compiler Abstraction and Memory Mapping

The objects (e.g., variables, functions, constants) are declared by compiler independent definitions – the compiler abstraction definitions. Each compiler abstraction definition is assigned to a memory section. The tables in the following sections contain the memory section names and the compiler abstraction definitions of the FiM and illustrate their assignment among each other.

For the purpose of explanation, the memory sections are grouped according to necessary access permissions. Within this document these groups are called **Memory Section Groups**. For the FiM the following groups can be identified:

- Memory Section Group **“Constant”**
- Memory Section Group **“Master”**
- Memory Section Group **“MasterSatAll”**
- Memory Section Group **“Restricted”**
- Memory Section Groups **“MasterSat<Name>”** (one per satellite)
- Memory Section Groups **“Sat<Name>”** (one per satellite)

Note
Depending on the partitioning use case, the memory sections must be mapped to the relevant parts in memory such that the access requirements explained in section 2.3.3 are fulfilled. From FiM version 10.00.00 onwards, the memory mapping must be done through the MICROSAR Classic component MemMap. For more detailed instructions, refer to [8].

3.2.1 Constant Sections (Memory Section Group “Constant”)

Table 3-3 shows the constant sections used by FiM.

All parts of the FiM need read access to these memory sections.

Compiler Abstraction Definitions / Memory Mapping Sections	FIM_CODE	FIM_CONST	FIM_PBCFG	FIM_PBCFG_ROOT
FIM_START_SEC_CODE FIM_STOP_SEC_CODE	■			
FIM_START_SEC_CONST_8 FIM_STOP_SEC_CONST_8		■		
FIM_START_SEC_CONST_16 FIM_STOP_SEC_CONST_16		■		
FIM_START_SEC_CONST_UNSPECIFIED FIM_STOP_SEC_CONST_UNSPECIFIED		■		
FIM_START_SEC_PBCFG FIM_STOP_SEC_PBCFG			■	
FIM_START_SEC_PBCFG_ROOT FIM_STOP_SEC_PBCFG_ROOT				■

Table 3-3 Compiler abstraction and memory mapping, (Memory Section Group “Constant”)

3.2.2 Variable Sections for FiM Master (Memory Section Group “Master”)

Table 3-4 shows variable memory sections of the Memory Section Group “Master”.

Required access permissions to the memory sections of this group are:

- read/write:
  - the partition that FiM master runs on
- read:
  - the partition that FiM_Init() and FiM_PostInit() are called from

Caution
If FiM master runs in a trusted partition and fulfills the requirement for freedom of interference, no other (untrusted) partition may have write access to the Memory Section Group “Master”.

Note
Since FiM master and DEM master must be mapped to the same OS partition, also DEM master has read/write access to these memory sections.

Memory Mapping Sections / Compiler Abstraction Definitions	FIM_VAR_NO_INIT	FIM_VAR_INIT	FIM_DEM_DATA	FIM_APPL_DATA
FIM_START_SEC_VAR_NO_INIT_16 FIM_STOP_SEC_VAR_NO_INIT_16	■			
FIM_START_SEC_VAR_NO_INIT_32 FIM_STOP_SEC_VAR_NO_INIT_32	■			
FIM_START_SEC_VAR_NO_INIT_UNSPECIFIED FIM_STOP_SEC_VAR_NO_INIT_UNSPECIFIED	■			
FIM_START_SEC_VAR_INIT_8 FIM_STOP_SEC_VAR_INIT_8		■		
DEM buffer used in communication (section depends on DEM implementation)			■	
Application or RTE buffer used in port communication (section depends on configuration and port mapping)				■

Table 3-4 Compiler abstraction and memory mapping, variable sections (Memory Section Group “Master”)

3.2.3 Variable Sections for FiM Master and all Satellites (Memory Section Group “MasterSatAll”)

Table 3-5 shows variable memory sections of the Memory Section Group “MasterSatAll”.

Required access permissions to the memory sections of this group are:

- read/write:
  - the partition(s) of FiM master and all satellites

Note
Since FiM master and DEM master must be mapped to the same OS partition, also DEM master has read/write access to these memory sections.

Memory Mapping Sections / Compiler Abstraction Definitions	FIM_VAR_NO_INIT	FIM_VAR_INIT	FIM_DEM_DATA	FIM_APPL_DATA
FIM_START_SEC_MASTERSATALL_VAR_NO_INIT_32 FIM_STOP_SEC_MASTERSATALL_VAR_NO_INIT_32	■			

Table 3-5 Compiler abstraction and memory mapping, variable sections (Memory Section Group “MasterSatAll”)

3.2.4 Variable Sections with Restricted Write Access (Memory Section Group “Restricted”)

Table 3-6 shows variable memory of the Memory Section Group “Restricted”.

Required access permissions to the memory sections of this group are:

- read/write:
  - the partition that FiM_Init() and FiM_PostInit() are called from
- read:
  - the partition(s) of FiM master and all satellites

Caution
If any part of the FiM runs in a trusted partition and fulfills the requirement for freedom of interference, then:

- no other (untrusted) partition may have write access to the Memory Section Group “Restricted” and
- the functions FiM_Init() and FiM_PostInit() must be called from one of FiM’s trusted partitions.

Memory Mapping Sections / Compiler Abstraction Definitions	FIM_VAR_NO_INIT	FIM_VAR_INIT
FIM_START_SEC_VAR_NO_INIT_16_RESTRICTED FIM_STOP_SEC_VAR_NO_INIT_16_RESTRICTED	■	
FIM_START_SEC_VAR_NO_INIT_UNSPECIFIED_RESTRICTED FIM_STOP_SEC_VAR_NO_INIT_UNSPECIFIED_RESTRICTED	■	
FIM_START_SEC_VAR_INIT_8_RESTRICTED FIM_STOP_SEC_VAR_INIT_8_RESTRICTED		■

Table 3-6 Compiler abstraction and memory mapping, variable sections (Memory Section Group “Restricted”)

3.2.5 Variable Sections for FiM Satellites and FiM Master (Memory Section Group “MasterSat<OS_APPLICATION_NAME>”)

Table 3-7 shows variable memory of the Memory Section Group “MasterSat<OS_APPLICATION_NAME>”, where “<OS_APPLICATION_NAME>” is the OS application name of the partition that the satellite is running on. There is one such group per satellite.

Required access permissions to the memory sections of this group are:

- read/write:
  - the partition that the FiM master is running on and
  - the partition that the satellite that the group stands for is running on
- read:
  - the partition(s) of all other satellites

Note
If a FiM satellite runs in a trusted partition and fulfills the requirement for freedom of interference, FiM master may still run in an untrusted partition and this untrusted partition has write access to the Memory Section Groups “MasterSat<Name>” of the FiM satellite running in a trusted partition.

Caution
This is only valid for the memory sections in the Memory Section Groups “MasterSat<Name>”. Do not allow write access from untrusted partitions to any other memory sections used by a part of the FiM that is running in a trusted partition.

Note
Since FiM master and DEM master must be mapped to the same OS partition, also DEM Master has read/write access to these memory sections.

Memory Mapping Sections / Compiler Abstraction Definitions	FIM_VAR_NO_INIT	FIM_APPL_DATA
FIM_START_SEC_INVALID_OSAPPLICATION_VAR_NO_INIT_32 FIM_STOP_SEC_INVALID_OSAPPLICATION_VAR_NO_INIT_32 FIM_START_SEC_<OS_APPLICATION_NAME>_VAR_NO_INIT_32 FIM_STOP_SEC_<OS_APPLICATION_NAME>_VAR_NO_INIT_32	■	
Application or RTE buffer used in port communication (section depends on configuration and port mapping)		■

Table 3-7 Compiler abstraction and memory mapping, variable sections (Memory Section Group “MasterSat<Name>”)

3.2.6 Variable Sections for FiM Satellites (Memory Section Group “Sat<OS_APPLICATION_NAME>”)

Table 3-8 shows variable memory of the Memory Section Group “Sat<OS_APPLICATION_NAME>”, where “<OS_APPLICATION_NAME>” is the OS application name of the partition that the satellite is running on. There is one such group per satellite.

Required access permissions to the memory sections of this group are:

- read/write:
  - the partition that the satellite that the group stands for is running on
- read:
  - the partition that FiM_Init() and FiM_PostInit() are called from

Caution
If a FiM satellite runs in a trusted partition and fulfills the requirement for freedom of interference, no other (untrusted) partition may have write access to its Memory Section Group “Sat<Name>”.

Memory Mapping Sections \ Compiler Abstraction Definitions	FIM_VAR_CLEARED
FIM_START_SEC_INVALID_OSAPPLICATION_VAR_CLEARED_8 FIM_STOP_SEC_INVALID_OSAPPLICATION_VAR_CLEARED_8 FIM_START_SEC_<OS_APPLICATION_NAME>_VAR_CLEARED_8 FIM_STOP_SEC_<OS_APPLICATION_NAME>_VAR_CLEARED_8	■

Table 3-8 Compiler abstraction and memory mapping, variable sections (satellites)

3.3 Synchronization

The FiM uses two mechanisms to maintain data consistency.

Where possible, the FiM uses a synchronization method based on atomic compare/exchange. Otherwise, the FiM relies on a critical section mechanism.

3.3.1 Atomic Compare/Exchange

Most hardware platforms supply instructions for efficient synchronization – test-and-set, compare-exchange or similar features. During integration, a suitable method for synchronization can be provided, e.g., based on a compiler intrinsic, or by a short inline function implementing the compare-exchange behavior using the mechanism provided by the hardware platform and compiler (see section 4.4.1.1).

As a fallback, the FiM supplements a default implementation using a critical section.

3.3.2 Critical Sections

To protect internal data structures against unwanted modification, the FiM uses “Critical Sections” for blocking concurrent access. AUTOSAR Schedule Manager APIs are used to handle critical sections (SchM_FiM.h is included).

Caution
You must take special care that the component implementing the critical sections (e.g., AUTOSAR Schedule Manager) is already started before the FiM is run.

FiM uses the following critical sections:

3.3.2.1 FIM_EXCLUSIVE_AREA_0

Length of locking is

- short during runtime to protect (re)setting a bit (only used if CompareAndSwap is not implemented by application)
- medium during initialization to protect the resetting of all inhibition states.

This Critical Section must **synchronize across all processor cores on which a FiM Satellite is located**, so it must be mapped to an adequate mechanism.

3.3.2.2 FIM_EXCLUSIVE_AREA_1

Length of locking is

- short during runtime to protect the inc-/decrementing of a counter,
- medium during initialization to protect the resetting of all counters.

This Critical Section must **synchronize locally on one processor core**, so it must be mapped to an adequate mechanism.

Note
Inter-core locking is not needed for FIM_EXCLUSIVE_AREA_1.

3.4 Integration into AUTOSAR 3 Stack

If the FiM is to be integrated into a stack according AUTOSAR 3 some manual adaptations need to be done that are described in this section.

Note
The FiM AUTOSAR 4 always requires a DEM according AUTOSAR 4. Integrating this DEM in an AUTOSAR 3 stack is not in the scope of this Technical Reference.

Note
Multi partitioning is not supported in AUTOSAR 3 environments.

Since the interface to the DEM is always according AUTOSAR 4 only the interfaces to the following modules need attention:

- RTE
- SchM

3.4.1 RTE

An AUTOSAR 3 RTE does not provide the necessary include file `Rte_FiM_Type.h`. Create it and include `Rte_Type.h`:

Rte_FiM_Type.h

```
#if !defined (RTE_FIM_TYPE_H)
# define RTE_FIM_TYPE_H
# include "Rte_Type.h"
#endif  /* RTE_FIM_TYPE_H */
```

Table 3-9 Rte_FiM_Type.h

3.4.2 SchM

3.4.2.1 Geny

The Vector MICROSAR Classic 3 SchM is configured in Geny.

In the SchM configuration create a BSW module support for FiM with two Exclusive Areas `FIM_EXCLUSIVE_AREA_0` and `FIM_EXCLUSIVE_AREA_1`.

3.4.2.2 DaVinci Configurator Pro 5

In DaVinci Configurator Pro 5 include a user configuration file in the FiM module containing:

User Configuration File

```
#define SchM_Enter_FiM_FIM_EXCLUSIVE_AREA_0() SchM_Enter_FiM(FIM_EXCLUSIVE_AREA_0)
#define SchM_Enter_FiM_FIM_EXCLUSIVE_AREA_1() SchM_Enter_FiM(FIM_EXCLUSIVE_AREA_1)
#define SchM_Exit_FiM_FIM_EXCLUSIVE_AREA_0()  SchM_Exit_FiM(FIM_EXCLUSIVE_AREA_0)
#define SchM_Exit_FiM_FIM_EXCLUSIVE_AREA_1()  SchM_Exit_FiM(FIM_EXCLUSIVE_AREA_1)
```

Table 3-10 User Configuration File

4 API Description

For an interface overview please see Figure 1-2.

4.1 Type Definitions

The types defined by the FiM are described in this chapter.

Type Name	C-Type	Description	Value Range
FiM_FunctionIdType	uint8, uint16	Type for the FunctionID	0…255 Size depends on number of FIDs.
			0…65535 Size depends on number of FIDs.

Table 4-1 Type definitions

4.2 Services provided by FiM

4.2.1 FiM_Init()

Prototype	
`void FiM_Init (const FiM_ConfigType* FiMConfigPtr)`	
Parameter	
`FiMConfigPtr`	pointer to a valid configuration for FiM
Return code	
`void`	N/A
Functional Description	
> sets up internal variables > the Function Inhibition Manager is not yet ready for operation afterwards > the FIM is set to state pre-initialized	
Particularities and Limitations	
> ServiceID = 0 > This function can be called from task context only. > If any part of the FiM runs in a trusted partition, this function may only be called from one of these trusted partitions. > This function is not reentrant. > This function is synchronous. > Precondition: Call Sequence 1: has to be called before the DEM is initialized > Precondition: Call Sequence 2: has to be called before other FiM APIs are called (except FiM_GetVersionInfo and FiM_InitMemory) > In the pre-compile variant, the ConfigPtr is unused.	

Table 4-2 FiM_Init()

4.2.2 FiM_DemInitMaster()

Prototype	
`void FiM_DemInitMaster (void)`	
Parameter	
`N/A`	N/A
Return code	
`void`	N/A
Functional Description	
> extension to Autosar. > this service (re-)initializes the FiM master > the FIM master is set to state initialized > called by DEM master during initialization after DEM master is prepared to receive queries for event status for initial FID pending status calculation > called by DEM master when the DEM detects a status change of a certain number of events (DEM implementation specific), e.g., after clearance of event memory	
Particularities and Limitations	
> ServiceID = 83 > This function can be called from task context only. > This function may only be called from partition that FiM (and DEM) master is running on. > This function is not reentrant. > This function is synchronous. > Precondition: Initialization: FiM master needs to be at least pre-initialized	

Table 4-3 FiM_DemInitMaster()

Caution
During the complete runtime of FiM_DemInitMaster, FiM_DemTriggerOnEventStatus must not be called!

Note: Calling of FiM_DemTriggerOnMonitorStatus is ok.

Caution
The FiM loops through all configured events and pending configurations and calls Dem_GetEventUdsStatus for each of them. Depending on the configuration, FiM_DemInitMaster can take a considerable time.

4.2.3 FiM_DemInitSatellite()

Prototype	
`void FiM_DemInitSatellite (ApplicationType applicationId)`	
Parameter	
`applicationId`	identification of satellite by assigned OS partition INVALID_OSAPPLICATION if no OS partition is assigned (only allowed in single partition use case)
Return code	
`void`	N/A
Functional Description	
> extension to Autosar. > this service (re-)initializes a FiM satellite > called by DEM satellite during initialization after DEM is prepared to receive queries for monitor status for initial inhibition status calculation > called by DEM satellite when the DEM detects a status change of a certain number of events (DEM implementation specific), e.g., after clearance of event memory	
Particularities and Limitations	
> ServiceID = 84 > This function can be called from task context only. > This function may only be called from the same partition as the DEM satellite identified via the parameter applicationId. > This function is not reentrant. > This function is synchronous. > Precondition: Initialization: FiM needs to be at least pre-initialized	

Table 4-4 FiM_DemInitSatellite()

Caution
The FiM loops through all configured events and inhibition configurations and calls Dem_GetMonitorStatus for each of them. Depending on the configuration, FiM_DemInitSatellite can take a considerable time.

Note
Usage of 0 instead of INVALID_OSAPPLICATION if no OS partition is assigned, has now been removed.

4.2.4 FiM_DemInit()

Prototype	
`void` **`FiM_DemInit`** `(void)`	
Parameter	
`N/A`	N/A
Return code	
`void`	N/A
Functional Description	
> this service (re-)initializes the FiM master and the only satellite > compatibility function for single partition use cases where there is only the master and one satellite > calls FiM_DemInitMaster(), FiM_DemInitSatellite() and FiM_PostInit()	
Particularities and Limitations	
> ServiceID = 3 > This function can be called from task context only. > This function may only be called from the partition that FiM master is running on and in single partition use case. > This function is not reentrant. > This function is synchronous. > Precondition: FiM master and satellite need to be at least pre-initialized > Precondition: Only one satellite is configured.	

Table 4-5 FiM_DemInit()

4.2.5 FiM_PostInit()

Prototype	
`void FiM_PostInit (void)`	
Parameter	
`N/A`	N/A
Return code	
`void`	N/A
Functional Description	
> extension to Autosar. > this service calculates and stores the cumulated initialization status of all FiM satellites > must be called during startup of BSW after master and all satellites have been initialized (except for single core systems that use FiM_DemInit())	
Particularities and Limitations	
> ServiceID = 85 > This function can be called from task context only. > If any part of the FiM runs in a trusted partition, this function may only be called from one of these trusted partitions. > This function is not reentrant. > This function is synchronous. > Precondition: Initialization: FiM master and all FiM satellites need to be initialized	

Table 4-6 FiM_PostInit()

4.2.6 FiM_DemTriggerOnMonitorStatus()

Prototype	
`void` **`FiM_DemTriggerOnMonitorStatus`** `(Dem_EventIdType EventId)`	
Parameter	
`EventId`	Identification of an Event by assigned event number. The Event Number is configured in the DEM. Min.: 1 (0: Indication of no Event or Failure) Max.: depending on configuration of Event Numbers in DEM (Max is either 255 or 65535)
Return code	
`void`	N/A
Functional Description	
> notifies the FiM that the monitor status of an event changed > called by DEM	
Particularities and Limitations	
> ServiceID = 2 > This function can be called from any context. > This function may only be called from the partition of the DEM satellite onto which the event is mapped or from the partition of the DEM master. > This function is reentrant. > This function is synchronous. > Precondition: Initialization: FiM has to be initialized	

Table 4-7 FiM_DemTriggerOnMonitorStatus()

Note
The function FiM_DemTriggerOnMonitorStatus has to look up the events that are configured in the FiM module. The search algorithm used depends on the configuration. If the event ids in FiM_EventIdTable are in continuously ascending order without gaps a direct table access is used. If they are in ascending order a binary search algorithm is used. Otherwise, a linear search is applied.

4.2.7 FiM_DemTriggerOnEventStatus()

Prototype	
`void FiM_DemTriggerOnEventStatus (Dem_EventIdType EventId, Dem_UdsStatusByteType EventStatusOld, Dem_UdsStatusByteType EventStatusNew)`	
Parameter	
`EventId`	Identification of an Event by assigned event number. The Event Number is configured in the DEM. Min.: 1 (0: Indication of no Event or Failure) Max.: depending on configuration of Event Numbers in DEM (either 255 or 65535)
`EventStatusOld`	extended event status before change
`EventStatusNew`	extended event status after change
Return code	
`void`	N/A
Functional Description	
> Extension to Autosar. > Notifies the FiM that the extended status of an event changed. > Called by DEM.	
Particularities and Limitations	
> ServiceID = 0x82 > This function can be called from any context. > This function may only be called from the partition that FiM (and DEM) master is running on. > This function is reentrant for different EventIds. > This function is synchronous. > Precondition: Initialization: FiM has to be initialized > Precondition: OBD license available	

Table 4-8 FiM_DemTriggerOnEventStatus()

Caution
FiM_DemTriggerOnEventStatus may not be interrupted by a call of FiM_DemTriggerOnEventStatus for the same event id!

Interruption of FiM_DemTriggerOnEventStatus by FiM_DemTriggerOnEventStatus is only allowed if the event ids of both calls differ.

Note
The function FiM_DemTriggerOnEventStatus has to look up the events that are configured in the FiM module. The search algorithm used depends on the configuration. If the event ids in FiM_EventIdPendingTable are in continuously ascending order without gaps a direct table access is used. If they are in ascending order a binary search algorithm is used. Otherwise a linear search is applied.

4.2.8 FiM_GetFunctionPermission()

Prototype	
`Std_ReturnType FiM_GetFunctionPermission (FiM_FunctionIdType FID, boolean* Permission)`	
Parameter	
`FID`	FunctionId as configured identifies a functionality Min.: 0 Max.: Result of configuration of FIDs in FIM (Max is either 255 or 65535)
`Permission`	TRUE: FID has permission to run FALSE: FID has no permission to run, i.e. shall not be executed
Return code	
`Std_ReturnType`	E_OK: The request is accepted E_NOT_OK: The request is not accepted, e.g. initialization of FIM not completed
Functional Description	
> Service reports the permission state of the functionality assigned to the FID. > Permission will be set to FALSE, if the FIM is not initialized or if the FID is not valid. > If development error reporting is enabled, an error will additionally be reported to the DET.	
Particularities and Limitations	
> ServiceID = 1 > This function can be called from any context. > This function is reentrant. > This function is synchronous. > Precondition: Initialization: FiM has to be initialized.	

Table 4-9 FiM_GetFunctionPermission()

Note
When no events are connected to the requested FID or all connected events are not available in DEM (signalized by DEM returning E_NOT_OK during Dem_GetMonitorStatus) FiM_GetFunctionPermission returns the FID as permitted.

Reasoning: No event inhibits the function.

Note
See section 2.6 for the effects of the configuration option “Cyclic FID Calculation” on this function.

4.2.9 FiM_GetFunctionPendingStatus()

Prototype	
`Std_ReturnType FiM_GetFunctionPendingStatus (FiM_FunctionIdType FID, boolean* pendingStatus)`	
Parameter	
`FID`	FunctionId as configured identifies a functionality Min.: 0 Max.: Result of configuration of FIDs in FIM (Max is either 255 or 65535)
`pendingStatus`	TRUE: any event connected to FID has status bit DEM_UDS_STATUS_PDTC set FALSE: no event connected to FID has status bit DEM_UDS_STATUS_PDTC set
Return code	
`Std_ReturnType`	E_OK: The request is accepted E_NOT_OK: The request is not accepted, e.g., initialization of FIM not completed
Functional Description	
> Extension to Autosar. > This function is used in context of IUMPR calculation for OBD. > Service reports the pending status of the event assigned to the FID. > Pending status will be set to FALSE, if the FIM is not initialized or if the FID is not valid. > If development error reporting is enabled, an error will additionally be reported to the DET. > See also [4] for a description of usage.	
Particularities and Limitations	
> ServiceID = 0x80 > This function can be called from any context. > This function is reentrant. > This function is synchronous. > Precondition: Initialization: FiM has to be initialized. > Precondition: OBD license available.	

Table 4-10 FiM_GetFunctionPendingStatus()

Note
When no events are connected to the requested FID or all connected events are not available in DEM (signalized by DEM returning E_NOT_OK during Dem_GetEventStatus) FiM_GetFunctionPendingStatus returns FALSE.

Reasoning: No event related to FID has a pending bit set.

4.2.10 FiM_GetVersionInfo()

Prototype	
`void` **`FiM_GetVersionInfo`** `(Std_VersionInfoType* versioninfo)`	
Parameter	
`versioninfo`	pointer to where to store the version information of this module
Return code	
`void`	N/A
Functional Description	
> This service returns the version information of the FiM. > The version information contains vendor ID, moduleID, major/minor/patch version number. > The version numbers are decimal coded. > This service is only available if enabled at compile time.	
Particularities and Limitations	
> ServiceID = 4 > This function can be called from any context. > This function is reentrant. > This function is synchronous. > Precondition: Configuration: FIM_VERSION_INFO_API == STD_ON	

Table 4-11 FiM_GetVersionInfo()

4.2.11 FiM_InitMemory()

Prototype	
`void` **`FiM_InitMemory`** `(void)`	
Parameter	
`N/A`	N/A
Return code	
`void`	N/A
Functional Description	
> Extension to Autosar. > Use this function to initialize static RAM variables of the master and satellites in case the start-up code is not used to initialize RAM.	
Particularities and Limitations	
> This function can be called from any context. > This function is reentrant. > This function is synchronous. > This function can only be called before memory protection is set up since it writes across partition boundaries.	

Table 4-12 FiM_InitMemory()

4.2.12 FiM_MainFunction()

Prototype	
`void` **`FiM_MainFunction`** `(void)`	
Parameter	
`N/A`	N/A
Return code	
`void`	N/A
Functional Description	

> If FiMCyclicFidCalculation is TRUE
  > Calculates the permission status of a number of FIDs from the inhibition status of the linked event / inhibition codes. This inhibition status is calculated during the call of FiM_DemTriggerOnMonitorStatus().
  > The number of FID permission states calculated during one call of FiM_MainFunction() can be limited using the parameter FiMMaxHandledFidsPerCycle
  > The calculation of FID permission states can be limited by configuration to those FIDs that are potentially affected by status changes of connected events since the last calculation (parameter FiMAffectedFIDsIdentification).
> If FiMCyclicFidCalculation is FALSE
  > Main function is empty and does not need to be scheduled.	

Particularities and Limitations	
> ServiceID = 5 > This function can be called from task context only. > This function may only be called from master partition. > This function is not reentrant. > This function is synchronous. > Precondition: Initialization: FiM has to be initialized	

Table 4-13 FiM_MainFunction()

4.3 Services used by FiM

Table 4-14 lists services which are provided by other components and are used by FiM. For details about prototype and functionality refer to the documentation of the providing component.

Component	API
Det	Det_ReportErrorStatus()
Dem	Dem_GetMonitorStatus()
Dem	Dem_GetEventUdsStatus()
EcuM	optional EcuM_BswErrorHook

Table 4-14 Services used by the FiM

4.3.1 EcuM_BswErrorHook()

Prototype	
`void` **`EcuM_BswErrorHook`** `(uint16 BswModuleId, uint8 ErrorId)`	
Parameter	
`BswModuleId`	Autosar ModuleId. The Dem will pass `FIM_MODULE_ID`.
`ErrorId`	Error code detailing the error cause, see Table 5-5
Return code	
`void`	N/A
Functional Description	
This function is called to report defunct configuration data passed to FiM_Init. The FiM will leave FiM_Init after a call to this function, without initializing. Further calls to the FiM module are not safe.	
Particularities and Limitations	
> This function is called in error cases, when initializing a Post-Build configuration fails. > It's not safe if it returns to the caller, especially if development error detection is disabled. > This function is called from FiM_Init().	

Table 4-15 EcuM_BswErrorHook()

4.4 Configurable Interfaces

4.4.1 Callouts

4.4.1.1 ApplFiM_SyncCompareAndSwap()

Caution
If a hardware specific operation is used to implement the CompareAndSwap functionality, it must be ensured that this operation works atomically also across processor cores for the full width of an uint32 on the chosen derivative. E.g., some PPC derivatives are known not to have implemented reservation logic between cores (e.g., for lwarx/stwcx). Other PPC operations that use the Decorated Storage Memory Controller (DSMC) may only synchronize 28 bits instead of 32 bits.

Prototype	
`boolean ApplFiM_SyncCompareAndSwap(volatile uint32* AddressPtr, uint32 OldValue, uint32 NewValue)`	
Parameter	
`AddressPtr`	Memory location to modify. The pointer is aligned to uint32. The pointer is only valid until the function returns.
`OldValue`	The comparison value
`NewValue`	The value to write
Return code	
TRUE	The new value was written to AddressPtr
FALSE	The new value was not written to AddressPtr
Functional Description	

The function is expected to implement the following operation **atomically also across processor cores**:
IF value at location AddressPtr EQUALS OldValue:
  STORE NewValue at location AddressPtr
  RETURN TRUE
OTHERWISE:
  RETURN FALSE
Many hardware platforms provide an operation which allows to implement this with a single instruction.	

Particularities and Limitations	
> This function shall be reentrant > This function shall be synchronous > Providing this function is optional, the FiM can use a default implementation. Since the default implementation is not optimized for the current platform, it is strongly recommended to use this callout. The configuration parameter `FiMGeneral/FiMUserDefinedCompareAndSwap` alters between external and internal implementation. > When using the external callout function, the FiM typically needs a function prototype which is delivered by adding an include file via parameter FiMGeneral/FiMHeaderFileInclusion, e.g., Appl_FiM.h.	
Call context	
> This function is called from APIs with unrestricted call context.	

Table 4-16 ApplFiM_SyncCompareAndSwap()

4.5 Service Ports

Note
Service ports can only be used when RTE support is configured.

4.5.1 Client Server Interface

A client server interface is related to a Provide Port at the server side and a Require Port at the client side.

4.5.1.1 Provide Ports on FiM Side

At the Provide Ports of the FiM the API functions described in section 4.2 are available as Runnable Entities. The Runnable Entities are invoked via Operations. The mapping from a SWC client call to an Operation is performed by the RTE. In this mapping the RTE adds Port Defined Argument Values to the client call of the SWC, if configured.

The following sub-chapters present the Provide Ports defined for the FiM and the Operations defined for the Provide Ports, the API functions related to the Operations and the Port Defined Argument Values to be added by the RTE.

4.5.1.1.1 FunctionInhibition

Operation	API Function	Port Defined Argument Values
FunctionInhibition	Fim_GetFunctionPermission	FunctionIdType FunctionId

Table 4-17 FunctionInhibition

Only available on FiM satellites for which a FID is configured.

4.5.1.2 Require Ports on FiM Side

At its Require Ports the FiM calls Operations. These Operations have to be provided by the SWCs by means of Runnable Entities. These Runnable Entities implement the callback functions expected by the FiM.

Note
Currently, the FiM does not have Require Ports.

5 Configuration

5.1 Configuration Variants

The FiM supports the configuration variants

- VARIANT-PRE-COMPILE
- VARIANT-POST-BUILD-LOADABLE

The configuration classes of the FiM parameters depend on the supported configuration variants. For their definitions please see the fim_bswmd.arxml file.

5.2 Configurable Attributes

Each configurable option is described within its online help in the DaVinci Configurator Pro 5 tool.

The parameters in Table 5-1 can be configured via the tool.

Configuration Parameter	Comment	Post-Build Loadable
FiMDataFixed	FALSE: calibration off TRUE: not supported	no
FiMEventUpdate TriggeredByDem[1]	TRUE: trigger on event active FALSE: not supported	no
FiMOptimizationFor InhibitionMasks	If enabled, code generator will try to optimize usage of inhibition masks. E.g., inhibition configurations with identical event id and function id are combined to a single configuration (e.g., FIM_NOT_TESTED/ FIM_LAST_FAILED to FIM_NOT_TESTED_OR_FAILED).	yes
FiMMax FidsPerEvent	If component vPblCalib is used (see section 5.3.1) this parameter determines the maximum number of inhibition configurations that can be calibrated for each DEM event. If component vPblCalib is not used this parameter is ignored.	no
FiMMax NrInhStates	Defines maximum number of inhibition states that can be stored in RAM (per satellite). This number is important for adding additional inhibition configurations during post-build phase.	no
FiMCyclic FidCalculation	How FID permission is calculated from event's inhibition status: TRUE: done on FiM main task for a number of FIDs FALSE: done during call to FiMGetFunctionPermisssion for requested FID	no
FiMAffectedFIDs Identification	TRUE: only those FIDs are calculated during cyclic calculation that might be affected by a changed event status. Upon call of FiM_DemTriggerOnMonitorStatus, those FIDs are tagged as affected that are connected to the passed event (independent of inhibition configuration or actual status change). During cyclic	no

[1] Translates to FIM_CYCLIC_EVENT_EVALUATION in code with inverted meaning.

Configuration Parameter	Comment	Post-Build Loadable
	calculation only FIDs are calculated that were tagged since the last calculation of that FID. Note: Memory consumption may increase significantly when enabling this option. FALSE: FIDs are calculated independent of whether the status potentially changed or not	
FiMMaxHandledFidsPerCycle	Maximum number of FIDs that are calculated during one call of FiM_MainFunction when cyclic FID calculation is selected.	yes

Table 5-1 Configurable Parameters

The following configuration parameters are currently not supported and therefore ignored:

- FiMMaxTotalLinks
- FiMMaxSummaryLinks
- FiMMaxSummaryEvents
- FiMMaxEventsPerFid
- FiMMaxEventFidLinks

5.2.1 Inhibition Configuration Codes

Inhibition configurations use event id / inhibition configuration code / FID triple combinations. Inhibition configuration codes are shown in Table 5-2.

Inhibition Configuration Code	Referenced Inhibition Configuration	Remarks
0x00	invalid	used to disable event/FID link
0x01	FIM_LAST_FAILED	DEM_MONITOR_STATUS_TF flag of DemMonitorStatus is set
0x02	FIM_NOT_TESTED	DEM_MONITOR_STATUS_TNCTOC flag of DemMonitorStatus is set
0x03	FIM_TESTED	DEM_MONITOR_STATUS_TNCTOC flag of DemMonitorStatus is not set
0x04	FIM_TESTED_AND_FAILED	DEM_MONITOR_STATUS_TF flag of DemMonitorStatus is set and DEM_MONITOR_STATUS_TNCTOC flag is not set

Table 5-2 Inhibition Configuration Codes

Note
For optimization purposes the FiM internally handles additional codes. These cannot be configured explicitly. Rather they are generated automatically.

5.3 Measurement and Calibration

5.3.1 Generation of A2L Measurements

The FiM module supports usage of MC Support Data for certain FiM variables. This allows A2L generation with the McDataConverter that is provided as part of DaVinci Configurator Pro 5. The measurements will be included in the generated file McData.a2l. To enable this feature, the “Generate Debug data” generation option needs to be set in DaVinci Configurator Pro 5. For more information regarding AUTOSAR A2L file creation, see [7].

Note
The creation of Debug Data requires an AMD.Dbg license.

The following measurable objects are generated when the option to generate debug data is enabled.

FiM FID Pending Counter		
Measurement Item	Measurement name	Description
FiM FID Pending Counter	FiM_FidPendingCounter_<FidShortname>	Pending Counter for each FID containing the number of events that currently lock the corresponding IUMPR ratio.

Table 5-3 FiM FID Pending Counter

FiM FID permission states can be measured per FID or via bitfields:

FiM FID Permission State		
Measurement Item	Measurement name	Description
Permission State of a FiM FID	FiM_Permission_<Number>.<FidShortname>	The permission state contains the information whether a functionality represented by its FID can be executed. If the permission state is TRUE, the functionality associated with the FID is permitted to be executed. If the permission state is FALSE, the functionality associated with the FID is not allowed to be executed.
Bitfield with FiM FID Permission States	FiM_PermissionWord_<Number>	The bitfield contains the permission states of up to 32 FIDs. The names of these FIDs are listed in the description string of the bitfield with the last FID in the string corresponding to the least significant bit. Each used bit indicates the permission state of a FID (TRUE if the bit is set, FALSE if the bit is not set).

Table 5-4 FiM FID Permission State

Caution
Note that each measurement name must be a valid AUTOSAR name with maximum 128 characters. If this limit is exceeded, the measurement name will be hashed and shortened so that its length does not exceed this limit.

5.3.2 Calibration

The FiM does not support a direct calibration via a calibration tool.

Post-Build Loadable has to be used instead, see section 5.4 for details.

The component vPblCalib can be used to connect to a standard compliant calibration tool and hide the Post-Build Loadable process from the user. See [6] for details.

5.4 Post-Build support

5.4.1 Post-Build Loadable

Although calibration already is a post build method of configuration, Vector provides a tool-based approach superior to calibration. While calibration only modifies existing configuration tables, the Post-Build Loadable approach also allows to validate the configuration change preventing misconfiguration, and to use compacted table structures – with benefits to run-time and ROM usage.

Note
Adding new FIDs to an existing configuration during Post-Build is not supported. If you have ‘inactive’ functions that are enabled by calibration or other means, set up the FID for this function at pre-compile time and disable it by not configuring an event-to-FID link to it.

5.4.1.1 Initialization

During the startup of the ECU, the FiM expects to receive the pointer to its configuration data in FiM_Init(). Typically, this pointer is passed by the MICROSAR Classic EcuM based on the post-build configuration. If no MICROSAR Classic EcuM is used, the procedure of how to find the proper initialization pointer is out of scope of this document.

The FiM module will verify the received pointer for three criteria before it is accepted to initialize the module. If the initialization fails, an EcuM error hook (see chapter 4.3.1) is called with an error code according to Table 5-5.

Error Code	Reason
ECUM_BSWERROR_NULLPTR	Initialization with a null pointer.
ECUM_BSWERROR_MAGICNUMBER	Magic pattern check failed. This pattern is appended at the end of the initialization root structure. An error here is a strong indication of random data, or a major incompatibility between the code and the configuration data.
ECUM_BSWERROR_COMPATIBILITYVERSION	The configuration data was created by an incompatible generator. This is also tested by verification of a ‘magic’ pattern, so initialization with random data can also cause this error code.

Table 5-5 Error Codes possible during Post-Build initialization failure

If no MICROSAR Classic EcuM is used, these error hooks and the error code constants have to be provided by the environment.

1. If the pointer equals NULL_PTR, initialization is rejected.
2. If the initialization structure does not end with the correct magic number, it is rejected.

3. If the initialization structure was created by an incompatible generator version, it is rejected (starting magic number check)

Caution
The verification steps performed during initialization are neither intended nor sufficient to detect corrupted configuration data. They are intended only to detect initialization with a random pointer, and to reject data created by an incompatible generator version.

5.4.1.2 Configuration of Post-Build Loadable

The configuration of post-build loadable is described in [5].

The FiM supports several configuration data elements for post-build loadable. Besides parameters marked in Table 5-1 this also includes:

- event ids known by FiM
- event-to-FID links
- inhibition configurations
- pending configurations

Note
Inhibition configurations can only reference events if there is a FiM satellite configured on the OS partition onto which the event is mapped in the DEM configuration.

Since FiM satellites cannot be changed during the post-build phase, no reference to an event can be added to the FiM configuration during post-build phase if an appropriate FiM satellite had not been configured at pre-compile time.

It is therefore recommended to configure a FiM satellite on each OS partition where a DEM satellite is configured.

6 AUTOSAR Standard Compliance

6.1 Deviations

See Table 2-2 Not supported AUTOSAR standard conform features.

6.2 Additions/ Extensions

FiM_InitMemory, see sections 4.2.11

FiM_DemInitMaster, see section 4.2.2.

FiM_DemInitSatellite, see section 4.2.3.

FiM_PostInit, see section 4.2.5.

FiM_GetFunctionPendingStatus, see section 4.2.9.

FiM_DemTriggerOnEventStatus, see section 4.2.7.

6.3 Limitations

Limitations are not known.

7 Glossary and Abbreviations

7.1 Glossary

Term	Description
DaVinci Configurator Pro 5	Configuration and generation tool for MICROSAR Classic components

Table 7-1 Glossary

7.2 Abbreviations

Abbreviation	Description
API	Application Programming Interface
AR	AUTOSAR
AUTOSAR	Automotive Open System Architecture
BSW	Basis Software
DEM	Diagnostic Event Manager
DET	Development Error Tracer
ECU	Electronic Control Unit
EcuM	ECU Manager
FiM	Function Inhibition Manager
HIS	Hersteller Initiative Software
ISR	Interrupt Service Routine
IUMPR	In Use Monitor Performance Ratio
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
OBD	Onboard Diagnostics
PPort	Provide Port
RPort	Require Port
RTE	Runtime Environment
SchM	Schedule Manager
SRS	Software Requirement Specification
SWC	Software Component
SWS	Software Specification

Table 7-2 Abbreviations

8 Contact

Visit our website for more information on

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

www.vector.com