Document Title	Specification of Function Inhibition Manager
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	082

Document Status	Final
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	4.3.1

Document Change History			
Date	**Release**	**Changed by**	**Description**
2017-12-08	4.3.1	AUTOSAR Release Management	• Minor corrections / clarifications / editorial changes; For details please refer to the ChangeDocumentation
2016-11-30	4.3.0	AUTOSAR Release Management	• Renaming of Event Status to Monitor Status following redesign of Dem/ DCM interface • Changed usage of Dem_GetEventStatus to Dem_GetMonitorStatus and renamed FiM_DemTriggerOnEventStatus to FiM_DemTriggerOnMonitorStatus Interfaces following redesign of Dem/ DCM interface • Removed requirement SWS_Fim_00073 • minor corrections / clarifications / editorial changes; For details please refer to the ChangeDocumentation
2015-07-31	4.2.2	AUTOSAR Release Management	• Fim considers EventAvailbilty/ EventSuppression • Modified Initialization Sequence • minor corrections / clarifications / editorial changes; For details please refer to the ChangeDocumentation

2014-10-31	4.2.1	AUTOSAR Release Management	• Simplification of FiM configuration • Support of "Monitored Components" • Postbuild configuration clean up • Editorial changes
2014-03-31	4.1.3	AUTOSAR Release Management	• Revised development error codes • Editorial changes
2013-10-31	4.1.2	AUTOSAR Release Management	• Change containers FiMFID and FiMInhibitationConfiguration • Editorial changes • Removed chapter(s) on change documentation
2013-03-15	4.1.1	AUTOSAR Administration	• Apply new requirement format and requirement IDs (leading zeros to reach 5 digits) • Move general requirements to AUTOSAR_SWS_BSWGeneral [1] • Add formal description of the Standardized AUTOSAR Interface for the Fim service. Types are formalized so that the types generated by the RTE can be used for the Fim APIs.
2011-12-22	4.0.3	AUTOSAR Administration	• Renaming of FiMCyclicEventEvaluation configuration parameter into FiMEventUpdateTriggeredByDem • Reformulation of [SWS_Fim_00070], SWS_Fim_00073 • Inhibition masks use TestFailed bit instead of TestFailedThisOperationCycle • File structure schema changed • Initialization sequence diagram added • Remove development error FIM_E_EVENTID_OUT_OF_RANGE

2010-09-30	3.1.5	AUTOSAR Administration	• Intra module checks updated • Corrected multiplicity of configuration parameters FiMInhChoicedemRef and FiMInhChoiceSumRef • Introduction of ImplementationDataType replacing IntegerType and Boolean • Clarification of chapter describing interaction between Dem and FiM 7.2.2.2 • Relocation of [SWS_Fim_00067] explaining evaluation by the FiM of Dem events • Addition of a new requirement describing the standardized AUTOSAR interface [SWS_Fim_00090]
2010-02-02	3.1.4	AUTOSAR Administration	• Legal disclaimer revised
2008-08-13	3.1.1	AUTOSAR Administration	• OBD related chapter added 7.2.3 • Corrected error description • Legal disclaimer revised
2007-12-21	3.0.1	AUTOSAR Administration	• Error classification extended to report invocation with NULL pointer • Corrected InternalBehavior of FiM to fit to API's reentrant behavior • Minimum value of parameter FimMaxSummaryLinks fixed • Document meta information extended • Small layout adaptations made
2007-01-24	2.1.15	AUTOSAR Administration	• "Advice for users" revised • "Revision Information" added

	2.1.14	AUTOSAR Administration	• Modification of the FiM data structure: Several summarized events can be assigned to the FimInhibition-Configuration • Inserted corrected sequence charts for FiM initialization phase and `FiM_DemTriggerOnEventStatus` • Added file MemMap.h to header file structure • Added requirement for extended header file structure (Schedule Manager) • Added SchM_FiM.h to header file structure • Legal disclaimer revised
2006-05-16	2.0	AUTOSAR Administration	Initial Release

Disclaimer

Table of Contents

1 Introduction and functional overview

The Function Inhibition Manager is responsible for providing a control mechanism for software components and the functionality therein. In this context, a functionality can be built up of the contents of one, several or parts of runnable entities with the same set of permission / inhibit conditions. By means of the FiM, inhibiting (deactivation of application function) these functionalities can be configured and even modified during runtime (post-built configuration).

Functionality and runnable entity are different and independent types of classifications. Runnable entities are mainly characterized by their scheduling requirements. In contrast to that, functionalities are classified by their inhibit conditions. The services of the FiM focus on functionalities in SW-Cs, however, they are not limited to them. Functionalities of the BSW can also use the FiM services.

The functionalities are assigned to an identifier (FID - function identifier) along with the inhibit conditions for that particular identifier. The functionalities poll for the permission state of their respective FIDs before execution. If an inhibit condition comes true for a particular identifier, the corresponding functionality shall not be executed anymore.

The FiM is closely related to the Dem since diagnostic events and their status information are supported as inhibit conditions. Hence, functionality which needs to be stopped in case of a failure, e.g. of a certain sensor, can be represented by a particular identifier. If the failure is detected and the event is reported to the Dem, the FiM then inhibits the FID and therefore the corresponding functionality.

In order to handle the relation of functionality and linked events, the identifier and inhibit conditions of the functionality have been introduced into the SW-C template (equivalence for BSW) and during configuration, data structures are built up to deal with the sensitiveness of the identifiers against certain events

Software components can be integrated into a new environment as a collection of events which can be configured without big effort. Furthermore, system analysis is supported when questions as, for example, "Which functionality is inhibited if a particular event is detected?" arise. The data basis of the FiM serves as documentation of the configured relations between events and the SW-C to be inhibited.

In AUTOSAR, the RTE deals with SW-C in terms of their interfaces and scheduling requirements. In contrast to that, the FiM deals with inhibit conditions and provides supporting mechanisms for controlling functionalities via respective identifiers (FID). Therefore, the FiM concept and RTE concept do not interfere with each other.

The basic targets of the FiM specification document are:

- Standardization of APIs
- Introduction of possible implementation approaches
- Provide the ability for a common approach of OEM and supplier

2 Acronyms and abbreviations

Abbreviation / Acronym:	**Description:**
Activity state	The activity state is the status of a software component being executed. The activity state results from the permission state as a precondition and physical enable condition, too. It is not calculated by the FiM and not available as a status variable. It can only be derived from local information within a software component. For further details, see chapter 7.2.1.6.
API	Application Programming Interface
BSW	Basic Software
Dem	Diagnostic Event Manager
ECU	Electronic Control Unit
FID	Function Identifier
FiM	Function Inhibition Manager
Functionality	Functionality comprises User-visible and User-non-visible functional aspects of a system (AUTOSAR_Glossary.pdf [2]). In addition to that - in the FiM context - a functionality can be built up of the contents of one, several or parts of runnable entities with the same set of permission / inhibit conditions. By means of the FiM, the inhibition of these functionalities can be configured and even modified by calibration. Each functionality is represented by a unique FunctionId. A functionality is characterized by a specific set of inhibit condition in contrast to runnable entities having specific scheduling conditions.
HW	Hardware
ID	Identification/Identifier
Inhibition Condition	The relation between one FID, an inhibition mask and the status of a Dem event/component. (see FiMInhibitionConfiguration)
ISO	International Standardization Organization
MIL	Malfunction Indication Light
Monitoring function	• Part of the Software Component. • Mechanism to monitor and finally to detect a fault of a certain sensor, actuator or could be a plausibility check • Reports states about events from internal processing of a SW-C or from further processing of return values of other basic software modules. • See also AUTOSAR_SWS_DiagnosticEventManager [3]
NVRAM	Non volatile Memory
OBD	On-board Diagnostics
OBDII	Emission-related On-board Diagnostics
OEM	Original Equipment Manufacturer
OS	Operating System
Permission state	The permission state contains the information whether a functionality, represented by its FID, can be executed or whether it shall not run. The state is controlled by the FiM based on reported events. For further details, see chapter 7.2.1.6.
RAM	Random Access Memory
ROM	Read-only Memory
RTE	Runtime Environment
Runnable entity	A Runnable Entity is a part of an Atomic Software-Component, which can be executed and scheduled independently from the other Runnable Entities of this Atomic Software-Component. It is described by a sequence of instructions that can be started by the RTE. Each runnable entity is associated with exactly one EntryPoint.
SW-C	Software Component
UDS	Unified Diagnostic Services
WP	Autosar Work Package

Abbreviation / Acronym:	Description:
Xxx_	Placeholder for an API provider

Table 2.1: Abbreviations and Acronyms

3 Related documentation

3.1 Input documents

[1] General Specification of Basic Software Modules
AUTOSAR_SWS_BSWGeneral

[2] Glossary
AUTOSAR_TR_Glossary

[3] Specification of Diagnostic Event Manager
AUTOSAR_SWS_DiagnosticEventManager

[4] Requirements on Function Inhibition Manager
AUTOSAR_SRS_FunctionInhibitionManager

[5] Virtual Functional Bus
AUTOSAR_EXP_VFB

[6] Software Component Template
AUTOSAR_TPS_SoftwareComponentTemplate

3.2 Related standards and norms

[13] IEC 7498-1 The Basic Model, IEC Norm, 1994

[14] D1.5-General Architecture; ITEA/EAST-EEA, Version 1.0; chapter 3, page 72 et seq.

[15] D2.1-Embedded Basic Software Structure Requirements; ITEA/EAST-EEA, Version 1.0 or higher

[16] D2.2-Description of existing solutions; ITEA/EAST-EEA, Version 1.0 or higher.

3.3 Related specification

AUTOSAR provides a General Specification on Basic Software modules [1, SWS BSW General], which is also valid for Function Inhibition Manager.

Thus, the specification SWS BSW General shall be considered as additional and required specification for Function Inhibition Manager.

4 Constraints and assumptions

[SWS_Fim_00007] ⌈ FID numbers shall be unique per FiM.⌋ *(SRS_Fim_04701)*

Since communication between software components and basic software is limited to one ECU, the FiM can only control FIDs being located on the same ECU. Note that the RTE does currently not support communication between basic software and software components located on different ECUs.

4.1 Limitations

Timing constrains have to be considered for the whole system. Note that the process and response times strongly depend on the implementation of the FiM module. Hence, if there are explicit needs for faster responses of the FiM than the cycle (time slice of the task) these needs have to be considered by the FiM implementation specifically by the affected application. Special measures have to be implemented by the FiM which are not explicitly specified in this AUTOSAR document, since here, the implementation is - on purpose - not prescribed.

[SWS_Fim_00043] ⌈ The FiM shall compute the permission of a FID independently of the state of other FIDs.⌋ *(SRS_Fim_04706)*

Interdependencies between FIDs are not supported by the FiM. That means an FID does not influence another FID.

4.2 Applicability to car domains

The FiM is designed to fulfill the design demands for ECUs with respect to a central handling of reactions of the system upon detected malfunctions, e.g. open circuit or shortcut. Therefore, the immediate domain of applicability of the FiM is currently body, chassis and powertrain ECUs. However, there is no reason that the FiM cannot be used in implementations of ECUs for other car domains as, for example, infotainment.

One major constraint is that the FiM alone will NOT be able to handle SW-Components that are:

1. time critical - They might be too slow for local reconfigurations (fast backup reaction in case of e.g. invalid signals).

2. physically interactive - They might not be sufficiently flexible.

3. safety critical - They might not have sufficient software integrity.

5 Dependencies on other modules

[SWS_Fim_00044] ⌈ The AUTOSAR **Function Inhibition Manager (FiM)** has interfaces and dependencies on the Diagnostic Event Manager (Dem), the Software Components (SW-C) with FID interface, the ECU State Manager, the RTE and the BSW modules supposed to be inhibited by the FiM.⌋ *(SRS_BSW_00384)*

- The **Diagnostic Event Manager (Dem)** is in charge of handling detected malfunctions denoted as events and reported by monitoring functions. The Dem informs and updates the Function Inhibition Manager (FiM) upon changes of the monitor status in order to stop or release functionalities according to assigned dependencies.
- **SW-Components (SW-C) with FID interface** query for permission to execute functionality identified by an FID at the FiM. The FIDs have to be provided by the SW components.
- **ECU State manager** is responsible for the basic initialization and de-initialization of BSW-components.
- **BSW module(s)** that are supposed to be inhibited by the FiM shall use the Fi M interface to ask for permission. Therefore, the affected BSW modules have to provide the corresponding configuration data (EventID - FID - Inhibition mask relation) at configuration time realized by using a template similar to the SW-component template. The interface handling for BSW modules corresponds to the interface handling for SW-components.
- **The RTE** implements scheduling mechanisms for BSW, e.g. assigns priority and memory protection to each BSW module used in an ECU.

5.1 Requirements

There are three sources of requirements for this specification:

- The requirements for the functionality of the FiM service are specified in [4]. In order to model the VFB view of the Service, the chapter on AUTOSAR Services of the VFB specification [5] has to be considered as an additional requirement.
- For the formal description of the SW-C attributes [6] gives the requirements.

5.1.1 Use Cases

On each ECU, typically one instance of the FiM Service and several Atomic Software Component instances using this Service are employed. The Atomic Software Components are named "clients" further on in this document.

Additionally, there are parts of the basic software, which either control the FiM Manager (e.g. the ECUState Manager for initialization and shutdown) or need to query the FiM for execution permission themselves.

5.2 File structure

[SWS_Fim_00029] ⌈ The FiM module shall adhere to the following include file structure:⌋ *(SRS_BSW_00346, SRS_BSW_00348, SRS_BSW_00381, SRS_BSW_00383, SRS_BSW_00412, SRS_BSW_00415, SRS_BSW_00447)*

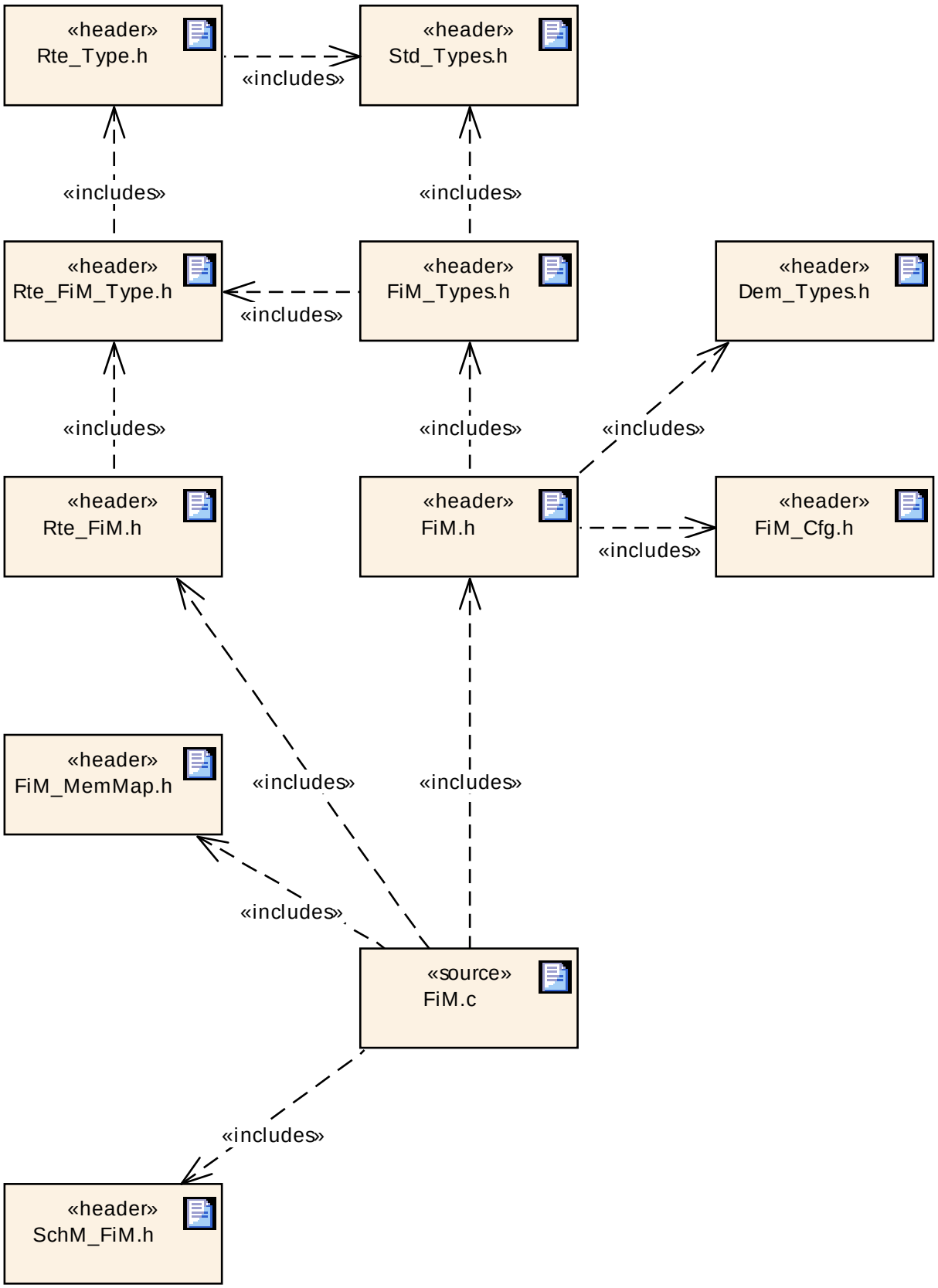
«header»
Rte_Type.h
«includes»
«header»
Std_Types.h
«includes»
«includes»
«header»
Rte_FiM_Type.h
«includes»
«header»
FiM_Types.h
«header»
Dem_Types.h
«includes»
«includes»
«includes»
«header»
Rte_FiM.h
«header»
FiM.h
«includes»
«header»
FiM_Cfg.h
«header»
FiM_MemMap.h
«includes»
«includes»
«includes»
«source»
FiM.c
«includes»
«header»
SchM_FiM.h

Figure 5.1

[SWS_Fim_00031] ⌈ The FiM module shall include the Dem.h file.⌋ *(SRS_BSW_00383)*

By this inclusion, EventId Symbols and the API to read the monitor status are included.

[SWS_Fim_00096] ⌈ The file FiM_Types.h shall include Rte_FiM_Type.h to include the types which are common used by BSW Modules and Software Components. FiM_Types.h and FiM.h shall only contain types that are not already defined in Rte_FiM_Type.h.⌋ *()*

6 Requirements traceability

Requirement	Description	Satisfied by
[SRS_BSW_00301]	All AUTOSAR Basic Software Modules shall only import the necessary information	[SWS_Fim_00999]
[SRS_BSW_00302]	All AUTOSAR Basic Software Modules shall only export information needed by other modules	[SWS_Fim_00999]
[SRS_BSW_00304]	All AUTOSAR Basic Software Modules shall use the following data types instead of native C data types	[SWS_Fim_00027]
[SRS_BSW_00305]	Data types naming convention	[SWS_Fim_00027]
[SRS_BSW_00306]	AUTOSAR Basic Software Modules shall be compiler and platform independent	[SWS_Fim_00999]
[SRS_BSW_00307]	Global variables naming convention	[SWS_Fim_00999]
[SRS_BSW_00308]	AUTOSAR Basic Software Modules shall not define global data in their header files, but in the C file	[SWS_Fim_00999]
[SRS_BSW_00309]	All AUTOSAR Basic Software Modules shall indicate all global data with read-only purposes by explicitly assigning the const keyword	[SWS_Fim_00999]
[SRS_BSW_00310]	API naming convention	[SWS_Fim_00004] [SWS_Fim_00006] [SWS_Fim_00011] [SWS_Fim_00021]
[SRS_BSW_00312]	Shared code shall be reentrant	[SWS_Fim_00011] [SWS_Fim_00021]
[SRS_BSW_00314]	All internal driver modules shall separate the interrupt frame definition from the service routine	[SWS_Fim_00999]
[SRS_BSW_00323]	All AUTOSAR Basic Software Modules shall check passed API parameters for validity	[SWS_Fim_00999]
[SRS_BSW_00325]	The runtime of interrupt service routines and functions that are running in interrupt context shall be kept short	[SWS_Fim_00999]
[SRS_BSW_00328]	All AUTOSAR Basic Software Modules shall avoid the duplication of code	[SWS_Fim_00999]
[SRS_BSW_00330]	It shall be allowed to use macros instead of functions where source code is used and runtime is critical	[SWS_Fim_00999]
[SRS_BSW_00331]	All Basic Software Modules shall strictly separate error and status information	[SWS_Fim_00015]

Requirement	Description	Satisfied by
[SRS_BSW_00333]	For each callback function it shall be specified if it is called from interrupt context or not	[SWS_Fim_00999]
[SRS_BSW_00334]	All Basic Software Modules shall provide an XML file that contains the meta data	[SWS_Fim_00999]
[SRS_BSW_00336]	Basic SW module shall be able to shutdown	[SWS_Fim_00999]
[SRS_BSW_00342]	It shall be possible to create an AUTOSAR ECU out of modules provided as source code and modules provided as object code, even mixed	[SWS_Fim_00999]
[SRS_BSW_00343]	The unit of time for specification and configuration of Basic SW modules shall be preferably in physical time unit	[SWS_Fim_00999]
[SRS_BSW_00344]	BSW Modules shall support link-time configuration	[SWS_Fim_00013]
[SRS_BSW_00345]	BSW Modules shall support pre-compile configuration	[SWS_Fim_00013]
[SRS_BSW_00346]	All AUTOSAR Basic Software Modules shall provide at least a basic set of module files	[SWS_Fim_00029]
[SRS_BSW_00347]	A Naming seperation of different instances of BSW drivers shall be in place	[SWS_Fim_00999]
[SRS_BSW_00348]	All AUTOSAR standard types and constants shall be placed and organized in a standard type header file	[SWS_Fim_00029]
[SRS_BSW_00353]	All integer type definitions of target and compiler specific scope shall be placed and organized in a single type header	[SWS_Fim_00999]
[SRS_BSW_00357]	For success/failure of an API call a standard return type shall be defined	[SWS_Fim_00999]
[SRS_BSW_00358]	The return type of init() functions implemented by AUTOSAR Basic Software Modules shall be void	[SWS_Fim_00004] [SWS_Fim_00006] [SWS_Fim_00045] [SWS_Fim_00059]
[SRS_BSW_00359]	All AUTOSAR Basic Software Modules callback functions shall avoid return types other than void if possible	[SWS_Fim_00999]
[SRS_BSW_00360]	AUTOSAR Basic Software Modules callback functions are allowed to have parameters	[SWS_Fim_00999]

Requirement	Description	Satisfied by
[SRS_BSW_00361]	All mappings of not standardized keywords of compiler specific scope shall be placed and organized in a compiler specific type and keyword header	[SWS_Fim_00999]
[SRS_BSW_00373]	The main processing function of each AUTOSAR Basic Software Module shall be named according the defined convention	[SWS_Fim_00060]
[SRS_BSW_00375]	Basic Software Modules shall report wake-up reasons	[SWS_Fim_00999]
[SRS_BSW_00377]	A Basic Software Module can return a module specific types	[SWS_Fim_00027]
[SRS_BSW_00378]	AUTOSAR shall provide a boolean type	[SWS_Fim_00999]
[SRS_BSW_00381]	The pre-compile time parameters shall be placed into a separate configuration header file	[SWS_Fim_00029]
[SRS_BSW_00383]	The Basic Software Module specifications shall specify which other configuration files from other modules they use at least in the description	[SWS_Fim_00029] [SWS_Fim_00031]
[SRS_BSW_00384]	The Basic Software Module specifications shall specify at least in the description which other modules they require	[SWS_Fim_00004] [SWS_Fim_00044]
[SRS_BSW_00386]	The BSW shall specify the configuration for detecting an error	[SWS_Fim_00999]
[SRS_BSW_00404]	BSW Modules shall support post-build configuration	[SWS_Fim_00062]
[SRS_BSW_00405]	BSW Modules shall support multiple configuration sets	[SWS_Fim_00062]
[SRS_BSW_00406]	A static status variable denoting if a BSW module is initialized shall be initialized with value 0 before any APIs of the BSW module is called	[SWS_Fim_00045] [SWS_Fim_00055] [SWS_Fim_00056] [SWS_Fim_00057] [SWS_Fim_00058] [SWS_Fim_00059]
[SRS_BSW_00409]	All production code error ID symbols are defined by the Dem module and shall be retrieved by the other BSW modules from Dem configuration	[SWS_Fim_00999]
[SRS_BSW_00412]	References to c-configuration parameters shall be placed into a separate h-file	[SWS_Fim_00029]
[SRS_BSW_00414]	Init functions shall have a pointer to a configuration structure as single parameter	[SWS_Fim_00004]

Requirement	Description	Satisfied by
[SRS_BSW_00415]	Interfaces which are provided exclusively for one module shall be separated into a dedicated header file	[SWS_Fim_00029]
[SRS_BSW_00416]	The sequence of modules to be initialized shall be configurable	[SWS_Fim_00004] [SWS_Fim_00018]
[SRS_BSW_00417]	Software which is not part of the SW-C shall report error events only after the DEM is fully operational.	[SWS_Fim_00999]
[SRS_BSW_00422]	Pre-de-bouncing of error status information is done within the DEM	[SWS_Fim_00999]
[SRS_BSW_00423]	BSW modules with AUTOSAR interfaces shall be describable with the means of the SW-C Template	[SWS_Fim_00999]
[SRS_BSW_00424]	BSW module main processing functions shall not be allowed to enter a wait state	[SWS_Fim_00999]
[SRS_BSW_00425]	The BSW module description template shall provide means to model the defined trigger conditions of schedulable objects	[SWS_Fim_00999]
[SRS_BSW_00426]	BSW Modules shall ensure data consistency of data which is shared between BSW modules	[SWS_Fim_00999]
[SRS_BSW_00427]	ISR functions shall be defined and documented in the BSW module description template	[SWS_Fim_00999]
[SRS_BSW_00428]	A BSW module shall state if its main processing function(s) has to be executed in a specific order or sequence	[SWS_Fim_00999]
[SRS_BSW_00429]	Access to OS is restricted	[SWS_Fim_00999]
[SRS_BSW_00432]	Modules should have separate main processing functions for read/receive and write/transmit data path	[SWS_Fim_00999]
[SRS_BSW_00433]	Main processing functions are only allowed to be called from task bodies provided by the BSW Scheduler	[SWS_Fim_00999]
[SRS_BSW_00447]	Standardizing Include file structure of BSW Modules Implementing Autosar Service	[SWS_Fim_00029]
[SRS_Fim_04700]	An Interface for querying the FID permission status shall be provided	[SWS_Fim_00010] [SWS_Fim_00011] [SWS_Fim_00090] [SWS_Fim_00094]
[SRS_Fim_04701]	The Functionalities supervised by the FIM shall be defined by static configuration	[SWS_Fim_00002] [SWS_Fim_00003] [SWS_Fim_00007]

Requirement	Description	Satisfied by
[SRS_Fim_04702]	The FIM shall support different inhibit options	[SWS_Fim_00012]
[SRS_Fim_04706]	Individual configuration of inhibit conditions of functionalities shall be available	[SWS_Fim_00008] [SWS_Fim_00013] [SWS_Fim_00016] [SWS_Fim_00043]
[SRS_Fim_04709]	The permission state shall be evaluated before executing functionalities	[SWS_Fim_00011]
[SRS_Fim_04712]	The permission states at start up shall be initialized	[SWS_Fim_00004] [SWS_Fim_00018]
[SRS_Fim_04713]	Methods for the computation of permission states shall be provided	[SWS_Fim_00009] [SWS_Fim_00015] [SWS_Fim_00020]
[SRS_Fim_04717]	The permission states shall be updated	[SWS_Fim_00021] [SWS_Fim_00022]
[SRS_Fim_04719]	Mechanism for summarized diagnostic event states shall be provided	[SWS_Fim_00061]
[SRS_Fim_04721]	OBD Functionalities shall be supported	[SWS_Fim_00999]
[SRS_Fim_04723]	The FIM shall provide a boolean configuration option per FID.	[SWS_Fim_00105] [SWS_Fim_00106] [SWS_Fim_00107] [SWS_Fim_00108]

7 Functional specification

7.1 Background & Rationale

The Function Inhibition Manager allows querying the permission / inhibition status of software components and the functionality therein. In the FiM context an FID (FID - function identifier) identifies an application functionality along with the inhibit conditions for that particular identifier. The functionalities poll for the permission state of their FID before execution. If an inhibit condition applies for a particular identifier, the corresponding functionality is not allowed to be executed anymore. By means of the FiM, the inhibition of these functionalities can be configured and even modified by calibration. Dem events and their status information are supported as inhibit conditions.

In order to handle the relation of functionality and associated affecting events, the identifier (FID) and inhibit conditions (events) of the functionality are included in the SW component template (equivalence for BSW). During configuration of the FiM, data structures (i.e. an inhibit matrix) are built up to deal with the sensitiveness of the identifiers against certain events.

7.2 Requirements

7.2.1 FiM core variables

7.2.1.1 Definition of 'Diagnostic Event'

A 'Diagnostic Event' is an identifier provided by the Dem to a specific diagnostic monitor function to report an error.

See AUTOSAR_SWS_DiagnosticEventManager document for further details [3].

7.2.1.2 Definition of 'Monitor Status'

A 'monitor status' is the status calculated by the Dem according to the reported values of monitor functions. Possible values are defined by Dem_MonitorStatusType.

See AUTOSAR_SWS_DiagnosticEventManager document for further details [3].

7.2.1.3 Definition of 'Monitored Component'

A 'Monitored Component' is an identifier provided by the Dem to a specific monitored component (hardware component or signal). The FAILED status of a 'monitored component' represents the result of all assigned monitoring functions and inherited failure information from other DemComponents.

See AUTOSAR_SWS_DiagnosticEventManager document for further details [3].

7.2.1.4 Definition of 'Summarized Event'

[SWS_Fim_00061] ⌈ The FiM configuration shall support summarizing events. A summarized event consists of multiple single diagnostic events.⌋ *(SRS_Fim_04719)*

During the configuration process, these single events can be combined to a summarized event (ECUC_FiM_00037). A summarized event simplifies dealing with the multiple events that are associated with or represented by the particular summarized event. For simplicity, this particular summarized event can be used as an inhibit condition in the SW-C templates.

[SWS_Fim_00064] ⌈ The FiM shall also be able to process the inhibit conditions of all FIDs associated to one summarized event if one of the Dem Events associated to this summarized event is reported to the FiM.⌋ *()*

Hence, the particular summarized event is just a representative of multiple diagnostic events (ref.10.2.3). A use case for summarized events is for example the combination of all error conditions that indicate a failed sensor:

A sensor X has multiple diagnostics, e.g. short cut ground, battery and open circuit: X_SCG, X_SCB and X_OC. The functions FID_0, FID_1, ..., FID_N are to be inhibited in case of this fault. A direct configuration requires 3 * N containers `FiMInhibitionConfiguration` with FIM_INH_EVENT_ID = X_SCG/SCB/OC and FIM_INH_FUNCTION_ID = FID_0/.../N.

With summarized events (`FiMSummaryEvent`), a group of events can be reused for several inhibition configurations, by selecting it as `FiMInhSumRef`. This may simplify configuration.

7.2.1.5 Definition of 'Function Identifier'

The Fim implements the calculation of function permissions. Object to those calculations are SW-Components or logical units, which receive the information "Permission granted" / "permission denied".

To address those components, these have to be configured in FIM and a Function Identifier is assigned to address them via interfaces.

[SWS_Fim_00002] ⌈ The configuration process shall guarantee that FunctionIds are unique per FiM. Two distinct functionalities with different dependencies on events shall never have the same FunctionId (see also [SWS_Fim_00007]).⌋ *(SRS_Fim_04701)*

[SWS_Fim_00003] ⌈ The FiM module's environment shall use the FunctionId to directly point to the associated functionality information (permission status etc.)⌋ *(SRS_Fim_04701)*

[SWS_Fim_00010] ⌈ The flow of information starts with the API call of the Dem providing changes of the event information. This information is processed and dependencies to FIDs are evaluated. Finally, the permission state of the FIDs is accessed via API through the RTE (Figure 7.1).⌋ *(SRS_Fim_04700)*

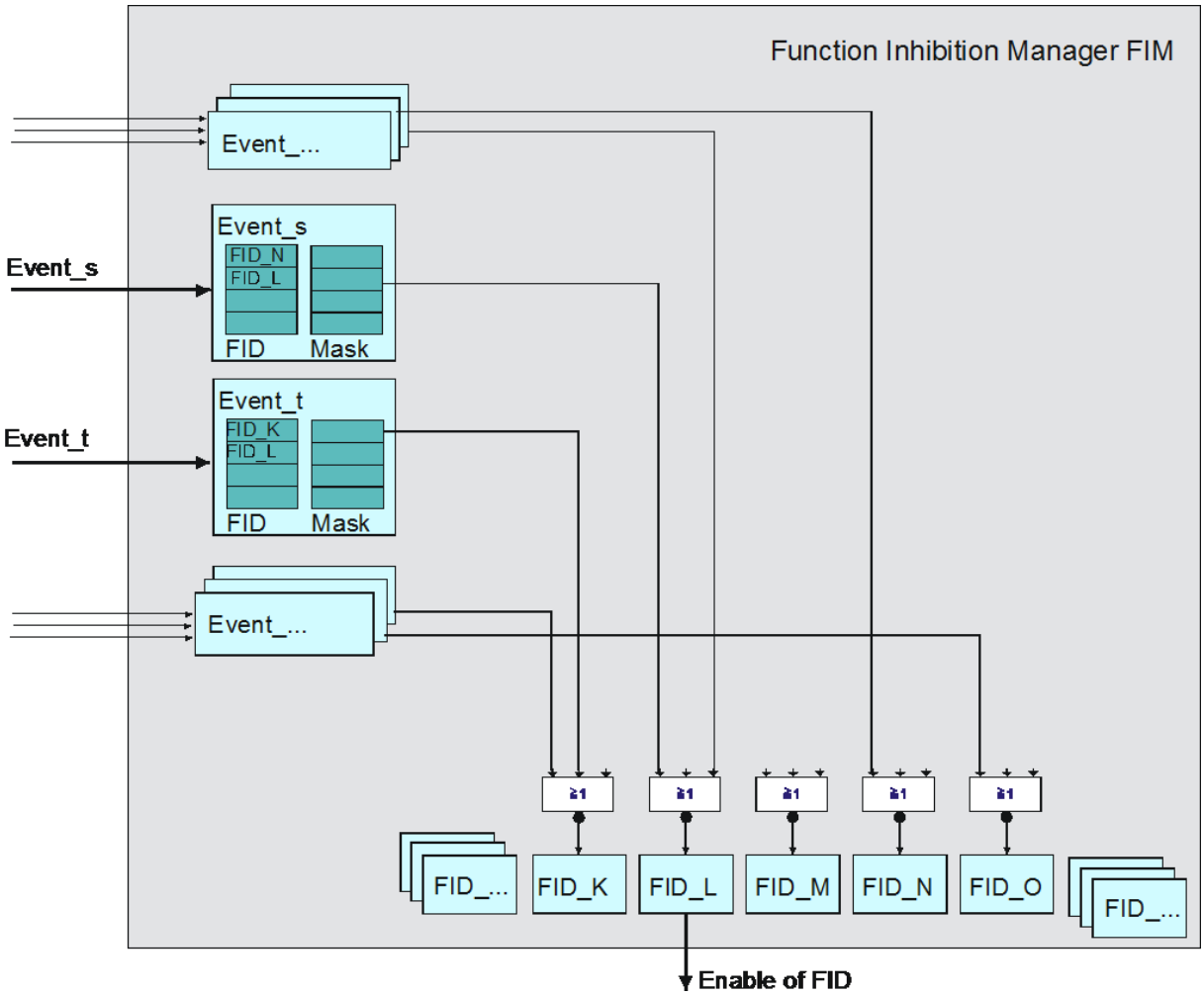

Figure 7.1: Logical information flow to determine FID permission states for an implementation with permission state stored in RAM

The permission state of each FID is calculated based on the EventIds assigned to a specific FID. Afterwards, the calculated permission states of each FID (e.g. FID_K) are "and-ed" to determine the resulting permission state. This implies an implementation where the FiM stores the permission state of the FIDs in RAM.

Alternatively, the FiM can poll the monitor status to re-calculate the permission state. The polling is triggered either by a functionality requesting its permission state (SW-C or BSW) or in a cyclic task. In this case, there is no increased process effort within the FiM at changes of any event.

7.2.1.6 Definition of 'Function Identifier permission state'

[SWS_Fim_00015] ⌈ The FID permission state contains the information whether a functionality represented by its FID can be executed. If the permission state == TRUE, the functionality associated with the FID is permitted to be executed. If the permission state == FALSE, the functionality associated with the FID is not allowed to be executed.⌋ *(SRS_BSW_00331, SRS_Fim_04713)*

The permission state is based on events reported by the Dem. Therefore, the permission state does not directly consider physical conditions (e.g. temperature, engine speed...) but those conditions reported to the Dem (e.g. sensor defect).

Additionally to the permission state as prerequisite, the activity state (is the function active or not) includes physical enable conditions representing whether the functionality is indeed executed or not, i.e. is active or not.

As stated above, one possible implementation is to provide the permission state in status variables. An alternative is to compute the permission on the query based on the underlying dependencies.

Hint: If the permission states are stored in status variables, they are unique values per FID. SW-components access the status via `FiM_GetFunctionPermission`.

[SWS_Fim_00009] ⌈ If the implementation uses status variables for the permission of the FIDs, the status variables shall be readable for tracking purposes by the calibration system (to be defined by AUTOSAR) during the development phase of the ECU.⌋ *(SRS_Fim_04713)*

7.2.2 FiM core functionalities

7.2.2.1 FiM Data Structure

[SWS_Fim_00013] ⌈ The configuration process of the FiM shall create data structures within the FiM module to store the inhibit relations (EventID - FID - applicable mask).⌋ *(SRS_BSW_00344, SRS_BSW_00345, SRS_Fim_04706)*

A configurable number of EventIds and inhibition masks are assigned to one FID. The number of EventIds and inhibit masks per FID have to match so that for each configured event, a corresponding inhibit mask exists.

The inhibition mask contains the inhibition conditions for a FID provided that the associated EventIds have a certain status (`Dem_EventStatusExtendedType`). These masks define which states of an event the FID is sensitive to. However, the mask does not only address certain bits according to the `Dem_EventStatusExtendedType`, it rather selects an algorithm to calculate the boolean inhibition condition from the `Dem_EventStatusExtendedType`.

The implementation of the FiM data structure cannot be prescribed. A possible implementation of the inhibit matrix could be a block of calibration values for each inhibit

source (=EventId). That means for each EventId a list of FIDs and masks is available that shall be inhibited by this EventId. A possible FiM structure consisting of such a configuration and a FID status array is exemplarily shown in Figure 2.

There is an inhibition mask assigned to every FID and both are assigned to a particular EventId. If this event has a certain state, the inhibition of the FID becomes active if the event state matches the configured mask.

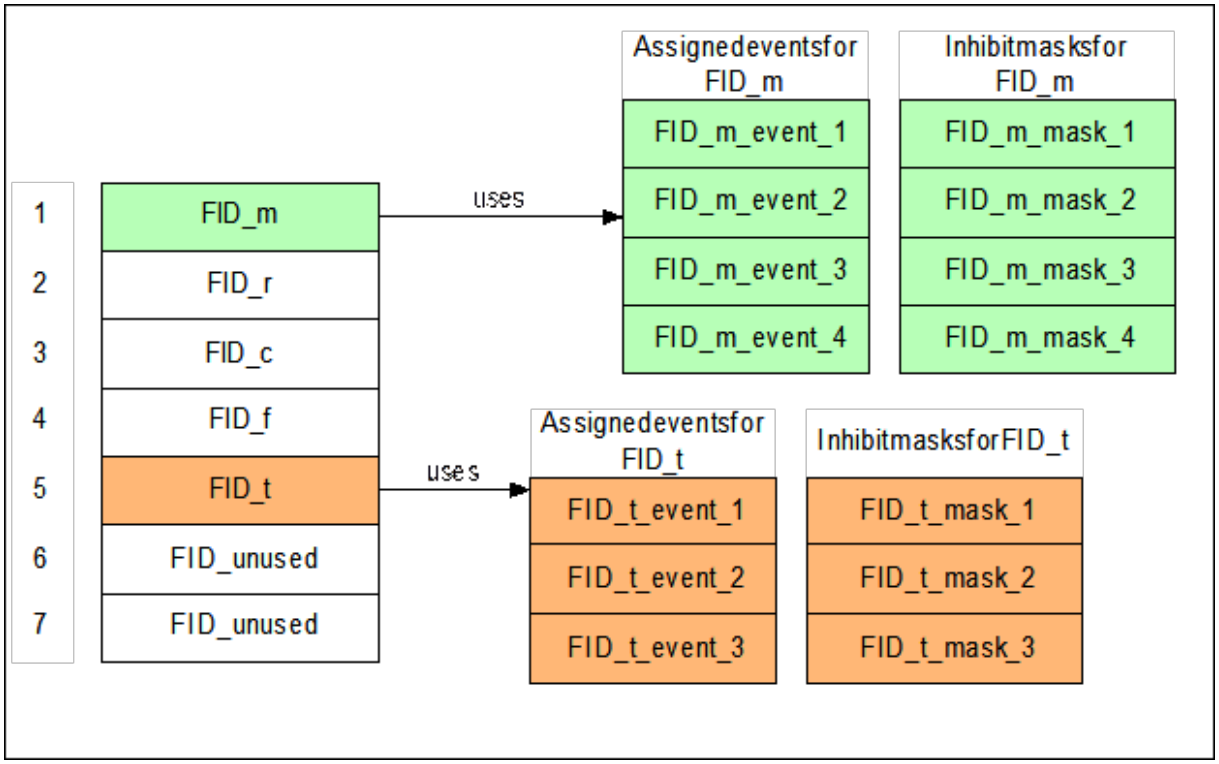

Figure 7.2: Inhibit Mask

[SWS_Fim_00008] ⌈ The FiM module shall provide the possibility to modify the inhibit conditions by post-built configuration.⌋ *(SRS_Fim_04706)*

Depending on the implementation, it might not be possible to:

- Add new events.
- Extend the number of inhibited FID's per event.
- Extend the specified configuration parameters concerning number of events, number of FIDs and number of links.

7.2.2.2 Interaction between Dem and Function Inhibition Manager (FiM)

[SWS_Fim_00022] ⌈ The purpose of the FiM module is to provide services to control (permit / inhibit) functionality within SW-Cs based on Dem events being supported as inhibit conditions.⌋ *(SRS_Fim_04717)*

[SWS_Fim_00065] ⌈ The Function Inhibition Manager shall use the FID - EventIDs - inhibition masks relations provided by the software components to determine the permission state for all configured FIDs.⌋ *()*

Upon changes in the monitor status of a reported event, the Dem informs the FiM about the monitor status change via the API function `FiM_DemTriggerOnMonitorStatus`, if DemTriggerFiMReports is enabled.

On being informed about a monitor status change, the Fim uses the Api `Dem_GetMonitorStatus` to recalculate the function inhibitions.

1. Note: From the function point of view, synchronous update of inhibit / release conditions can be made either within or outside of `FiM_MainFunction` API.

As mentioned in chapter 4.1, the implementation of the FiM highly depends on requirements (e.g. timing requirements) derived from applications. If an application requires fast reaction times the FiM has to provide FID information sufficiently fast to allow triggering limp-home functionality.

The API `FiM_DemTriggerOnMonitorStatus` is only relevant if a status variable per FID is stored. In an alternative implementation when no status is stored and the permission status is calculated every time when queried, the API `FiM_DemTriggerOnMonitorStatus` is without effect.

As an example of implementation, Figure 3 shows the calculation of a single Event Id-FID link. On the left hand side, the monitor status is reported by the Dem as Dem_EventStatusExtendedType. This status is compared to the mask configured for the EventId associated with the FID.

An inhibition counter is assigned to each FID. The inhibition counter contains the number of currently inhibiting EventIds.

If the calculation is performed cyclically (monitor status is read through `Dem_GetMonitorStatus`), the inhibition counter shall be incremented if the status and the mask match; otherwise, the inhibition counter is not updated. This is applicable for `FiM_GetFunctionPermission` (if the permission state has to be computed upon the query) and `FiM_MainFunction` APIs.

In the trigger on monitor status change, the stored currently inhibiting EventIds (inhibition counter) shall be used for the computation for the permission state. If there is an monitor status change reported by `FiM_DemTriggerOnMonitorStatus`, then the following shall be performed:

a. If the change in status for the EventId results in a released state (mask does not match with the monitor status), then the inhibition counter has to be decremented.

b. If the change in status for the EventId results in an inhibited state (mask matches with the monitor status), then the inhibition counter has to be incremented.

If the inhibition counter is > 0, then the FID permission state shall be set to FALSE, otherwise the FID permission state shall be set to TRUE.

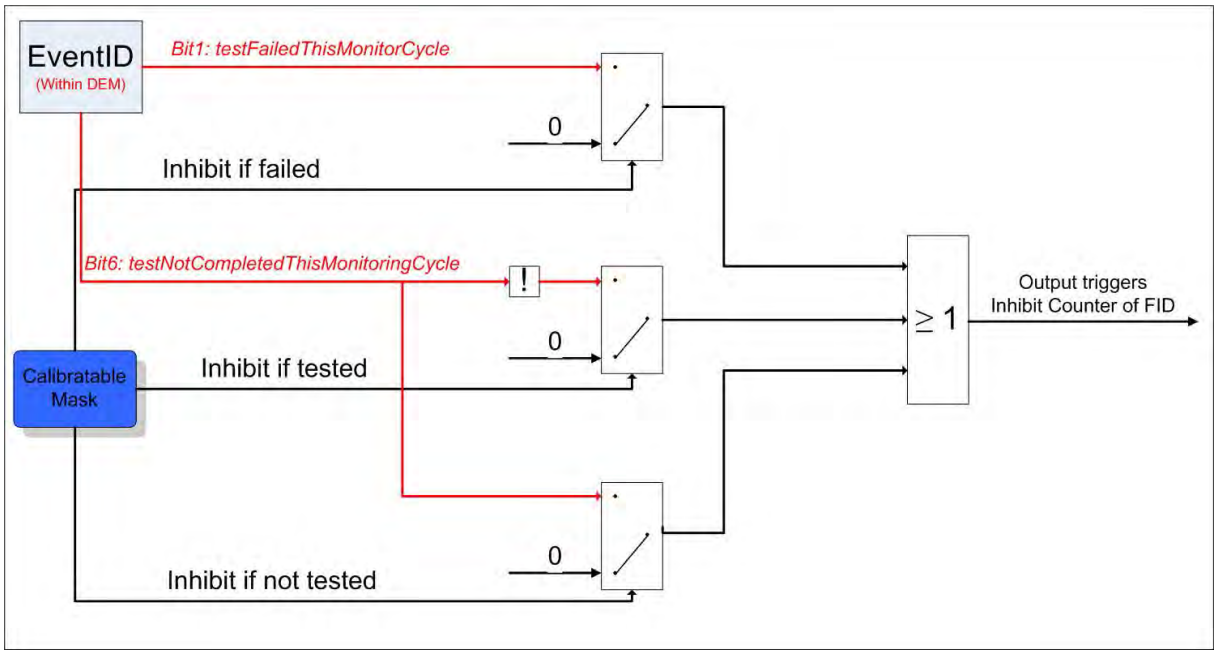

Figure 7.3: Calculation of permission state based on monitor status information

[SWS_Fim_00012] ⌈ The FiM module shall calculate the inhibit status based on the actual status of the inhibit source and the calibrated mask which exists for each inhibit source (ref. 10.2.7). The FiM module shall inhibit the FID if the Monitor status is equal to the calibrated mask (=Defect, Tested, NotTested). The inhibition is deactivated if the mask of the event does not match anymore the calibrated value.⌋*(SRS_Fim_04702)*

Optionally, the tested status can be used for inhibiting. Depending on the inhibition condition, the inhibition can be active if the event has status "Tested" or "NotTested". If no tested value is selected, the tested status is not relevant.

The available combinations of status flags are assigned to a predefined value which has verbal representation like "Tested", "Not_Tested" or Last_Failed".

[SWS_Fim_00098] ⌈ The Function Inhibition Manager shall use the FID - DemComponentId - inhibition configuration to determine the permission state for the configured FID.

Upon changes of the FAILED status of a `DemComponent`, the function status shall be recalculated. Whenever the component status is FAILED (`ComponentFailedStatus` = TRUE), the FID is inhibited.⌋*()*

[SWS_Fim_00099] ⌈ If the FIM is configured for cyclically polling the status, the FIM shall use the API `Dem_GetComponentFailed` to get the current FAILED status of a component.⌋*()*

[SWS_Fim_00100] ⌈ If the FIM is configured for being triggered on eventStatus (FiMCyclicEventEvaluation), the FIM shall accept the status changed information of a `DemComponent` by providing the function `FiM_DemTriggerOnComponentStatus`.⌋*()*

7.2.2.3 Interaction between SW-Components and Function Inhibition Manager (FiM)

[SWS_Fim_00016] ⌈ The configuration engineer shall provide at compile time the inhibit conditions for each FID required for handling the dependencies of functionalities and events in the FiM module.⌋ *(SRS_Fim_04706)*

Note, that modifications by calibration shall be possible. The configuration mechanism of the FiM using SW-component template contents shall consider these requirements.

First, the FID needs to be introduced and allocated. Furthermore, for each FID a list of events plus associated mask causing the inhibition of the FID shall be provided by the SW-component. Chapter 10 introduces how the SW-component template considers these configuration requirements.

During the configuration process, the data structures are built up. Depending on the implementation this could, e.g. be a mapping of an event onto all affected FIDs or alternatively vice versa, a mapping of a FID onto all events affecting it.

Controlling implies that within the implemented functionality, the permission of a FID is queried via AUTOSAR service.

[SWS_Fim_00020] ⌈ The FiM module shall ensure an immediate control of functionality by synchronously responding to an incoming permission query. The FiM module shall realize this behavior either by storing the permission state as a status variable or by evaluation of the event states upon permission query.⌋ *(SRS_Fim_04713)*

[SWS_Fim_00105] ⌈ If a function (FID) is set to not available using the interface `FiM_SetFunctionAvailable`, its permission state `FiM_GetFunctionPermission` shall always return FALSE⌋ *(SRS_Fim_04723)*

7.2.2.4 Application example for FiM usage

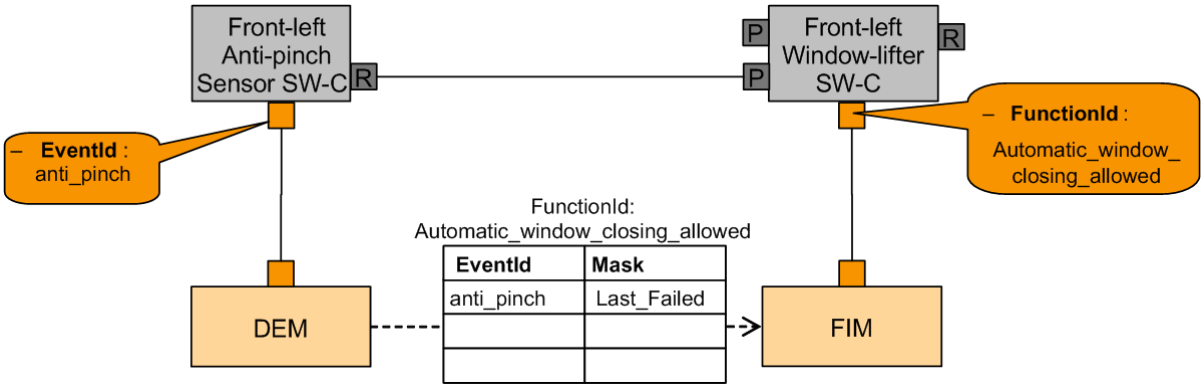

Figure 7.4: FiM usage

- The configuration of the FiM actually establishes the relationship between the EventId and the assigned FunctionId(s)

- The required information is:
  - For each FunctionId: How does the status of the FunctionId depend on the status of one/several EventIds?
    - The mask determines the relationship between the EventId status and the inhibit status of the FunctionId.
    - The row result is 'OR'ed to come up with the overall result for one FunctionId if it depends on several EventIds.

7.2.2.5 Initialization

[SWS_Fim_00018] ⌈ If Dem events status information is used, the FiM module shall compute the permission states for all FIDs at its initialization based on all restored monitor status information (not only events stored in the fault memory) of the Dem.⌋ *(SRS_BSW_00416, SRS_Fim_04712)*

7.2.3 OBD-Functionality

7.2.3.1 In-Use-Monitor Performance Ratio (IUMPR) Support

In order to track the behavior of diagnostic functions in every day usage, in particular the capability to find malfunctions, the regulations require the tracking of this performance in relation to a standardized driving profile. This is called "In-Use Monitor Performance Ratio" (IUMPR) defined as the number of times a fault could have been found (=numerator) divided by the number of times the standardized driving profile has been fulfilled (=denominator). The relevant data recording is allocated in the Dem based on FIDs and EventIDs.

Thus, based on the FiM configuration of the referenced FIDs it can be evaluated whether a Ratio Id specific data record needs to be stopped. In particular, IUMPR tracking shall be stopped as long as the entry remains visible in service $07.

The Dem may use the FiM configuration for its IUMPR calculation or by call of `FiM_GetFunctionPermission` of a dedicated FID.

Note: The FiM does not provide special OBDII functionality but uses already existing mechanisms for OBDII.

7.2.4 Auxiliary explanations and definitions

7.2.4.1 Output for other WPs

In order to be runtime-efficient, the monitor status information needs to be evaluated quickly, e.g. in the `FiM_Init` function. If Dem and FiM are implemented as one package, the Dem-APIs with access to monitor status information are not necessarily used and so direct access to monitor status information is allowed (see AUTOSAR conformance classes).

7.3 Error classification

7.3.1 Development Errors

[SWS_Fim_00076] ⌈ The Development Errors Types are shown in table Table 7.1. ⌋*()*

Type or error	Related error code	Value [hex]
API function called before the FiM module has been full initialized or after the FiM module has been shut down	FIM_E_UNINIT	0x01
`FiM_GetFunctionPermission` called with wrong FID	FIM_E_FID_OUT_OF_RANGE FIM_E_FID_OUT_OF_RANGE	0x02
Dem calls FiM with invalid EventId	FIM_E_EVENTID_OUT_OF_RANGE	0x03
API is invoked with NULL Pointer.	FIM_E_PARAM_POINTER	0x04
Invalid configuration set selection	FIM_E_INIT_FAILED	0x05

Table 7.1: Development Errors Types

7.3.2 Runtime Errors

There are no runtime errors.

7.3.3 Transient Faults

There ar no transient faults.

7.3.4 Production Errors

There are no productions errors.

7.4 Configuration Constraints

[SWS_Fim_CONSTR_0001] ⌈ For each configured FiMInhibitionConfiguration, at least one of FiMInhSumRef or FiMInhEventRef or FiMInhComponentRef shall be configured. ⌋*()*

8 API specification

8.1 Imported types

In this chapter, all types included from the following files are listed:

[SWS_Fim_00081] ⌈

Module	*Imported Type*
Dem	Dem_ComponentIdType Dem_EventIdType Dem_MonitorStatusType
SchM	SchM_ReturnType
Std_Types	Std_ReturnType Std_VersionInfoType

Table 8.1: FiM_ImportedTypes

⌋()

8.2 Type definitions

8.2.1 FiM_ConfigType

[SWS_Fim_00092] ⌈

Name:	`FiM_ConfigType`		
Type:	`Structure`		
Range:	`--`		implementation specific
Description:	This type defines a data structure for the post build parameters of the FIM. At initialization the FIM gets a pointer to a structure of this type to get access to its configuration data, which is necessary for initializsation.		

Table 8.2: FiM_ConfigType

⌋()

8.3 Function definitions

This is a list of functions provided for upper layer modules.

8.3.1 Interface ECUState Manager <-> FiM

8.3.1.1 FiM_Init

[SWS_Fim_00077] ⌈

Service name:	FiM_Init	
Syntax:	`void FiM_Init( const FiM_ConfigType* FiMConfigPtr )`	
Service ID[hex]:	0x00	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	FiMConfigPtr	–
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	This service initializes the FIM.	

Table 8.3: FiM_Init

⌋() Note: see Chapter 9.1

[SWS_Fim_00004] ⌈ The FiM calculates the permission states based on monitor status information stored in the Dem and inhibition mask configuration.⌋ *(SRS_BSW_00310, SRS_BSW_00358, SRS_BSW_00384, SRS_BSW_00414, SRS_BSW_00416, SRS_Fim_04712)*

[SWS_Fim_00045] ⌈ If development error detection is turned on the FiM module shall report an error to the DET if it has not successfully completed the initialization and has detected not permitted access.⌋ *(SRS_BSW_00358, SRS_BSW_00406)*

[SWS_Fim_00059] ⌈ A static status variable denoting if the FiM is initialized shall be initialized with value 0 before any APIs of the FiM is called.

`FiM_Init` shall set the static status variable to a value not equal to 0.⌋ *(SRS_BSW_00358, SRS_BSW_00406)*

In order to restore the permission states quickly, it is recommended that the Dem provides direct access to monitor status information if Dem and FiM are implemented as a cluster. In this case, the FiM needs to have knowledge about the data structure of the Dem so that it can directly access EventId states.

[SWS_Fim_00072] ⌈ If Dem and FiM are implemented as two separate modules, the FiM module shall access the EventId states through the API call `Dem_GetMonitorStatus`.⌋ *()*

Note: There is no explicit action during shutdown. The permission states remain valid until the ECU is shut down since they directly depend on the monitor status information.

8.3.2 Interface SW-Components <-> FiM

8.3.2.1 FiM_GetFunctionPermission

[SWS_Fim_00011] ⌈

Service name:	FiM_GetFunctionPermission	
Syntax:	`Std_ReturnType FiM_GetFunctionPermission( FiM_FunctionIdType FID, boolean* Permission )`	
Service ID[hex]:	0x01	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	FID	Identification of a functionality by assigned FID. The FunctionId is configured in the FIM. Min.: 1 (0: Indication of no functionality) Max.: Result of configuration of FIDs in FIM (Max is either 255 or 65535)
Parameters (inout):	None	
Parameters (out):	Permission	TRUE: FID has permission to run FALSE: FID has no permission to run, i.e. shall not be executed
Return value:	Std_ReturnType	E_OK: The request is accepted E_NOT_OK: The request is not accepted, ie. initialization of FIM not completed
Description:	This service reports the permission state to the functionality.	

Table 8.4: FiM_GetFunctionPermission

⌋*(SRS_BSW_00310, SRS_BSW_00312, SRS_Fim_04700, SRS_Fim_04709)*

[SWS_Fim_00066] ⌈ The SW Components and the BSW shall use the function `FiM_GetFunctionPermission` to query for the permission to execute a certain functionality represented by the respective FID.⌋*()*

[SWS_Fim_00025] ⌈ The function `FiM_GetFunctionPermission` shall deliver the return value synchronously to enable direct use of this information for controlling and executing the underlying code in the software component.⌋*()*

[SWS_Fim_00055] ⌈ If development error detection for the module FiM is enabled: the function `FiM_GetFunctionPermission` shall perform a plausibility check on the FID range. If a FID is out of range, the function shall raise a development error and return no permission (FALSE).⌋*(SRS_BSW_00406)*

[SWS_Fim_00056] ⌈ If development error detection for the module FiM is enabled: the function `FiM_GetFunctionPermission` shall check that the initialization of the module FiM has been completed. If the function detects that the initialization is not complete, it shall raise a development error and return no permission (FALSE).⌋ *(SRS_BSW_00406)*

8.3.2.2 FiM_ SetFunctionAvailable

[SWS_Fim_00106] ⌈

Service name:	FiM_SetFunctionAvailable	
Syntax:	`Std_ReturnType FiM_SetFunctionAvailable( FiM_FunctionIdType FID, boolean Availability )`	
Service ID[hex]:	0x07	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	FID Availability	Identification of a functionality by assigned FID. The permission of the requested FID: TRUE: Function is available. FALSE: Function is not available.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: The request is accepted E_NOT_OK: Request is not accepted (e.g. invalid FID is given)
Description:	This service sets the availability of a function. The function is only available if FiMAvailabilitySupport is configured as True.	

Table 8.5: FiM_SetFunctionAvailable

⌋*(SRS_Fim_04723)*

8.3.3 Interface Dem <-> FiM

8.3.3.1 FiM_DemTriggerOnMonitorStatus

[SWS_Fim_00021] ⌈

Service name:	FiM_DemTriggerOnMonitorStatus	
Syntax:	`void FiM_DemTriggerOnMonitorStatus( Dem_EventIdType EventId )`	
Service ID[hex]:	0x02	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	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.: Result of configuration of Event Numbers in DEM (Max is either 255 or 65535)
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	

Description:	This service is provided to be called by the Dem in order to inform the Fim about monitor status changes.

Table 8.6: FiM_DemTriggerOnMonitorStatus

⌋*(SRS_BSW_00310, SRS_BSW_00312, SRS_Fim_04717)*

[SWS_Fim_00057] ⌈ If development error detection for the module FiM is enabled: the function `FiM_DemTriggerOnMonitorStatus` shall perform a plausibility check on the EventId. If the requested EventId is not existing in the Dem configuration, the function shall raise the development error `FIM_E_EVENTID_OUT_OF_RANGE`.⌋ *(SRS_BSW_00406)*

[SWS_Fim_00058] ⌈ If development error detection for the module FiM is enabled: The function `FiM_DemTriggerOnMonitorStatus` shall check for complete initialization of the FiM. If the function detects that the initialization is not complete, it shall raise a development error.⌋ *(SRS_BSW_00406)*

8.3.3.2 FiM_ DemTriggerOnComponentStatus

[SWS_Fim_00101] ⌈

Service name:	FiM_DemTriggerOnComponentStatus	
Syntax:	`void FiM_DemTriggerOnComponentStatus( Dem_ComponentIdType ComponentId, boolean ComponentFailedStatus )`	
Service ID[hex]:	0x06	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	ComponentId	Identification of a DemComponent.
	ComponentFailedStatus	New FAILED status of the component.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Triggers on changes of the component failed status.	

Table 8.7: FiM_DemTriggerOnComponentStatus

⌋*()*

8.3.3.3 FiM_DemInit

[SWS_Fim_00006] ⌈

Service name:	FiM_DemInit

Syntax:	`void FiM_DemInit(`
	`void`
	`)`

Service ID[hex]:	0x03
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	This service re-initializes the FIM.

Table 8.8: FiM_DemInit

⌋*(SRS_BSW_00310, SRS_BSW_00358)*

[SWS_Fim_00069] ⌈ The function `FiM_DemInit` shall re-compute the permission state for all FIDs.⌋*()*

[SWS_Fim_00082] ⌈ If Dem and FiM are implemented as two separate modules, the function `FiM_DemInit` shall synchronously access the EventId states via the function `Dem_GetMonitorStatus`.⌋*()*

In case Dem and FiM are implemented as one bundle, the FiM module needs to have knowledge about the data structure of the Dem so that it can directly access the EventId states.

8.3.3.4 FiM_GetVersionInfo

[SWS_Fim_00078] ⌈

Service name:	FiM_GetVersionInfo	

Syntax:	`void FiM_GetVersionInfo(`
	`Std_VersionInfoType* versioninfo`
	`)`	

Service ID[hex]:	0x04	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	versioninfo	Pointer to where to store the version information of this module.
Return value:	None	
Description:	This service returns the version information of this module.	

Table 8.9: FiM_GetVersionInfo

⌋()

8.3.4 Call-back notifications

This chapter lists all functions provided by the FiM module and used by lower layer modules.

No callback notification is specified.

8.3.5 Scheduled functions

This chapter lists all functions provided by the FiM module and called directly by the Basic Software Module Scheduler.

8.3.5.1 FiM_MainFunction

[SWS_Fim_00060] ⌈

Service name:	FiM_MainFunction
Syntax:	`void FiM_MainFunction( void )`
Service ID[hex]:	0x05
Description:	–

Table 8.10: FiM_MainFunction

⌋(*SRS_BSW_00373*)

The evaluation of permission states can be performed either on event change or cyclically.

[SWS_Fim_00070] ⌈ If FiM module polls monitor status (as defined in configuration parameter `FiMEventUpdateTriggeredByDem` = FALSE) and decides to do it in a cyclic manner, `FiM_MainFunction` shall be used to calculate the permission states of all EventIds using their inhibition masks. The API `Dem_GetMonitorStatus` shall be used to get status information of EventIds.⌋()

[SWS_Fim_00097] ⌈ If `Dem_GetMonitorStatus` returns E_NOT_OK, the FIM shall not consider this event in its inhibition mask calculation⌋()

[SWS_Fim_00067] ⌈ The FiM shall perform the evaluation of actual EventIds status information cyclically for all the EventIds using the inhibition mask and then calculate the corresponding FID permission states. FiM shall access the monitor status information using the API `Dem_GetMonitorStatus` if Dem and FiM are implemented as

separate modules. FiM shall access the monitor status structure of Dem if Dem and Fi M are implemented as a bundle.⌋()

8.3.6 Expected Interfaces

This chapter lists all functions the module FiM requires from other modules.

8.3.6.1 Mandatory Interfaces

This chapter defines all interfaces, which are required to fulfill the core functionality of the module.

[SWS_Fim_00079] ⌈

API function	*Description*
Dem_GetMonitorStatus	Gets the current monitor status for an event.
SchM_ActMainFunction_FiM	Invokes the SchM_ActMainFunction function to trigger the activation of a corresponding main processing function.
SchM_CancelMainFunction_FiM	Invokes the SchM_CancelMainFunction function to trigger the cancellation of the requested activation of a corresponding main processing function.

Table 8.11: FiM Mandatory Interfaces

⌋()

8.3.6.2 Optional Interfaces

This chapter defines all interfaces, which are required to fulfill an optional functionality of the module.

[SWS_Fim_00080] ⌈

API function	*Description*
Det_ReportError	Service to report development errors.

Table 8.12: FiM Optional Interfaces

⌋()

8.4 Service interfaces

This chapter specifies the ports and port interfaces to operate the FiM functionality over the VFB.

8.4.1 Client-Server-Interfaces

8.4.1.1 FiM_FunctionInhibition

Using the concepts of the SW-C template, the interface is defined as follows:

[SWS_Fim_00090] ⌈

Name	FunctionInhibition	
Comment	The SW Components can use this service to query for the permission to execute a certain functionality represented by a FID.	
IsService	true	
Variation	--	
Possible Errors	0	E_OK
	1	E_NOT_OK

Table 8.13: Service Interface FunctionInhibition

Operations

GetFunctionPermission			
Comments	Get the permission state of the respective FID.		
Variation	--		
Parameters	Permission	Comment	The permission of the requested FID. TRUE: FID has permission to run FALSE: FID has no permission to run, i.e. shall not be executed
		Type	boolean
		Variation	--
		Direction	OUT
Possible Errors	E_OK	Operation successful	
	E_NOT_OK	The request is not accepted, i.e. initialization of FIM not completed	

Table 8.14: Operation GetFunctionPermission

⌋(*SRS_Fim_04700*)

8.4.1.2 FiM_ControlFunctionAvailable

Using the concepts of the SW-C template, the interface is defined as follows:

[SWS_Fim_00107] ⌈

Name	ControlFunctionAvailable	
Comment	SW Components can use this service to set the availability of a function.	
IsService	true	
Variation	({ecuc(FiM/FiMGeneral/FiMAvailabilitySupport)} == True)	
Possible Errors	0	E_OK
	1	E_NOT_OK

Table 8.15: Service Interface ControlFunctionAvailable

Operations

SetFunctionAvailable			
Comments	Sets the availability of a function.		
Variation	--		
Parameters	Availability	Comment	The permission of the requested FID: TRUE: Function is available. FALSE: Function is not available.
		Type	boolean
		Variation	--
		Direction	IN
Possible Errors	E_OK	Operation successful	
	E_NOT_OK	The request is not accepted	

Table 8.16: Operation SetFunctionAvailable

⌋(*SRS_Fim_04723*)

8.4.2 Implementation Data Types

8.4.2.1 FiM_FunctionIdType

[SWS_Fim_00027] ⌈

Name	FiM_FunctionIdType		
Kind	Type		
Derived from	*Base Type*	*Variation*	
	uint16	platform depended	
	uint8	platform depended	
Description	Type for the FunctionID		
Range	0..255, 0..65535		Identifier of functionality Configurable, size depends on System complexity. Remark: Not all numbers are valid. The FIM data generation tool shall only assign valid values.

Variation	--

Table 8.17: Implementation Data Type FiM_FunctionIdType

⌋(*SRS_BSW_00304*, *SRS_BSW_00305*, *SRS_BSW_00377*)

8.4.3 Ports

[SWS_Fim_00094] ⌈

Name	Func_{Name}		
Kind	ProvidedPort	***Interface***	FunctionInhibition
Description	A client can query the FiM for execution permission for a specific function. The FIDs which represent the functions are not directly used by the client SW-C. Instead, the mechanism of "port-defined argument values" is used and every FID is mapped to a separate port that is responsible for the data exchange via RTE.		
Port Defined Argument Value(s)	Type	FiM_FunctionIdType	
	Value	{ecuc(FiM/FiMConfigSet/FiMFID/ FiMFunctionId.value)}	
Variation	Name = {ecuc(FiM/FiMConfigSet/FiMFID.SHORT-NAME)}		

Table 8.18: Port Func_{Name}

⌋(*SRS_Fim_04700*)

[SWS_Fim_00108] ⌈

Name	Control_{Name}		
Kind	ProvidedPort	***Interface***	ControlFunctionAvailable
Description	A client can set the availability for a specific function.		
Port Defined Argument Value(s)	Type	FiM_FunctionIdType	
	Value	{ecuc(FiM/FiMConfigSet/FiMFID/ FiMFunctionId.value)}	
Variation	({ecuc(FiM/FiMGeneral/FiMAvailabilitySupport)} == True) Name = {ecuc(FiM/FiMConfigSet/FiMFID.SHORT-NAME)}		

Table 8.19: Port Control_{Name}

⌋(*SRS_Fim_04723*)

8.4.4 Internal Behavior

The InternalBehavior of the FiM Service is only seen by the local RTE. Additionally to the definition of the function identifiers as port defined arguments, the InternalBehavior has to specify the operation invoked runnables:

```
Internal Behavior FiM {

// definition of associated operation-invoked RTE-events not shown

// (it is done in the same way as for any SWC type)

// section "runnable entities":

RunnableEntity GetFunctionPermission

symbol "FiMGetFunctionPermission"

canbeInvokedConcurrently = TRUE

}
```

9 Sequence diagrams

9.1 Initialization sequence of FiM

[SWS_Fim_00102] ⌈ The initialization of Dem and Fim shall always follow the below order :

step 0) `Dem_PreInit`

step 1) Non-volatile memory data has to be available

step 2) `FiM_Init` (setting up internal variables); after `FiM_Init`, the Fim is not yet ready to be used.

step 3) `Dem_Init`: do the internal DEM initialization and use `FiM_DemInit` to finally initialize the FIM⌋*()*

Note: From step 3 onwards, the Dem and Fim are finally initialized and ready to be used.

[SWS_Fim_00103] ⌈ `FiM_DemInit` shall only be used during first `Dem_PreInit` after system start-up.⌋*()*

[SWS_Fim_00104] ⌈ `FiM_GetFunctionPermission` shall not be used before full initialization of FIM (`FiM_DemInit`).⌋*()*

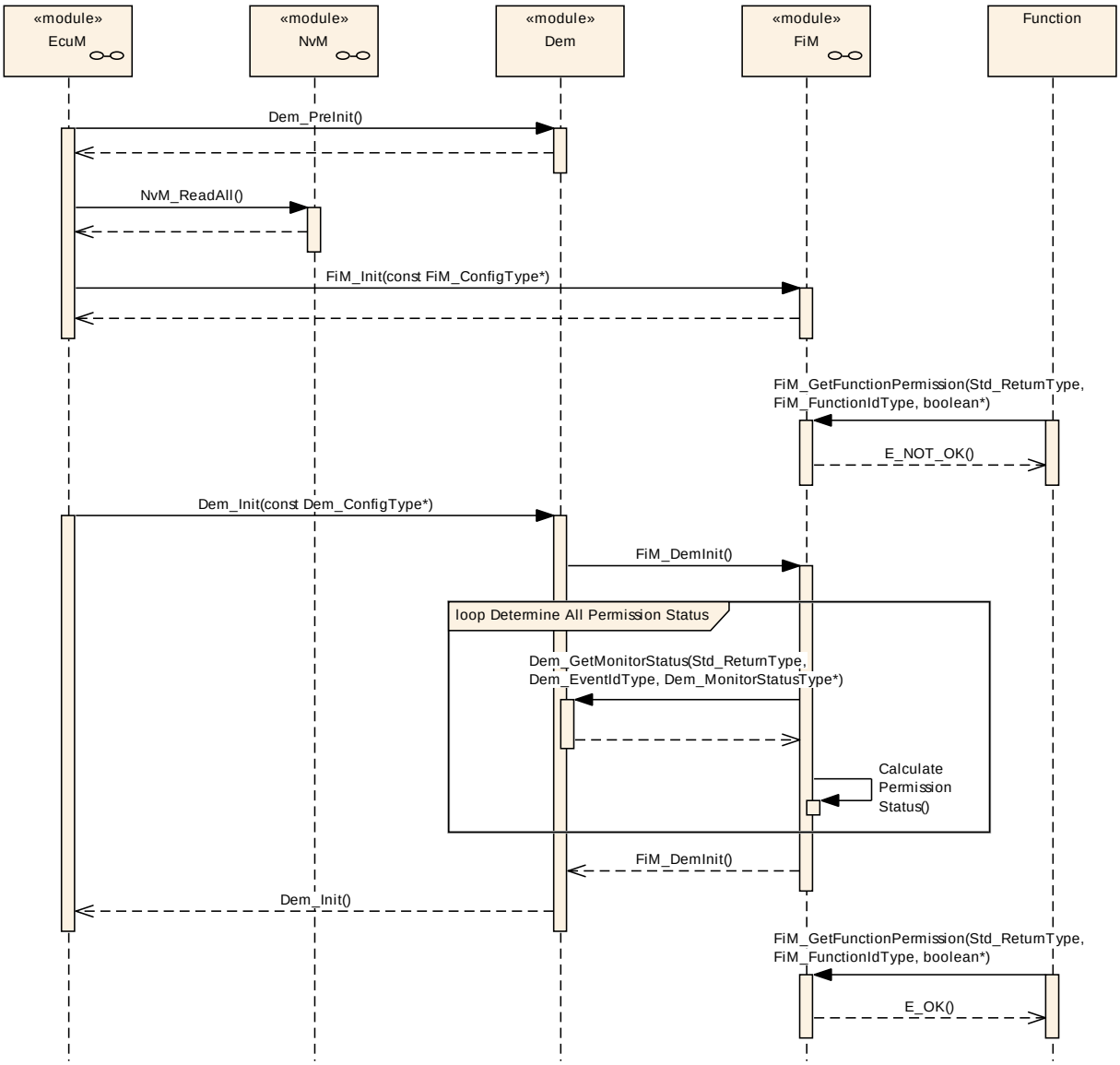

Figure 9.1: Initialization sequence of FiM

9.2 FiM_DemTriggerOnMonitorStatus

The sequence diagram below illustrates how the Dem informs the FiM about the change of a certain monitor status by calling `FiM_DemTriggerOnMonitorStatus`. Furthermore, it indicates how the FID is affected by requesting permission status using `FiM_GetFunctionPermission`.

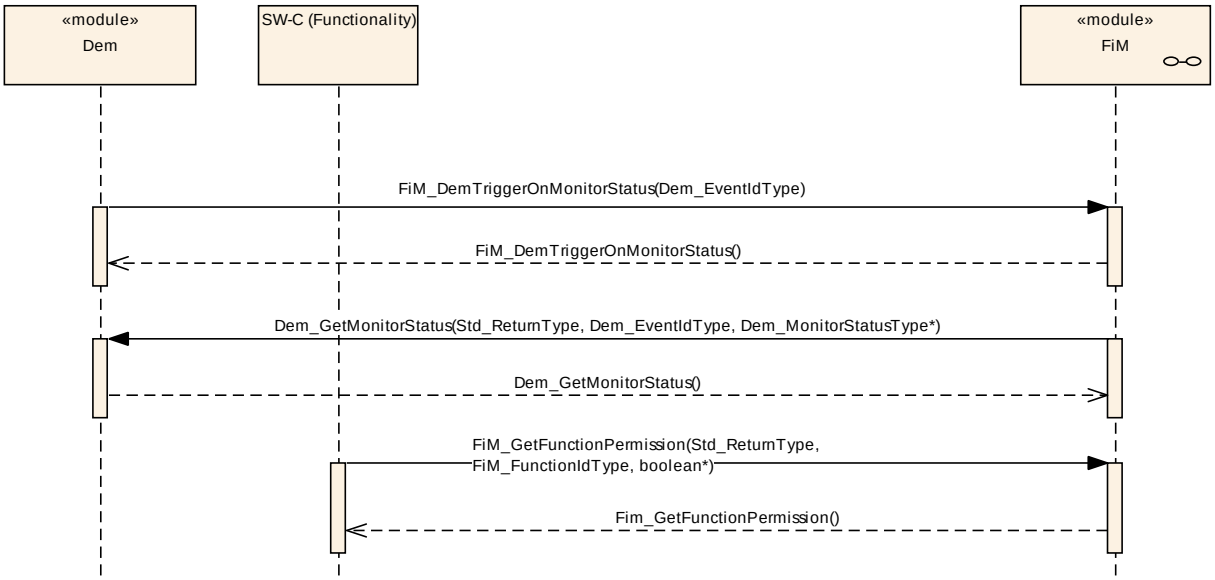

Figure 9.2: FiM_DemTriggerOnMonitorStatus

10 Configuration specification

In general, this chapter defines configuration parameters and their clustering into containers. In order to support the specification, Chapter 10.1 describes fundamentals. It also specifies a template (table) you shall use for the parameter specification. We intend to leave Chapter 10.1 in the specification to guarantee comprehension.

Chapter 10.2 specifies the structure (containers) and the parameters of the module Fi M.

Chapter 10.3 specifies published information of the module FiM.

10.1 How to read this chapter

For details refer to the chapter 10.1 "Introduction to configuration specification" in SWS_BSWGeneral [1].

10.2 Containers and configuration parameters

The following chapters summarize all configuration parameters. The detailed meanings of the parameters are described in Chapter 7 and Chapter 7.3.

[SWS_Fim_00062] ⌈

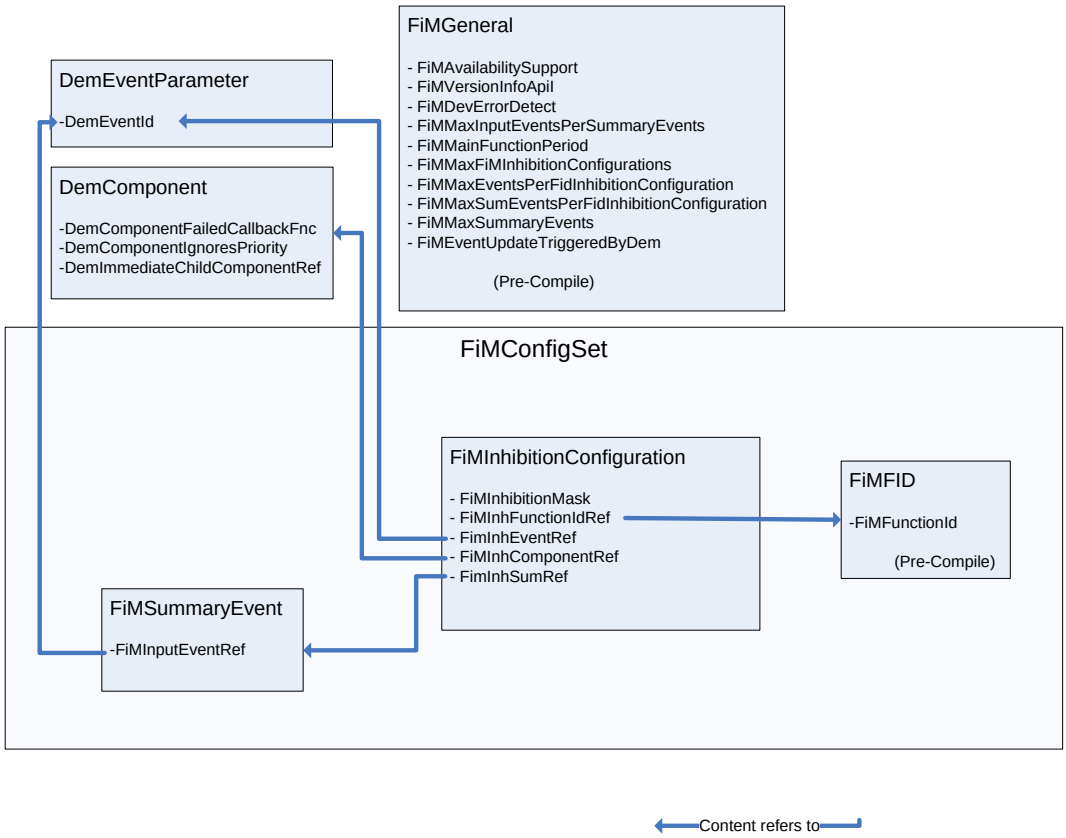

Figure 10.1: FiM configuration

(SRS_BSW_00404, SRS_BSW_00405)

10.2.1 FiM

Module SWS Item	ECUC_FiM_00612	
Module Name	FiM	
Module Description	Configuration of the FiM (Function Inhibition Manager) module.	
Post-Build Variant Support	true	
Supported Config Variants	VARIANT-POST-BUILD, VARIANT-PRE-COMPILE	
Included Containers		
Container Name	**Multiplicity**	**Scope / Dependency**
FiMConfigSet	1	This container contains the configuration parameters and sub containers of the FiM module supporting multiple configuration sets.
FiMGeneral	1	

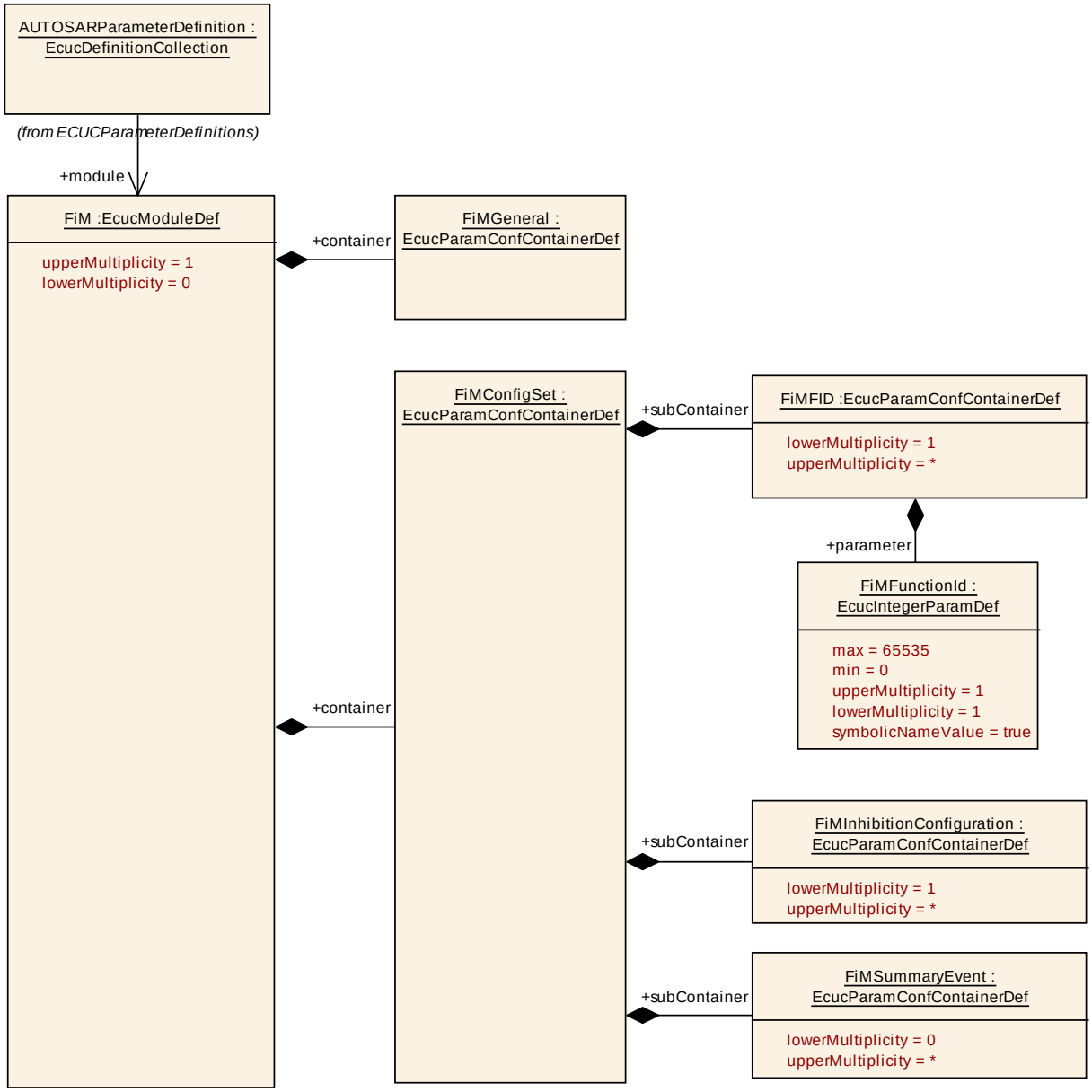

Figure 10.2: Configuration overview for FiM

10.2.2 FiMGeneral

SWS Item	[ECUC_FiM_00040]
Container Name	FiMGeneral
Description	
Configuration Parameters	

Name	FiMAvailabilitySupport [ECUC_FiM_00610]		
Parent Container	FiMGeneral		
Description	This configuration parameter specifies, if the Fim shall support the service to set the Availabity of a Funtionality. true: Service is supported. false: Service is not supported		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value	false		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	FiMDevErrorDetect [ECUC_FiM_00087]		
Parent Container	FiMGeneral		
Description	Switches the development error detection and notification on or off. • true: detection and notification is enabled. • false: detection and notification is disabled.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value	false		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	FiMEventUpdateTriggeredByDem [ECUC_FiM_00086]
Parent Container	FiMGeneral
Description	This configuration parameter specifies the way FIM obtains status of EventIds. TRUE: the DEM informs FIM about changes of monitor status, FALSE: the FIM polls monitor status from the DEM module either cyclically or on demand.
Multiplicity	1
Type	EcucBooleanParamDef
Default Value	
Post-Build Variant Value	false

Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	FiMMainFunctionPeriod [ECUC_FiM_00611]		
Parent Container	FiMGeneral		
Description	Allow to configure the time for the periodic cyclic task. Please note: This configuration value shall be equal to the value in the Basic Software Scheduler configuration of the RTE module. The AUTOSAR configuration standard is to use SI units, so this parameter is defined as float value in seconds. FiM configuration tools shall convert this float value to the appropriate value format for the use in the software implementation of FiM.		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	]0 .. INF[		
Default Value			
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	FiMMaxEventsPerFidInhibitionConfiguration [ECUC_FiM_00608]		
Parent Container	FiMGeneral		
Description	This configuration parameter specifies the total maximum number of inhibiting events in a FiMInhibitionConfiguration. Its applicable for post build configuration versions only and may be used to allocate the maximum size of memory to store and execute the configuration.		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	1 .. 65535		
Default Value			
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	FiMMaxFiMInhibitionConfigurations [ECUC_FiM_00606]		
Parent Container	FiMGeneral		
Description	This configuration parameter specifies the total maximum number of FiMInhibitionConfigurations. Its applicable for post build configuration versions only and may be used to allocate the maximum size of memory to store and execute the configuration.		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	1 .. 65535		
Default Value			
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	FiMMaxInputEventsPerSummaryEvents [ECUC_FiM_00609]		
Parent Container	FiMGeneral		
Description	This configuration parameter specifies the total maximum number of input events per summary event. Its applicable for post build configuration versions only and may be used to allocate the maximum size of memory to store and execute the configuration.		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	1 .. 65535		
Default Value			
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	FiMMaxSumEventsPerFidInhibitionConfiguration [ECUC_FiM_00607]		
Parent Container	FiMGeneral		
Description	This configuration parameter specifies the total maximum number of inhibiting summary events in a FiMInhibitionConfiguration. Its applicable for post build configuration versions only and may be used to allocate the maximum size of memory to store and execute the configuration.		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	1 .. 65535		

Default Value			
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	FiMMaxSummaryEvents [ECUC_FiM_00091]		
Parent Container	FiMGeneral		
Description	This configuration parameter specifies the maximum number of summarized events that can be configured.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default Value			
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	FiMVersionInfoApi [ECUC_FiM_00094]		
Parent Container	FiMGeneral		
Description	This configuration parameter is used to switch on or to switch off the API to get the version information.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value	false		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

No Included Containers

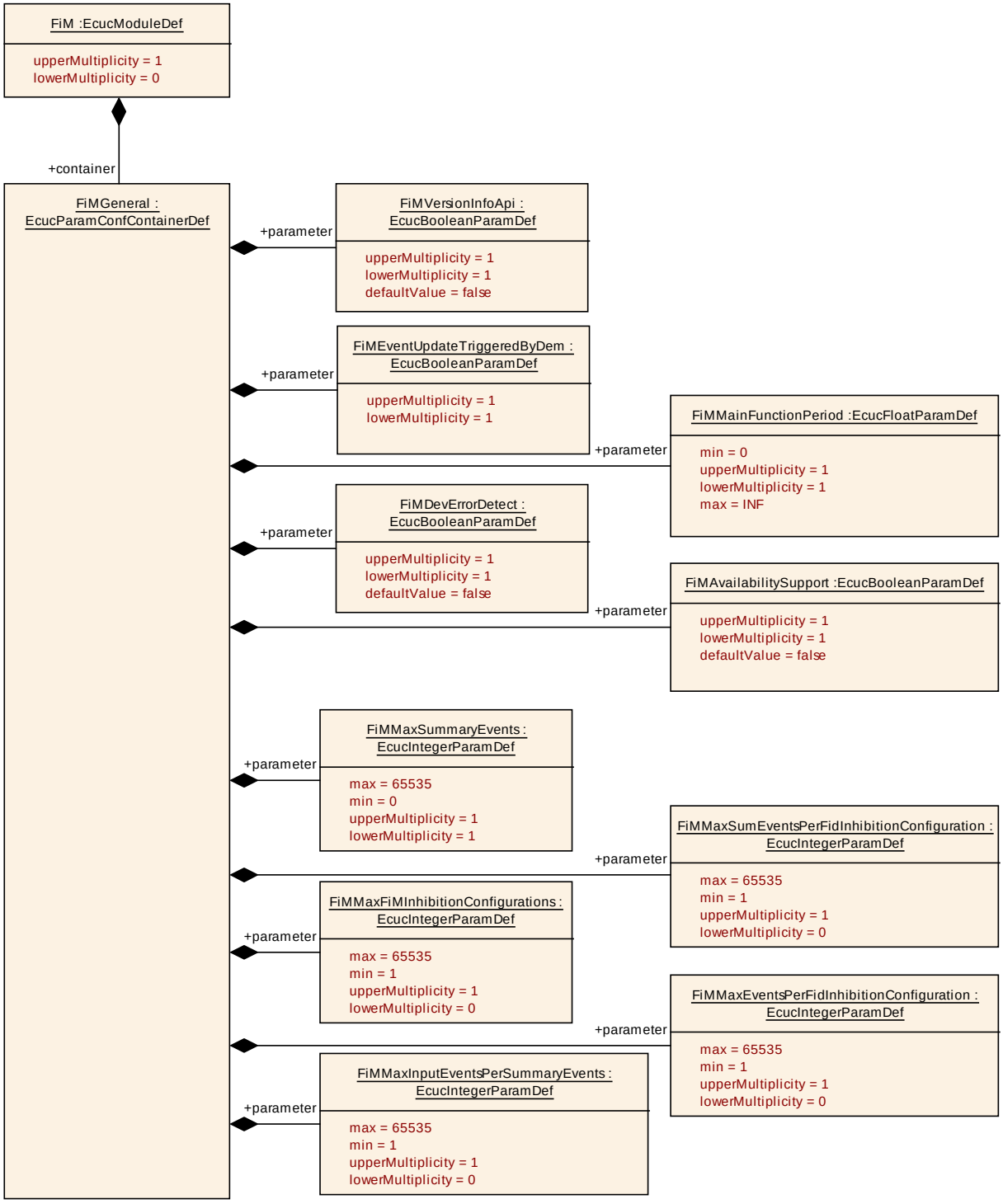

Figure 10.3: Configuration overview for FiMGeneral

10.2.3 FiMConfigSet

SWS Item	[ECUC_FiM_00601]
Container Name	FiMConfigSet

Description	This container contains the configuration parameters and sub containers of the FiM module supporting multiple configuration sets.
Configuration Parameters	

Included Containers		
Container Name	**Multiplicity**	**Scope / Dependency**
FiMFID	1..*	This container includes symbolic names of all FIDs.
FiMInhibition Configuration	1..*	This container includes all configuration parameters concerning the relationship between event and FID.
FiMSummaryEvent	0..*	The summarized EventId definition record consists of a summarized event ID and specific Dem Events. This record means that a particular FID that has to be disabled in case of summarized event (defined above) is to be disabled in any of the specific events. A possible solution could be assigning events as summarized events along with a list of specific events. During the configuration process the summarized event substitutes the referenced single events. However, it is not outlined how this requirement is solved - whether by configuration process or by implementation within the FiM. The FiM configuration tool could also build up a suitable data structure for summarized events and deal with it in the FiM implementation.

10.2.4 FiMFID

SWS Item	[ECUC_FiM_00039]
Container Name	FiMFID
Description	This container includes symbolic names of all FIDs.
Configuration Parameters	

Name	FiMFunctionId [ECUC_FiM_00085]	
Parent Container	FiMFID	
Description	Unique identifier of a FimFunctionId. This parameter should not be changeable by user, because the Id should be generated by Fim itself to prevent gaps and multiple use of an Id. Note: The implementer can add the attribute 'withAuto' to the parameter definition which indicates that the value can be calculated by the generator automatically. When 'withAuto' is set to 'true' for this parameter definition the 'isAutoValue' can be set to 'true'. If 'isAutoValue' is set to 'true' the actual value will not be considered during ECU Configuration but will be (re-)calculated by the code generator and stored in the value attribute afterwards.	
Multiplicity	1	
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)	
Range	0 .. 65535	
Default Value		

Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: ECU		

No Included Containers

10.2.5 FiMInhibitionConfiguration

SWS Item	[ECUC_FiM_00038]		
Container Name	FiMInhibitionConfiguration		
Description	This container includes all configuration parameters concerning the relationship between event and FID.		
Post-Build Variant Multiplicity	true		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Configuration Parameters			

Name	FiMInhInhibitionMask [ECUC_FiM_00096]	
Parent Container	FiMInhibitionConfiguration	
Description	The configuration parameter is used to specify the inhibition mask for an event - FID relation.	
Multiplicity	1	
Type	EcucEnumerationParamDef	
Range	FIM_LAST_FAILED	Last Failed - DEM_UDS_STATUS_TF flag of Dem Eventstatus is set Use case: Re-configuration, avoiding follow-up errors
	FIM_NOT_TESTED	Not Tested this cycle - DEM_UDS_STATUS_TNCTOC flag of Dem Eventstatus is set. Use case: Scheduling of monitors.
	FIM_TESTED	Tested - DEM_UDS_STATUS_TNCTOC flag of Dem Eventstatus is not set. Use case: Self deactivation, check during driving cycle.

	FIM_TESTED_AND_FAILED	Tested and Failed - DEM_UDS_STATUS_TF flag of Dem Eventstatus is set and DEM_UDS_STATUS_TNCTOC flag is not set Use case: Avoiding deadlocks, repeated monitoring.	
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

Name	FiMInhComponentRef [ECUC_FiM_00605]		
Parent Container	FiMInhibitionConfiguration		
Description	Reference to a DemComponent which is necessary for function permission.		
Multiplicity	0..*		
Type	Reference to DemComponent		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

Name	FiMInhEventRef [ECUC_FiM_00100]		
Parent Container	FiMInhibitionConfiguration		
Description	Selection of an single DEM Event.		
Multiplicity	0..*		
Type	Symbolic name reference to DemEventParameter		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD

Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

Name	FiMInhFunctionIdRef [ECUC_FiM_00095]		
Parent Container	FiMInhibitionConfiguration		
Description			
Multiplicity	1		
Type	Reference to FiMFID		
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

Name	FiMInhSumRef [ECUC_FiM_00102]		
Parent Container	FiMInhibitionConfiguration		
Description	Selection of a summarized Event.		
Multiplicity	0..*		
Type	Reference to FiMSummaryEvent		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

No Included Containers

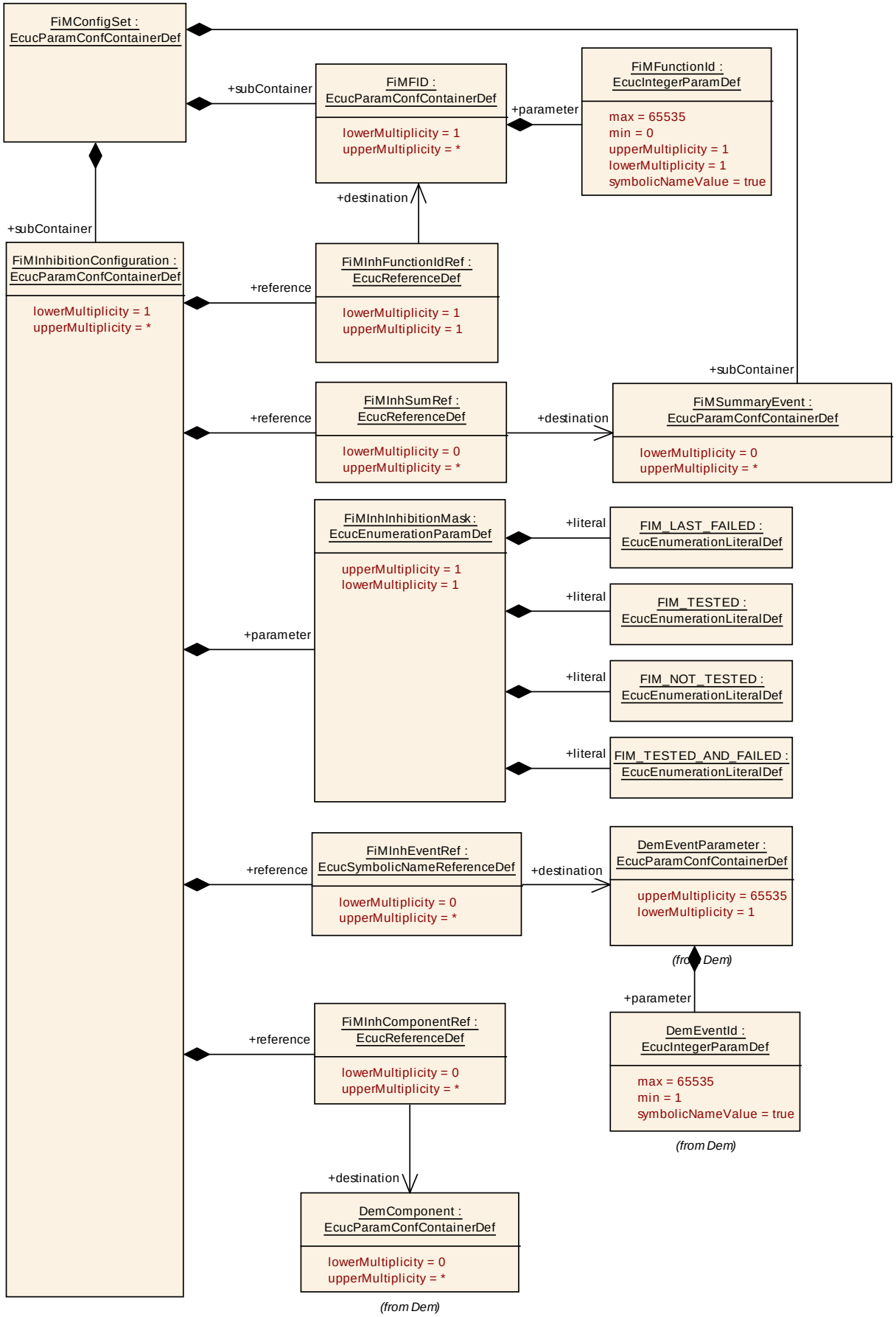

Figure 10.4: Configuration overview for FiMInhibitionConfiguration

10.2.6 FiMSummaryEvent

SWS Item	[ECUC_FiM_00603]		
Container Name	FiMSummaryEvent		
Description	The summarized EventId definition record consists of a summarized event ID and specific Dem Events. This record means that a particular FID that has to be disabled in case of summarized event (defined above) is to be disabled in any of the specific events. A possible solution could be assigning events as summarized events along with a list of specific events. During the configuration process the summarized event substitutes the referenced single events. However, it is not outlined how this requirement is solved - whether by configuration process or by implementation within the FiM. The FiM configuration tool could also build up a suitable data structure for summarized events and deal with it in the FiM implementation.		
Post-Build Variant Multiplicity	true		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Configuration Parameters			

Name	FiMInputEventRef [ECUC_FiM_00604]		
Parent Container	FiMSummaryEvent		
Description	Reference to DemEventParameters combined to this summarized event.		
Multiplicity	1..*		
Type	Symbolic name reference to DemEventParameter		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

No Included Containers

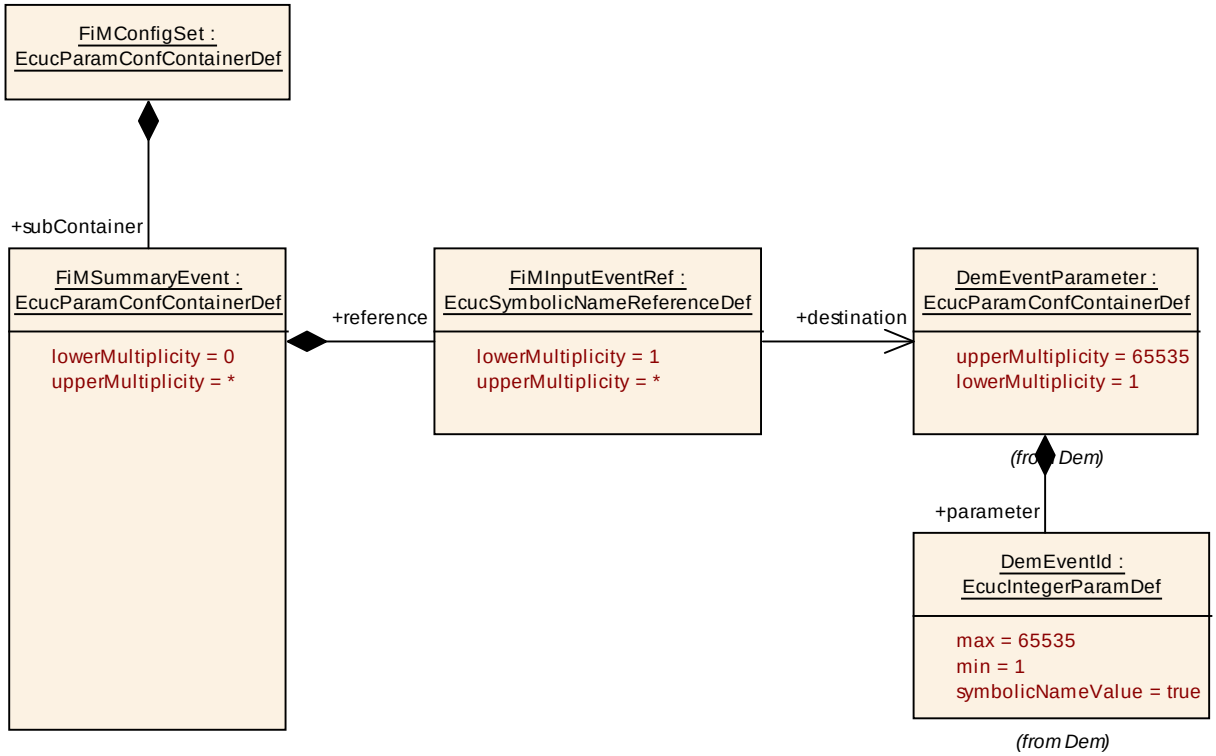

Figure 10.5: Configuration overview for FiMSummaryEvent

10.3 Published Information

For details refer to the chapter 10.3 "Published Information" in SWS_BSWGeneral[1].

A Not applicable requirements

[SWS_Fim_00999] ⌈ These requirements are not applicable to this specification.⌋ *(SRS_BSW_00301, SRS_BSW_00302, SRS_BSW_00306, SRS_BSW_00307, SRS_BSW_00308, SRS_BSW_00309, SRS_BSW_00314, SRS_BSW_00323, SRS_BSW_00325, SRS_BSW_00328, SRS_BSW_00330, SRS_BSW_00333, SRS_BSW_00334, SRS_BSW_00336, SRS_BSW_00342, SRS_BSW_00343, SRS_BSW_00347, SRS_BSW_00353, SRS_BSW_00357, SRS_BSW_00359, SRS_BSW_00360, SRS_BSW_00361, SRS_BSW_00375, SRS_BSW_00378, SRS_BSW_00386, SRS_BSW_00409, SRS_BSW_00417, SRS_BSW_00422, SRS_BSW_00423, SRS_BSW_00424, SRS_BSW_00425, SRS_BSW_00426, SRS_BSW_00427, SRS_BSW_00428, SRS_BSW_00429, SRS_BSW_00432, SRS_BSW_00433, SRS_Fim_04721)*