Document Title	Specification of FlexRay Driver
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	26

Document Status	published
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	R23-11

Document Change History

Date	Release	Changed by	Description
2023-11-23	R23-11	AUTOSAR Release Management	• Remove all HandleId configuration parameters
2022-11-24	R22-11	AUTOSAR Release Management	• Changed `Kind` and `Derived from` of `Fr_ConfigType` • Changed SWS_Fr_00602 to [SWS_Fr_NA_00602]
2021-11-25	R21-11	AUTOSAR Release Management	• No content changes
2020-11-30	R20-11	AUTOSAR Release Management	• Reworked chapter "Error Classification" • Changed exposure of `Fr_ConfigType`
2019-11-28	R19-11	AUTOSAR Release Management	• Enhanced multi core usage support • Changed Document Status from Final to published
2018-10-31	4.4.0	AUTOSAR Release Management	• Supports BusMirror concept • Enhanced multi core usage (DRAFT) • Editorial changes
2017-12-08	4.3.1	AUTOSAR Release Management	• Removed references to HIS • Renamed "default error" to "development error" • minor corrections / clarifications / editorial changes; for details please refer to the ChangeDocumentation

▽

△

2016-11-30	4.3.0	AUTOSAR Release Management	• Added TX conflict detectionsupport • Editorial changes
2015-07-31	4.2.2	AUTOSAR Release Management	• Changed development errors to default errors
2014-10-31	4.2.1	AUTOSAR Release Management	• Removed obsolete configuration parameters • Improved description of "Extended Production Errors"
2014-03-31	4.1.3	AUTOSAR Release Management	• Removed `NULL_PTR` Det check for `Fr_Init`(). • Splitted config parameter FrBufferReconfig into 3 config parameters `FrPrepareLPduSupport`, `FrReconfigLPduSupport` and `FrDisableLPduSupport`. • Replaced Dem events by genuine uppercase letters • Removed integrator requirement for `Fr_GeneralTypes.h`
2013-10-31	4.1.2	AUTOSAR Release Management	• Added new DET error `FR_E_INV_FRAMELIST_SIZE` • Editorial changes • Removed chapter(s) on change documentation
2013-03-15	4.1.1	AUTOSAR Administration	• Clarifications and corrections of existing requirements • Reclassification of production errors to extended production errors.
2011-12-22	4.0.3	AUTOSAR Administration	• Small corrections and clarification on existing features
2010-09-30	3.1.5	AUTOSAR Administration	• New service for reading the FlexRay configuration parameters at runtime • Update of configuration parameters according to the FlexRay Protocol Specification 3.0

▽

△

2010-02-02	3.1.4	AUTOSAR Administration	• Added support for FlexRay Protocol 3.0 compliant FlexRay controllers • Added receive-FIFO support including Message-ID filtering • New services to retrieve diagnostic- and status information (clock correction, sync frame tables, aggregated channel status, slot status) • Added transmission cancelation support. • Removed relative timer support • Legal disclaimer revised
2008-08-13	3.1.1	AUTOSAR Administration	• Legal disclaimer revised
2008-02-01	3.0.2	AUTOSAR Administration	• Added NM-Vector Support • Added dynamic frame length support for dynamic FlexRay segment • Added API service for coldstart control • Document meta information extended • Small layout adaptations made
2007-12-21	3.0.1	AUTOSAR Administration	• Renamed members of Fr_POCStatusType according to FlexRay Protocol Specification 2.1 • Renamed API function Fr_TransmitTxLSdu(), Fr_CheckTxLSduStatus(), Fr_ReceiveRxLSdu() and related API types. • Added new API function Fr_PrepareLPdu() • Added new API function Fr_StopMTS() • Added reference to BSW Scheduler SWS • Reworked API function DET and DEM reporting • Reworked DET error codes ▽

▽

△

			△ • Updated traceability table • Legal disclaimer revised • Release Notes added • ”Advice for users” revised • ”Revision Information” added
2006-05-16	2.0	AUTOSAR Administration	• Second release
2005-05-31	1.0	AUTOSAR Administration	• Initial release

Contents

1 Introduction and functional overview

This specification describes the functionality, API and the configuration for the AUTOSAR Basic Software module FlexRay Driver (Fr).

The FlexRay Driver abstracts the hardware related implementation details of specific FlexRay Communication Controllers (CC). This specification basically relies on FlexRay CCs compliant to the FlexRay specification [1]. Additionally older FlexRay controllers compliant to FlexRay specification [2] are supported by this specification. Different behaviours in this SWS resulting from the different supported FlexRay specifications are pointed out as footnotes or remarks where applicable.

All supported features of a FlexRay controller are encapsulated within the Fr module and shall be accessed via this uniform interface only. The APIs provide abstract functional operations that are mapped to a sequence of hardware accesses depending on the actual implemented Fr module. Thus, the FlexRay Interface (FrIf), as the user of the Fr module, is independent of the underlying FlexRay CC hardware. The Fr module doesn't have a main-function or an ISR. All Fr module API functions are executed only in the context of the FrIf.

A single Fr module supports only a single type of FlexRay CC hardware implementation. The Fr supports multiple FlexRay CCs of this single hardware implementation. The FlexRay Driver's prefix is uniquely assigned per Fr module to allow usage of different FlexRay Drivers, the names of which are separated by namespace. The FrIf can access different FlexRay CC hardware implementations using different FlexRay Drivers. The FrIf configuration determines which driver from among different types is used to access a particular CC.

The configuration of the Fr module shall be done at system configuration time, with the Fr module's specific configuration being generated by a Module Configuration Generator (MCG), which translates the parameters out of the ECU configuration parameters to Fr module specific configuration data structures.

Figure 1.1 depicts the basic structure of the FlexRay stack. One FrIf accesses several CCs using one or several FlexRay Drivers.

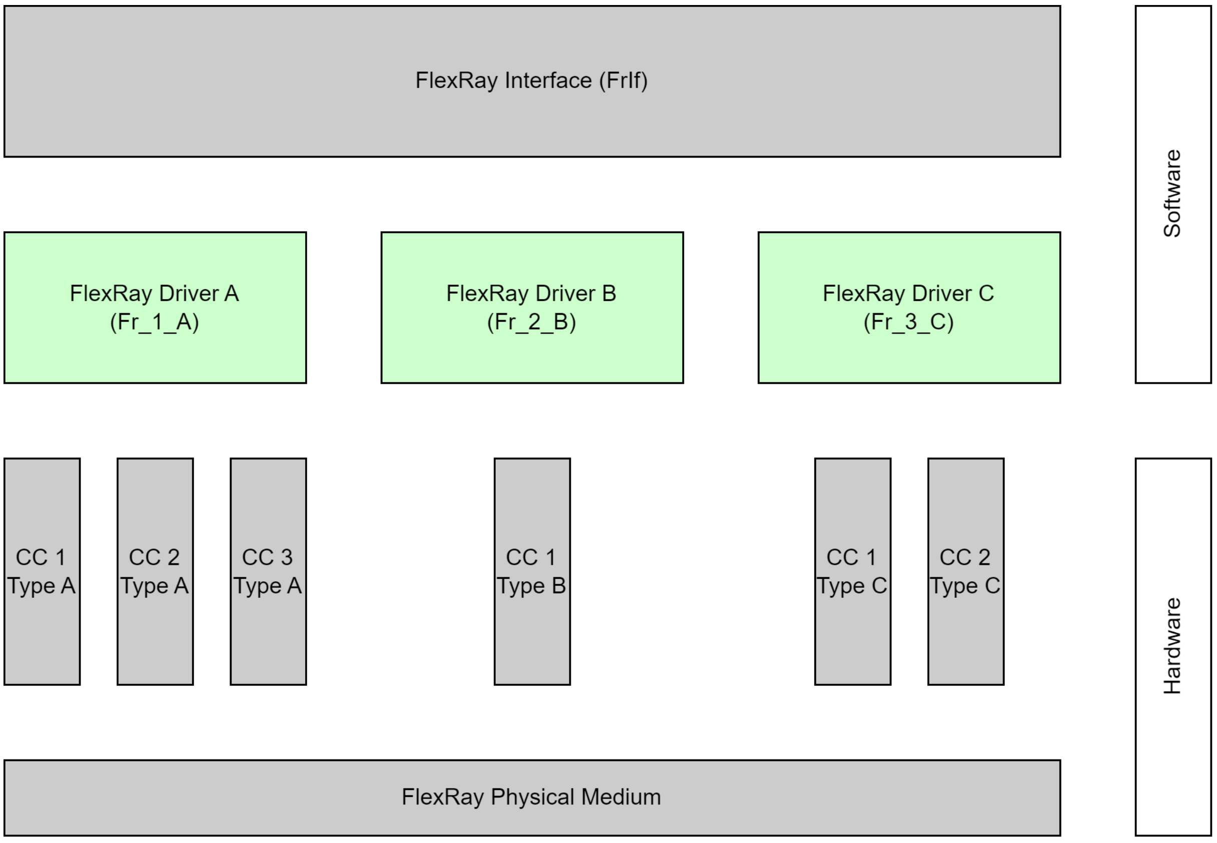

Figure 1.1: FlexRay stack module overview

2 Acronyms and Abbreviations

The glossary below includes acronyms and abbreviations relevant to the FlexRay Driver module that are not included in the [3, AUTOSAR glossary].

Abbreviation / Acronym:	**Description:**
API	Application Programming Interface
AUTOSAR	Automotove Open Systems Architecture
BSW	Basic Software
DEM/Dem	Autosar Module: Diagnostic Event Manager
DET/Det	Autosar Module: Default Error Tracer
ECU	Electronic Control Unit
MCG	Module Configuration Generator
CC	Communication Controller
CHI	Controller Host Interface
FIFO	First In First Out buffer
Fr	Autosar Module: FlexRay Driver
FrIf	Autosar Module: FlexRay Interface
FrTp	Autosar Module: FlexRay Transport Protocol
FrTrcv	Autosar Module: FlexRay Transceiver Driver
ID/Id	Identifier
ISR	Interrupt Service Routine
LPdu	Datalink layer Protocol Datagram Unit
MCAL	Microcontroller Abstraction Layer
MCU	Microcontroller Unit
MISRA	Motor Industry Software Reliability Association
NIT	FlexRay Network Idle Time
n/a	Not Applicable
OS	Operating System
PLL	Phase Locked Loop
POC	Protocol Operation Control (see [1] for details)
POCState	Actual CC internal state of the POC. This state might differ from vPOC!State in certain cases, e.g., after FREEZE command invocation (see [1] for details).
SchM	Autosar Module: Schedule Manager
SRS	System Requirements Specification
SW	SoftWare
SW-C	SoftWare Component
vPOC	Data structure provided from the CC to the host at the CHI, which contains the actual POC status of the CC (see [1] for details).
XML	Extensible Markup Language

Table 2.1: Acronyms and abbreviations used in the scope of this Document

Term:	Description:
absolute timer	An absolute timer is set to and triggered by an absolute global time of a FlexRay cluster. The FlexRay global time consists of a cycle and a macrotick offset
buffer	A buffer in the context of the Fr SWS describes a hardware transmit/receive resource, part of the FlexRay controller that is mapped to a FlexRay slot, channel, cycle for transmission or reception.
cluster	A communication system of multiple nodes connected to each other.
Macrotick	The macrotick represents the smallest unit of the global synchronized time of a FlexRay cluster.
Synchronized	A FlexRay CC is considered synchronized, to the FlexRay cluster connected to, as long as the following condition holds true: `((!vPOC!Freeze) && (vPOC!State == NORMAL_ACTIVE) || (vPOC!State == NORMAL_PASSIVE))`

Table 2.2: Terms used in the scope of this Document

3 Related documentation

3.1 Input documents & related standards and norms

[1] ISO 17458-1:2013, Road vehicles - FlexRay communication system
https://www.iso.org

[2] FlexRay Communications System Protocol Specification V2.1
http://www.flexray.com/

[3] Glossary
AUTOSAR_FO_TR_Glossary

[4] General Specification of Basic Software Modules
AUTOSAR_CP_SWS_BSWGeneral

[5] Specification of Standard Types
AUTOSAR_CP_SWS_StandardTypes

[6] Specification of FlexRay Interface
AUTOSAR_CP_SWS_FlexRayInterface

3.2 Related specification

AUTOSAR provides a General Specification on Basic Software modules [4], which is also valid for FlexRay Driver.

Thus, the specification SWS BSW General shall be considered as additional and required specification for FlexRay Driver.

4 Constraints and assumptions

4.1 Limitations

[SWS_Fr_00449] ⌈In the dynamic segment of each FlexRay Communication Cycle, a transmit/receive buffer of a FlexRay Communication Controller shall be used only one particular LPdu. This limits the reconfiguration possibilities and thus restricts the number of transmittable (sent and received) LPdus per dynamic segment to the accumulated number (over all CCs on one ECU) of transmit/receive buffers connected to one cluster. This limitation results from the unpredictability of the time of transmission of an LPdu within the dynamic segment. Because of that a point in time for the reconfiguration of a certain buffer for multiple usages within the dynamic segment cannot be predetermined.⌋*()*

4.2 Applicability to car domains

The FlexRay Communication stack can be used wherever high data rates and fault tolerant communication (in conjunction with [1]) are required. Furthermore it enables the synchronized operation of several ECUs within a car.

5 Dependencies to other modules

This chapter lists the modules interacting with the Fr module.

Modules that use Fr module:

- The FrIf is the only user of the Fr module (except initialization by EcuM). It uses the Fr module(s) to access possibly different FlexRay Communication Controllers in a uniform and abstract way.
- The EcuM initializes the Fr module by calling Fr_Init.

Modules used by the Fr module:

- **[SWS_Fr_00453]** ⌈The Fr module shall use the BSW Scheduler mechanisms for data consistency when required.⌋*()*

Other Module dependencies:

- **[SWS_Fr_00454]** ⌈On certain systems the CC might share resources with other components (e.g., the MCU), and might depend on their configurations. If those resources are within the scope of the other modules (e.g., PLL configuration, memory mapping), then the Fr module doesn't configure those components but requires that their initialization precede the Fr module's initialization.⌋*()*

5.1 File structure

This section gives an overview about the files and their relations required for a proper implementation of the Fr module. Please note that the file structure is not completely specified but the implementation shall use at least the files and the file structure presented in this section.

5.1.1 Code file structure

[SWS_Fr_00116] ⌈The code file structure shall not be completely defined within this specification.⌋*(SRS_BSW_00346, SRS_BSW_00380)*

5.1.2 Header file structure

[SWS_Fr_00464] ⌈The file `Fr.h` shall contain all types and function prototypes required by the Fr module's environment.⌋*()*

[SWS_Fr_00117] ⌈`Fr_GeneralTypes.h` shall contain all types and constants that are shared among the AUTOSAR FlexRay modules Fr, FrIf and FrTrcv.⌋*()*

6 Requirements Tracing

Requirement	Description	Satisfied by
[SRS_BSW_00003]	All software modules shall provide version and identification information	[SWS_Fr_00080]
[SRS_BSW_00305]	Data types naming convention	[SWS_Fr_00077]
[SRS_BSW_00307]	Global variables naming convention	[SWS_Fr_00098]
[SRS_BSW_00308]	AUTOSAR Basic Software Modules shall not define global data in their header files, but in the C file	[SWS_Fr_00102]
[SRS_BSW_00334]	All Basic Software Modules shall provide an XML file that contains the meta data	[SWS_Fr_00080]
[SRS_BSW_00336]	Basic SW module shall be able to shutdown	[SWS_Fr_00014]
[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_Fr_00097]
[SRS_BSW_00345]	BSW Modules shall support pre-compile configuration	[SWS_Fr_00027]
[SRS_BSW_00346]	All AUTOSAR Basic Software Modules shall provide at least a basic set of module files	[SWS_Fr_00116]
[SRS_BSW_00347]	A Naming seperation of different instances of BSW drivers shall be in place	[SWS_Fr_00076]
[SRS_BSW_00348]	All AUTOSAR standard types and constants shall be placed and organized in a standard type header file	[SWS_Fr_00099]
[SRS_BSW_00353]	All integer type definitions of target and compiler specific scope shall be placed and organized in a single type header	[SWS_Fr_00099]
[SRS_BSW_00358]	The return type of init() functions implemented by AUTOSAR Basic Software Modules shall be void	[SWS_Fr_00032]
[SRS_BSW_00374]	All Basic Software Modules shall provide a readable module vendor identification	[SWS_Fr_00080]
[SRS_BSW_00379]	All software modules shall provide a module identifier in the header file and in the module XML description file.	[SWS_Fr_00080]
[SRS_BSW_00380]	Configuration parameters being stored in memory shall be placed into separate c-files	[SWS_Fr_00116]
[SRS_BSW_00404]	BSW Modules shall support post-build configuration	[SWS_Fr_00027] [SWS_Fr_00032]
[SRS_BSW_00405]	BSW Modules shall support multiple configuration sets	[SWS_Fr_00032]
[SRS_BSW_00407]	Each BSW module shall provide a function to read out the version information of a dedicated module implementation	[SWS_Fr_00070]

▽

△

Requirement	Description	Satisfied by
[SRS_BSW_00411]	All AUTOSAR Basic Software Modules shall apply a naming rule for enabling/disabling the existence of the API	[SWS_Fr_00070] [SWS_Fr_00340]
[SRS_BSW_00413]	An index-based accessing of the instances of BSW modules shall be done	[SWS_Fr_00075]
[SRS_BSW_00414]	Init functions shall have a pointer to a configuration structure as single parameter	[SWS_Fr_00032]
[SRS_BSW_00438]	Configuration data shall be defined in a structure	[SWS_Fr_00137]
[SRS_BSW_00441]	Naming convention for type, macro and function	[SWS_Fr_00505] [SWS_Fr_00506] [SWS_Fr_00507] [SWS_Fr_00508] [SWS_Fr_00509] [SWS_Fr_00511] [SWS_Fr_00512] [SWS_Fr_00514]
[SRS_Fr_05003]	Slot/Cycle Multiplexing shall be supported	[SWS_Fr_00005] [SWS_Fr_00092] [SWS_Fr_00093] [SWS_Fr_00094]
[SRS_Fr_05005]	The CC Hardware FIFO Mechanism shall be supported	[SWS_Fr_00593] [SWS_Fr_00594] [SWS_Fr_00595] [SWS_Fr_00596] [SWS_Fr_00597]
[SRS_Fr_05006]	Abstraction of FlexRay-Specific Features shall be provided	[SWS_Fr_00593]
[SRS_Fr_05011]	Initialization of the Low-Level Parameters shall be available	[SWS_Fr_00017]
[SRS_Fr_05012]	Initialization of the FlexRay CC Transmit/Receive Buffers shall be available	[SWS_Fr_00148]
[SRS_Fr_05019]	FlexRay Global Time shall be provided	[SWS_Fr_00042]
[SRS_Fr_05024]	The software interface of the Driver shall be independent of the CC buffers' configuration	[SWS_Fr_00005] [SWS_Fr_00092] [SWS_Fr_00093] [SWS_Fr_00094] [SWS_Fr_00440] [SWS_Fr_00441] [SWS_Fr_00610]
[SRS_Fr_05044]	CC's Absolute Timer shall be provided	[SWS_Fr_00033] [SWS_Fr_00095]
[SRS_Fr_05046]	Absolute Alarms of a CC shall be enabled	[SWS_Fr_00034]
[SRS_Fr_05047]	Absolute Alarms of a CC shall be disabled	[SWS_Fr_00035]
[SRS_Fr_05048]	Absolute Alarms of a CC shall be acknowledged	[SWS_Fr_00036]
[SRS_Fr_05055]	Timer Interrupts during Shutdown shall be avoided	[SWS_Fr_00106]
[SRS_Fr_05058]	The configuration of the FlexRay Driver shall be defined at system configuration time.	[SWS_Fr_00480]
[SRS_Fr_05059]	The Driver shall be configure the CC's transmit/receive buffers	[SWS_Fr_00148] [SWS_Fr_00524] [SWS_Fr_00539]
[SRS_Fr_05064]	Abstraction of FlexRay CC-specific Implementation shall be provided	[SWS_Fr_00465] [SWS_Fr_00466]
[SRS_Fr_05065]	The FlexRay Driver shall be able to communicate with at least four Flex Ray CCs of the same type	[SWS_Fr_00467]

▽

△

Requirement	Description	Satisfied by
[SRS_Fr_05072]	The FlexRay Driver shall raise an error if the FlexRay Time Services function is called after the communication of the CC is Out of Sync	[SWS_Fr_00044]
[SRS_Fr_05106]	The Buffer of a specific CC in Normal Active Mode shall be reconfigurable	[SWS_Fr_00107]
[SRS_Fr_05109]	The FlexRay Driver shall provide a software interface to start-up a specific FlexRay CC	[SWS_Fr_00010]
[SRS_Fr_05114]	A FlexRay CC Communication shall be aborted when wanted	[SWS_Fr_00011]
[SRS_Fr_05115]	The FlexRay CC Communication shall be halted when wanted	[SWS_Fr_00014]
[SRS_Fr_05116]	Initialization of FlexRay CC shall be available	[SWS_Fr_00017]
[SRS_Fr_05117]	A Wake-Up Pattern shall be sent on a specific channel of a CC	[SWS_Fr_00009] [SWS_Fr_00091]
[SRS_Fr_05120]	FlexRay CC POC Status shall be provided	[SWS_Fr_00012]
[SRS_Fr_05125]	The FlexRay Driver shall provide services to handle interrupts of a Flex Ray Communication Controller.	[SWS_Fr_00034] [SWS_Fr_00035] [SWS_Fr_00036] [SWS_Fr_00108]
[SRS_Fr_05169]	Timer Interrupts during Start-up shall be avoided	[SWS_Fr_00152]

Table 6.1: RequirementsTracing

7 Functional specification

7.1 General description

[SWS_Fr_00465] ⌈A single Fr module offers a uniform way to use features of FlexRay CCs independent of the CC implementation, thus hiding the actual hardware implementation (registers, buffers, etc.) from upper layers.⌋*(SRS_Fr_05064)*

[SWS_Fr_00466] ⌈The Fr module maps abstract functional requests to sequences of CC specific hardware accesses.⌋*(SRS_Fr_05064)*

A detailed description for all API services can be found in chapter 8.

7.2 Implementation Requirements

This chapter lists requirements that shall be fulfilled by Fr module implementations.

[SWS_Fr_00076] ⌈The Fr module's implementer shall replace all prefixes Fr within the Fr specification by a vendor specific prefix `Fr_<Vendor Id>_<Vendor specific name>` during implementation to allow the usage of different FlexRay Drivers within one build system. The Fr module's implementer shall apply this rule to all prefixes within filenames, Fr module specific datatypes, Fr module specific constants, Fr module specific global variables and API functions.⌋*(SRS_BSW_00347)*

[SWS_Fr_00097] ⌈The Fr module shall implement the API functions specified by the Fr SWS as real C-code functions and shall not implement the API functions as macros.⌋*(SRS_BSW_00342)*

[SWS_Fr_00479] ⌈The rationale of [SWS_Fr_00097] is to allow object code module integration.⌋*()*

[SWS_Fr_00102] ⌈None of the Fr module's header files shall define global variables.⌋*(SRS_BSW_00308)*

[SWS_Fr_00106] ⌈The Fr module or the underlying hardware or both shall stop FlexRay timers in case of loss of synchronization.⌋*(SRS_Fr_05055)*

The implementation may assume that

- The Fr module's environment shall call all LPdu-based services (`Fr_TransmitTxLPdu`(), `Fr_ReceiveRxLPdu`(), `Fr_CheckTxLPduStatus`(), `Fr_PrepareLPdu`()) synchronous to the FlexRay global time (at predefined determined points in time) in case of proper system operation.
- The Fr module's environment may call all non LPdu-based services at any time independent from the FlexRay global time.

7.3 Indexing Scheme

Users of the Fr identify Fr resources by using an indexing scheme as depicted in Figure 7.1.

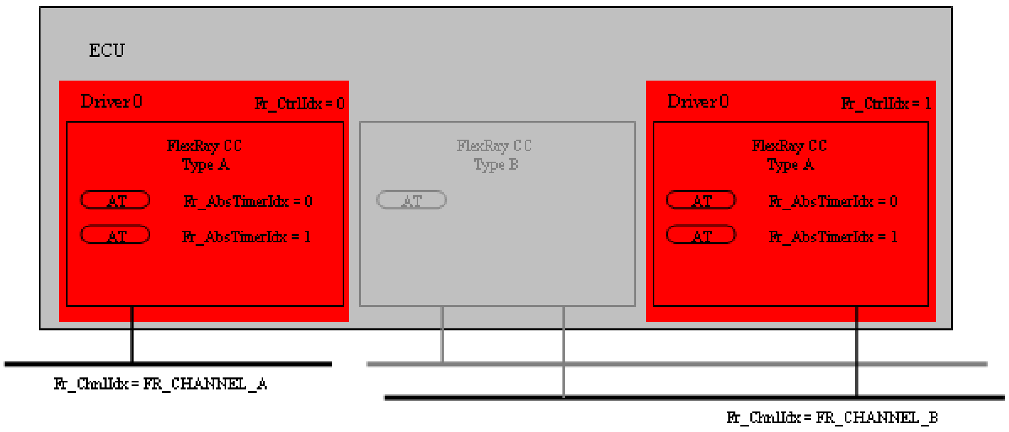

Figure 7.1: FlexRay Driver indexing scheme

The following Fr resources are available:

[SWS_Fr_00075] ⌈CCs are identified via controller indices (`Fr_CtrlIdx`).⌋*(SRS_BSW_00413)*

[SWS_Fr_00467] ⌈Each driver's CCs are identified by controller indices 0 to (n-1) where n is the number of CCs controlled by the particular Fr.⌋*(SRS_Fr_05065)*

[SWS_Fr_00344] ⌈For each FlexRay CC the connected channels are identified by channel indices (`Fr_ChnlIdx`).⌋*()*

[SWS_Fr_00468] ⌈A dedicated type that holds the enumerations `FR_CHANNEL_A`, `FR_CHANNEL_B` or `FR_CHANNEL_AB` represents the channel index.⌋*()*

[SWS_Fr_00469] ⌈Channel indices are only valid within a tuple <`Fr_CtrlIdx`, `Fr_ChnlIdx`>.⌋*()*

[SWS_Fr_00005] ⌈Each FlexRay frame processed by Fr API functions is identified by an LPdu index (`Fr_LPduIdx`).⌋*(SRS_Fr_05003, SRS_Fr_05024)*

[SWS_Fr_00470] ⌈Each LPdu carries the LSdu. Each controller's LPdus are identified by LPdu indices from 0 to (n-1) where n is the number of LPdus processed by the corresponding CC.⌋*()*

[SWS_Fr_00471] ⌈LPdu indices are only valid within a tuple <`Fr_CtrlIdx`, `Fr_LPduIdx`>.⌋*()*

[SWS_Fr_00472] ⌈An `Fr_LPduIdx` uniquely identifies the following parameters of a FlexRay frame as a key: {Slot ID, Channel, cycle repetition, base cycle, transmit/receive}.⌋*()*

[SWS_Fr_00345] ⌈Each FlexRay CC contains absolute timers. Absolute FlexRay timers are identified via absolute timer indices (`Fr_AbsTimerIdx`).⌋*()*

[SWS_Fr_00473] ⌈Each CC's absolute timers are identified by absolute timer indices from 0 to (n-1), where n is the number of absolute timers controlled by the particular CC.⌋*()*

[SWS_Fr_00474] ⌈Absolute timer indices are only valid within a tuple <Fr_CtrlIdx, Fr_AbsTimerIdx>.⌋*()*

The FlexRay Driver numbering scheme (Figure 7.1) assigns indices to these items on a per-driver basis. Note that only the FlexRay CCs handled by one specific Fr module (i.e., the FlexRay CCs of type A in the example given) are being assigned indices within the context of this Fr module. All other CCs (e.g., the FlexRay CC of type B) are not handled by this Fr module and thus no indices have been assigned to these FlexRay CCs within the context of this Fr module.

7.4 POC state machine control

[SWS_Fr_00477] ⌈Since a FlexRay CC is condition-based, it internally maintains a state machine, the Protocol Operation Control (POC) state machine. The state transitions are driven both by hardware related events as well as by commands passed by the host at the CHI (see [1] for details).⌋*()*

[SWS_Fr_00478] ⌈The CHI commands driving the POC state machine are incorporated into several Fr module API functions. API functions affecting the POC state of a FlexRay CC are:

- `Fr_StartCommunication`()
- `Fr_HaltCommunication`()
- `Fr_AbortCommunication`()
- `Fr_SendWUP`()
- `Fr_ControllerInit`()

⌋*()*

[SWS_Fr_00438] ⌈All API functions other than those listed above shall not change the POC state of the FlexRay CC. Figure 3 shows the POC states of the FlexRay CC and the transitions applicable to the Fr module API functions. Note that

- certain transitions (marked with *)) are performed by the invocation of a single API function call (`Fr_ControllerInit`()).
- certain transitions might be implicitly performed by the FlexRay CC without external command invocation (dotted arrow)

- certain transitions specified cannot be performed by the current Fr module API (not drawn in Figure 3 - compare to [5]).

⌋*()*

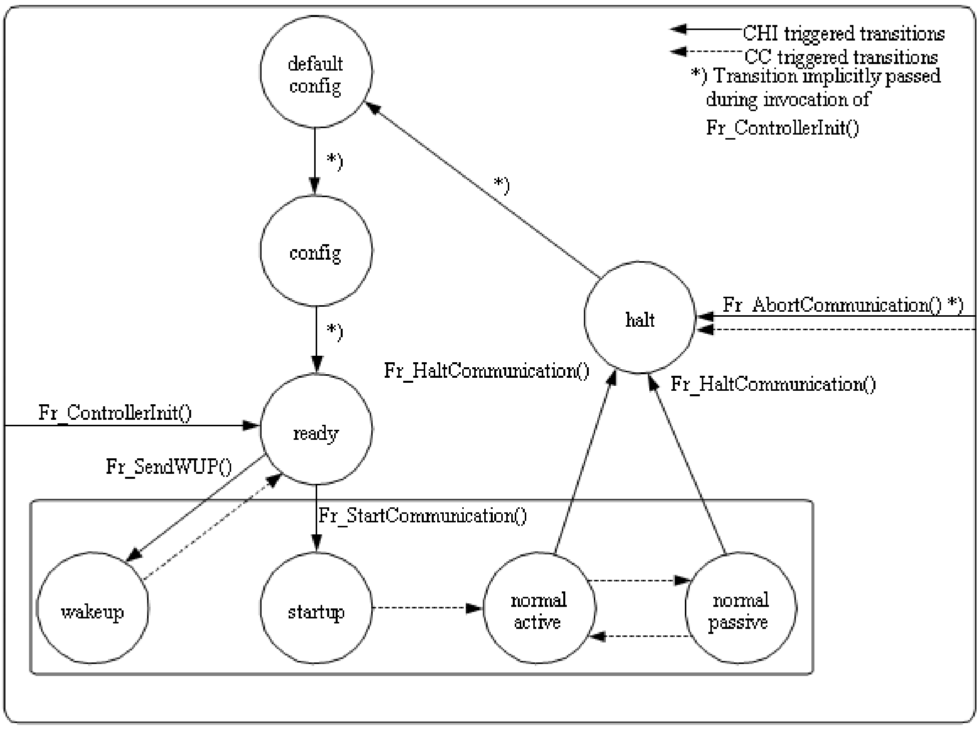

Figure 7.2: FlexRay Driver POC State Machine Control

7.5 FIFO support and message ID filtering

To efficiently support reception in certain use-cases, FlexRay controllers might support receive-FIFOs. The receive-FIFOs accept FlexRay frames based on a set of configured filter criterias which match FlexRay specific properties such as frameID, cycle, channel, as well as protocol add-ons like the message ID, in hardware.

[SWS_Fr_00593] ⌈The hardware receive-FIFO shall be used if the FIFO filter-criterias as configured can be applied to the hardware FIFO.⌋*(SRS_Fr_05005, SRS_Fr_05006)*

[SWS_Fr_00594] ⌈All LPdus (as configured within FrIf) matching a receive-FIFO's filter-criteria shall be assigned to the respective receive-FIFO.⌋*(SRS_Fr_05005)*

[SWS_Fr_00595] ⌈No specific buffers shall be assigned to LPdus that are assigned to a receive-FIFO.⌋*(SRS_Fr_05005)*

[SWS_Fr_00596] ⌈If `Fr_ReceiveRxLPdu`() is called for an LPdu assigned to the receive FIFO, the service `Fr_ReceiveRxPdu`() consumes the first valid frame out of the respective FIFO and returns it as received frame. There is no receive-FIFO specific API, thus keeping the upper layers unaffected.⌋*(SRS_Fr_05005)*

Hint: This restricted implementation of the receive-FIFO covers a very typical use-case (FrTp):

- All received (L)Pdus assigned to the FIFO shall be processed by a single upper layer module.
- The upper layer does not care about the specific assignment of (L)Pdus to FlexRay FrameTriggerings.

[SWS_Fr_00597] ⌈LPdus received via the FIFO shall be returned in the same order as they were received on the FlexRay network.⌋*(SRS_Fr_05005)*

7.6 Configuration description

[SWS_Fr_00080] ⌈The Fr module shall provide an XML file that contains the data, which is required for the SW identification and configuration parameters. This file shall describe vendor specific configuration parameters if applicable.⌋*(SRS_BSW_00003, SRS_BSW_00334, SRS_BSW_00374, SRS_BSW_00379)*

[SWS_Fr_00480] ⌈A driver MCG reads the ECU configuration parameters of the Fr and the FrIf modules. While cluster related configuration parameters are contained in the FrIf module's configuration, CC related configuration parameters are contained in the Fr module's configuration. The Fr module's specific configuration tool shall read both ECU module configurations to derive the configuration parameters for all FlexRay CCs mapped to the Fr module.⌋*(SRS_Fr_05058)*

[SWS_Fr_00481] ⌈All frame transmission/reception related configuration parameters are located only in the FrIf module description (within configuration containers '`FrIfLPdu`' and '`FrIfFrameTriggering`'). The Fr must be able to handle all transmission/reception requests of all related LPdus. The LPdus within the FrIf configuration contain both an LPduIdx which is passed to the Fr API as well as a link to a frame triggering that holds the link of the LPdu to the FlexRay network (assignment to Slot, channel, cycle).

The CC configuration parameters related to frame transmission and reception shall be derived from the communication matrix the CC is mapped to within the FrIf.⌋*()*

[SWS_Fr_00482] ⌈For optimization purposes the Fr MCG shall read the FrIf job list for detecting the points in time certain actions on the Fr will be synchronously invoked by the FrIf (see [6] for the FrIf configuration parameters).⌋*()*

[SWS_Fr_00483] ⌈Based on those invocation times the Fr MCG might decide certain resource alignment optimizations for transmission and reception (share buffers among different LPdus).⌋*()*

[SWS_Fr_00003] ⌈If the FrIf job list contains dedicated buffer reconfiguration entries that allow for optimization, then the Fr module's MCG may decide to share one buffer for several LPdus within the static segment.⌋*()*

[SWS_Fr_00624] ⌈If an LPdu is dynamically reconfigurable ('`FrIfReconfigurable`' set to true) the MCG shall decide to assign a single exclusive hardware message buffer to those LPdus.⌋*()*

[SWS_Fr_00484] ⌈The Fr MCG shall have knowledge about the capabilities of the CC and the corresponding driver, therefore this tool is called driver dependent.⌋*()*

[SWS_Fr_00485] ⌈If an Fr MCG is unable to map all required communication operations to the available resources, then it has to report that conflict[1].⌋*()*

[SWS_Fr_00486] ⌈The number of supported FlexRay CCs is defined at configuration time.⌋*()*

[SWS_Fr_00487] ⌈The MCG shall ensure the consistency of the generated configuration.⌋*()*

[SWS_Fr_00027] ⌈The Fr module shall support the pre-compile-time and post-build-time configuration classes.⌋*(SRS_BSW_00345, SRS_BSW_00404)*

An assignment of those configuration classes to configuration parameters can be found in chapter 10.

Hint: The description of the software configuration itself is not part of this specification but very implementation specific.

A detailed description of all Fr related configuration parameters is specified in chapter 10 of this document. Additionally the configuration parameters of the FrIf (see chapter 10 of [6]) shall be evaluated for Fr module configuration.

7.7 Error Classification

Section "Error Handling" of the document [4] "General Specification of Basic Software Modules" describes the error handling of the Basic Software in detail. Above all, it constitutes a classification scheme consisting of five error types which may occur in BSW modules.

Based on this foundation, the following section specifies particular errors arranged in the respective subsections below.

[1]This can result from either from running out of resources (e.g. buffers) or the mapping of the configuration to the particular device is not supported (e.g. configuration features supported in [1], but the device is compliant to [2]).

7.7.1 Development Errors

[SWS_Fr_91003] Definiton of development errors in module Fr ⌈

Type of error	*Related error code*	*Error value*
parameter timer index exceeds number of available timers	FR_E_INV_TIMER_IDX	0x01
invalid pointer in parameter list	FR_E_PARAM_POINTER	0x02
parameter offset exceeds bounds	FR_E_INV_OFFSET	0x03
invalid controller index	FR_E_INV_CTRL_IDX	0x04
invalid channel index	FR_E_INV_CHNL_IDX	0x05
parameter cycle exceeds 63	FR_E_INV_CYCLE	0x06
Fr module was not initialized	FR_E_INIT_FAILED	0x08
Payload length parameter has an invalid value	FR_E_INV_LENGTH	0x0A
invalid LPdu index	FR_E_INV_LPDU_IDX	0x0B
invalid FlexRay header CRC	FR_E_INV_HEADERCRC	0x0C
invalid value passed as parameter Fr_Config ParamIdx	FR_E_INV_CONFIG_IDX	0x0D
invalid framelist size value	FR_E_INV_FRAMELIST_SIZE	0x0E

⌋*()*

7.7.2 Runtime Errors

[SWS_Fr_91004] Definiton of runtime errors in module Fr ⌈

Type of error	*Related error code*	*Error value*
Fr CC is not in the expected POC state	FR_E_INV_POCSTATE	0x09

⌋*()*

7.7.3 Transient Faults

There are no transient faults.

7.7.4 Production Errors

There are no production errors.

7.7.5 Extended Production Errors

[SWS_Fr_00498] ⌈

Error Name:	FR_E_CTRL_TESTRESULT [_<Fr_CtrlIdx>]	
Short Description:	FlexRay Controller hardware error	
Long Description:	This extended production error indicates hardware errors of the FlexRay communication controller. Please note that this extended production error does not address Flexray protocol errors detected on the network.	
Detection Criteria:	Fail	Every API function accessing hardware registers of the FlexRay controller might detect a missbehavior of the device compared to the device specification. For details see requirements: [SWS_Fr_00147], [SWS_Fr_00176], [SWS_Fr_00181], [SWS_Fr_00520], [SWS_Fr_00186], [SWS_Fr_00190], [SWS_Fr_00195], [SWS_Fr_00201], [SWS_Fr_00216], [SWS_Fr_00223], [SWS_Fr_00613], [SWS_Fr_00232], [SWS_Fr_00243], [SWS_Fr_00248], [SWS_Fr_00529], [SWS_Fr_00543], [SWS_Fr_00255], [SWS_Fr_00261], [SWS_Fr_00552], [SWS_Fr_00560], [SWS_Fr_00568], [SWS_Fr_00580], [SWS_Fr_00589], [SWS_Fr_00272], [SWS_Fr_00286], [SWS_Fr_00297], [SWS_Fr_00308], [SWS_Fr_00319], [SWS_Fr_00331], [SWS_Fr_00652]
	Pass	During FlexRay communication controller initialization (function Fr_ControllerInit()) the proper operation of the hardware registers is successfully verified. For details see requirements: [SWS_Fr_00598]
Secondary Parameters:	In case of successful device operation verification (function Fr_ControllerInit()) report PASS. In case of an error always report FAIL.	
Time Required:	If a FlexRay Controller hardware error occurs it shall be immediately reported as error.	
Monitor Frequency:	continuous	

⌋*()*

[SWS_Fr_00600] ⌈

Error Name:	FR_E_LPDU_SLOTSTATUS [_<LPduIdx>]
Short Description:	FlexRay Protocol communication error
Long Description:	This production error indicates Flexray protocol communication errors on the network for a particular LPdu.

▽

△

Detection Criteria:	Fail	Each time FlexRay slot status with at least one of the following FlexRay protocol errors active: • vSS!SyntaxError • vSS!ContentError • vSS!Bviolation For details see requirements: [SWS_Fr_00605], [SWS_Fr_00606]
	Pass	Each time FlexRay slot status with none one of the following FlexRay protocol errors active: • vSS!SyntaxError • vSS!ContentError • vSS!Bviolation For details see requirements: [SWS_Fr_00627], [SWS_Fr_00629]
Secondary Parameters:	Detection of production error events is active only during ongoing FlexRay communication.	
Time Required:	The time for detecting an evident FlexRay protocol error in a particular slot strongly depends on the period of the actual FlexRay slot and thus on the FlexRay protocol configuration parameters.	
Monitor Frequency:	continuous	

⌋*()*

7.8 Security Events

The module does not report security events.

8 API specification

[SWS_Fr_00098] ⌈All API functions or global variables, whether they are specified or not shall follow the naming scheme `Fr_<name>`, where the first letter of each word in <name> is written uppercase and the remainder of the word lowercase.⌋*(SRS_BSW_00307)*

8.1 Imported types

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

[SWS_Fr_00099] Definition of imported datatypes of module Fr ⌈

Module	***Header File***	***Imported Type***
Dem	Rte_Dem_Type.h	Dem_EventIdType
	Rte_Dem_Type.h	Dem_EventStatusType
Std	Std_Types.h	Std_ReturnType
	Std_Types.h	Std_VersionInfoType

⌋*(SRS_BSW_00348, SRS_BSW_00353)*

8.2 Macro definitions

8.2.1 Configuration parameter index macros

The following table lists macros which list symbolic names that can be passed into API function `Fr_ReadCCConfig`() as parameter `Fr_ConfigParamIdx` (see subsection 8.4.32).

Each macro (index) uniquely identifies a configuration parameter which value can be read out of the controller's configuration using `Fr_ReadCCConfig`().

Macro name:	**Value:**	**Mapps to configuration parameter:**
FR_CIDX_GDCYCLE	0	FrIfGdCycle
FR_CIDX_PMICROPERCYCLE	1	FrPMicroPerCycle
FR_CIDX_PDLISTENTIMEOUT	2	FrPdListenTimeout
FR_CIDX_GMACROPERCYCLE	3	FrIfGMacroPerCycle
FR_CIDX_GDMACROTICK	4	FrIfGdMacrotick
FR_CIDX_GNUMBEROFMINISLOTS	5	FrIfGNumberOfMinislots
FR_CIDX_GNUMBEROFSTATICSLOTS	6	FrIfGNumberOfStaticSlots
FR_CIDX_GDNIT	7	FrIfGdNit
FR_CIDX_GDSTATICSLOT	8	FrIfGdStaticSlot
FR_CIDX_GDWAKEUPRXWINDOW	9	FrIfGdWakeupRxWindow

▽

△

Macro name:	Value:	Mapps to configuration parameter:
FR_CIDX_PKEYSLOTID	10	FrPKeySlotId
FR_CIDX_PLATESTTX	11	FrPLatestTx
FR_CIDX_POFFSETCORRECTIONOUT	12	FrPOffsetCorrectionOut
FR_CIDX_POFFSETCORRECTIONSTART	13	FrPOffsetCorrectionStart
FR_CIDX_PRATECORRECTIONOUT	14	FrPRateCorrectionOut
FR_CIDX_PSECONDKEYSLOTID	15	FrPSecondKeySlotId
FR_CIDX_PDACCEPTEDSTARTUPRANGE	16	FrPdAcceptedStartupRange
FR_CIDX_GCOLDSTARTATTEMPTS	17	FrIfGColdStartAttempts
FR_CIDX_GCYCLECOUNTMAX	18	FrIfGCycleCountMax
FR_CIDX_GLISTENNOISE	19	FrIfGListenNoise
FR_CIDX_GMAXWITHOUTCLOCKCORRECTFATAL	20	FrIfGMaxWithoutClockCorrectFatal
FR_CIDX_GMAXWITHOUTCLOCKCORRECTPASSIVE	21	FrIfGMaxWithoutClockCorrectPassive
FR_CIDX_GNETWORKMANAGEMENTVECTORLENGTH	22	FrIfGNetworkManagementVector-Length
FR_CIDX_GPAYLOADLENGTHSTATIC	23	FrIfGPayloadLengthStatic
FR_CIDX_GSYNCFRAMEIDCOUNTMAX	24	FrIfGSyncFrameIDCountMax
FR_CIDX_GDACTIONPOINTOFFSET	25	FrIfGdActionPointOffset
FR_CIDX_GDBIT	26	FrIfGdBit
FR_CIDX_GDCASRXLOWMAX	27	FrIfGdCasRxLowMax
FR_CIDX_GDDYNAMICSLOTIDLEPHASE	28	FrIfGdDynamicSlotIdlePhase
FR_CIDX_GDMINISLOTACTIONPOINTOFFSET	29	FrIfGdMiniSlotActionPointOffset
FR_CIDX_GDMINISLOT	30	FrIfGdMinislot
FR_CIDX_GDSAMPLECLOCKPERIOD	31	FrIfGdSampleClockPeriod
FR_CIDX_GDSYMBOLWINDOW	32	FrIfGdSymbolWindow
FR_CIDX_GDSYMBOLWINDOWACTIONPOINTOFFSET	33	FrIfGdSymbolWindowActionPointOffset
FR_CIDX_GDTSSTRANSMITTER	34	FrIfGdTssTransmitter
FR_CIDX_GDWAKEUPRXIDLE	35	FrIfGdWakeupRxIdle
FR_CIDX_GDWAKEUPRXLOW	36	FrIfGdWakeupRxLow
FR_CIDX_GDWAKEUPTXACTIVE	37	FrIfGdWakeupTxActive
FR_CIDX_GDWAKEUPTXIDLE	38	FrIfGdWakeupTxIdle
FR_CIDX_PALLOWPASSIVETOACTIVE	39	FrPAllowPassiveToActive
FR_CIDX_PCHANNELS	40	FrPChannels
FR_CIDX_PCLUSTERDRIFTDAMPING	41	FrPClusterDriftDamping
FR_CIDX_PDECODINGCORRECTION	42	FrPDecodingCorrection
FR_CIDX_PDELAYCOMPENSATIONA	43	FrPDelayCompensationA
FR_CIDX_PDELAYCOMPENSATIONB	44	FrPDelayCompensationB
FR_CIDX_PMACROINITIALOFFSETA	45	FrPMacroInitialOffsetA
FR_CIDX_PMACROINITIALOFFSETB	46	FrPMacroInitialOffsetB
FR_CIDX_PMICROINITIALOFFSETA	47	FrPMicroInitialOffsetA
FR_CIDX_PMICROINITIALOFFSETB	48	FrPMicroInitialOffsetB
FR_CIDX_PPAYLOADLENGTHDYNMAX	49	FrPPayloadLengthDynMax
FR_CIDX_PSAMPLESPERMICROTICK	50	FrPSamplesPerMicrotick
FR_CIDX_PWAKEUPCHANNEL	51	FrPWakeupChannel
FR_CIDX_PWAKEUPPATTERN	52	FrPWakeupPattern
FR_CIDX_PDMICROTICK	53	FrPdMicrotick
FR_CIDX_GDIGNOREAFTERTX	54	FrIfGdIgnoreAfterTx

▽

△

Macro name:	Value:	Mapps to configuration parameter:
FR_CIDX_PALLOWHALTDUETOCLOCK	55	FrPAllowHaltDueToClock
FR_CIDX_PEXTERNALSYNC	56	FrPExternalSync
FR_CIDX_PFALLBACKINTERNAL	57	FrPFallBackInternal
FR_CIDX_PKEYSLOTONLYENABLED	58	FrPKeySlotOnlyEnabled
FR_CIDX_PKEYSLOTUSEDFORSTARTUP	59	FrPKeySlotUsedForStartup
FR_CIDX_PKEYSLOTUSEDFORSYNC	60	FrPKeySlotUsedForSync
FR_CIDX_PNMVECTOREARLYUPDATE	61	FrPNmVectorEarlyUpdate
FR_CIDX_PTWOKEYSLOTMODE	62	FrPTwoKeySlotMode

8.3 Type definitions

[SWS_Fr_00499] ⌈The content of `Fr_GeneralTypes.h` shall be protected by a `FR_GENERAL_TYPES` define.⌋*()*

[SWS_Fr_00500] ⌈If different FlexRay drivers are used, only one instance of this file has to be included in the source tree. For implementation all `Fr_GeneralTypes.h` related types in the documents mentioned before shall be considered.⌋*()*

[SWS_Fr_00077] ⌈All types whether they are specified or implementation dependant shall follow the naming scheme `Fr_<name>Type`, where the first letter of each word in <name> is written uppercase and the remainder of the word is written lowercase.⌋*(SRS_BSW_00305)*

8.3.1 Fr_ConfigType

[SWS_Fr_91001] Definition of datatype Fr_ConfigType ⌈

Name	Fr_ConfigType
Kind	Structure
Description	This type contains the implementation-specific post build configuration structure.
Available via	Fr.h

⌋*()*

[SWS_Fr_00648] ⌈Rules of [SWS_Fr_00076] shall be applied to Fr_ConfigType.⌋*()*

8.3.2 Fr_POCStateType

[SWS_Fr_00505] Definition of datatype Fr_POCStateType ⌈

Name	Fr_POCStateType		
Kind	Enumeration		
Range	FR_POCSTATE_CONFIG	0x00	Represents literal CONFIG of formal type definition T_POCState.
	FR_POCSTATE_DEFAULT_CONFIG	0x01	Represents literal DEFAULT_CONFIG of formal type definition T_POCState.
	FR_POCSTATE_HALT	0x02	Represents literal HALT of formal type definition T_POCState.
	FR_POCSTATE_NORMAL_ACTIVE	0x03	Represents literal NORMAL_ACTIVE of formal type definition T_POCState.
	FR_POCSTATE_NORMAL_PASSIVE	0x04	Represents literal NORMAL_PASSIVE of formal type definition T_POCState.
	FR_POCSTATE_READY	0x05	Represents literal READY of formal type definition T_POCState.
	FR_POCSTATE_STARTUP	0x06	Represents literal STARTUP of formal type definition T_POCState.
	FR_POCSTATE_WAKEUP	0x07	Represents literal WAKEUP of formal type definition T_POCState.
Description	This formal definition refers to the description of type T_POCState in chapter 2.2.1.3 POC status of [12].		
Available via	Fr_GeneralTypes.h		

⌋*(SRS_BSW_00441)*

8.3.3 Fr_SlotModeType

[SWS_Fr_00506] Definition of datatype Fr_SlotModeType ⌈

Name	Fr_SlotModeType		
Kind	Enumeration		
Range	FR_SLOTMODE_KEYSLOT	0x00	Represents literal KEYSLOT of formal type definition T_SlotMode.
	FR_SLOTMODE_ALL_PENDING	0x01	Represents literal ALL_PENDING of formal type definition T_SlotMode.
	FR_SLOTMODE_ALL	0x02	Represents literal ALL of formal type definition T_SlotMode.
Description	This formal definition refers to the description of type T_SlotMode in chapter 2.2.1.3 POC status of [12].		
Available via	Fr_GeneralTypes.h		

⌋*(SRS_BSW_00441)* [1]

[1]For FlexRay 2.1 Rev A compliant FlexRay controllers see literal SINGLESLOT instead of KEYSLOT in [2].

8.3.4 Fr_ErrorModeType

[SWS_Fr_00507] Definition of datatype Fr_ErrorModeType ⌈

Name	Fr_ErrorModeType		
Kind	Enumeration		
Range	FR_ERRORMODE_ACTIVE	0x00	Represents literal ACTIVE of formal type definition T_ErrorMode.
	FR_ERRORMODE_ PASSIVE	0x01	Represents literal PASSIVE of formal type definition T_ErrorMode.
	FR_ERRORMODE_ COMM_HALT	0x02	Represents literal COMM_HALT of formal type definition T_ErrorMode.
Description	This formal definition refers to the description of type T_ErrorMode in chapter 2.2.1.3 POC status of [12].		
Available via	Fr_GeneralTypes.h		

⌋*(SRS_BSW_00441)*

8.3.5 Fr_WakeupStatusType

[SWS_Fr_00508] Definition of datatype Fr_WakeupStatusType ⌈

Name	Fr_WakeupStatusType		
Kind	Enumeration		
Range	FR_WAKEUP_UNDEFINED	0x00	Represents literal UNDEFINED of formal type definition T_WakeupStatus.
	FR_WAKEUP_RECEIVED_ HEADER	0x01	Represents literal RECEIVED_HEADER of formal type definition T_WakeupStatus.
	FR_WAKEUP_RECEIVED_ WUP	0x02	Represents literal RECEIVED_WUP of formal type definition T_WakeupStatus.
	FR_WAKEUP_COLLISION_ HEADER	0x03	Represents literal COLLISION_HEADER of formal type definition T_WakeupStatus.
	FR_WAKEUP_COLLISION_ WUP	0x04	Represents literal COLLISION_WUP of formal type definition T_WakeupStatus.
	FR_WAKEUP_COLLISION_ UNKNOWN	0x05	Represents literal COLLISION_UNKNOWN of formal type definition T_WakeupStatus.
	FR_WAKEUP_ TRANSMITTED	0x06	Represents literal TRANSMITTED of formal type definition T_WakeupStatus.
Description	This formal definition refers to the description of type T_WakeupStatus in chapter 2.2.1.3 POC status of [12].		
Available via	Fr_GeneralTypes.h		

⌋*(SRS_BSW_00441)*

8.3.6 Fr_StartupStateType

[SWS_Fr_00509] Definition of datatype Fr_StartupStateType ⌈

Name	Fr_StartupStateType		
Kind	Enumeration		
Range	FR_STARTUP_UNDEFINED	0x00	Represents literal UNDEFINED of formal type definition T_StartupState.
	FR_STARTUP_COLDSTART_LISTEN	0x01	Represents literal COLDSTART_LISTEN of formal type definition T_StartupState.
	FR_STARTUP_INTEGRATION_COLDSTART_CHECK	0x02	Represents literal INTEGRATION_COLDSTART_CHECK of formal type definition T_StartupState.
	FR_STARTUP_COLDSTART_JOIN	0x03	Represents literal COLDSTART_JOIN of formal type definition T_StartupState.
	FR_STARTUP_COLDSTART_COLLISION_RESOLUTION	0x04	Represents literal COLDSTART_COLLISION_RESOLUTION of formal type definition T_StartupState.
	FR_STARTUP_COLDSTART_CONSISTENCY_CHECK	0x05	Represents literal COLDSTART_CONSISTENCY_CHECK of formal type definition T_StartupState.
	FR_STARTUP_INTEGRATION_LISTEN	0x06	Represents literal INTEGRATION_LISTEN of formal type definition T_StartupState.
	FR_STARTUP_INITIALIZE_SCHEDULE	0x07	Represents literal INITIALIZE_SCHEDULE of formal type definition T_StartupState.
	FR_STARTUP_INTEGRATION_CONSISTENCY_CHECK	0x08	Represents literal INTEGRATION_CONSISTENCY_CHECK of formal type definition T_StartupState.
	FR_STARTUP_COLDSTART_GAP	0x09	Represents literal COLDSTART_GAP of formal type definition T_StartupState.
	FR_STARTUP_EXTERNAL_STARTUP	0x0a	Represents literal EXTERNAL_STARTUP of formal type definition T_StartupState.
Description	This formal definition refers to the description of type T_StartupState in chapter 2.2.1.3 POC status of [12].		
Available via	Fr_GeneralTypes.h		

⌋ *(SRS_BSW_00441)*

Note: `Fr_StartupStateType` contains the superset of FlexRay 2.1 and FlexRay 3.0 specification. Thus state `FR_STARTUP_EXTERNAL_STARTUP` cannot be reached on FlexRay 2.1 compliant FlexRay controllers.

8.3.7 Fr_POCStatusType

[SWS_Fr_00510] Definition of datatype Fr_POCStatusType ⌈

Name	Fr_POCStatusType	
Kind	Structure	
Elements	CHIHaltRequest	
	Type	boolean

▽

△

	Comment	–
	ColdstartNoise	
	Type	boolean
	Comment	–
	ErrorMode	
	Type	Fr_ErrorModeType
	Comment	–
	Freeze	
	Type	boolean
	Comment	–
	SlotMode	
	Type	Fr_SlotModeType
	Comment	–
	StartupState	
	Type	Fr_StartupStateType
	Comment	–
	State	
	Type	Fr_POCStateType
	Comment	–
	WakeupStatus	
	Type	Fr_WakeupStatusType
	Comment	–
	CHIReadyRequest	
	Type	boolean
	Comment	–
Description	This formal definition refers to the description of type T_POCStatus in chapter 2.2.1.3 POC status of [12].	
Available via	Fr_GeneralTypes.h	

⌋*()*

8.3.8 Fr_TxLPduStatusType

[SWS_Fr_00511] Definition of datatype Fr_TxLPduStatusType ⌈

Name	Fr_TxLPduStatusType		
Kind	Enumeration		
Range	FR_TRANSMITTED	0x00	LPdu has been transmitted
	FR_TRANSMITTED_ CONFLICT	0x01	A transmission conflict has occurred.
	FR_NOT_TRANSMITTED	0x02	LPdu has not been transmitted
Description	These values are used to determine whether a LPdu has been transmitted or not.		
Available via	Fr_GeneralTypes.h		

⌋*(SRS_BSW_00441)*

8.3.9 Fr_RxLPduStatusType

[SWS_Fr_00512] Definition of datatype Fr_RxLPduStatusType ⌈

Name	Fr_RxLPduStatusType		
Kind	Enumeration		
Range	FR_RECEIVED	0x00	LPdu has been received
	FR_NOT_RECEIVED	0x01	LPdu has not been received
	FR_RECEIVED_MORE_ DATA_AVAILABLE	0x02	LPdu has been received. More instances of this LPdu are available (FIFO usage).
Description	These values are used to determine if a LPdu has been received or not.		
Available via	Fr_GeneralTypes.h		

⌋*(SRS_BSW_00441)*

8.3.10 Fr_ChannelType

[SWS_Fr_00514] Definition of ImplementationDataType Fr_ChannelType ⌈

Name	Fr_ChannelType		
Kind	Enumeration		
Range	FR_CHANNEL_A	0x01	Refers to channel A of a CC.
	FR_CHANNEL_B	0x02	Refers to channel B of a CC.
	FR_CHANNEL_AB	0x03	Refers to both channels (A and B) of a CC.
Description	The values are used to reference channels on a CC.		
Variation	–		
Available via	Fr_GeneralTypes.h		

⌋*(SRS_BSW_00441)*

8.3.11 Fr_SlotAssignmentType

[SWS_Fr_91002] Definition of datatype Fr_SlotAssignmentType ⌈

Name	Fr_SlotAssignmentType	
Kind	Structure	
Elements	Cycle	
	Type	uint8
	Comment	Cycle in which the frame is transmitted / received.
	SlotId	
	Type	uint16
	Comment	Slot ID of the frame.
	channelId	
	Type	Fr_ChannelType
	Comment	Channel of the frame.

▽

△

Description	This structure contains information about the assignment of a FlexRay frame to a cycle and a slot ID.
Available via	Fr_GeneralTypes.h

⌋*()*

8.4 Function definitions

During specification of the API functions the following guidelines were applied:

- The API functions of the Fr module shall have the return type `Std_ReturnType` or `void` (no return code).
- If an API function of the Fr module has the return type `Std_ReturnType`, and if the function performs its service successfully, then it shall return `E_OK` otherwise `E_NOT_OK`.
- If the Fr module's environment is passing input parameters by a reference, then the Fr SWS shall use the const qualifier (`const type *`) to guarantee that it doesn't change the input parameter.
- For output parameters, a memory address to store the parameter is passed as an argument.
- If API functions of the Fr module successfully finish (return `E_OK`), then all output parameters shall be written with with valid values.
- If API functions of the Fr module erroneously finish (return `E_NOT_OK`), then no output parameter shall be written. Output parameters shall keep their original values in this case.

8.4.1 Fr_Init

[SWS_Fr_00032] Definition of API function Fr_Init ⌈

Service Name	Fr_Init	

Syntax	`void Fr_Init (`
	`  const Fr_ConfigType* Fr_ConfigPtr`
	`)`	

Service ID [hex]	0x1c	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	Fr_ConfigPtr	Address to an Fr dependant configuration structure that contains all information for operating the Fr subsequently.
Parameters (inout)	None	
Parameters (out)	None	

▽

△

Return value	None
Description	Initializes the Fr.
Available via	Fr.h

⌋*(SRS_BSW_00358, SRS_BSW_00404, SRS_BSW_00405, SRS_BSW_00414)*

CC precondition for the function `Fr_Init`(): None.

[SWS_Fr_00137] ⌈The function `Fr_Init`() shall internally store the configuration address to enable subsequent API calls to access the configuration.⌋*(SRS_BSW_00438)*

8.4.2 Fr_ControllerInit

[SWS_Fr_00017] Definition of API function Fr_ControllerInit ⌈

Service Name	Fr_ControllerInit	
Syntax	`Std_ReturnType Fr_ControllerInit ( uint8 Fr_CtrlIdx )`	
Service ID [hex]	0x00	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant for the same device	
Parameters (in)	Fr_CtrlIdx	Index of FlexRay CC within the context of the FlexRay Driver.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: API call finished successfully. `E_NOT_OK`: API call aborted due to errors.
Description	Initialzes a FlexRay CC.	
Available via	Fr.h	

⌋*(SRS_Fr_05116, SRS_Fr_05011)*

[SWS_Fr_00148] ⌈The function `Fr_ControllerInit`() shall perform the following tasks on FlexRay CC `Fr_CtrIdx`:

1. Switch CC into 'POC:config' (from any other POCState).
2. Configure all FlexRay cluster and node configuration parameters (e.g., cycle length, macrotick duration).
3. Configure all transmit/receive resources (e.g., buffer initialization) according to the frame triggering configuration parameters contained in the FrIf.
4. Switch CC into 'POC:ready'
5. Return E_OK.

⌋*(SRS_Fr_05059, SRS_Fr_05012)*

CC post condition for the function `Fr_ControllerInit`(): CC `Fr_CtrlIdx` shall be left in POCState 'POC:ready'.

[SWS_Fr_00149] ⌈The function Fr_ControllerInit() shall ensure that no transmission requests are pending.⌋*()*

[SWS_Fr_00150] ⌈The function Fr_ControllerInit() shall ensure that no reception indications are pending.⌋*()*

[SWS_Fr_00151] ⌈The function Fr_ControllerInit() shall ensure that no interrupts are pending.⌋*()*

[SWS_Fr_00152] ⌈The function Fr_ControllerInit() shall ensure that all timers are disabled.⌋*(SRS_Fr_05169)*

[SWS_Fr_00153] ⌈The function Fr_ControllerInit() shall ensure that all interrupts are disabled.⌋*()*

[SWS_Fr_00515] ⌈The function Fr_ControllerInit() shall disable all LPdus which are dynamically reconfigurable (see Fr_ReconfigLPdu(), Fr_DisableLPdu()).⌋*()*

[SWS_Fr_00147] ⌈If the function Fr_ControllerInit() detects errors while testing the CC (CC test), then it shall repeat the test procedure a configurable number (FrCtrlTestCount) of times. If all tests fail, then it calls Dem_SetEventStatus() (FR_E_CTRL_TESTRESULT, DEM_EVENT_STATUS_FAILED) and returns E_NOT_OK.⌋*()*

[SWS_Fr_00647] ⌈The CC test as described in [SWS_Fr_00147] shall verify (read back and compare to reference values held in the configuration) that the node and cluster FlexRay parameters were written properly into the FlexRay CC.⌋*()*

[SWS_Fr_00598] ⌈If the function Fr_ControllerInit() passes the CC test within a number of configurable (FrCtrlTestCount) times, then it calls Dem_SetEventStatus() (FR_E_CTRL_TESTRESULT, DEM_EVENT_STATUS_PASSED).⌋*()*

[SWS_Fr_00143] ⌈If development error detection for the Fr module is enabled, and if the function Fr_ControllerInit() is called before the successful initialization of Fr, then the function Fr_ControllerInit() shall raise the development error FR_E_INIT_FAILED.⌋*()*

[SWS_Fr_00144] ⌈If development error detection for the Fr module is enabled, then the function Fr_ControllerInit() shall check the validity of the parameter Fr_CtrlIdx. If Fr_CtrlIdx is invalid, then the function Fr_ControllerInit() shall raise the development error FR_E_INV_CTRL_IDX.⌋*()*

8.4.3 Fr_StartCommunication

[SWS_Fr_00010] Definition of API function Fr_StartCommunication ⌈

Service Name	Fr_StartCommunication	
Syntax	`Std_ReturnType Fr_StartCommunication ( uint8 Fr_CtrlIdx )`	
Service ID [hex]	0x03	
Sync/Async	Asynchronous	
Reentrancy	Non Reentrant for the same device	
Parameters (in)	Fr_CtrlIdx	Index of FlexRay CC within the context of the FlexRay Driver.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	E_OK: API call finished successfully. E_NOT_OK: API call aborted due to errors.
Description	Starts communication.	
Available via	Fr.h	

⌋*(SRS_Fr_05109)*

Note: The Fr module's environment shall only call the function Fr_StartCommunication() when CC Fr_CtrlIdx is in POCState 'POC:ready'.

[SWS_Fr_00177] ⌈The function Fr_StartCommunication() shall perform the following tasks on FlexRay CC Fr_CtrIdx:

1. Invoke the CC CHI command 'RUN', which initiates the startup procedure within the FlexRay CC.
2. Return E_OK.

⌋*()*

The function call of Fr_StartCommunication() changes the CC POCState to POC:startup which is a transitional state. In the case when communication startup succeeds, the CC wil change the POCState to 'POC:normal active' or 'POC:normal passive'. It is not guaranteed that the FlexRay CC will reside in the 'POC:normal active' or 'POC:normal passive' state after a call of the function Fr_StartCommunication().

[SWS_Fr_00176] ⌈If the function Fr_StartCommunication() is able to and detects a hardware error while performing the requested functionality, then it shall call Dem_SetEventStatus() (FR_E_CTRL_TESTRESULT, DEM_EVENT_STATUS_FAILED) and return E_NOT_OK.⌋*()*

[SWS_Fr_00173] ⌈If development error detection for the Fr module is enabled, and if the function Fr_StartCommunication() is called before successful initialization of the Fr, then the function Fr_StartCommunication() shall raise the development error FR_E_INIT_FAILED.⌋*()*

[SWS_Fr_00174] ⌈If development error detection for the Fr module is enabled, and the function Fr_StartCommunication() shall check the validity of the parameter

Fr_CtrlIdx. If Fr_CtrlIdx is invalid, then the function Fr_StartCommunication() shall raise the development error FR_E_INV_CTRL_IDX.⌋()

[SWS_Fr_00175] ⌈The function Fr_StartCommunication() shall check whether the CC Fr_CtrlIdx's POCState is in POC:ready. If the POCState is not POC:ready, then the function Fr_StartCommunication() shall raise the runtime error FR_E_INV_POCSTATE.⌋()

8.4.4 Fr_AllowColdstart

[SWS_Fr_00114] Definition of API function Fr_AllowColdstart ⌈

Service Name	Fr_AllowColdstart	

Syntax	`Std_ReturnType Fr_AllowColdstart (`
	`  uint8 Fr_CtrlIdx`
	`)`	

Service ID [hex]	0x23	
Sync/Async	Asynchronous	
Reentrancy	Non Reentrant for the same device	
Parameters (in)	Fr_CtrlIdx	Index of FlexRay CC within the context of the FlexRay Driver.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	E_OK: API call finished successfully. E_NOT_OK: API call aborted due to errors.
Description	Invokes the CC CHI command 'ALLOW_COLDSTART'.	
Available via	Fr.h	

⌋()

Note: The Fr Module's environment shall only call the function Fr_AllowColdstart() when the CC Fr_CtrlIdx is in POCState 'POC:ready or POC:startup.

[SWS_Fr_00182] ⌈The function Fr_AllowColdstart() shall perform the following tasks on FlexRay CC Fr_CtrIdx:

1. Invoke the CC CHI command 'ALLOW_COLDSTART'.
2. Return E_OK.

⌋()

[SWS_Fr_00181] ⌈If the function Fr_AllowColdstart() is able to and detects a hardware error while performing the requested functionality, then it shall call Dem_SetEventStatus() (FR_E_CTRL_TESTRESULT, DEM_EVENT_STATUS_FAILED) and return E_NOT_OK.⌋()

[SWS_Fr_00178] ⌈If development error detection for the Fr module is enabled, and if the function Fr_AllowColdstart() is called before the successful initialization of Fr, then the function Fr_AllowColdstart() shall raise the development error FR_E_INIT_FAILED.⌋()

[SWS_Fr_00179] ⌈If development error detection for the Fr module is enabled, then the function Fr_AllowColdstart() shall check the validity of the parameter Fr_CtrlIdx. If Fr_CtrlIdx is invalid, then the function Fr_AllowColdstart() shall raise the development error FR_E_INV_CTRL_IDX.⌋*()*

[SWS_Fr_00180] ⌈The function Fr_AllowColdstart() shall check the CC Fr_CtrlIdx's POCState. If the POCState is POC:default config, POC:config, or POC:halt, then the function Fr_AllowColdstart() shall raise the runtime error FR_E_INV_POCSTATE.⌋*()*

8.4.5 Fr_AllSlots

[SWS_Fr_00516] Definition of API function Fr_AllSlots ⌈

Service Name	Fr_AllSlots	

Syntax	`Std_ReturnType Fr_AllSlots (`
	`  uint8 Fr_CtrlIdx`
	`)`	

Service ID [hex]	0x24	
Sync/Async	Asynchronous	
Reentrancy	Non Reentrant for the same device	
Parameters (in)	Fr_CtrlIdx	Index of FlexRay CC within the context of the FlexRay Driver.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	E_OK: API call finished successfully. E_NOT_OK: API call aborted due to errors.
Description	Invokes the CC CHI command 'ALL_SLOTS'.	
Available via	Fr.h	

⌋*()*

Note: The Fr module's environment shall only call the function Fr_AllSlots() when CC Fr_CtrlIdx is synchronous to the FlexRay global time.

[SWS_Fr_00518] ⌈The function Fr_AllSlots() shall perform the following tasks on FlexRay CC Fr_CtrIdx:

1. Invoke the CC CHI command 'ALL_SLOTS', which requests a switch from key slot only mode to all slots transmission mode at the beginning of the next communication cycle.
2. Return E_OK.

⌋*()*

Note: The function Fr_AllSlots() requests to switch from key slot only mode to all slots transmission mode at the beginning of the next communication cycle.

[SWS_Fr_00520] ⌈If the function Fr_AllSlots() is able to and detects a hardware error while performing the requested functionality, then it shall call Dem_SetEventStatus() (FR_E_CTRL_TESTRESULT, DEM_EVENT_STATUS_FAILED) and return E_NOT_OK.⌋*()*

[SWS_Fr_00521] ⌈If development error detection for the Fr module is enabled, and if the function Fr_AllSlots() is called before the successful initialization of Fr, then the function Fr_AllSlots() shall raise the development error FR_E_INIT_FAILED.⌋*()*

[SWS_Fr_00522] ⌈If development error detection for the Fr module is enabled, then the function Fr_AllSlots() shall check the validity of the parameter Fr_CtrlIdx. If Fr_CtrlIdx is invalid, then the function Fr_AllSlots() shall raise the development error FR_E_INV_CTRL_IDX.⌋*()*

[SWS_Fr_00523] ⌈The function Fr_AllSlots() shall check whether the CC Fr_CtrlIdx is synchronous to the FlexRay global time. If the CC Fr_CtrlIdx is not synchronous to the FlexRay global time, then the function Fr_AllSlots() shall raise the runtime error FR_E_INV_POCSTATE.⌋*()*

8.4.6 Fr_HaltCommunication

[SWS_Fr_00014] Definition of API function Fr_HaltCommunication ⌈

Service Name	Fr_HaltCommunication	

Syntax	`Std_ReturnType Fr_HaltCommunication (`
	`  uint8 Fr_CtrlIdx`
	`)`	

Service ID [hex]	0x04	
Sync/Async	Asynchronous	
Reentrancy	Non Reentrant for the same device	
Parameters (in)	Fr_CtrlIdx	Index of FlexRay CC within the context of the FlexRay Driver.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	E_OK: API call finished successfully. E_NOT_OK: API call aborted due to errors.
Description	Invokes the CC CHI command 'DEFERRED_HALT'.	
Available via	Fr.h	

⌋*(SRS_BSW_00336, SRS_Fr_05115)*

Note: The Fr module's environment shall only call the function Fr_HaltCommunication() when CC Fr_CtrlIdx is synchronous to the FlexRay global time.

[SWS_Fr_00187] ⌈The function Fr_HaltCommunication() shall perform the following tasks on FlexRay CC Fr_CtrIdx:

1. Invoke the CC CHI command 'DEFERRED_HALT'[2].
2. Return E_OK.

⌋*()*

[2]Invoke the command 'HALT' for FlexRay Controllers compliant to [2].

Note: The function Fr_HaltCommunication() requests the halt state which shall be reached by the end of the current FlexRay communication cycle but might not be reached immediately.

[SWS_Fr_00186] ⌈If the function Fr_HaltCommunication() is able to and detects a hardware error while performing the requested functionality, then it shall call Dem_SetEventStatus() (FR_E_CTRL_TESTRESULT, DEM_EVENT_STATUS_FAILED) and return E_NOT_OK.⌋*()*

[SWS_Fr_00183] ⌈If development error detection for the Fr module is enabled, and if the function Fr_HaltCommunication() is called before the successful initialization of Fr, then the function Fr_HaltCommunication() shall raise the development error FR_E_INIT_FAILED.⌋*()*

[SWS_Fr_00184] ⌈If development error detection for the Fr module is enabled, then the function Fr_HaltCommunication() shall check the validity of the parameter Fr_CtrlIdx. If Fr_CtrlIdx is invalid, then the function Fr_HaltCommunication() shall raise the development error FR_E_INV_CTRL_IDX.⌋*()*

[SWS_Fr_00185] ⌈The function Fr_HaltCommunication() shall check whether the CC Fr_CtrlIdx is synchronous to the FlexRay global time. If the CC Fr_CtrlIdx is not synchronous to the FlexRay global time, then the function Fr_HaltCommunication() shall raise the runtime error FR_E_INV_POCSTATE.⌋*()*

8.4.7 Fr_AbortCommunication

[SWS_Fr_00011] Definition of API function Fr_AbortCommunication ⌈

Service Name	Fr_AbortCommunication	

Syntax	`Std_ReturnType Fr_AbortCommunication (`
	`  uint8 Fr_CtrlIdx`
	`)`	

Service ID [hex]	0x05	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant for the same device	
Parameters (in)	Fr_CtrlIdx	Index of FlexRay CC within the context of the FlexRay Driver.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	E_OK: API call finished successfully. E_NOT_OK: API call aborted due to errors.
Description	Invokes the CC CHI command 'FREEZE'.	
Available via	Fr.h	

⌋*(SRS_Fr_05114)*

[SWS_Fr_00191] ⌈The function Fr_AbortCommunication() shall perform the following tasks on FlexRay CC Fr_CtrIdx:

- Invoke the CC CHI command 'FREEZE', which immediately aborts communication (if active) and changes to the POC:halt state from any previous POCState.

- Return E_OK.

⌋()

Note: The function Fr_AbortCommunication() leaves the CC Fr_CtrlIdx in POC-State POC:halt (vPOC!Freeze is set).

[SWS_Fr_00190] ⌈If the function Fr_AbortCommunication() is able to and detects a hardware error while performing the requested functionality, then it shall call Dem_SetEventStatus() (FR_E_CTRL_TESTRESULT, DEM_EVENT_STATUS_FAILED) and return E_NOT_OK.⌋*()*

[SWS_Fr_00188] ⌈If development error detection for the Fr module is enabled, and if the function Fr_AbortCommunication() is called before the successful initialization of Fr, then the function Fr_AbortCommunication() shall raise the development error FR_E_INIT_FAILED.⌋*()*

[SWS_Fr_00189] ⌈If development error detection for the Fr module is enabled, then the function Fr_AbortCommunication() shall check the validity of the parameter Fr_CtrlIdx. If Fr_CtrlIdx is invalid, then the function Fr_AbortCommunication() shall raise the development error FR_E_INV_CTRL_IDX.⌋*()*

8.4.8 Fr_SendWUP

[SWS_Fr_00009] Definition of API function Fr_SendWUP ⌈

Service Name	Fr_SendWUP	

Syntax	`Std_ReturnType Fr_SendWUP (`
	`  uint8 Fr_CtrlIdx`
	`)`	

Service ID [hex]	0x06	
Sync/Async	Asynchronous	
Reentrancy	Non Reentrant for the same device	
Parameters (in)	Fr_CtrlIdx	Index of FlexRay CC within the context of the FlexRay Driver.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	E_OK: API call finished successfully. E_NOT_OK: API call aborted due to errors.
Description	Invokes the CC CHI command 'WAKEUP'.	
Available via	Fr.h	

⌋*(SRS_Fr_05117)*

Note: The Fr module's environment shall only call Fr_SendWUP() when CC Fr_CtrlIdx is in POCState 'POC:ready'.

[SWS_Fr_00196] ⌈The function Fr_SendWUP() shall perform the following tasks on FlexRay CC Fr_CtrIdx:

1. Invoke the CC CHI command 'WAKEUP', which initiates the wakeup transmission procedure on the configured FlexRay channel.

2. Return E_OK.

⌋()

Note: The function Fr_SendWUP() changes the CC Fr_CtrlIdx POCState to POC:wakeup, which is a transitional state. After wakeup procedure completion, the CC will reach POC:ready again.

Note: Sending a wakeup pattern does not necessarily cause all cluster nodes to be awoken afterwards. The function Fr_SendWUP() just invokes the wakeup symbol transmission procedure on a certain FlexRay CC.

[SWS_Fr_00195] ⌈If the function Fr_SendWUP() is able to and detects a hardware error while performing the requested functionality, then it shall call Dem_SetEventStatus() (FR_E_CTRL_TESTRESULT, DEM_EVENT_STATUS_FAILED) and return E_NOT_OK.⌋()

[SWS_Fr_00192] ⌈If development error detection for the Fr module is enabled, and if the function Fr_SendWUP() is called before the successful initialization of Fr, then the function Fr_SendWUP() shall raise the development error FR_E_INIT_FAILED.⌋()

[SWS_Fr_00193] ⌈If development error detection for the Fr module is enabled, then the function Fr_SendWUP() shall check the validity of the parameter Fr_CtrlIdx. If Fr_CtrlIdx is invalid, then the function Fr_SendWUP() shall raise the development error FR_E_INV_CTRL_IDX.⌋()

[SWS_Fr_00194] ⌈The function Fr_SendWUP() shall check whether the CC Fr_CtrlIdx's POCState is POC:ready. If the POCState is not POC:ready, then the function Fr_SendWUP() shall raise the runtime error FR_E_INV_POCSTATE.⌋()

8.4.9 Fr_SetWakeupChannel

[SWS_Fr_00091] Definition of API function Fr_SetWakeupChannel ⌈

Service Name	Fr_SetWakeupChannel	

Syntax	`Std_ReturnType Fr_SetWakeupChannel (`
	`  uint8 Fr_CtrlIdx,`
	`  Fr_ChannelType Fr_ChnlIdx`
	`)`	

Service ID [hex]	0x07	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant for the same device	
Parameters (in)	Fr_CtrlIdx	Index of FlexRay CC within the context of the FlexRay Driver.
	Fr_ChnlIdx	Index of FlexRay channel within the context of the FlexRay CC Fr_CtrlIdx. Valid values are FR_CHANNEL_A and FR_ CHANNEL_B.
Parameters (inout)	None	
Parameters (out)	None	

▽

△

Return value	Std_ReturnType	E_OK: API call finished successfully. E_NOT_OK: API call aborted due to errors.
Description	Sets a wakeup channel.	
Available via	Fr.h	

⌋*(SRS_Fr_05117)*

[SWS_Fr_00202] ⌈The function Fr_SetWakeupChannel() shall perform the following tasks on FlexRay CC Fr_CtrlIdx:

1. Change the CC's POCState to POC:config by invoking the CHI command 'CONFIG'.
2. Configure the wakeup channel according to parameter Fr_ChnlIdx.
3. Change the CC's POCState to POC:ready again by invoking the CHI command 'CONFIG_COMPLETE'.
4. Return E_OK.

⌋*()*

[SWS_Fr_00201] ⌈If the function Fr_SetWakeupChannel() is able to and detects a hardware error while performing the requested functionality, then it shall call Dem_SetEventStatus() (FR_E_CTRL_TESTRESULT, DEM_EVENT_STATUS_FAILED) and return E_NOT_OK.⌋*()*

[SWS_Fr_00197] ⌈If development error detection for the Fr module is enabled, and if the function Fr_SetWakeupChannel() is called before the successful initialization of Fr, then the function Fr_SetWakeupChannel() shall raise the development error FR_E_INIT_FAILED.⌋*()*

[SWS_Fr_00198] ⌈If development error detection for the Fr module is enabled, then the function Fr_SetWakeupChannel() shall check the validity of the parameter Fr_CtrlIdx. If Fr_CtrlIdx is invalid, then the function Fr_SetWakeupChannel() shall raise the development error FR_E_INV_CTRL_IDX.⌋*()*

[SWS_Fr_00199] ⌈If development error detection for the Fr module is enabled, then the function Fr_SetWakeupChannel() shall check the validity of the parameter Fr_ChnlIdx. If Fr_ChnlIdx is invalid, then the function Fr_SetWakeupChannel() shall raise the development error FR_E_INV_CHNL_IDX.⌋*()*

[SWS_Fr_00200] ⌈The function Fr_SetWakeupChannel() shall check whether the CC Fr_CtrlIdx's POCState is POC:ready. If the POCState is not 'POC:ready', then the function Fr_SetWakeupChannel() shall raise the runtime error FR_E_INV_POCSTATE.⌋*()*

8.4.10 Fr_GetPOCStatus

[SWS_Fr_00012] Definition of API function Fr_GetPOCStatus ⌈

Service Name	Fr_GetPOCStatus	

Syntax	`Std_ReturnType Fr_GetPOCStatus (`
	`  uint8 Fr_CtrlIdx,`
	`  Fr_POCStatusType* Fr_POCStatusPtr`
	`)`	

Service ID [hex]	0x0a	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant for the same device	
Parameters (in)	Fr_CtrlIdx	Index of FlexRay CC within the context of the FlexRay Driver.
Parameters (inout)	None	
Parameters (out)	Fr_POCStatusPtr	Address the output value is stored to.
Return value	Std_ReturnType	E_OK: API call finished successfully. E_NOT_OK: API call aborted due to errors.
Description	Gets the POC status.	
Available via	Fr.h	

⌋*(SRS_Fr_05120)*

CC precondition for the function Fr_GetPOCStatus(): None.

[SWS_Fr_00217] ⌈The function Fr_GetPOCStatus() shall perform the following tasks on FlexRay CC Fr_CtrIdx:

1. Query the CC's actual POC status by reading the CHI variable 'vPOC' and write the result to parameter Fr_POCStatusPtr.
2. Return E_OK.

⌋*()*

[SWS_Fr_00216] ⌈If the function Fr_GetPOCStatus() is able to and detects a hardware error while performing the requested functionality, then it shall call Dem_SetEventStatus() (FR_E_CTRL_TESTRESULT, DEM_EVENT_STATUS_FAILED) and return E_NOT_OK.⌋*()*

[SWS_Fr_00213] ⌈If development error detection for the Fr module is enabled, and if the function Fr_GetPOCStatus() is called before the successful initialization of Fr, then the function Fr_GetPOCStatus() shall raise the development error FR_E_INIT_FAILED.⌋*()*

[SWS_Fr_00214] ⌈If development error detection for the Fr module is enabled, then the function Fr_GetPOCStatus() shall check the validity of the parameter Fr_CtrlIdx. If Fr_CtrlIdx is invalid, then the function Fr_GetPOCStatus() shall raise the development error FR_E_INV_CTRL_IDX.⌋*()*

[SWS_Fr_00215] ⌈If development error detection for the Fr module is enabled, then the function Fr_GetPOCStatus() shall check whether the parameter Fr_POCStatusPtr

is a NULL pointer (NULL_PTR). If Fr_POCStatusPtr is a NULL pointer, then the function Fr_GetPOCStatus() shall raise the development error FR_E_PARAM_POINTER.⌋
()

8.4.11 Fr_TransmitTxLPdu

[SWS_Fr_00092] Definition of API function Fr_TransmitTxLPdu ⌈

Service Name	Fr_TransmitTxLPdu	

Syntax	`Std_ReturnType Fr_TransmitTxLPdu (`
	`  uint8 Fr_CtrlIdx,`
	`  uint16 Fr_LPduIdx,`
	`  const uint8* Fr_LSduPtr,`
	`  uint8 Fr_LSduLength,`
	`  Fr_SlotAssignmentType* Fr_SlotAssignmentPtr`
	`)`	

Service ID [hex]	0x0b	
Sync/Async	Asynchronous	
Reentrancy	Non Reentrant for the same device	
Parameters (in)	Fr_CtrlIdx	Index of FlexRay CC within the context of the FlexRay Driver.
	Fr_LPduIdx	This index is used to uniquely identify a FlexRay frame.
	Fr_LSduPtr	This reference points to a buffer where the assembled LSdu to be transmitted within this LPdu is stored at.
	Fr_LSduLength	Determines the length of the data (in Bytes) to be transmitted.
Parameters (inout)	None	
Parameters (out)	Fr_SlotAssignmentPtr	This reference points to the memory location where the actual cycle, slot ID, and channel of the frame identified by Fr_LPduIdx shall be stored. A NULL_PTR indicates that the information is not required by the caller.
Return value	Std_ReturnType	E_OK: API call finished successfully. E_NOT_OK: API call aborted due to errors.
Description	Transmits data on the FlexRay network.	
Available via	Fr.h	

⌋*(SRS_Fr_05003, SRS_Fr_05024)*

CC precondition for the function Fr_TransmitTxLPdu(): None.

[SWS_Fr_00224] ⌈The function Fr_TransmitTxLPdu() shall perform the following tasks on FlexRay CC Fr_CtrIdx:

1. Figure out the physical resource (e.g., a buffer) mapped to the transmission of the FlexRay frame identified by Fr_LPduIdx.
2. If FrExtendedLPduReporting is enabled and Fr_SlotAssignment is not a NULL pointer, copy expected cycle and slot ID of the transmitted frame to Fr_SlotAssignment.
3. If the transmit Lpdu supports dynamic payload length (configuration parameter FrIfAllowDynamicLSduLength is set to true), then the transmission resource shall be reconfigured to match the payload length Fr_LsduLength

passed to the API. Note that the dynamic payload length is only applicable to frames within the dynamic FlexRay segment.

4. Copy Fr_LsduLength bytes from address Fr_LsduPtr into the FlexRay CC's transmission resource and then activate it for transmission.

5. Return E_OK.

⌋*()*

[SWS_Fr_00440] ⌈If a transmit resource is shared between more than 1 Lpdu (using the reconfiguration mechanism of Fr_PrepareLPdu), then the function Fr_TransmitTxLPdu() must ensure that the transmit resource is correctly configured to match the properties of Fr_LpduIdx. This means that if a transmit operation (Fr_TransmitTxLPdu()) is called for a particular Fr_LpduIdx and the Lpdu shares a single buffer with another Lpdu, then it shall check at the time of service invocation whether the buffer is configured according to the Lpdu to be processed.

The function Fr_TransmitTxLPdu() shall return E_NOT_OK and abort the function execution if a wrong buffer configuration is detected.⌋*(SRS_Fr_05024)*

[SWS_Fr_00225] ⌈The Fr module shall ensure that the payload data is transmitted on the FlexRay network in the same byte order as was passed by the parameter Fr_LsduPtr in the function Fr_TransmitTxLPdu(). (first byte = lowest address, last byte = highest address).⌋*()*

[SWS_Fr_00223] ⌈If the function Fr_TransmitTxLPdu() is able to and detects a hardware error while performing the requested functionality, then it shall call Dem_SetEventStatus() (FR_E_CTRL_TESTRESULT, DEM_EVENT_STATUS_FAILED) and return E_NOT_OK.⌋*()*

[SWS_Fr_00218] ⌈If development error detection for the Fr module is enabled, and if the function Fr_TransmitTxLPdu() is called before the successful initialization of Fr, then the function Fr_TransmitTxLPdu() shall raise the development error FR_E_INIT_FAILED.⌋*()*

[SWS_Fr_00219] ⌈If development error detection for the Fr module is enabled, then the function Fr_TransmitTxLPdu() shall check the validity of the parameter Fr_CtrlIdx. If Fr_CtrlIdx is invalid, then the function Fr_TransmitTxLPdu() shall raise the development error FR_E_INV_CTRL_IDX.⌋*()*

[SWS_Fr_00220] ⌈If development error detection for the Fr module is enabled, then the function Fr_TransmitTxLPdu() shall check the validity of the parameter Fr_LpduIdx. If Fr_LpduIdx is invalid, then the function Fr_TransmitTxLPdu() shall raise the development error FR_E_INV_LPDU_IDX.⌋*()*

[SWS_Fr_00221] ⌈If development error detection for the Fr module is enabled, then the function Fr_TransmitTxLPdu() shall check the validity of the parameter Fr_LsduLength. If Fr_LsduLength is invalid, then the function Fr_TransmitTxLPdu() shall raise the development error FR_E_INV_LENGTH.⌋*()*

[SWS_Fr_00222] ⌈If development error detection for the Fr module is enabled, then the function Fr_TransmitTxLPdu() shall check whether the parameter Fr_LsduPtr is a NULL pointer (NULL_PTR). If Fr_LsduPtr is a NULL pointer, then the function Fr_TransmitTxLPdu() shall raise the development error FR_E_PARAM_POINTER.⌋*()*

8.4.12 Fr_CancelTxLPdu

[SWS_Fr_00610] Definition of API function Fr_CancelTxLPdu ⌈

Service Name	Fr_CancelTxLPdu	

Syntax	`Std_ReturnType Fr_CancelTxLPdu (`
	`  uint8 Fr_CtrlIdx,`
	`  uint16 Fr_LPduIdx`
	`)`	

Service ID [hex]	0x2d	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant for the same device	
Parameters (in)	Fr_CtrlIdx	Index of FlexRay CC within the context of the FlexRay Driver.
	Fr_LPduIdx	This index is used to uniquely identify a FlexRay frame
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	E_OK: API call finished successfully. E_NOT_OK: API call aborted due to errors.
Description	Cancels the already pending transmission of a LPdu contained in a controllers physical transmit resource (e.g. message buffer).	
Available via	Fr.h	

⌋*(SRS_Fr_05024)*

CC precondition for the function Fr_CancelTxLPdu(): None.

[SWS_Fr_00611] ⌈The function Fr_CancelTxLPdu() shall perform the following tasks on FlexRay CC Fr_CtrIdx:

- Figure out the physical resource (e.g., a buffer) mapped to the transmission of the FlexRay frame identified by Fr_LpduIdx.
- If the physical resource figured out is actively pending for transmission, then the transmit request of this particular resource shall be terminated and E_OK returned. If no transmission is pending E_NOT_OK shall be returned, indicating that no such cancelation took place.

⌋*()*

[SWS_Fr_00612] ⌈If a transmit resource is shared between more than 1 Lpdu (using reconfiguration mechanism of Fr_PrepareLPdu()), then the function Fr_CancelTxLPdu() must ensure that the transmit resource is correctly configured to match the properties of Fr_LpduIdx. This means that if a cancel transmit operation (Fr_CancelTxLPdu()) is called for a particular Fr_LpduIdx and the Lpdu shares a single buffer with another Lpdu, then it shall check at the time of service invocation that the buffer is configured according to the Lpdu to be processed.

The function Fr_CancelTxLPdu() shall return E_NOT_OK and abort the function execution if a wrong configuration is detected.⌋*()*

[SWS_Fr_00613] ⌈If the function Fr_CancelTxLPdu() is able to and detects a hardware error while performing the requested functionality, then it shall call Dem_SetEventStatus() (FR_E_CTRL_TESTRESULT, DEM_EVENT_STATUS_FAILED) and return E_NOT_OK.⌋*()*

[SWS_Fr_00614] ⌈If development error detection for the Fr module is enabled, and if the function Fr_CancelTxLPdu() is called before the successful initialization of Fr, then the function Fr_CancelTxLPdu() shall raise the development error FR_E_INIT_FAILED.⌋*()*

[SWS_Fr_00615] ⌈If development error detection for the Fr module is enabled, then the function Fr_CancelTxLPdu() shall check the validity of the parameter Fr_CtrlIdx. If Fr_CtrlIdx is invalid, then the function Fr_CancelTxLPdu() shall raise the development error FR_E_INV_CTRL_IDX.⌋*()*

[SWS_Fr_00616] ⌈If development error detection for the Fr module is enabled, then the function Fr_CancelTxLPdu() shall check the validity of the parameter Fr_LpduIdx. If Fr_LpduIdx is invalid, then the function Fr_CancelTxLPdu() shall raise the development error FR_E_INV_LPDU_IDX.⌋*()*

8.4.13 Fr_ReceiveRxLPdu

[SWS_Fr_00093] Definition of API function Fr_ReceiveRxLPdu ⌈

Service Name	Fr_ReceiveRxLPdu	

Syntax	`Std_ReturnType Fr_ReceiveRxLPdu (`
	`  uint8 Fr_CtrlIdx,`
	`  uint16 Fr_LPduIdx,`
	`  uint8* Fr_LSduPtr,`
	`  Fr_RxLPduStatusType* Fr_LPduStatusPtr,`
	`  uint8* Fr_LSduLengthPtr,`
	`  Fr_SlotAssignmentType* Fr_SlotAssignmentPtr`
	`)`	

Service ID [hex]	0x0c	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant for the same device	
Parameters (in)	Fr_CtrlIdx	Index of FlexRay CC within the context of the FlexRay Driver.
	Fr_LPduIdx	This index is used to uniquely identify a FlexRay frame.
Parameters (inout)	None	
Parameters (out)	Fr_LSduPtr	This reference points to the buffer where the LSdu to be received shall be stored.
	Fr_LPduStatusPtr	This reference points to the memory location where the status of the LPdu shall be stored
	Fr_LSduLengthPtr	This reference points to the memory location where the length of the LSdu (in bytes) shall be stored. This length represents the number of bytes copied to Fr_LSduPtr.

▽

△

	Fr_SlotAssignmentPtr	This reference points to the memory location where the actual cycle, slot ID, and channel of the frame identified by Fr_LPduIdx shall be stored. A NULL_PTR indicates that the information is not required by the caller.
Return value	Std_ReturnType	`E_OK`: API call finished successfully. `E_NOT_OK`: API call aborted due to errors.
Description	Receives data from the FlexRay network.	
Available via	Fr.h	

⌋*(SRS_Fr_05003, SRS_Fr_05024)*

CC precondition for the function `Fr_ReceiveRxLPdu`(): None.

[SWS_Fr_00233] ⌈The function `Fr_ReceiveRxLPdu`() shall perform the following tasks on FlexRay CC `Fr_CtrIdx`:

1. Figure out the physical resource (e.g., a buffer, a receive-FIFO) mapped to the reception of the FlexRay frame as identified by `Fr_LpduIdx`.
2. Figure out whether a new FlexRay frame instance has been received within the receive resource as figured in bullet 1. If the receive resource is a FIFO, then consume the first element out of the FIFO.
3. If `FrExtendedLPduReporting` is enabled and `Fr_SlotAssignment` is not a NULL pointer and a new FlexRay frame has been accepted, copy cycle and slot ID of the frame to `Fr_SlotAssignment`.
4. If a new FlexRay frame has been accepted, then copy the received payload data to address `Fr_LsduPtr`, store the number of bytes copied to `Fr_LsduLengthPtr` and store the status `FR_RECEIVED` to `Fr_RxLPduStatusPtr`. If a FIFO is used as received resource and the FIFO is not empty, then store the status `FR_RECEIVED_MORE_DATA_AVAILABLE` to `Fr_RxLPduStatusPtr`.
5. If no new frame has been accepted, then do not copy any payload data to `Fr_LsduPtr`, write `0` to the parameter `Fr_LsduLengthPtr` and store the status `FR_NOT_RECEIVED` to `Fr_RxLPduStatusPtr`.
6. Return `E_OK`.

⌋*()*

[SWS_Fr_00441] ⌈If a receive resource is shared between more than 1 LPdus (using reconfiguration mechanism of `Fr_PrepareLPdu`()), then the function `Fr_ReceiveRxLPdu`() must ensure that the receive resource is correctly configured to match the properties of `Fr_LpduIdx`. This means that if a receive operation (`Fr_ReceiveRxLPdu`()) is called for a particular `Fr_LPduIdx` and the LPdu shares a single buffer with another LPdu, then it shall check that at the time of service invocation the buffer is configured according to the Lpdu to be processed.

The function `Fr_ReceiveRxLPdu`() shall return `E_NOT_OK` and abort the function execution if a wrong buffer configuration is detected.⌋*(SRS_Fr_05024)*

[SWS_Fr_00234] ⌈The function Fr_ReceiveRxLPdu() shall ensure that the payload data is copied to Fr_LSduPtr in the same byte order as it was received on the FlexRay bus. (first byte = lowest address, last byte = highest address).⌋*()*

[SWS_Fr_00604] ⌈If stringent check is disabled by configuration parameter (FrRxStringentCheck is false), then received data is accepted if the SlotStatus shows a valid frame (vSS!ValidFrame). Otherwise FR_NOT_RECEIVED is written to Fr_RxLPduStatusPtr and 0 is written to Fr_LSduLengthPtr.⌋*()*

[SWS_Fr_00603] ⌈If stringent check is enabled by configuration parameter (FrRxStringentCheck is true), then received data is accepted only if the SlotStatus shows a valid frame (vSS!ValidFrame) and there was no single SlotStatus error bit set (vSS!SyntaxError, vSS!ContentError, vSS!Bviolation). Otherwise FR_NOT_RECEIVED is written to Fr_RxLPduStatusPtr and 0 is written to Fr_LsduLengthPtr.⌋*()*

[SWS_Fr_00236] ⌈The function Fr_ReceiveRxLPdu() shall ensure that FR_RECEIVED is returned only for non-Nullframes.⌋*()*

[SWS_Fr_00237] ⌈The function Fr_ReceiveRxLPdu() shall ensure that the function returns FR_RECEIVED only once per received frame.⌋*()*

[SWS_Fr_00645] ⌈If stringent length check is enabled by configuration parameter (FrRxStringentLengthCheck is true), then received data is accepted only if the received payload length exactly matches the expected payload length as provided by configuration parameter FrIfLSduLength. Otherwise FR_NOT_RECEIVED is written to Fr_RxLPduStatusPtr and 0 is written to Fr_LSduLengthPtr .⌋*()*

[SWS_Fr_00239] ⌈The function Fr_ReceiveRxLPdu() shall ensure that the number of payload bytes copied to Fr_LsduPtr, and therefore the payload length stored to Fr_LSduLengthPtr are limited both by the received payload length as well as by the configuration parameter FrIfLSduLength configured in the FrIf.

This enables

- the partly reception of large FlexRay frames (e.g., enables local resource optimizations, support for transparent frame extensions).
- the reception of short FlexRay frames. (e.g., frames with dynamic payload length).

⌋*()*

[SWS_Fr_00232] ⌈If the function Fr_ReceiveRxLPdu() is able to and detects a hardware error while performing the requested functionality, then it shall call Dem_SetEventStatus() (FR_E_CTRL_TESTRESULT, DEM_EVENT_STATUS_FAILED) and return E_NOT_OK.⌋*()*

[SWS_Fr_00605] ⌈If the optional configuration parameter FrIfDemFTSlotStatusRef exists and a single slot status error bit (vSS!SyntaxError, vSS!ContentError, vSS!Bviolation) is set, then the slot status information shall be reported to DEM as Dem_SetEventStatus() (FR_E_LPDU_SLOTSTATUS, DEM_EVENT_STATUS_FAILED).⌋*()*

[SWS_Fr_00627] ⌈If the optional configuration parameter FrIfDemFTSlotStatusRef exists and no single slot status error bit (vSS!SyntaxError, vSS!ContentError, vSS!Bviolation) is set, then the slot status information shall be reported to DEM as Dem_SetEventStatus() (FR_E_LPDU_SLOTSTATUS, DEM_EVENT_STATUS_PASSED).⌋*()*

[SWS_Fr_00628] ⌈Dem_SetEventStatus() shall only be called if the optional configuration parameter FrIfDemFTSlotStatusRef exists.⌋*()*

[SWS_Fr_00226] ⌈If development error detection for the Fr module is enabled, and if the function Fr_ReceiveRxLPdu() is called before the successful initialization of Fr, then the function Fr_ReceiveRxLPdu() shall raise the development error FR_E_INIT_FAILED.⌋*()*

[SWS_Fr_00227] ⌈If development error detection for the Fr module is enabled, then the function Fr_ReceiveRxLPdu() shall check the validity of the parameter Fr_CtrlIdx. If Fr_CtrlIdx is invalid, then the function Fr_ReceiveRxLPdu() shall raise the development error FR_E_INV_CTRL_IDX.⌋*()*

[SWS_Fr_00228] ⌈If development error detection for the Fr module is enabled, then the function Fr_ReceiveRxLPdu() shall check the validity of the parameter Fr_LpduIdx. If Fr_LpduIdx is invalid, then the function Fr_ReceiveRxLPdu() shall raise the development error FR_E_INV_LPDU_IDX.⌋*()*

[SWS_Fr_00229] ⌈If development error detection for the Fr module is enabled, then the function Fr_ReceiveRxLPdu() shall check whether the parameter Fr_LsduPtr is a NULL pointer (NULL_PTR). If Fr_LsduPtr is a NULL pointer, then the function Fr_ReceiveRxLPdu() shall raise the development error FR_E_PARAM_POINTER.⌋*()*

[SWS_Fr_00230] ⌈If development error detection for the Fr module is enabled, then the function Fr_ReceiveRxLPdu() shall check whether the parameter Fr_RxLPduStatusPtr is a NULL pointer (NULL_PTR). If Fr_RxLPduStatusPtr is a NULL pointer, then the function Fr_ReceiveRxLPdu() shall raise the development error FR_E_PARAM_POINTER.⌋*()*

[SWS_Fr_00231] ⌈If development error detection for the Fr module is enabled, then the function Fr_ReceiveRxLPdu() shall check whether the parameter Fr_LsduLengthPtr is a NULL pointer (NULL_PTR). If Fr_LsduLengthPtr is a NULL pointer, then the function Fr_ReceiveRxLPdu() shall raise the development error FR_E_PARAM_POINTER.⌋*()*

8.4.14 Fr_CheckTxLPduStatus

[SWS_Fr_00094] Definition of API function Fr_CheckTxLPduStatus ⌈

Service Name	Fr_CheckTxLPduStatus	
Syntax	`Std_ReturnType* Fr_CheckTxLPduStatus ( uint8 Fr_CtrlIdx, uint16 Fr_LPduIdx, Fr_TxLPduStatusType* Fr_TxLPduStatusPtr, Fr_SlotAssignmentType* Fr_SlotAssignmentPtr )`	
Service ID [hex]	0x0d	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant for the same device	
Parameters (in)	Fr_CtrlIdx	Index of FlexRay CC within the context of the FlexRay Driver.
	Fr_LPduIdx	This index is used to uniquely identify a FlexRay frame
Parameters (inout)	None	
Parameters (out)	Fr_TxLPduStatusPtr	This reference is used to store the transmit status of the LPdu
	Fr_SlotAssignmentPtr	This reference points to the memory location where the actual cycle, slot ID, and channel of the frame identified by Fr_LPduIdx shall be stored. A NULL_PTR indicates that the information is not required by the caller.
Return value	Std_ReturnType*	`E_OK`: API call finished successfully. `E_NOT_OK`: API call aborted due to errors.
Description	Checks the transmit status of the LSdu.	
Available via	Fr.h	

⌋*(SRS_Fr_05003, SRS_Fr_05024)*

CC precondition for the function `Fr_CheckTxLPduStatus`(): None.

[SWS_Fr_00244] ⌈The function `Fr_CheckTxLPduStatus`() shall perform the following tasks on FlexRay CC `Fr_CtrIdx`:

1. Figure out the physical resource (e.g., a buffer) mapped to the transmission of the FlexRay frame identified by `Fr_LpduIdx`.
2. Check whether the transmission resource as figured in bullet 1 is pending for transmission[3] and if a TX conflict (vss!TxConflict) occurred for this resource.
3. If `FrExtendedLPduReporting` is enabled and `Fr_SlotAssignment` is not a NULL pointer, copy cycle and slot ID of the checked frame to `Fr_SlotAssignment`.
4. If a transmission request is pending, then store the status `FR_NOT_TRANSMITTED` to `Fr_TxLPduStatusPtr`.
5. If no transmission request is pending and no TX conflict occurred, then store the status `FR_TRANSMITTED` to `Fr_TxLPduStatusPtr`.
6. If no transmission request is pending and a TX conflict occurred, then store the status `FR_TRANSMITTED_CONFLICT` to `Fr_TxLPduStatusPtr`.

[3]The returned status does not check whether a transmission has been really performed, but returns whether a transmission resource is empty or not.

7. Return E_OK.

⌋()

[SWS_Fr_00243] ⌈If the function Fr_CheckTxLPduStatus() is able to and detects a hardware error while performing the requested functionality, then it shall call Dem_SetEventStatus() (FR_E_CTRL_TESTRESULT, DEM_EVENT_STATUS_FAILED) and return E_NOT_OK.⌋*()*

[SWS_Fr_00606] ⌈If the optional configuration parameter FrIfDemFTSlotStatusRef exists and a single slot status error bit (vSS!SyntaxError, vSS!ContentError, vSS!Bviolation) is set, then the slot status information shall be reported to DEM as Dem_SetEventStatus() (FR_E_LPDU_SLOTSTATUS, DEM_EVENT_STATUS_FAILED).⌋*()*

[SWS_Fr_00629] ⌈If the optional configuration parameter FrIfDemFTSlotStatusRef exists and no single slot status error bit (vSS!SyntaxError, vSS!ContentError, vSS!Bviolation) is set, then the slot status information shall be reported to DEM as Dem_SetEventStatus() (FR_E_LPDU_SLOTSTATUS, DEM_EVENT_STATUS_PASSED).⌋*()*

[SWS_Fr_00630] ⌈Dem_SetEventStatus() shall be called only if the optional configuration parameter FrIfDemFTSlotStatusRef exists.⌋*()*

[SWS_Fr_00240] ⌈If development error detection for the Fr module is enabled, and if the function Fr_CheckTxLPduStatus() is called before the successful initialization of Fr, then the function Fr_CheckTxLPduStatus() shall raise the development error FR_E_INIT_FAILED.⌋*()*

[SWS_Fr_00241] ⌈If development error detection for the Fr module is enabled, then the function Fr_CheckTxLPduStatus() shall check the validity of the parameter Fr_CtrlIdx. If Fr_CtrlIdx is invalid, then the function Fr_CheckTxLPduStatus() shall raise the development error FR_E_INV_CTRL_IDX.⌋*()*

[SWS_Fr_00242] ⌈If development error detection for the Fr module is enabled, then the function Fr_CheckTxLPduStatus() shall check the validity of the parameter Fr_LpduIdx. If Fr_LpduIdx is invalid, then the function Fr_CheckTxLPduStatus() shall raise the development error FR_E_INV_LPDU_IDX.⌋*()*

[SWS_Fr_00343] ⌈If development error detection for the Fr module is enabled, then the function Fr_CheckTxLPduStatus() shall check whether the parameter Fr_TxLPduStatusPtr is a NULL pointer (NULL_PTR). If Fr_TxLPduStatusPtr is a NULL pointer, then the function Fr_CheckTxLPduStatus() shall raise the development error FR_E_PARAM_POINTER.⌋*()*

8.4.15 Fr_PrepareLPdu

[SWS_Fr_00107] Definition of API function Fr_PrepareLPdu ⌈

Service Name	Fr_PrepareLPdu	

Syntax	`Std_ReturnType Fr_PrepareLPdu (`
	`  uint8 Fr_CtrlIdx,`
	`  uint16 Fr_LPduIdx`
	`)`	

Service ID [hex]	0x1f	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant for the same device	
Parameters (in)	Fr_CtrlIdx	Index of FlexRay CC within the context of the FlexRay Driver.
	Fr_LPduIdx	This index is used to uniquely identify a FlexRay frame
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: API call finished successfully. `E_NOT_OK`: API call aborted due to errors.
Description	Prepares a LPdu.	
Available via	Fr.h	

⌋*(SRS_Fr_05106)*

CC precondition for the function `Fr_PrepareLPdu`(): None.

[SWS_Fr_00249] ⌈The function `Fr_PrepareLPdu`() shall perform the following tasks on FlexRay CC `Fr_CtrIdx`:

1. Figure out the physical resource (e.g., a buffer) mapped to the processing of the FlexRay frame identified by `Fr_LpduIdx`.
2. Configure the physical resource (a buffer) appropriate for `Fr_LpduIdx` operation (SlotId, Cycle filter, payload length, header CRC, etc.) if the MCG uses the reconfiguration feature[4].
3. Return `E_OK`.

⌋*()*

[SWS_Fr_00250] ⌈The function `Fr_PrepareLPdu`() shall be pre compile time configurable On/Off by the configuration parameter: `FrPrepareLPduSupport`.⌋*()*

[SWS_Fr_00248] ⌈If the function `Fr_PrepareLPdu`() is able to and detects a hardware error while performing the requested functionality, then it shall call `Dem_SetEventStatus`() (`FR_E_CTRL_TESTRESULT`, `DEM_EVENT_STATUS_FAILED`) and return `E_NOT_OK`.⌋*()*

[4]If the MCG decides to save message buffers using message buffer reconfiguration it assigns two different LPdus (A and B) a single message buffer X. Each LPdu is linked to a (different) FrameTriggering configuration which contains the slot/cycle/channel assignment. Depending whether LPdu A or B is passed to `Fr_PrepareLPdu`(), the message buffer X is configured according to the slot/cycle/channel assignment of the related LPdu.

[SWS_Fr_00245] ⌈If development error detection for the Fr module is enabled, and if the function Fr_PrepareLPdu() is called before the successful initialization of Fr, then the function Fr_PrepareLPdu() shall raise the development error FR_E_INIT_FAILED.⌋*()*

[SWS_Fr_00246] ⌈If development error detection for the Fr module is enabled, then the function Fr_PrepareLPdu() shall check the validity of the parameter Fr_CtrlIdx. If Fr_CtrlIdx is invalid, then the function Fr_PrepareLPdu() shall raise the development error FR_E_INV_CTRL_IDX.⌋*()*

[SWS_Fr_00247] ⌈If development error detection for the Fr module is enabled, then the function Fr_PrepareLPdu() shall check the validity of the parameter Fr_LpduIdx. If Fr_LpduIdx is invalid, then the function Fr_PrepareLPdu() shall raise the development error FR_E_INV_LPDU_IDX.⌋*()*

8.4.16 Fr_ReconfigLPdu

[SWS_Fr_00524] Definition of API function Fr_ReconfigLPdu ⌈

Service Name	Fr_ReconfigLPdu	

Syntax	`Std_ReturnType Fr_ReconfigLPdu (`
	`  uint8 Fr_CtrlIdx,`
	`  uint16 Fr_LPduIdx,`
	`  uint16 Fr_FrameId,`
	`  Fr_ChannelType Fr_ChnlIdx,`
	`  uint8 Fr_CycleRepetition,`
	`  uint8 Fr_CycleOffset,`
	`  uint8 Fr_PayloadLength,`
	`  uint16 Fr_HeaderCRC`
	`)`	

Service ID [hex]	0x25	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant for the same device	
Parameters (in)	Fr_CtrlIdx	Index of FlexRay CC within the context of the FlexRay Driver.
	Fr_LPduIdx	This index is used to uniquely identify a FlexRay frame
	Fr_FrameId	FlexRay Frame ID the FrIf_LPdu shall be configured to.
	Fr_ChnlIdx	FlexRay Channel the FrIf_LPdu shall be configured to.
	Fr_CycleRepetition	Cycle Repetition part of the cycle filter mechanism FrIf_LPdu shall be configured to.
	Fr_CycleOffset	Cycle Offset part of the cycle filter mechanism FrIf_LPdu shall be configured to.
	Fr_PayloadLength	Payloadlength in units of bytes the FrIf_LPduIdx shall be configured to.
	Fr_HeaderCRC	Header CRC the FrIf_LPdu shall be configured to.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	E_OK: API call finished successfully. E_NOT_OK: API call aborted due to errors.

▽

△

Description	Reconfigures a given LPdu according to the parameters (FrameId, Channel, CycleRepetition, CycleOffset, PayloadLength, HeaderCRC) at runtime.
Available via	Fr.h

⌋*(SRS_Fr_05059)*

CC precondition for the function Fr_ReconfigLPdu(): None.

[SWS_Fr_00525] ⌈The function Fr_ReconfigLPdu() shall perform the following tasks on FlexRay CC Fr_CtrlIdx:

- Figure out the physical resource (e.g., a buffer) mapped to the processing of the FlexRay frame as identified by Fr_LpduIdx.
- Configure the physical resource (a buffer) according to the parameters given at the API. The Lpdu direction is statically associated with the Lpdu and cannot be changed by this service.
- Return E_OK.

⌋*()*

[SWS_Fr_00526] ⌈Whether an Lpdu is dynamically reconfigurable is determined via the configuration parameter FrIfReconfigurable which is a property of the FrIfLPdu configuration parameter container.⌋*()*

[SWS_Fr_00527] ⌈Since FlexRay supports only even number of bytes as payload length, the parameter Fr_PayloadLength must be internally rounded to the next higher even number if it is odd.⌋*()*

[SWS_Fr_00528] ⌈The function Fr_ReconfigLPdu() shall be pre compile time configurable On/Off by the configuration parameter: FrReconfigLPduSupport.⌋*()*

[SWS_Fr_00529] ⌈If the function Fr_ReconfigLPdu() is able to and detects a hardware error while performing the requested functionality, then it shall call Dem_SetEventStatus() (FR_E_CTRL_TESTRESULT, DEM_EVENT_STATUS_FAILED) and return E_NOT_OK.⌋*()*

[SWS_Fr_00530] ⌈If development error detection for the Fr module is enabled, and if the function Fr_ReconfigLPdu() is called before the successful initialization of Fr, then the function Fr_ReconfigLPdu() shall raise the development error FR_E_INIT_FAILED.⌋*()*

[SWS_Fr_00531] ⌈If development error detection for the Fr module is enabled, then the function Fr_ReconfigLPdu() shall check the validity of the parameter Fr_CtrlIdx. If Fr_CtrlIdx is invalid, then the function Fr_ReconfigLPdu() shall raise the development error FR_E_INV_CTRL_IDX.⌋*()*

[SWS_Fr_00532] ⌈If development error detection for the Fr module is enabled, then the function Fr_ReconfigLPdu() shall check the validity of the parameter Fr_LpduIdx. If Fr_LpduIdx is invalid, then the function Fr_ReconfigLPdu() shall raise the development error FR_E_INV_LPDU_IDX.⌋*()*

[SWS_Fr_00533] ⌈If development error detection for the Fr module is enabled, then the function Fr_ReconfigLPdu() shall check the validity of the parameter Fr_ChnlIdx. If Fr_ChnlIdx is invalid, then the function Fr_ReconfigLPdu() shall raise the development error FR_E_INV_CHNL_IDX.⌋*()*

[SWS_Fr_00534] ⌈If development error detection for the Fr module is enabled, then the function Fr_ReconfigLPdu() shall check the validity of the parameter Fr_CycleRepetition. If Fr_CycleRepetition is invalid, then the function Fr_ReconfigLPdu() shall raise the development error FR_E_INV_CYCLE.⌋*()*

[SWS_Fr_00535] ⌈Valid values[5] for parameter Fr_CycleRepetition are 1, 2, 4, 5, 8, 10, 16, 20, 32, 40, 50 and 64.⌋*()*

[SWS_Fr_00536] ⌈If development error detection for the Fr module is enabled, then the function Fr_ReconfigLPdu() shall check the parameter Fr_CycleOffset for being valid. If Fr_CycleOffset is invalid, then the function Fr_ReconfigLPdu() shall raise the development error FR_E_INV_CYCLE.⌋*()*

[SWS_Fr_00537] ⌈Valid values for parameter Fr_CycleOffset are 0 to (Fr_CycleRepetition - 1).⌋*()*

[SWS_Fr_00538] ⌈If development error detection for the Fr module is enabled, then the function Fr_ReconfigLPdu() shall check the validity of the parameter Fr_PayloadLength. If Fr_PayloadLength is invalid, then the function Fr_ReconfigLPdu() shall raise the development error FR_E_INV_LENGTH.⌋*()*

[SWS_Fr_00634] ⌈If development error detection for the Fr module is enabled, then the function Fr_ReconfigLPdu() shall check the parameter Fr_HeaderCRC for being valid[6]. If Fr_HeaderCRC is invalid, then the function Fr_ReconfigLPdu() shall raise the development error FR_E_INV_HEADERCRC.⌋*()*

8.4.17 Fr_DisableLPdu

[SWS_Fr_00539] Definition of API function Fr_DisableLPdu ⌈

Service Name	Fr_DisableLPdu

Syntax	`Std_ReturnType Fr_DisableLPdu (`
	`  uint8 Fr_CtrlIdx,`
	`  uint16 Fr_LPduIdx`
	`)`

Service ID [hex]	0x26
Sync/Async	Synchronous
Reentrancy	Non Reentrant for the same device

▽

[5]For FlexRay Controllers compliant to [2] only cycle repetition values 1, 2, 4, 8, 16, 32 and 64 shall be supported.

[6]This does not mean that the CRC shall be recalculated. Instead the CRC shall be checked whether it fits in the allowed value range (0 - 2047) or not.

△

Parameters (in)	Fr_CtrlIdx	Index of FlexRay CC within the context of the FlexRay Driver.
	Fr_LPduIdx	This index is used to uniquely identify a FlexRay frame
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	E_OK: API call finished successfully. E_NOT_OK: API call aborted due to errors.
Description	Disables the hardware resource of a LPdu for transmission/reception.	
Available via	Fr.h	

⌋*(SRS_Fr_05059)*

CC precondition for the function Fr_DisableLPdu(): None.

[SWS_Fr_00540] ⌈The function Fr_DisableLPdu() shall perform the following tasks on FlexRay CC Fr_CtrIdx:

1. Figure out the physical resource (e.g., a buffer) mapped to the processing of the FlexRay frame identified by Fr_LpduIdx.
2. Configure the physical resource (a buffer) in a way that it doesn't take part in the transmission/reception process.
3. Return E_OK.

⌋*()*

[SWS_Fr_00541] ⌈Only Lpdus that can be dynamically reconfigured can be disabled by this service (see Fr_ReconfigLPdu()).⌋*()*

[SWS_Fr_00542] ⌈The function Fr_DisableLPdu() shall be pre compile time configurable On/Off by the configuration parameter: FrDisableLPduSupport.⌋*()*

[SWS_Fr_00543] ⌈If the function Fr_DisableLPdu() is able to and detects a hardware error while performing the requested functionality, then it shall call Dem_SetEventStatus() (FR_E_CTRL_TESTRESULT, DEM_EVENT_STATUS_FAILED) and return E_NOT_OK.⌋*()*

[SWS_Fr_00544] ⌈If development error detection for the Fr module is enabled, and if the function Fr_DisableLPdu() is called before the successful initialization of Fr, then the function Fr_DisableLPdu() shall raise the development error FR_E_INIT_FAILED and return E_NOT_OK.⌋*()*

[SWS_Fr_00545] ⌈If development error detection for the Fr module is enabled, then the function Fr_DisableLPdu() shall check the validity of the parameter Fr_CtrlIdx. If Fr_CtrlIdx is invalid, then the function Fr_DisableLPdu() shall raise the development error FR_E_INV_CTRL_IDX.⌋*()*

[SWS_Fr_00546] ⌈If development error detection for the Fr module is enabled, then the function Fr_DisableLPdu() shall check the validity of the parameter Fr_LpduIdx. If Fr_LpduIdx is invalid, then the function Fr_DisableLPdu() shall raise the development error FR_E_INV_LPDU_IDX.⌋*()*

8.4.18 Fr_GetGlobalTime

[SWS_Fr_00042] Definition of API function Fr_GetGlobalTime ⌈

Service Name	Fr_GetGlobalTime	

Syntax	`Std_ReturnType Fr_GetGlobalTime (`
	`  uint8 Fr_CtrlIdx,`
	`  uint8* Fr_CyclePtr,`
	`  uint16* Fr_MacroTickPtr`
	`)`	

Service ID [hex]	0x10	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant for the same device	
Parameters (in)	Fr_CtrlIdx	Index of FlexRay CC within the context of the FlexRay Driver.
Parameters (inout)	None	
Parameters (out)	Fr_CyclePtr	Address where the current FlexRay communication cycle value shall be stored.
	Fr_MacroTickPtr	Address where the current macrotick value shall be stored.
Return value	Std_ReturnType	`E_OK`: API call finished successfully. `E_NOT_OK`: API call aborted due to errors.
Description	Gets the current global FlexRay time. Important Note: Fr_GetGlobalTime may be called within an exclusive area.	
Available via	Fr.h	

⌋*(SRS_Fr_05019)*

Note: The Fr module's environment shall only call `Fr_GetGlobalTime`() if the CC `Fr_CtrlIdx` is synchronous to FlexRay global time.

[SWS_Fr_00256] ⌈The function `Fr_GetGlobalTime`() shall perform the following tasks on FlexRay CC `Fr_CtrIdx`:

1. Read the current global FlexRay time and write it to the output parameters `Fr_CyclePtr` and `Fr_MacrotickPtr`.
2. Return `E_OK`.

⌋*()*

[SWS_Fr_00257] ⌈The function `Fr_GetGlobalTime`() shall ensure that the time information is consistent and valid. This means that the values returned of both parameters (`Fr_CyclePtr` and `Fr_MacrotickPtr`) must be taken from a single point in time. The resulting global time shall always strictly increase over time (until it wraps around at the time-boundary).⌋*()*

[SWS_Fr_00044] ⌈The function `Fr_GetGlobalTime`() shall ensure that the time information is valid and up to date (synchronized CC), - otherwise the output parameters shall not be written and `E_NOT_OK` returned.⌋*(SRS_Fr_05072)*

[SWS_Fr_00255] ⌈If the function `Fr_GetGlobalTime`() is able to and detects a hardware error while performing the requested functionality, then it shall call `Dem_SetEventStatus`() (`FR_E_CTRL_TESTRESULT`, `DEM_EVENT_STATUS_FAILED`) and return `E_NOT_OK`.⌋*()*

[SWS_Fr_00251] ⌈If development error detection for the Fr module is enabled, and if the function Fr_GetGlobalTime() is called before the successful initialization of Fr, then the function Fr_GetGlobalTime() shall raise the development error FR_E_INIT_FAILED.⌋*()*

[SWS_Fr_00252] ⌈If development error detection for the Fr module is enabled, then the function Fr_GetGlobalTime() shall check the validity of the parameter Fr_CtrlIdx. If Fr_CtrlIdx is invalid, then the function Fr_GetGlobalTime() shall raise the development error FR_E_INV_CTRL_IDX.⌋*()*

[SWS_Fr_00253] ⌈If development error detection for the Fr module is enabled, then the function Fr_GetGlobalTime() shall check whether the parameter Fr_CyclePtr is a NULL pointer (NULL_PTR). If Fr_CyclePtr is a NULL pointer, the function Fr_GetGlobalTime() shall raise the development error FR_E_PARAM_POINTER.⌋*()*

[SWS_Fr_00254] ⌈If development error detection for the Fr module is enabled, then the function Fr_GetGlobalTime() shall check whether the parameter Fr_MacroTickPtr is a NULL pointer (NULL_PTR). If Fr_MacroTickPtr is a NULL pointer, the function Fr_GetGlobalTime() shall raise the development error FR_E_PARAM_POINTER.⌋*()*

8.4.19 Fr_GetNmVector

[SWS_Fr_00113] Definition of API function Fr_GetNmVector ⌈

Service Name	Fr_GetNmVector	

Syntax	`Std_ReturnType Fr_GetNmVector (`
	`  uint8 Fr_CtrlIdx,`
	`  uint8* Fr_NmVectorPtr`
	`)`	

Service ID [hex]	0x22	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant for the same device	
Parameters (in)	Fr_CtrlIdx	Index of FlexRay CC within the context of the FlexRay Driver.
Parameters (inout)	None	
Parameters (out)	Fr_NmVectorPtr	Address where the NmVector of the last communication cycle shall be stored.
Return value	Std_ReturnType	E_OK: API call finished successfully. E_NOT_OK: API call aborted due to errors.
Description	Gets the network management vector of the last communication cycle.	
Available via	Fr.h	

⌋*()*

Note: The Fr module's environment shall only call the function Fr_GetNmVector() when the CC Fr_CtrlIdx is synchronous to FlexRay global time.

Note: The NM-Vector will be updated at a defined point in time (see [1]). At this point of time the service Fr_GetNmVector() shall not be called, in order to ensure a consistent NM-Vector value.

[SWS_Fr_00262] ⌈The function Fr_GetNmVector() shall perform the following tasks on FlexRay CC Fr_CtrlIdx:

1. Read the current accrued network management vector out of the FlexRay CC and then write it to the output parameter Fr_NmVectorPtr. The number of bytes written to the output parameter is constant and is known at configuration time (FrIf configuration parameter FrIfGNetworkManagementVectorLength).

2. Return E_OK.

⌋*()*

[SWS_Fr_00263] ⌈The function Fr_GetNmVector() shall ensure that the FlexRay CC is synchronous to global time when the data is read - otherwise the output parameters shall not be written and E_NOT_OK returned.⌋*()*

[SWS_Fr_00264] ⌈The function Fr_GetNmVector() shall ensure that the payload data is copied to Fr_NmVectorPtr in the same byte order as they were received on the FlexRay bus. (first byte = lowest address, last byte = highest address).⌋*()*

[SWS_Fr_00261] ⌈If the function Fr_GetNmVector() is able to and detects a hardware error while performing the requested functionality, then it shall call Dem_SetEventStatus() (FR_E_CTRL_TESTRESULT, DEM_EVENT_STATUS_FAILED) and return E_NOT_OK.⌋*()*

[SWS_Fr_00258] ⌈If development error detection for the Fr module is enabled, and if the function Fr_GetNmVector() is called before the successful initialization of Fr, then the function Fr_GetNmVector() shall raise the development error FR_E_INIT_FAILED.⌋*()*

[SWS_Fr_00259] ⌈If development error detection for the Fr module is enabled, then the function Fr_GetNmVector() shall check the validity of the parameter Fr_CtrlIdx. If Fr_CtrlIdx is invalid, then the function Fr_GetNmVector() shall raise the development error FR_E_INV_CTRL_IDX.⌋*()*

[SWS_Fr_00260] ⌈If development error detection for the Fr module is enabled, then the function Fr_GetNmVector() shall check whether the parameter Fr_NmVectorPtr is a NULL pointer (NULL_PTR). If Fr_NmVectorPtr is a NULL pointer, then the function Fr_GetNmVector() shall raise the development error FR_E_PARAM_POINTER.⌋*()*

8.4.20 Fr_GetNumOfStartupFrames

[SWS_Fr_00547] Definition of API function Fr_GetNumOfStartupFrames ⌈

Service Name	Fr_GetNumOfStartupFrames	

Syntax	`Std_ReturnType Fr_GetNumOfStartupFrames (`
	`  uint8 Fr_CtrlIdx,`
	`  uint8* Fr_NumOfStartupFramesPtr`
	`)`	

Service ID [hex]	0x27	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant for the same device	
Parameters (in)	Fr_CtrlIdx	Index of FlexRay CC within the context of the FlexRay Driver.
Parameters (inout)	None	
Parameters (out)	Fr_NumOfStartupFrames Ptr	Address where the number of startup frames seen within the last even/odd cycle pair shall be stored.
Return value	Std_ReturnType	`E_OK`: API call finished successfully. `E_NOT_OK`: API call aborted due to errors.
Description	Gets the current number of startup frames seen on the cluster. See variable vStartupPairs of [12] for details.	
Available via	Fr.h	

⌋*()* [7]

Note: The Fr module's environment shall only call `Fr_GetNumOfStartupFrames`() if the CC `Fr_CtrlIdx` is synchronous to FlexRay global time.

[SWS_Fr_00549] ⌈The function `Fr_GetNumOfStartupFrames`() shall perform the following tasks on FlexRay CC `Fr_CtrIdx`:

1. Read the number of aligned startup frame pairs received or transmitted during the previous double cycle, aggregated across both channels and write it to the output parameter `Fr_NumOfStartupFramesPtr`.
2. Return `E_OK`.

⌋*()*

[SWS_Fr_00550] ⌈If the hardware doesn't support accumulating the number of startupframes, (FlexRay 2.1 Rev A compliant hardware), then the driver shall always assume 2 startup frames available.⌋*()*

[SWS_Fr_00551] ⌈The function `Fr_GetNumOfStartupFrames`() shall ensure that the information is valid and up to date (synchronized CC) - otherwise the output parameters shall not be written and `E_NOT_OK` returned.⌋*()*

[SWS_Fr_00552] ⌈If the function `Fr_GetNumOfStartupFrames`() is able to and detects a hardware error while performing the requested functionality, then it shall call `Dem_SetEventStatus`() (`FR_E_CTRL_TESTRESULT`, `DEM_EVENT_STATUS_FAILED`) and return `E_NOT_OK`.⌋*()*

[7] FlexRay 2.1 Rev A compliant controllers do not support `vStartupPairs`. See [SWS_Fr_00550] for FlexRay 2.1 Rev A controllers implementation constraints.

[SWS_Fr_00553] ⌈If development error detection for the Fr module is enabled, and if the function Fr_GetNumOfStartupFrames() is called before the successful initialization of Fr, then the function Fr_GetNumOfStartupFrames() shall raise the development error FR_E_INIT_FAILED.⌋*()*

[SWS_Fr_00554] ⌈If development error detection for the Fr module is enabled, then the function Fr_GetNumOfStartupFrames() shall check the validity of the parameter Fr_CtrlIdx. If Fr_CtrlIdx is invalid, then the function Fr_GetNumOfStartupFrames() shall raise the development error FR_E_INV_CTRL_IDX.⌋*()*

[SWS_Fr_00555] ⌈If development error detection for the Fr module is enabled, then the function Fr_GetNumOfStartupFrames() shall check whether the parameter Fr_NumOfStartupFramesPtr is a NULL pointer (NULL_PTR). If Fr_NumOfStartupFramesPtr is a NULL pointer, then the function Fr_GetNumOfStartupFrames() shall raise the development error FR_E_PARAM_POINTER.⌋*()*

8.4.21 Fr_GetChannelStatus

[SWS_Fr_00556] Definition of API function Fr_GetChannelStatus ⌈

Service Name	Fr_GetChannelStatus	
Syntax	`Std_ReturnType Fr_GetChannelStatus ( uint8 Fr_CtrlIdx, uint16* Fr_ChannelAStatusPtr, uint16* Fr_ChannelBStatusPtr )`	
Service ID [hex]	0x28	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant for the same device	
Parameters (in)	Fr_CtrlIdx	Index of FlexRay CC within the context of the FlexRay Driver.
Parameters (inout)	None	
Parameters (out)	Fr_ChannelAStatusPtr	Address where the bitcoded channel A status information shall be stored.
	Fr_ChannelBStatusPtr	Address where the bitcoded channel B status information shall be stored.
Return value	Std_ReturnType	E_OK: API call finished successfully. E_NOT_OK: API call aborted due to errors.
Description	Gets the channel status information.	
Available via	Fr.h	

⌋*()*

Note: The Fr module's environment shall only call Fr_GetChannelStatus() if the CC Fr_CtrlIdx is synchronous to FlexRay global time.

[SWS_Fr_00558] ⌈The function Fr_GetChannelStatus() shall perform the following tasks on FlexRay CC Fr_CtrIdx:

1. Read the aggregated channel status, NIT status, symbol window status and write it to the output parameter Fr_ChannelAStatusPtr/ Fr_ChannelBStatusPtr. The value of *Fr_ChannelAStatusPtr/

*Fr_ChannelBStatusPtr is bitcoded with the following meaning (Bit 0 = LSB, Bit 15 = MSB)[8]:

Bit 0 Channel A/B aggregated channel status vSS!ValidFrame

Bit 1 Channel A/B aggregated channel status vSS!SyntaxError

Bit 2 Channel A/B aggregated channel status vSS!ContentError

Bit 3 Channel A/B aggregated channel status additional communication

Bit 4 Channel A/B aggregated channel status vSS!Bviolation

Bit 5 Channel A/B aggregated channel status vSS!TxConflict

Bit 6 Not used (0)

Bit 7 Not used (0)

Bit 8 Channel A/B symbol window status data vSS!ValidMTS

Bit 9 Channel A/B symbol window status data vSS!SyntaxError

Bit 10 Channel A/B symbol window status data vSS!Bviolation

Bit 11 Channel A/B symbol window status data vSS!TxConflict

Bit 12 Channel A/B NIT status data vSS!SyntaxError

Bit 13 Channel A/B NIT status data vSS!Bviolation

Bit 14 Not used (0)

Bit 15 Not used (0)

2. Reset the aggregated channel status information within the FlexRay controller.
3. Return E_OK.

⌋*()*

[SWS_Fr_00559] ⌈The function Fr_GetChannelStatus() shall ensure that the information is valid and up to date (synchronized CC) - otherwise the output parameters shall not be written and E_NOT_OK returned.⌋*()*

[SWS_Fr_00560] ⌈If the function Fr_GetChannelStatus() is able to and detects a hardware error while performing the requested functionality, then it shall call Dem_SetEventStatus() (FR_E_CTRL_TESTRESULT, DEM_EVENT_STATUS_FAILED) and return E_NOT_OK.⌋*()*

[SWS_Fr_00561] ⌈If development error detection for the Fr module is enabled, and if the function Fr_GetChannelStatus() is called before the successful initialization of Fr, then the function Fr_GetChannelStatus() shall raise the development error FR_E_INIT_FAILED.⌋*()*

[8] Bit 5 and Bit 11 shall be set to 0 for FlexRay 2.1 compliant controllers, since vSS!TxConflict is not supported on this hardware.

[SWS_Fr_00562] ⌈If development error detection for the Fr module is enabled, then the function Fr_GetChannelStatus() shall check the validity of the parameter Fr_CtrlIdx. If Fr_CtrlIdx is invalid, then the function Fr_GetChannelStatus() shall raise the development error FR_E_INV_CTRL_IDX.⌋*()*

[SWS_Fr_00563] ⌈If development error detection for the Fr module is enabled, then the function Fr_GetChannelStatus() shall check whether the parameter Fr_ChannelAStatusPtr is a NULL pointer (NULL_PTR). If Fr_ChannelAStatusPtr is a NULL pointer, then the function Fr_GetChannelStatus() shall raise the development error FR_E_PARAM_POINTER.⌋*()*

[SWS_Fr_00607] ⌈If development error detection for the Fr module is enabled, then the function Fr_GetChannelStatus() shall check whether the parameter Fr_ChannelBStatusPtr is a NULL pointer (NULL_PTR). If Fr_ChannelBStatusPtr is a NULL pointer, then the function Fr_GetChannelStatus() shall raise the development error FR_E_PARAM_POINTER.⌋*()*

8.4.22 Fr_GetClockCorrection

[SWS_Fr_00564] Definition of API function Fr_GetClockCorrection ⌈

Service Name	Fr_GetClockCorrection	

Syntax	`Std_ReturnType Fr_GetClockCorrection (`
	`  uint8 Fr_CtrlIdx,`
	`  sint16* Fr_RateCorrectionPtr,`
	`  sint32* Fr_OffsetCorrectionPtr`
	`)`	

Service ID [hex]	0x29	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant for the same device	
Parameters (in)	Fr_CtrlIdx	Index of FlexRay CC within the context of the FlexRay Driver.
Parameters (inout)	None	
Parameters (out)	Fr_RateCorrectionPtr	Address where the current rate correction value shall be stored.
	Fr_OffsetCorrectionPtr	Address where the current offset correction value shall be stored.
Return value	Std_ReturnType	E_OK: API call finished successfully. E_NOT_OK: API call aborted due to errors.
Description	Gets the current clock correction values. See variables vInterimRateCorrection and vInterim OffsetCorrection of [12] for details.	
Available via	Fr.h	

⌋*()* [9] [10]

[SWS_Fr_00566] ⌈The function Fr_GetClockCorrection() shall perform the following tasks on FlexRay CC Fr_CtrlIdx:

[9] vInterimRate Correction maps to vRateCorrection for FlexRay 2.1 compliant controllers, see [2]
[10] vInterimOffsetCorrection maps to vOffsetCorrection for FlexRay 2.1 compliant controllers, see [2]

1. Read the rate correction value (vInterimRateCorrection[11]) and write it as signed integer to the output parameter Fr_RateCorrectionPtr. Read the offset correction value (vInterimOffsetCorrection[12]) and write it as signed integer to the output parameter Fr_OffsetCorrectionPtr
2. Return E_OK.

⌋*()*

[SWS_Fr_00568] ⌈If the function Fr_GetClockCorrection() is able to and detects a hardware error while performing the requested functionality, then it shall call Dem_SetEventStatus() (FR_E_CTRL_TESTRESULT, DEM_EVENT_STATUS_FAILED) and return E_NOT_OK.⌋*()*

[SWS_Fr_00569] ⌈If development error detection for the Fr module is enabled, and if the function Fr_GetClockCorrection() is called before the successful initialization of Fr, then the function Fr_GetClockCorrection() shall raise the development error FR_E_INIT_FAILED.⌋*()*

[SWS_Fr_00570] ⌈If development error detection for the Fr module is enabled, then the function Fr_GetClockCorrection() shall check the validity of the parameter Fr_CtrlIdx. If Fr_CtrlIdx is invalid, then the function Fr_GetClockCorrection() shall raise the development error FR_E_INV_CTRL_IDX.⌋*()*

[SWS_Fr_00571] ⌈If development error detection for the Fr module is enabled, then the function Fr_GetClockCorrection() shall check whether the parameter Fr_RateCorrectionPtr is a NULL pointer (NULL_PTR). If Fr_RateCorrectionPtr is a NULL pointer, then the function Fr_GetClockCorrection() shall raise the development error FR_E_PARAM_POINTER.⌋*()*

[SWS_Fr_00572] ⌈If development error detection for the Fr module is enabled, then the function Fr_GetClockCorrection() shall check whether the parameter Fr_OffsetCorrectionPtr is a NULL pointer (NULL_PTR). If Fr_OffsetCorrectionPtr is a NULL pointer, then the function Fr_GetClockCorrection() shall raise the development error FR_E_PARAM_POINTER.⌋*()*

[11]vInterimRate Correction maps to vRateCorrection for FlexRay 2.1 compliant controllers, see [2]
[12]vInterimOffsetCorrection maps to vOffsetCorrection for FlexRay 2.1 compliant controllers, see [2]

8.4.23 Fr_GetSyncFrameList

[SWS_Fr_00573] Definition of API function Fr_GetSyncFrameList ⌈

Service Name	Fr_GetSyncFrameList	

Syntax	`Std_ReturnType Fr_GetSyncFrameList (`
	`  uint8 Fr_CtrlIdx,`
	`  uint8 Fr_ListSize,`
	`  uint16* Fr_ChannelAEvenListPtr,`
	`  uint16* Fr_ChannelBEvenListPtr,`
	`  uint16* Fr_ChannelAOddListPtr,`
	`  uint16* Fr_ChannelBOddListPtr`
	`)`	

Service ID [hex]	0x2a	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant for the same device	
Parameters (in)	Fr_CtrlIdx	Index of FlexRay CC within the context of the FlexRay Driver.
	Fr_ListSize	Size of the arrays passed via parameters: Fr_ChannelAEvenList Ptr Fr_ChannelBEvenListPtr Fr_ChannelAOddListPtr Fr_Channel BOddListPtr. The service must ensure to not write more entries into those arrays than granted by this parameter.
Parameters (inout)	None	
Parameters (out)	Fr_ChannelAEvenListPtr	Address the list of syncframes on channel A within the even communication cycle is written to. The exact number of elements written to the list is limited by parameter Fr_ListSize. Unused list elements are filled with the value '0' to indicate that no more syncframe has been seen.
	Fr_ChannelBEvenListPtr	Address the list of syncframes on channel B within the even communication cycle is written to. The exact number of elements written to the list is limited by parameter Fr_ListSize. Unused list elements are filled with the value '0' to indicate that no more syncframe has been seen.
	Fr_ChannelAOddListPtr	Address the list of syncframes on channel A within the odd communication cycle is written to. The exact number of elements written to the list is limited by parameter Fr_ListSize. Unused list elements are filled with the value '0' to indicate that no more syncframe has been seen.
	Fr_ChannelBOddListPtr	Address the list of syncframes on channel B within the odd communication cycle is written to. The exact number of elements written to the list is limited by parameter Fr_ListSize. Unused list elements are filled with the value '0' to indicate that no more syncframe has been seen.
Return value	Std_ReturnType	`E_OK`: API call finished successfully. `E_NOT_OK`: API call aborted due to errors.
Description	Gets a list of syncframes received or transmitted on channel A and channel B via the even and odd communication cycle. See variables vsSyncIdListA and vsSyncIdListB of [12] for details.	
Available via	Fr.h	

⌋*()*

[SWS_Fr_00575] ⌈The function `Fr_GetSyncFrameList`() shall perform the following tasks on FlexRay CC `Fr_CtrlIdx`:

1. Read the list of syncframes received in the last even communication cycle on channel A and write it as array to the memory location `Fr_ChannelAEvenListPtr`.

2. Read the list of syncframes received in the last even communication cycle on channel B and write it as array to the memory location `Fr_ChannelBEvenListPtr`.
3. Read the list of syncframes received in the last odd communication cycle on channel A and write it as array to the memory location `Fr_ChannelAOddListPtr`.
4. Read the list of syncframes received in the last odd communication cycle on channel B and write it as array to the memory location `Fr_ChannelBOddListPtr`.
5. Return `E_OK`.

⌋*()*

[SWS_Fr_00576] ⌈The size of the array written to `Fr_ChannelAEvenListPtr`, `Fr_ChannelBEvenListPtr`, `Fr_ChannelAOddListPtr` and `Fr_ChannelBOddListPtr` shall be limited to `Fr_ListSize` (array elements 0 to (`Fr_ListSize` - 1)).⌋*()*

[SWS_Fr_00577] ⌈Unused array elements shall be set to 0, indicating no valid sync frame.⌋*()*

[SWS_Fr_00578] ⌈A maximum number of 15 syncframes shall be supported.⌋*()*

[SWS_Fr_00580] ⌈If the function `Fr_GetSyncFrameList`() is able to and detects a hardware error while performing the requested functionality, then it shall call `Dem_SetEventStatus`() (`FR_E_CTRL_TESTRESULT`, `DEM_EVENT_STATUS_FAILED`) and return `E_NOT_OK`.⌋*()*

[SWS_Fr_00581] ⌈If development error detection for the Fr module is enabled, and if the function `Fr_GetSyncFrameList`() is called before the successful initialization of Fr, then the function `Fr_GetSyncFrameList`() shall raise the development error `FR_E_INIT_FAILED`.⌋*()*

[SWS_Fr_00582] ⌈If development error detection for the Fr module is enabled, then the function `Fr_GetSyncFrameList`() shall check the validity of the parameter `Fr_CtrlIdx`. If `Fr_CtrlIdx` is invalid, then the function `Fr_GetSyncFrameList`() shall raise the development error `FR_E_INV_CTRL_IDX`.⌋*()*

[SWS_Fr_00667] ⌈If development error detection for the Fr module is enabled, then the function `Fr_GetSyncFrameList`() shall check the validity of the parameter `Fr_ListSize`. If `Fr_ListSize` is larger than 15, then the function `Fr_GetSyncFrameList`() shall raise the development error `FR_E_INV_FRAMELIST_SIZE`.⌋*()*

[SWS_Fr_00583] ⌈If development error detection for the Fr module is enabled, then the function `Fr_GetSyncFrameList`() shall check whether the parameter `Fr_ChannelAEvenListPtr` is a NULL pointer (`NULL_PTR`). If `Fr_ChannelAEvenListPtr` is a NULL pointer, then the function `Fr_GetSyncFrameList`() shall raise the development error `FR_E_PARAM_POINTER`.⌋*()*

[SWS_Fr_00584] ⌈If development error detection for the Fr module is enabled, then the function Fr_GetSyncFrameList() shall check whether the parameter Fr_ChannelBEvenListPtr is a NULL pointer (NULL_PTR). If Fr_ChannelBEvenListPtr is a NULL pointer, then the function Fr_GetSyncFrameList() shall raise the development error FR_E_PARAM_POINTER.⌋ *()*

[SWS_Fr_00585] ⌈If development error detection for the Fr module is enabled, then the function Fr_GetSyncFrameList() shall check whether the parameter Fr_ChannelAOddListPtr is a NULL pointer (NULL_PTR). If Fr_ChannelAOddListPtr is a NULL pointer, then the function Fr_GetSyncFrameList() shall raise the development error FR_E_PARAM_POINTER.⌋ *()*

[SWS_Fr_00586] ⌈If development error detection for the Fr module is enabled, then the function Fr_GetSyncFrameList() shall check whether the parameter Fr_ChannelBOddListPtr is a NULL pointer (NULL_PTR). If Fr_ChannelBOddListPtr is a NULL pointer, then the function Fr_GetSyncFrameList() shall raise the development error FR_E_PARAM_POINTER.⌋ *()*

8.4.24 Fr_GetWakeupRxStatus

[SWS_Fr_00587] Definition of API function Fr_GetWakeupRxStatus ⌈

Service Name	Fr_GetWakeupRxStatus	

Syntax	`Std_ReturnType Fr_GetWakeupRxStatus (`
	`  uint8 Fr_CtrlIdx,`
	`  uint8* Fr_WakeupRxStatusPtr`
	`)`	

Service ID [hex]	0x2b	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant for the same device	
Parameters (in)	Fr_CtrlIdx	Index of FlexRay CC within the context of the FlexRay Driver.
Parameters (inout)	None	
Parameters (out)	Fr_WakeupRxStatusPtr	Address where bitcoded wakeup reception status shall be stored. Bit 0: Wakeup received on channel A indicator Bit 1: Wakeup received on channel B indicator Bit 2-7: Unused
Return value	Std_ReturnType	E_OK: API call finished successfully. E_NOT_OK: API call aborted due to errors.
Description	Gets the wakeup received information from the FlexRay controller.	
Available via	Fr.h	

⌋ *()*

[SWS_Fr_00588] ⌈The function Fr_GetWakeupRxStatus() shall perform the following tasks on FlexRay CC Fr_CtrIdx:

1. Read the wakeup pattern received indicators for channel A and channel B and write it to the output parameter Fr_WakeupRxStatusPtr. The value of

⋆Fr_WakeupRxStatusPtr is bitcoded with the following meaning (Bit 0 = LSB, Bit 7 = MSB):

Bit 0 Wakeup pattern received on channel A (1), otherwise (0)

Bit 1 Wakeup pattern received on channel B (1), otherwise (0)

Bit 2 Not used (always 0)

Bit 3 Not used (always 0)

Bit 4 Not used (always 0)

Bit 5 Not used (always 0)

Bit 6 Not used (always 0)

Bit 7 Not used (always 0)

2. Reset the wakeup received indication status information within the FlexRay controller.
3. Return E_OK.

⌋*()*

[SWS_Fr_00589] ⌈If the function Fr_GetWakeupRxStatus() is able to and detects a hardware error while performing the requested functionality, then it shall call Dem_SetEventStatus() (FR_E_CTRL_TESTRESULT, DEM_EVENT_STATUS_FAILED) and return E_NOT_OK.⌋*()*

[SWS_Fr_00590] ⌈If development error detection for the Fr module is enabled, and if the function Fr_GetWakeupRxStatus() is called before the successful initialization of Fr, then the function Fr_GetWakeupRxStatus() shall raise the development error FR_E_INIT_FAILED.⌋*()*

[SWS_Fr_00591] ⌈If development error detection for the Fr module is enabled, then the function Fr_GetWakeupRxStatus() shall check the validity of the parameter Fr_CtrlIdx. If Fr_CtrlIdx is invalid, then the function Fr_GetWakeupRxStatus() shall raise the development error FR_E_INV_CTRL_IDX.⌋*()*

[SWS_Fr_00592] ⌈If development error detection for the Fr module is enabled, then the function Fr_GetWakeupRxStatus() shall check whether the parameter Fr_WakeupRxStatusPtr is a NULL pointer (NULL_PTR). If Fr_WakeupRxStatusPtr is a NULL pointer, then the function Fr_GetWakeupRxStatus() shall raise the development error FR_E_PARAM_POINTER.⌋*()*

8.4.25 Fr_SetAbsoluteTimer

[SWS_Fr_00033] Definition of API function Fr_SetAbsoluteTimer ⌈

Service Name	Fr_SetAbsoluteTimer	

Syntax	`Std_ReturnType Fr_SetAbsoluteTimer (`
	`  uint8 Fr_CtrlIdx,`
	`  uint8 Fr_AbsTimerIdx,`
	`  uint8 Fr_Cycle,`
	`  uint16 Fr_Offset`
	`)`	

Service ID [hex]	0x11	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant for the same device	
Parameters (in)	Fr_CtrlIdx	Index of FlexRay CC within the context of the FlexRay Driver.
	Fr_AbsTimerIdx	Index of absolute timer within the context of the FlexRay CC.
	Fr_Cycle	Absolute cycle the timer shall elapse in.
	Fr_Offset	Offset within cycle Fr_Cycle in units of macrotick the timer shall elapse at.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: API call finished successfully. `E_NOT_OK`: API call aborted due to errors.
Description	Sets the absolute FlexRay timer.	
Available via	Fr.h	

⌋*(SRS_Fr_05044)*

Note: The Fr module's environment shall only call `Fr_SetAbsoluteTimer`() when the CC `Fr_CtrlIdx` is synchronous to FlexRay global time (at the moment of timer activation).

[SWS_Fr_00273] ⌈The function `Fr_SetAbsoluteTimer`() shall perform the following tasks:

1. Program the absolute FlexRay timer `Fr_AbsTimerIdx` according to the parameters `Fr_Cycle` and `Fr_Offset`.
2. Return `E_OK`.

⌋*()*

[SWS_Fr_00272] ⌈If the function `Fr_SetAbsoluteTimer`() is able to and detects a hardware error while performing the requested functionality, then it shall call `Dem_SetEventStatus`() (`FR_E_CTRL_TESTRESULT`, `DEM_EVENT_STATUS_FAILED`) and return `E_NOT_OK`.⌋*()*

[SWS_Fr_00267] ⌈If development error detection for the Fr module is enabled, and if the function `Fr_SetAbsoluteTimer`() is called before the successful initialization of Fr, then the function `Fr_SetAbsoluteTimer`() shall raise the development error `FR_E_INIT_FAILED`.⌋*()*

[SWS_Fr_00268] ⌈If development error detection for the Fr module is enabled, then the function `Fr_SetAbsoluteTimer`() shall check the validity of the parameter

Fr_CtrlIdx. If Fr_CtrlIdx is invalid, then the function Fr_SetAbsoluteTimer() shall raise the development error FR_E_INV_CTRL_IDX.⌋*()*

[SWS_Fr_00269] ⌈If development error detection for the Fr module is enabled, then the function Fr_SetAbsoluteTimer() shall check the validity of the parameter Fr_AbsTimerIdx. If Fr_AbsTimerIdx is invalid, then the function Fr_SetAbsoluteTimer() shall raise the development error FR_E_INV_TIMER_IDX.⌋*()*

[SWS_Fr_00270] ⌈If development error detection for the Fr module is enabled, then the function Fr_SetAbsoluteTimer() shall check the validity of the parameter Fr_Cycle. If Fr_Cycle is invalid, then the function Fr_SetAbsoluteTimer() shall raise the development error FR_E_INV_CYCLE.⌋*()*

[SWS_Fr_00271] ⌈If development error detection for the Fr module is enabled, then the function Fr_SetAbsoluteTimer() shall check the validity of the parameter Fr_Offset. If Fr_Offset is invalid, then the function Fr_SetAbsoluteTimer() shall raise the development error FR_E_INV_OFFSET.⌋*()*

[SWS_Fr_00436] ⌈The function Fr_SetAbsoluteTimer() shall check whether the CC Fr_CtrlIdx is synchronous to the FlexRay global time. If the CC Fr_CtrlIdx is not synchronous to the FlexRay global time, then the function Fr_SetAbsoluteTimer() shall raise the runtime error FR_E_INV_POCSTATE.⌋*()*

8.4.26 Fr_CancelAbsoluteTimer

[SWS_Fr_00095] Definition of API function Fr_CancelAbsoluteTimer ⌈

Service Name	Fr_CancelAbsoluteTimer	

Syntax	`Std_ReturnType Fr_CancelAbsoluteTimer (`
	`  uint8 Fr_CtrlIdx,`
	`  uint8 Fr_AbsTimerIdx`
	`)`	

Service ID [hex]	0x13	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant for the same device	
Parameters (in)	Fr_CtrlIdx	Index of FlexRay CC within the context of the FlexRay Driver.
	Fr_AbsTimerIdx	Index of absolute timer within the context of the FlexRay CC.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	E_OK: API call finished successfully. E_NOT_OK: API call aborted due to errors.
Description	Stops an absolute timer.	
Available via	Fr.h	

⌋*(SRS_Fr_05044)*

CC precondition for the function Fr_CancelAbsoluteTimer(): None.

[SWS_Fr_00287] ⌈The function Fr_CancelAbsoluteTimer() shall perform the following tasks:

1. Stop the absolute timer Fr_AbsTimerIdx.
2. Return E_OK.

⌋()

[SWS_Fr_00286] ⌈If the function Fr_CancelAbsoluteTimer() is able to and detects a hardware error while performing the requested functionality, then it shall call Dem_SetEventStatus() (FR_E_CTRL_TESTRESULT, DEM_EVENT_STATUS_FAILED) and return E_NOT_OK.⌋*()*

[SWS_Fr_00283] ⌈If development error detection for the Fr module is enabled, and if the function Fr_CancelAbsoluteTimer() is called before the successful initialization of Fr, then the function Fr_CancelAbsoluteTimer() shall raise the development error FR_E_INIT_FAILED.⌋*()*

[SWS_Fr_00284] ⌈If development error detection for the Fr module is enabled, then the function Fr_CancelAbsoluteTimer() shall check the validity of the parameter Fr_CtrlIdx. If Fr_CtrlIdx is invalid, then the function Fr_CancelAbsoluteTimer() shall raise the development error FR_E_INV_CTRL_IDX.⌋*()*

[SWS_Fr_00285] ⌈If development error detection for the Fr module is enabled, then the function Fr_CancelAbsoluteTimer() shall check the validity of the parameter Fr_AbsTimerIdx. If Fr_AbsTimerIdx is invalid, then the function Fr_CancelAbsoluteTimer() shall raise the development error FR_E_INV_TIMER_IDX.⌋*()*

8.4.27 Fr_EnableAbsoluteTimerIRQ

[SWS_Fr_00034] Definition of API function Fr_EnableAbsoluteTimerIRQ ⌈

Service Name	Fr_EnableAbsoluteTimerIRQ	
Syntax	`Std_ReturnType Fr_EnableAbsoluteTimerIRQ ( uint8 Fr_CtrlIdx, uint8 Fr_AbsTimerIdx )`	
Service ID [hex]	0x15	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant for the same device	
Parameters (in)	Fr_CtrlIdx	Index of FlexRay CC within the context of the FlexRay Driver.
	Fr_AbsTimerIdx	Index of absolute timer within the context of the FlexRay CC.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	E_OK: API call finished successfully. E_NOT_OK: API call aborted due to errors.
Description	Enables the interrupt line of an absolute timer.	
Available via	Fr.h	

⌋*(SRS_Fr_05125, SRS_Fr_05046)*

CC precondition for the function Fr_EnableAbsoluteTimerIRQ(): None.

[SWS_Fr_00298] ⌈The function Fr_EnableAbsoluteTimerIRQ() shall perform the following tasks:

1. Enable the interrupt line related to timer Fr_AbsTimerIdx.
2. Return E_OK.

⌋*()*

[SWS_Fr_00297] ⌈If the function Fr_EnableAbsoluteTimerIRQ() is able to and detects a hardware error while performing the requested functionality, then it shall call Dem_SetEventStatus() (FR_E_CTRL_TESTRESULT, DEM_EVENT_STATUS_FAILED) and return E_NOT_OK.⌋*()*

[SWS_Fr_00294] ⌈If development error detection for the Fr module is enabled, and if the function Fr_EnableAbsoluteTimerIRQ() is called before the successful initialization of Fr, then the function Fr_EnableAbsoluteTimerIRQ() shall raise the development error FR_E_INIT_FAILED.⌋*()*

[SWS_Fr_00295] ⌈If development error detection for the Fr module is enabled, then the function Fr_EnableAbsoluteTimerIRQ() shall check the validity of the parameter Fr_CtrlIdx. If Fr_CtrlIdx is invalid, then the function Fr_EnableAbsoluteTimerIRQ() shall raise the development error FR_E_INV_CTRL_IDX.⌋*()*

[SWS_Fr_00296] ⌈If development error detection for the Fr module is enabled, then the function Fr_EnableAbsoluteTimerIRQ() shall check the validity of the parameter Fr_AbsTimerIdx. If Fr_AbsTimerIdx is invalid, then the function Fr_EnableAbsoluteTimerIRQ() shall raise the development error FR_E_INV_TIMER_IDX.⌋*()*

8.4.28 Fr_AckAbsoluteTimerIRQ

[SWS_Fr_00036] Definition of API function Fr_AckAbsoluteTimerIRQ ⌈

Service Name	Fr_AckAbsoluteTimerIRQ	

Syntax	`Std_ReturnType Fr_AckAbsoluteTimerIRQ (`
	`  uint8 Fr_CtrlIdx,`
	`  uint8 Fr_AbsTimerIdx`
	`)`	

Service ID [hex]	0x17	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant for the same device	
Parameters (in)	Fr_CtrlIdx	Index of FlexRay CC within the context of the FlexRay Driver.
	Fr_AbsTimerIdx	Index of absolute timer within the context of the FlexRay CC.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	E_OK: API call finished successfully. E_NOT_OK: API call aborted due to errors.
Description	Resets the interrupt condition of an absolute timer.	
Available via	Fr.h	

⌋*(SRS_Fr_05125, SRS_Fr_05048)*

CC precondition for the function Fr_AckAbsoluteTimerIRQ(): None.

[SWS_Fr_00309] ⌈The function Fr_AckAbsoluteTimerIRQ() shall perform the following tasks:

1. Reset the interrupt condition of absolute timer Fr_AbsTimerIdx.
2. Return E_OK.

⌋*()*

[SWS_Fr_00308] ⌈If the function Fr_AckAbsoluteTimerIRQ() is able to and detects a hardware error while performing the requested functionality, then it shall call Dem_SetEventStatus() (FR_E_CTRL_TESTRESULT, DEM_EVENT_STATUS_FAILED) and return E_NOT_OK.⌋*()*

[SWS_Fr_00305] ⌈If development error detection for the Fr module is enabled, and if the function Fr_AckAbsoluteTimerIRQ() is called before the successful initialization of Fr, then the function Fr_AckAbsoluteTimerIRQ() shall raise the development error FR_E_INIT_FAILED.⌋*()*

[SWS_Fr_00306] ⌈If development error detection for the Fr module is enabled, then the function Fr_AckAbsoluteTimerIRQ() shall check the validity of the parameter Fr_CtrlIdx. If Fr_CtrlIdx is invalid, then the function Fr_AckAbsoluteTimerIRQ() shall raise the development error FR_E_INV_CTRL_IDX.⌋*()*

[SWS_Fr_00307] ⌈If development error detection for the Fr module is enabled, then the function Fr_AckAbsoluteTimerIRQ() shall check the validity of the parameter Fr_AbsTimerIdx. If Fr_AbsTimerIdx is invalid, then the function Fr_AckAbsoluteTimerIRQ() shall raise the development error FR_E_INV_TIMER_IDX.⌋*()*

8.4.29 Fr_DisableAbsoluteTimerIRQ

[SWS_Fr_00035] Definition of API function Fr_DisableAbsoluteTimerIRQ ⌈

Service Name	Fr_DisableAbsoluteTimerIRQ	

Syntax	`Std_ReturnType Fr_DisableAbsoluteTimerIRQ (`
	`  uint8 Fr_CtrlIdx,`
	`  uint8 Fr_AbsTimerIdx`
	`)`	

Service ID [hex]	0x19	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant for the same device	
Parameters (in)	Fr_CtrlIdx	Index of FlexRay CC within the context of the FlexRay Driver.
	Fr_AbsTimerIdx	Index of absolute timer within the context of the FlexRay CC.
Parameters (inout)	None	
Parameters (out)	None	

▽

△

Return value	Std_ReturnType	E_OK: API call finished successfully. E_NOT_OK: API call aborted due to errors.
Description	Disables the interrupt line of an absolute timer.	
Available via	Fr.h	

⌋*(SRS_Fr_05125, SRS_Fr_05047)*

CC precondition for the function Fr_DisableAbsoluteTimerIRQ(): None.

[SWS_Fr_00320] ⌈The function Fr_DisableAbsoluteTimerIRQ() shall perform the following tasks:

1. Disable the interrupt line related to absolute timer Fr_AbsTimerIdx.
2. Return E_OK.

⌋*()*

[SWS_Fr_00319] ⌈If the function Fr_DisableAbsoluteTimerIRQ() is able to and detects a hardware error while performing the requested functionality, then it shall call Dem_SetEventStatus() (FR_E_CTRL_TESTRESULT, DEM_EVENT_STATUS_FAILED) and return E_NOT_OK.⌋*()*

[SWS_Fr_00316] ⌈If development error detection for the Fr module is enabled, and if the function Fr_DisableAbsoluteTimerIRQ() is called before the successful initialization of Fr, then the function Fr_DisableAbsoluteTimerIRQ() shall raise the development error FR_E_INIT_FAILED.⌋*()*

[SWS_Fr_00317] ⌈If development error detection for the Fr module is enabled, then the function Fr_DisableAbsoluteTimerIRQ() shall check the validity of the parameter Fr_CtrlIdx. If Fr_CtrlIdx is invalid, then the function Fr_DisableAbsoluteTimerIRQ() shall raise the development error FR_E_INV_CTRL_IDX.⌋*()*

[SWS_Fr_00318] ⌈If development error detection for the Fr module is enabled, then the function Fr_DisableAbsoluteTimerIRQ() shall check the validity of the parameter Fr_AbsTimerIdx. If Fr_AbsTimerIdx is invalid, then the function Fr_DisableAbsoluteTimerIRQ() shall raise the development error FR_E_INV_TIMER_IDX.⌋*()*

8.4.30 Fr_GetAbsoluteTimerIRQStatus

[SWS_Fr_00108] Definition of API function Fr_GetAbsoluteTimerIRQStatus ⌈

Service Name	Fr_GetAbsoluteTimerIRQStatus

Syntax	`Std_ReturnType Fr_GetAbsoluteTimerIRQStatus (`
	`  uint8 Fr_CtrlIdx,`
	`  uint8 Fr_AbsTimerIdx,`
	`  boolean* Fr_IRQStatusPtr`
	`)`

▽

△

Service ID [hex]	0x20	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant for the same device	
Parameters (in)	Fr_CtrlIdx	Index of FlexRay CC within the context of the FlexRay Driver.
	Fr_AbsTimerIdx	Index of absolute timer within the context of the FlexRay CC.
Parameters (inout)	None	
Parameters (out)	Fr_IRQStatusPtr	Address the output value is stored to.
Return value	Std_ReturnType	`E_OK`: API call finished successfully. `E_NOT_OK`: API call aborted due to errors.
Description	Gets IRQ status of an absolute timer.	
Available via	Fr.h	

⌋*(SRS_Fr_05125)*

CC precondition for the function `Fr_GetAbsoluteTimerIRQStatus`(): None.

[SWS_Fr_00332] ⌈The function `Fr_GetAbsoluteTimerIRQStatus`() shall perform the following tasks:

1. Check whether the interrupt of absolute timer `Fr_AbsTimerIdx` is pending. Write `TRUE` to output parameter `Fr_IRQStatusPtr` in case the interrupt is pending, `FALSE` otherwise.
2. Return `E_OK`.

⌋*()*

[SWS_Fr_00331] ⌈If the function `Fr_GetAbsoluteTimerIRQStatus`() is able to and detects a hardware error while performing the requested functionality, then it shall call `Dem_SetEventStatus`() (`FR_E_CTRL_TESTRESULT`, `DEM_EVENT_STATUS_FAILED`) and return `E_NOT_OK`.⌋*()*

[SWS_Fr_00327] ⌈If development error detection for the Fr module is enabled, and if the function `Fr_GetAbsoluteTimerIRQStatus`() is called before the successful initialization of Fr, then the function `Fr_GetAbsoluteTimerIRQStatus`() shall raise the development error `FR_E_INIT_FAILED`.⌋*()*

[SWS_Fr_00328] ⌈If development error detection for the Fr module is enabled, then the function `Fr_GetAbsoluteTimerIRQStatus`() shall check the validity of the parameter `Fr_CtrlIdx`. If `Fr_CtrlIdx` is invalid, then the function `Fr_GetAbsoluteTimerIRQStatus`() shall raise the development error `FR_E_INV_CTRL_IDX`.⌋*()*

[SWS_Fr_00329] ⌈If development error detection for the Fr module is enabled, then the function `Fr_GetAbsoluteTimerIRQStatus`() shall check the validity of the parameter `Fr_AbsTimerIdx`. If `Fr_AbsTimerIdx` is invalid, then the function `Fr_GetAbsoluteTimerIRQStatus`() shall raise the development error `FR_E_INV_TIMER_IDX`.⌋*()*

[SWS_Fr_00330] ⌈If development error detection for the Fr module is enabled, then the function `Fr_GetAbsoluteTimerIRQStatus`() shall check whether the parameter `Fr_IRQStatusPtr` is a NULL pointer (`NULL_PTR`). If `Fr_IRQStatusPtr` is a

NULL pointer, then the function Fr_GetAbsoluteTimerIRQStatus() shall raise the development error FR_E_PARAM_POINTER.⌋*()*

8.4.31 Fr_GetVersionInfo

[SWS_Fr_00070] Definition of API function Fr_GetVersionInfo ⌈

Service Name	Fr_GetVersionInfo	
Syntax	`void Fr_GetVersionInfo ( Std_VersionInfoType* VersioninfoPtr )`	
Service ID [hex]	0x1b	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	VersioninfoPtr	Pointer to where to store the version information of this module.
Return value	None	
Description	Returns the version information of this module.	
Available via	Fr.h	

⌋*(SRS_BSW_00407, SRS_BSW_00411)*

[SWS_Fr_00340] ⌈If development error detection for the Fr module is enabled, then the function Fr_GetVersionInfo() shall check whether the parameter VersioninfoPtr is a NULL pointer (NULL_PTR). If VersioninfoPtr is a NULL pointer, then the function Fr_GetVersionInfo() shall raise the development error FR_E_PARAM_POINTER and return.⌋*(SRS_BSW_00411)*

8.4.32 Fr_ReadCCConfig

[SWS_Fr_00651] Definition of API function Fr_ReadCCConfig ⌈

Service Name	Fr_ReadCCConfig	
Syntax	`Std_ReturnType Fr_ReadCCConfig ( uint8 Fr_CtrlIdx, uint8 Fr_ConfigParamIdx, uint32* Fr_ConfigParamValuePtr )`	
Service ID [hex]	0x2e	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant for the same device	
Parameters (in)	Fr_CtrlIdx	Index of FlexRay CC within the context of the FlexRay Driver.
	Fr_ConfigParamIdx	Index that identifies the configuration parameter to read. See macros FR_CIDX_<config_parameter_name>.
Parameters (inout)	None	

▽

△

Parameters (out)	Fr_ConfigParamValuePtr	Address the output value is stored to.
Return value	Std_ReturnType	E_OK: API call finished successfully. E_NOT_OK: API call aborted due to errors.
Description	Reads a FlexRay protocol configuration parameter for a particular FlexRay controller out of the module's configuration.	
Available via	Fr.h	

⌋*()*

The function Fr_ReadCCConfig() shall perform the following tasks:

1. Read the value of the configuration parameter requested by Fr_ConfigParamIdx from the configuration and write it to output parameter *Fr_ConfigParamValuePtr.
2. Return E_OK.

[SWS_Fr_00652] ⌈If the function Fr_ReadCCConfig() is able to and detects a hardware error while performing the requested functionality, then it shall call Dem_SetEventStatus() (FR_E_CTRL_TESTRESULT, DEM_EVENT_STATUS_FAILED) and return E_NOT_OK.⌋*()*

[SWS_Fr_00653] ⌈If development error detection for the Fr module is enabled, and if the function Fr_ReadCCConfig() is called before the successful initialization of Fr, then the function Fr_ReadCCConfig() shall raise the development error FR_E_INIT_FAILED.⌋*()*

[SWS_Fr_00654] ⌈If development error detection for the Fr module is enabled, then the function Fr_ReadCCConfig() shall check the validity of the parameter Fr_CtrlIdx. If Fr_CtrlIdx is invalid, then the function Fr_ReadCCConfig() shall raise the development error FR_E_INV_CTRL_IDX.⌋*()*

[SWS_Fr_00655] ⌈If development error detection for the Fr module is enabled, then the function Fr_ReadCCConfig() shall check the validity of the parameter Fr_ConfigParamIdx. If Fr_ConfigParamIdx is invalid, then the function Fr_ReadCCConfig() shall raise the development error FR_E_INV_CONFIG_IDX.⌋*()* [13]

[SWS_Fr_00656] ⌈If development error detection for the Fr module is enabled, then the function Fr_ReadCCConfig() shall check whether the parameter Fr_ConfigParamValuePtr is a NULL pointer (NULL_PTR). If Fr_ConfigParamValuePtr is a NULL pointer, then the function Fr_ReadCCConfig() shall raise the development error FR_E_PARAM_POINTER.⌋*()*

Configuration parameters values are specified as integer, float, enumeration or boolean. In order to map those values to the output parameter of type uint32, the following generic rules for conversion shall be applied for integer and float:

- **[SWS_Fr_00658]** ⌈integers are mapped 1 to 1.⌋*()*

[13]Valid values are listed in subsection 8.2.1 "Configuration parameter index macros" and and in requirements [SWS_Fr_00662], [SWS_Fr_00663], [SWS_Fr_00664], [SWS_Fr_00665], [SWS_Fr_00666].

- **[SWS_Fr_00659]** ⌈floats (units of seconds) are converted to units of nanoseconds (with nanosecond granularity) and converted to uint32.⌋*()*
- **[SWS_Fr_00661]** ⌈booleans shall output 1 for true and 0 for false.⌋*()*

For configuration parameters specified as enumeration type, the following mappings shall be applied:

[SWS_Fr_00662] ⌈If parameter `Fr_ConfigParamIdx` is set to `FR_CIDX_PCHANNELS` (`FrPChannels`) then the value stored at `Fr_ConfigParamValuePtr` shall be interpreted as the following literals

0 FR_CHANNEL_A

1 FR_CHANNEL_B

2 FR_CHANNEL_AB

⌋*()*

[SWS_Fr_00663] ⌈If parameter `Fr_ConfigParamIdx` is set to `FR_CIDX_PSAMPLESPERMICROTICK` (`FrPSamplesPerMicrotick`) then the value stored at `Fr_ConfigParamValuePtr` shall be interpreted as the following literals

0 N1SAMPLES

1 N2SAMPLES

2 N4SAMPLES

⌋*()*

[SWS_Fr_00664] ⌈If parameter `Fr_ConfigParamIdx` is set to `FR_CIDX_PWAKEUPCHANNEL` (`FrPWakeupChannel`) then the value stored at `Fr_ConfigParamValuePtr` shall be interpreted as the following literals

0 FR_CHANNEL_A

1 FR_CHANNEL_B

⌋*()*

[SWS_Fr_00665] ⌈If parameter `Fr_ConfigParamIdx` is set to `FR_CIDX_PDMICROTICK` (`FrPdMicrotick`) then the value stored at `Fr_ConfigParamValuePtr` shall be interpreted as the following literals

0 T12_5NS

1 T25NS

2 T50NS

3 T100NS

4 T200NS

⌋()

[SWS_Fr_00666] ⌈If parameter `Fr_ConfigParamIdx` is set to `FR_CIDX_GDSAMPLECLOCKPERIOD` (`FrIfGdSampleClockPeriod`) then the value stored at `Fr_ConfigParamValuePtr` shall be interpreted as the following literals

0 T12_5NS

1 T25NS

2 T50NS

⌋()

8.5 Callback notifications

The FlexRay Driver does not call any callbacks.

8.6 Scheduled functions

The FlexRay driver, which is executed in the context of the FlexRay Interface has no function to be scheduled.

8.7 Expected interfaces

In this chapter all interfaces required from other modules are listed.

8.7.1 Mandatory interfaces

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

[SWS_Fr_00390] Definition of mandatory interfaces in module Fr ⌈

API Function	*Header File*	*Description*
Dem_SetEventStatus	Dem.h	Called by SW-Cs or BSW modules to report monitor status information to the Dem. BSW modules calling Dem_SetEventStatus can safely ignore the return value. This API will be available only if ({Dem/Dem ConfigSet/DemEventParameter/DemEvent ReportingType} == STANDARD_REPORTING)
Det_ReportRuntimeError	Det.h	Service to report runtime errors. If a callout has been configured then this callout shall be called.

⌋()

8.7.2 Optional interfaces

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

[SWS_Fr_00391] Definition of optional interfaces in module Fr ⌈

API Function	***Header File***	***Description***
Det_ReportError	Det.h	Service to report development errors.
SchM_Enter_Fr	<none>	Invokes the SchM_Enter function to enter a module local exclusive area.
SchM_Exit_Fr	<none>	Invokes the SchM_Exit function to exit an exclusive area.

⌋*()*

Further optional interfaces might be accessed in case the Fr uses other modules for accessing the CC hardware.

8.7.3 Configurable interfaces

There are no configurable interfaces related to the FlexRay Driver.

8.8 Service Interfaces

No service interfaces provided.

9 Sequence diagrams

The usage of the driver is depicted in the Sequence diagrams of the FlexRay Interface.

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 FlexRay Driver.

Chapter 10.3 specifies published information of the module FlexRay Driver.

10.1 How to read this chapter

For details refer to the chapter 10.1 “Introduction to configuration specification” in SWS_BSWGeneral.

[SWS_Fr_00670] ⌈The Flexray Driver module shall reject configurations with partition mappings which are not supported by the implementation.⌋*()*

10.2 Containers and configuration parameters

The following chapters summarize all configuration parameters. The detailed meanings of the parameters describe Chapter 7 and Chapter 8.

10.2.1 Fr

SWS Item	**[ECUC_Fr_00456]**
Module Name	Fr
Description	Configuration of the Fr (FlexRay driver) module.
Post-Build Variant Support	true
Supported Config Variants	VARIANT-POST-BUILD, VARIANT-PRE-COMPILE

Included Containers		
Container Name	**Multiplicity**	**Scope / Dependency**
FrGeneral	1	General configuration (parameters) of the FlexRay Driver module.
FrMultipleConfiguration	1	This container contains the configuration parameters and sub containers of the AUTOSAR Fr module.

10.2.2 FrGeneral

SWS Item	**[ECUC_Fr_00392]**
Container Name	FrGeneral
Parent Container	Fr
Description	General configuration (parameters) of the FlexRay Driver module.
Configuration Parameters	

SWS Item	**[ECUC_Fr_00001]**		
Parameter Name	FrCtrlTestCount		
Parent Container	FrGeneral		
Description	Maxmimum number of iterations the FlexRay controller hardware test is performed during controller initialization.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 255		
Default value	1		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

SWS Item	**[ECUC_Fr_00393]**		
Parameter Name	FrDevErrorDetect		
Parent Container	FrGeneral		
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		

SWS Item	**[ECUC_Fr_00455]**		
Parameter Name	FrDisableLPduSupport		
Parent Container	FrGeneral		
Description	Enables or disabled API function Fr_DisableLPdu.		
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		

SWS Item	**[ECUC_Fr_00459]**		
Parameter Name	FrExtendedLPduReporting		
Parent Container	FrGeneral		
Description	Enables or disables reporting of actual cycle and slot ID by Fr_TransmitTxLPdu, Fr_ReceiveRxLPdu, and Fr_CheckTxLPduStatus.		
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		

SWS Item	**[ECUC_Fr_00439]**		
Parameter Name	FrIndex		
Parent Container	FrGeneral		
Description	Specifies the InstanceId of this module instance. If only one instance is present it shall have the Id 0.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 255		
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		

SWS Item	**[ECUC_Fr_00394]**		
Parameter Name	FrNumCtrlSupported		
Parent Container	FrGeneral		
Description	Determines the maximum number of communication controllers that the driver supports.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 256		
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		

SWS Item	**[ECUC_Fr_00453]**		
Parameter Name	FrPrepareLPduSupport		
Parent Container	FrGeneral		
Description	Enables or disables API function Fr_PrepareLPdu.		
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		

SWS Item	**[ECUC_Fr_00454]**		
Parameter Name	FrReconfigLPduSupport		
Parent Container	FrGeneral		
Description	Enables or disabled API function Fr_ReconfigLPdu.		
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		

SWS Item	**[ECUC_Fr_00002]**		
Parameter Name	FrRxStringentCheck		
Parent Container	FrGeneral		
Description	If stringent check is enabled (true), received frames are accepted only if no slot status error occurred.		
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		

SWS Item	**[ECUC_Fr_00016]**
Parameter Name	FrRxStringentLengthCheck
Parent Container	FrGeneral
Description	If stringent check is enabled (true), received frames are accepted only if the received payload length matches the configured payload length.
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		

SWS Item	**[ECUC_Fr_00396]**		
Parameter Name	FrVersionInfoApi		
Parent Container	FrGeneral		
Description	Enables/disables the existence of the Fr_GetVersionInfo API.		
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		

SWS Item	**[ECUC_Fr_00457]**		
Parameter Name	FrEcucPartitionRef		
Parent Container	FrGeneral		
Description	Maps the Flexray driver to zero or multiple ECUC partitions to make the modules API available in this partition. The Flexray driver will operate as an independent instance in each of the partitions.		
Multiplicity	0..*		
Type	Reference to EcucPartition		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: ECU		

No Included Containers

[SWS_Fr_CONSTR_00001] ⌈The module will operate as an independent instance in each of the partitions, means the called API will only target the partition it is called in. (see FrEcucPartitionRef)⌋*()*

10.2.3 FrController

SWS Item	**[ECUC_Fr_00083]**		
Container Name	FrController		
Parent Container	FrMultipleConfiguration		
Description	Configuration of the individual controller.		
Configuration Parameters			

SWS Item	**[ECUC_Fr_00400]**		
Parameter Name	FrCtrlIdx		
Parent Container	FrController		
Description	Determines index of CC within Fr.		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 255		
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 withAuto = true		

SWS Item	**[ECUC_Fr_00402]**		
Parameter Name	FrPAllowHaltDueToClock		
Parent Container	FrController		
Description	Boolean flag that controls the transition to the POC:halt state due to a clock synchronization errors. If set to true, the CC is allowed to transition to POC:halt. If set to false, the CC will not transition to the POC:halt state but will enter or remain in the POC:normal passive state (self healing would still be possible)		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	–		
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		

SWS Item	**[ECUC_Fr_00403]**	
Parameter Name	FrPAllowPassiveToActive	
Parent Container	FrController	
Description	Number of consecutive even/odd cycle pairs that must have valid clock correction terms before the CC will be allowed to transition from the POC:normal passive state to POC:normal active state. If set to zero, the CC is not allowed to transition from POC:normal passive to POC:normal active	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	0 .. 31	

▽

△

Default value	–		
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		

SWS Item	**[ECUC_Fr_00404]**		
Parameter Name	FrPChannels		
Parent Container	FrController		
Description	Channels to which the node is connected. Implementation Type: Fr_ChannelType		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	FR_CHANNEL_A	Cluster uses channel A	
	FR_CHANNEL_AB	Cluster uses channel A and B	
	FR_CHANNEL_B	Cluster uses channel B	
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		

SWS Item	**[ECUC_Fr_00405]**		
Parameter Name	FrPClusterDriftDamping		
Parent Container	FrController		
Description	Local cluster drift damping factor used for rate correction [Microticks]. Remark: Upper limit 10 for FlexRay Protocol 3.0 compliance.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 20		
Default value	–		
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		

SWS Item	**[ECUC_Fr_00428]**	
Parameter Name	FrPdAcceptedStartupRange	
Parent Container	FrController	
Description	Expanded range of measured clock deviation allowed for startup frames during integration [Microticks]. Remark: Upper limit 1875 for FlexRay Protocol 2.1 Rev A compliance. Remark: Lower limit 29 for FlexRay Protocol 3.0 compliance.	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	0 .. 2743	
Default value	–	

▽

△

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		

SWS Item	**[ECUC_Fr_00406]**		
Parameter Name	FrPDecodingCorrection		
Parent Container	FrController		
Description	Value used by the receiver to calculate the difference between primary time reference point and secondary time reference point [Microticks]. Remark: Lower limit 14 for Flex Ray Protocol 2.1 Rev. A compliance. Upper limit 136 for FlexRay Protocol 3.0 compliance.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	12 .. 143		
Default value	–		
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		

SWS Item	**[ECUC_Fr_00407]**		
Parameter Name	FrPDelayCompensationA		
Parent Container	FrController		
Description	Value used to compensate for reception delays on the indicated channel. This covers assumed propagation delay up to cPropagationDelayMax for microticks in the range of 0.0125us to 0.05us [Microticks]. Remark: Lower limit 4 for FlexRay Protocol 3.0 compliance. Remark: Upper limit 200 for FlexRay Protocol 2.1 Rev A compliance.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 211		
Default value	–		
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		

SWS Item	**[ECUC_Fr_00408]**
Parameter Name	FrPDelayCompensationB
Parent Container	FrController
Description	Value used to compensate for reception delays on the indicated channel. This covers assumed propagation delay up to cPropagationDelayMax for microticks in the range of 0.0125us to 0.05us [Microticks]. Remark: Lower limit 4 for FlexRay Protocol 3.0 compliance. Remark: Upper limit 200 for FlexRay Protocol 2.1 Rev A compliance.
Multiplicity	1
Type	EcucIntegerParamDef

▽

△

Range	0 .. 211		
Default value	–		
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		

SWS Item	**[ECUC_Fr_00429]**		
Parameter Name	FrPdListenTimeout		
Parent Container	FrController		
Description	Value for the startup listen timeout and wakeup listen timeout. Although this is a node local parameter, the real time equivalent of this value should be the same for all nodes in the cluster [Microticks]. Remark: Lower limit 1926 for FlexRay Protocol 3.0 compliance. Upper limit 1283846 for FlexRay Protocol 2.1 Rev. A compliance.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1284 .. 2567692		
Default value	–		
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		

SWS Item	**[ECUC_Fr_00431]**		
Parameter Name	FrPdMicrotick		
Parent Container	FrController		
Description	Duration of a microtick. Remark: Allowed range T12_5NS, T25NS, T50NS for FlexRay Protocol 3.0 compliance.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	T100NS	100 ns	
	T12_5NS	12.5 ns	
	T200NS	200 ns	
	T25NS	25 ns	
	T50NS	50 ns	
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		

SWS Item	**[ECUC_Fr_00448]**
Parameter Name	FrPExternalSync
Parent Container	FrController

▽

△

Description	Flag indicating whether the node is externally synchronized (operating as time gateway sink in an TT-E cluster) or locally synchronized. If FrPExternalSync is set to 'true' then FrPTwoKeySlotMode must also be set to 'true'. Remarks: Set to 'false' for FlexRay Protocol 2.1 Rev. A compliance.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	–		
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		

SWS Item	**[ECUC_Fr_00447]**		
Parameter Name	FrPFallBackInternal		
Parent Container	FrController		
Description	Flag indicating whether a time gateway sink node will switch to local clock operation when synchronization with the time gateway source node is lost (FrPFallBackInternal = true) or will instead go to POC:ready (FrPFallBackInternal =false). Remarks: Set to 'false' for FlexRay Protocol 2.1 Rev. A compliance.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	–		
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		

SWS Item	**[ECUC_Fr_00411]**		
Parameter Name	FrPKeySlotId		
Parent Container	FrController		
Description	ID of the key slot, i.e., the slot used to transmit the startup frame, sync frame, or designated key slot frame. If this parameter is set to zero the node does not have a key slot. For Fr3.0: if the value is not provided in System Description it shall be configured to 0. For Fr2.1: if the value is not provided in System Description it is driver implementation specific which value to configure.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 1023		
Default value	–		
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		

SWS Item	[ECUC_Fr_00425]		
Parameter Name	FrPKeySlotOnlyEnabled		
Parent Container	FrController		
Description	Flag indicating whether or not the node shall enter key slot only mode following startup. Remarks: This parameter maps to FlexRay Protocol 2.1 Rev. A parameter pSingleSlot Enabled.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	–		
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		

SWS Item	[ECUC_Fr_00412]		
Parameter Name	FrPKeySlotUsedForStartup		
Parent Container	FrController		
Description	Flag indicating whether the key slot is used to transmit a startup frame. If FrPKeySlot UsedForStartup is set to true then FrPKeySlotUsedForSync must also be set to true. If FrPTwoKeySlotMode is set to true then both FrPKeySlotUsedForSync and FrPKeySlot UsedForStartup must also be set to true.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	–		
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		

SWS Item	[ECUC_Fr_00413]		
Parameter Name	FrPKeySlotUsedForSync		
Parent Container	FrController		
Description	Flag indicating whether the key slot is used to transmit a sync frame. If FrPKeySlot UsedForStartup is set to true then FrPKeySlotUsedForSync must also be set to true. If FrPTwoKeySlotMode is set to true then both FrPKeySlotUsedForSync and FrPKeySlot UsedForStartup must also be set to true.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	–		
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		

SWS Item	**[ECUC_Fr_00414]**		
Parameter Name	FrPLatestTx		
Parent Container	FrController		
Description	Number of the last minislot in which a frame transmission can start in the dynamic segment. Remark: Upper limit 7980 for FlexRay Protocol 2.1 Rev A compliance.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 7988		
Default value	–		
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		

SWS Item	**[ECUC_Fr_00415]**		
Parameter Name	FrPMacroInitialOffsetA		
Parent Container	FrController		
Description	Integer number of macroticks between the static slot boundary and the following macrotick boundary of the secondary time reference point based on the nominal macrotick duration [Macroticks].		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	2 .. 68		
Default value	–		
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		

SWS Item	**[ECUC_Fr_00416]**		
Parameter Name	FrPMacroInitialOffsetB		
Parent Container	FrController		
Description	Integer number of macroticks between the static slot boundary and the following macrotick boundary of the secondary time reference point based on the nominal macrotick duration [Macroticks].		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	2 .. 68		
Default value	–		
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		

SWS Item	**[ECUC_Fr_00417]**		
Parameter Name	FrPMicroInitialOffsetA		
Parent Container	FrController		
Description	Number of microticks between the secondary time reference point and the macrotick boundary immediately following the secondary time reference point. The parameter depends on FrPDelayCompensationA and therefore it has to be set independently for each channel [Microticks].		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 239		
Default value	–		
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		

SWS Item	**[ECUC_Fr_00418]**		
Parameter Name	FrPMicroInitialOffsetB		
Parent Container	FrController		
Description	Number of microticks between the secondary time reference point and the macrotick boundary immediately following the secondary time reference point. The parameter depends on FrPDelayCompensationB and therefore it has to be set independently for each channel [Microticks].		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 239		
Default value	–		
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		

SWS Item	**[ECUC_Fr_00419]**		
Parameter Name	FrPMicroPerCycle		
Parent Container	FrController		
Description	Nominal number of microticks in the communication cycle of the local node. If nodes have different microtick durations this number will differ from node to node [Microticks]. Remark: Lower limit 960 for FlexRay Protocol 3.0 compliance. Upper limit 640000 for FlexRay Protocol 2.1 Rev A compliance.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	640 .. 1280000		
Default value	–		
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		

SWS Item	**[ECUC_Fr_00444]**		
Parameter Name	FrPNmVectorEarlyUpdate		
Parent Container	FrController		
Description	Flag indicating when the update of the Network Management Vector in the CHI shall take place. If FrPNmVectorEarlyUpdate is set to false, the update shall take place after the NIT. If FrPNmVectorEarlyUpdate is set to true, the update shall take place after the end of the static segment. Remarks: Set to 'false' for FlexRay Protocol 2.1 Rev. A compliance.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	–		
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		

SWS Item	**[ECUC_Fr_00421]**		
Parameter Name	FrPOffsetCorrectionOut		
Parent Container	FrController		
Description	Magnitude of the maximum permissible offset correction value [Microticks]. Remark: Upper limit 15567 for FlexRay Protocol 2.1 Rev A compliance. Remark: Lower limit 15 for FlexRay Protocol 3.0 compliance.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	13 .. 16082		
Default value	–		
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		

SWS Item	**[ECUC_Fr_00450]**		
Parameter Name	FrPOffsetCorrectionStart		
Parent Container	FrController		
Description	Start of the offset correction phase within the NIT, expressed as the number of macroticks from the start of cycle [Macroticks]. Remark: This parameter maps to Flex Ray Protocol 2.1 Rev. A parameter gOffsetCorrectionStart. Remark: Lower limit 9 for FlexRay Protocol 2.1 Rev A compliance.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	7 .. 15999		
Default value	–		
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		

SWS Item	**[ECUC_Fr_00422]**		
Parameter Name	FrPPayloadLengthDynMax		
Parent Container	FrController		
Description	Maximum payload length for dynamic frames [16 bit words].		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 127		
Default value	–		
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		

SWS Item	**[ECUC_Fr_00423]**		
Parameter Name	FrPRateCorrectionOut		
Parent Container	FrController		
Description	Magnitude of the maximum permissible rate correction value and the maximum drift offset between two nodes operating with unsynchronized clocks for one communication cycle [Microticks]. Remarks: This parameter maps to FlexRay Protocol 2.1 Rev. A parameter pdMaxDrift. Lower limit 3 for FlexRay Protocol 3.0 compliance. Upper limit 1923 for FlexRay Protocol 2.1 Rev A compliance.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	2 .. 3846		
Default value	–		
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		

SWS Item	**[ECUC_Fr_00424]**		
Parameter Name	FrPSamplesPerMicrotick		
Parent Container	FrController		
Description	Number of samples per microtick. Remark: Allowed range N1SAMPLES, N2SAMPLES for FlexRay Protocol 3.0 compliance.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	N1SAMPLES	1 sample	
	N2SAMPLES	2 samples	
	N4SAMPLES	4 samples	
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		

SWS Item	**[ECUC_Fr_00445]**		
Parameter Name	FrPSecondKeySlotId		
Parent Container	FrController		
Description	ID of the second key slot, in which a second startup frame shall be sent when operating as a coldstart node in a TT-L or TT-D cluster. If this parameter is set to zero the node does not have a second key slot. Remark: Set to 0 for FlexRay Protocol 2.1 Rev A compliance.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 1023		
Default value	–		
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		

SWS Item	**[ECUC_Fr_00446]**		
Parameter Name	FrPTwoKeySlotMode		
Parent Container	FrController		
Description	Flag indicating whether node operates as a coldstart node in a TT-E or TT-L cluster. If p TwoKeySlotMode is set to true then both pKeySlotUsedForSync and pKeySlotUsedFor Startup must also be set to true. If pExternalSync is set to true then pTwoKeySlotMode must also be set to true. Remark: Set to false for FlexRay Protocol 2.1 Rev A compliance.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	–		
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		

SWS Item	**[ECUC_Fr_00426]**		
Parameter Name	FrPWakeupChannel		
Parent Container	FrController		
Description	Channel used by the node to send a wakeup pattern. FrPWakeupChannel must be selected from among the channels configured by FrPChannels.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	FR_CHANNEL_A	–	
	FR_CHANNEL_B	–	
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		

SWS Item	[ECUC_Fr_00427]		
Parameter Name	FrPWakeupPattern		
Parent Container	FrController		
Description	Number of repetitions of the wakeup symbol that are combined to form a wakeup pattern when the node enters the POC:wakeup send state. Remark: Lower limit 2 for FlexRay Protocol 2.1 Rev A compliance.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 63		
Default value	–		
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		

SWS Item	[ECUC_Fr_00458]		
Parameter Name	FrCtrlEcucPartitionRef		
Parent Container	FrController		
Description	Maps one single Flexray controller to zero or one ECUC partitions. The ECUC partition referenced is a subset of the ECUC partitions where the Flexray driver is mapped to.		
Multiplicity	0..1		
Type	Reference to EcucPartition		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	Pre-compile time	X	All Variants
	Link time	–	
	Post-build time	–	
Value Configuration Class	Pre-compile time	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: ECU		

Included Containers		
Container Name	**Multiplicity**	**Scope / Dependency**
FrAbsoluteTimer	1..*	Specifies the absolute timer configuration parameters of the Fr.
FrControllerDemEventParameterRefs	0..1	Container for the references to DemEventParameter elements which shall be invoked using the API Dem_SetEventStatus in case the corresponding error occurs. The EventId is taken from the referenced DemEventParameter's DemEventId symbolic value. The standardized errors are provided in this container and can be extended by vendor-specific error references.
FrFifo	0..*	One First In First Out (FIFO) queued receive structure, defining the admittance criteria to the FIFO, and mandating the ability to admit messages into the FIFO based on Message Id filtering criteria.

[SWS_Fr_CONSTR_00002] ⌈The ECUC partitions referenced by `FrCtrlEcucPartitionRef` shall be a subset of the ECUC partitions referenced by `FrEcucPartitionRef`.⌋*()*

[SWS_Fr_CONSTR_00003] ⌈`FrController` and `FrTrcvChannel` of one communication channel shall all reference the same ECUC partition.⌋*()*

[SWS_Fr_CONSTR_00004] ⌈If `FrEcucPartitionRef` references one or more ECUC partitions, `FrCtrlEcucPartitionRef` shall have a multiplicity of one and reference one of these ECUC partitions as well.⌋*()*

10.2.4 FrAbsoluteTimer

SWS Item	**[ECUC_Fr_00432]**
Container Name	FrAbsoluteTimer
Parent Container	FrController
Description	Specifies the absolute timer configuration parameters of the Fr.
Configuration Parameters	

SWS Item	**[ECUC_Fr_00433]**		
Parameter Name	FrAbsTimerIdx		
Parent Container	FrAbsoluteTimer		
Description	Contains the index of an absolute timer contained in Fr on a certain FlexRay CC.		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 254		
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 withAuto = true		

No Included Containers

10.2.5 FrControllerDemEventParameterRefs

SWS Item	**[ECUC_Fr_00452]**
Container Name	FrControllerDemEventParameterRefs
Parent Container	FrController
Description	Container for the references to DemEventParameter elements which shall be invoked using the API Dem_SetEventStatus in case the corresponding error occurs. The EventId is taken from the referenced DemEventParameter's DemEventId symbolic value. The standardized errors are provided in this container and can be extended by vendor-specific error references.
Configuration Parameters	

SWS Item	[ECUC_Fr_00005]		
Parameter Name	FR_E_CTRL_TESTRESULT		
Parent Container	FrControllerDemEventParameterRefs		
Description	Reference to DEM event Id that is reported for FlexRay controller hardware test failure. If this parameter is not configured, no event reporting happens.		
Multiplicity	0..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

10.2.6 FrFifo

SWS Item	[ECUC_Fr_00009]
Container Name	FrFifo
Parent Container	FrController
Description	One First In First Out (FIFO) queued receive structure, defining the admittance criteria to the FIFO, and mandating the ability to admit messages into the FIFO based on Message Id filtering criteria.
Configuration Parameters	

SWS Item	[ECUC_Fr_00006]		
Parameter Name	FrAdmitWithoutMessageId		
Parent Container	FrFifo		
Description	Determines whether or not frames received in the dynamic segment that don't contain a message ID will be admitted into the FIFO.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	–		
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		

SWS Item	**[ECUC_Fr_00007]**		
Parameter Name	FrBaseCycle		
Parent Container	FrFifo		
Description	FIFO cycle counter acceptance criteria.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 63		
Default value	–		
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		

SWS Item	**[ECUC_Fr_00449]**		
Parameter Name	FrChannels		
Parent Container	FrFifo		
Description	FIFO channel admittance criteria.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	FR_CHANNEL_A	Frames received on channel A	
	FR_CHANNEL_AB	Frames received on channel A and B	
	FR_CHANNEL_B	Frames received on channel B	
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		

SWS Item	**[ECUC_Fr_00008]**		
Parameter Name	FrCycleRepetition		
Parent Container	FrFifo		
Description	FIFO cycle counter acceptance criteria. Valid values are 1,2,4,5,8,10,16,20,32,40,50,64. Remark: Values 1,2,4,8,16,32,64 are valid only for FlexRay Protocol 2.1 Rev A compliance.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 64		
Default value	–		
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		

SWS Item	[ECUC_Fr_00010]		
Parameter Name	FrFifoDepth		
Parent Container	FrFifo		
Description	FrFifoDepth configures the maximum number of rx-frames which can be contained in the FIFO.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 2048		
Default value	–		
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		

SWS Item	[ECUC_Fr_00011]		
Parameter Name	FrMsgIdMask		
Parent Container	FrFifo		
Description	FIFO message identifier acceptance criteria (Mask filter).		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	–		
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		

SWS Item	[ECUC_Fr_00012]		
Parameter Name	FrMsgIdMatch		
Parent Container	FrFifo		
Description	FIFO message identifier acceptance criteria (Match filter).		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	–		
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		

Included Containers		
Container Name	**Multiplicity**	**Scope / Dependency**
FrRange	1..*	FIFO Frame Id range acceptance criteria.

10.2.7 FrRange

SWS Item	**[ECUC_Fr_00013]**
Container Name	FrRange
Parent Container	FrFifo
Description	FIFO Frame Id range acceptance criteria.
Configuration Parameters	

SWS Item	**[ECUC_Fr_00014]**		
Parameter Name	FrRangeMax		
Parent Container	FrRange		
Description	Last FrameId of this range that will be accepted by the FIFO.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 2047		
Default value	–		
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		

SWS Item	**[ECUC_Fr_00015]**		
Parameter Name	FrRangeMin		
Parent Container	FrRange		
Description	First FrameId of this range that will be accepted by the FIFO.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 2047		
Default value	–		
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		

No Included Containers

10.2.8 FrMultipleConfiguration

SWS Item	**[ECUC_Fr_00397]**
Container Name	FrMultipleConfiguration
Parent Container	Fr

▽

△

Description	This container contains the configuration parameters and sub containers of the AUTOSAR Fr module.
Configuration Parameters	

Included Containers		
Container Name	**Multiplicity**	**Scope / Dependency**
FrController	1..*	Container to hold multiple configuration sets.

10.3 Published Information

For details refer to the chapter 10.3 “Published Information” in SWS_BSWGeneral.

A Not applicable requirements

[SWS_Fr_NA_00602] ⌈These requirements are not applicable to this specification.⌋*(SRS_BSW_00306, SRS_BSW_00312, SRS_BSW_00314, SRS_BSW_00325, SRS_BSW_00327, SRS_BSW_00328, SRS_BSW_00330, SRS_BSW_00331, SRS_BSW_00333, SRS_BSW_00335, SRS_BSW_00341, SRS_BSW_00343, SRS_BSW_00344, SRS_BSW_00359, SRS_BSW_00360, SRS_BSW_00373, SRS_BSW_00375, SRS_BSW_00377, SRS_BSW_00386, SRS_BSW_00410, SRS_BSW_00415, SRS_BSW_00416, 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_BSW_00437, SRS_BSW_00439, SRS_BSW_00440, SRS_BSW_00447, SRS_BSW_00449, SRS_BSW_00450, SRS_BSW_00005, SRS_BSW_00006, SRS_BSW_00009, SRS_BSW_00010, SRS_BSW_00161, SRS_BSW_00162, SRS_BSW_00164, SRS_BSW_00168, SRS_BSW_00170, SRS_BSW_00172, SRS_Fr_05000, SRS_Fr_05001, SRS_Fr_05002, SRS_Fr_05033, SRS_Fr_05053, SRS_Fr_05052)*