Document Title	Specification of FlexRay Driver
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	026
Document Classification	Standard

Document Version	2.5.0
Document Status	Final
Part of Release	4.0
Revision	3

Document Change History

Date	Version	Changed by	Change Description
16.12.2011	2.5.0	AUTOSAR Administration	• Small corrections and clarification on existing features
18.10.2010	2.4.0	AUTOSAR Administration	• New service for reading the FlexRay configuration parameters at runtime • Update of configuration parameters according to the FlexRay Protocol Specification 3.0
30.11.2009	2.3.0	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
23.06.2008	2.2.1	AUTOSAR Administration	Legal disclaimer revised
28.11.2007	2.2.0	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

Document Change History			
Date	**Version**	**Changed by**	**Change Description**
31.01.2007	2.1.0	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
28.06.2006	2.0.0	AUTOSAR Administration	Second release
04.08.2005	1.0.0	AUTOSAR Administration	Initial release

Disclaimer

Advice for users

AUTOSAR Specification Documents may contain exemplary items (exemplary reference models, "use cases", and/or references to exemplary technical solutions, devices, processes or software).

Any such exemplary items are contained in the Specification Documents for illustration purposes only, and they themselves are not part of the AUTOSAR Standard. Neither their presence in such Specification Documents, nor any later documentation of AUTOSAR conformance of products actually implementing such exemplary items, imply that intellectual property rights covering such exemplary items are licensed under the same rules as applicable to the AUTOSAR Standard.

Table of Contents

1 Introduction and functional overview

The FlexRay Driver (Fr) abstracts the hardware related implementation details of specific FlexRay Communication Controllers (CC). This specification basically relies on FlexRay CCs compliant to the FlexRay specification [12]. Additionally older FlexRay controllers compliant to FlexRay specification [13] 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 depicts the basic structure of the FlexRay stack. One FrIf accesses several CCs using one or several FlexRay Drivers.

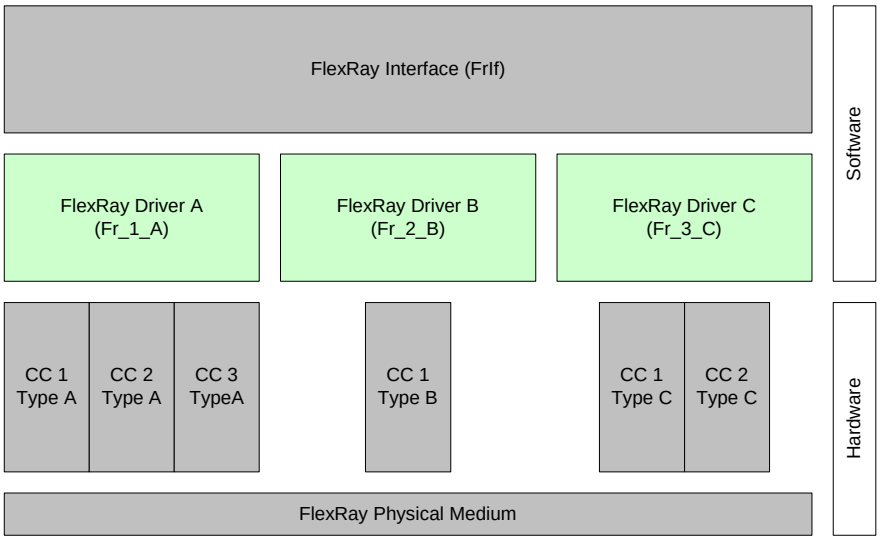

Figure 1: FlexRay stack module overview

2 Acronyms and abbreviations

Abbreviation:	*Description:*
API	Application Programming Interface
AUTOSAR	Automotove Open Systems Architecture
BSW	Basic Software
DEM/Dem	Autosar Module: Diagnostic Event Manager
DET/Det	Autosar Module: Development 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
HIS	Herstellerinitiative Software (see http://www.automotive-his.de/)
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 [12] 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 [12] 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 [12] for details).
XML	Extensible Markup Language

2.1 Glossary of terms

Term:	*Definition:*
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))

3 Related documentation

3.1 Input documents

[1] List of Basic Software Modules,
AUTOSAR_TR_BSWModuleList.pdf

[2] Layered Software Architecture,
AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf

[3] General Requirements on Basic Software Modules,
AUTOSAR_SRS_BSWGeneral.pdf

[4] Specification of ECU Configuration,
AUTOSAR_TPS_ECUConfiguration.pdf

[5] Specification of Standard Types,
AUTOSAR_SWS_StandardTypes.pdf

[6] Specification of Platform Types,
AUTOSAR_SWS_PlatformTypes.pdf

[7] Specification of FlexRay Interface,
AUTOSAR_SWS_FlexRayInterface.pdf

[8] Specification of FlexRay Transceiver Driver,
AUTOSAR_SWS_FlexRayTransceiver.pdf

[9] Specification of BSW Scheduler,
AUTOSAR_SWS_BSW_Scheduler.pdf

[10] Specification of Memory Mapping
AUTOSAR_SWS_MemoryMapping.pdf

[11] AUTOSAR Basic Software Module Description Template
AUTOSAR_TPS_BSWModuleDescriptionTemplate.pdf

3.2 Related standards and norms

[12] 2009, FlexRay Consortium, FlexRay Communication Systems Protocol Specification, Version 3.0 (unreleased draft).

[13] 2005, FlexRay Consortium, FlexRay Communication Systems Protocol Specification, Version 2.1 Revision A.

[14] 2006, Gemeinsames Subset der MISRA C Guidelines, Version 2.0, http://www.automotive-his.de/download/HIS_SubSet_MISRA_C_2.0.pdf

4 Constraints and assumptions

4.1 Limitations

[FR449]「 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 [12]) 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:

- **[FR451]** 「 The Fr module shall use the Development Error Tracer (DET) to report development errors. 」()
- **[FR452]** 「 The Fr module shall use the Diagnostic Event Manager (DEM) to report diagnostic-relevant events and states. 」()
- **[FR453]** 「 The Fr module shall use the BSW Scheduler mechanisms for data consistency when required. 」()

Other Module dependencies:

- **[FR454]** 「 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.

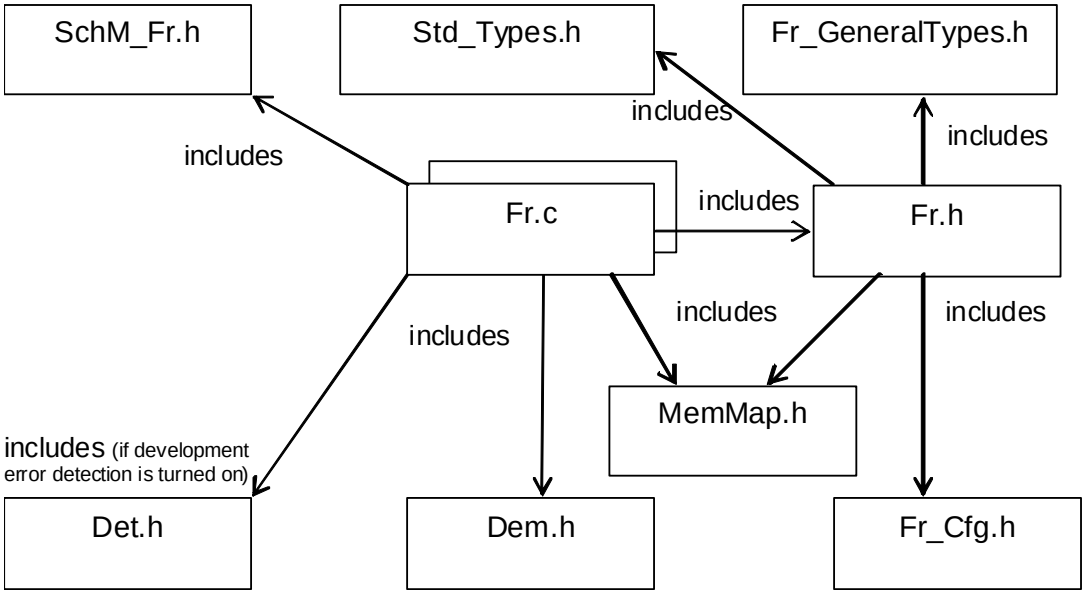

Figure 2 FlexRay Driver header file structure

Figure 2 shows the relation of source code files and header files that shall be used for implementation.

[FR459] ⌈ The *Fr.h* file shall include *Fr_GeneralTypes.h*. ⌋ ()

[FR460] ⌈ The *Fr.h* file shall include *Std_Types.h*. ⌋ ()

[FR461] ⌈ The *Fr.h* file shall include *Fr_Cfg.h*. ⌋ ()

[FR462] ⌈ The *Fr.c* file shall include *Fr.h*. ⌋ ()

[FR074] ⌈ All files related to the Fr module shall follow the naming convention *Fr[_<description>].<extension>* . ⌋ (BSW00300)

5.1.1 Code file structure

[FR115] ⌈ The implementation of the Fr module shall contain the following source code files:

- *Fr.c* for the general source code or *Fr_<purpose>.c* if the implementation is separated into several source code files. ⌋ (BSW00346, BSW00380, BSW158)

[FR116] ⌈ The code file structure shall not be completely defined within this specification. ⌋ (BSW00346, BSW00380, BSW158)

[FR463] ⌈ At this point it shall be pointed out that the code-file structure shall include the following files named:

- Fr_PBcfg.c – for post build time configurable parameters. 」()

5.1.2 Header file structure

[FR101]「 The implementation of the Fr module shall provide the header file *Fr.h*, which is the main module interface file. 」(BSW00302)

[FR464]「 The file *Fr.h* shall contain all types and function prototypes required by the Fr module's environment. 」()

[FR063]「 The implementation of the Fr module shall provide the header file *Fr_Cfg.h*, which shall contain the pre-compile-time configuration parameters. 」(BSW00381, BSW00412, BSW00419)

[FR117]「 *Fr_GeneralTypes.h* shall contain all types and constants that are shared among the AUTOSAR FlexRay modules Fr, FrIf and FrTrcv. 」()

[FR455]「 The integrator of the FlexRay modules shall provide the file *Fr_GeneralTypes.h*. 」()

[FR071]「 The Fr module shall include the *Dem.h* file. 」(BSW00409)

[FR456]「 The above inclusion also ensures that APIs to report errors as well the required Event Id symbols are included. 」()

[FR457]「 This specification defines the name of the Event Id symbols, which are provided by XML to the DEM configuration tool.
The DEM configuration tool assigns ECU dependent values to the Event Id symbols and publishes the symbols in *Dem_IntErrId.h*. 」()

[FR118]「 The Fr module source code file(s) shall include *Det.h* file. This inclusion ensures the includsion of APIs to report development errors. 」()

[FR111]「 The Fr module source code file(s) shall include *SchM_Fr.h* if data consistency mechanisms of the BSW scheduler are required as described in [9]. 」(BSW00435)

[FR112]「 All Fr module files shall include *MemMap.h* and apply the memory mapping abstraction mechanisms as specified by [10]. 」(BSW00436)

6 Requirements traceability

Document: AUTOSAR requirements on Basic Software, general

Requirement		***Satisfied by***
BSW003	Version identification	FR080
BSW00300	Module naming convention	FR074
BSW00301	Limit imported information	Figure 2
BSW00302	Limit exported information	FR101
BSW00304	AUTOSAR integer data types	Chapter 0
BSW00305	Self-defined data types naming convention	FR077, FR110
BSW00306	Avoid direct use of compiler and platform specific keywords	not applicable for Fr SWS
BSW00307	Global variables naming convention	FR098
BSW00308	Definition of global data	FR102
BSW00309	Global data with read-only constraint	FR085
BSW00310	API naming convention	Chapter 0
BSW00312	Shared code shall be reentrant	not applicable for Fr SWS
BSW00314	Separation of interrupt frames and service routines	not applicable for Fr SWS
BSW00318	Format of module version numbers	FR649
BSW00321	Enumeration of module version numbers	FR650
BSW00323	API parameter checking	Chapter 0
BSW00325	Runtime of interrupt service routines	not applicable for Fr SWS
BSW00326	Transition from ISRs to OS tasks	not applicable for Fr SWS
BSW00327	Error values naming convention	FR078
BSW00328	Avoid duplication of code	not applicable for Fr SWS
BSW00329	Avoidance of generic interfaces	not applicable for Fr SWS
BSW00330	Usage of macros / inline functions instead of functions	not applicable for Fr SWS
BSW00331	Separation of error and status values	not applicable for Fr SWS
BSW00333	Documentation of callback function context	not applicable for Fr SWS
BSW00334	Provision of XML file	FR080
BSW00335	Status values naming convention	not applicable for Fr SWS
BSW00336	Shutdown interface	FR014
BSW00337	Classification of errors	Chapter 7.7
BSW00338	Detection and Reporting of development errors	FR026, FR127
BSW00339	Reporting of production relevant error status	FR028
BSW00341	Microcontroller compatibility documentation	not applicable for Fr SWS
BSW00342	Usage of source code and object code	FR097
BSW00343	Specification and configuration of time	not applicable for Fr SWS
BSW00344	Reference to link-time configuration	not applicable for Fr SWS
BSW00345	Pre-compile-time configuration	FR027
BSW00346	Basic set of module files	FR115, FR116
BSW00347	Naming separation of different instances of BSW drivers	FR076
BSW00348	Standard type header	FR099, Figure 2
BSW00350	Development error detection keyword	FR026, FR029,
BSW00353	Platform specific type header	FR099, Figure 2
BSW00355	Do not redefine AUTOSAR integer data types	Chapter 8.3
BSW00357	Standard API return type	Chapter 0
BSW00358	Return type of init() functions	FR032
BSW00359	Return type of callback functions	not applicable for Fr SWS
BSW00360	Parameters of callback functions	not applicable for Fr SWS
BSW00361	Compiler specific language extension header	FR099, Figure 2
BSW00369	Do not return development error codes via API	Chapter 0

BSW00370	Separation of callback interface from API	not applicable for Fr SWS
BSW00371	Do not pass function pointers via API	not applicable for Fr SWS
BSW00373	Main processing function naming convention	not applicable for Fr SWS
BSW00374	Module vendor identification	FR080
BSW00375	Notification of wake-up reason	not applicable for Fr SWS
BSW00376	Return type and parameters of main processing functions	not applicable for Fr SWS
BSW00377	Module specific API return types	not applicable for Fr SWS
BSW00378	AUTOSAR boolean type	Chapter 8.3 Chapter 0
BSW00379	Module identification	FR080
BSW00380	Separate C-Files for configuration parameters	FR115, FR116
BSW00381	Separate configuration header file for pre-compile time parameters	FR063
BSW00383	List dependencies of configuration files	Chapter 5
BSW00384	List dependencies to other modules	Chapter 5
BSW00385	List possible error notifications	Chapter 7.7
BSW00386	Configuration for detecting an error	not applicable for Fr SWS
BSW00387	Specify the configuration class of callback function	not applicable for Fr SWS
BSW00388	Introduce containers	FR067
BSW00389	Containers shall have names	Chapter 10.2
BSW00390	Parameter content shall be unique within the module	Chapter 10.2
BSW00391	Parameter shall have unique names	Chapter 10.2
BSW00392	Parameters shall have a type	Chapter 10.2
BSW00393	Parameters shall have a range	Chapter 10.2
BSW00394	Specify the scope of the parameters	Chapter 10.2
BSW00395	List the required parameters (per parameter)	Chapter 10.2
BSW00396	Configuration classes	Chapter 10.2
BSW00397	Pre-compile-time parameters	Chapter 10.2
BSW00398	Link-time parameters	Chapter 10.2
BSW00399	Loadable Post-build time parameters	Chapter 10.2
BSW004	Version check	FR030
BSW00400	Selectable Post-build time parameters	Chapter 10.2
BSW00401	Documentation of multiple instances of configuration parameters	Chapter 10.2
BSW00402	Published information	FR649
BSW00404	Reference to post build time configuration	FR032, FR027
BSW00405	Reference to multiple configuration sets	FR032
BSW00406	Check module initialization	Chapter 0
BSW00407	Function to read out published parameters	FR070,
BSW00408	Configuration parameter naming convention	Chapter 10.2
BSW00409	Header files for production code error IDs	FR071
BSW00410	Compiler switches shall have defined values	not applicable for Fr SWS
BSW00411	Get version info keyword	FR070, FR340, FR341, FR342
BSW00412	Separate H-File for configuration parameters	FR063
BSW00413	Accessing instances of BSW modules	FR075
BSW00414	Parameter of init function	FR032
BSW00415	User dependent include files	not applicable for Fr SWS
BSW00416	Sequence of Initialization	not applicable for Fr SWS
BSW00417	Reporting of Error Events by Non-Basic Software	not applicable for Fr SWS
BSW00419	Separate C-Files for pre-compile time configuration parameters	FR063
BSW00422	Pre-de-bouncing of production relevant error status	not applicable for Fr SWS
BSW00423	Usage of SW-C template to describe BSW modules with AUTOSAR Interfaces	not applicable for Fr SWS
BSW00424	BSW main processing function task allocation	not applicable for Fr SWS
BSW00425	Trigger conditions for schedulable objects	not applicable for Fr SWS

BSW00426	Exclusive areas in BSW modules	not applicable for Fr SWS
BSW00427	ISR description for BSW modules	not applicable for Fr SWS
BSW00428	Execution order dependencies of main processing functions	not applicable for Fr SWS
BSW00429	Restricted BSW OS functionality access	not applicable for Fr SWS
BSW00432	Modules should have separate main processing functions for read/receive and write/transmit data path	not applicable for Fr SWS
BSW00433	Calling of main processing functions	not applicable for Fr SWS
BSW00435	Header File Structure for the Basic Software Scheduler	FR111
BSW00436	Module Header File Structure for the Memory Mapping	FR112
BSW00437	NoInit-Area in RAM	not applicable for Fr SWS
BSW00438	Post Build Configuration Data Structure	FR137
BSW00439	Declaration of interrupt handlers and ISRs	not applicable for Fr SWS
BSW00440	Function prototype for callback functions of AUTOSAR Services	not applicable for Fr SWS
BSW00441	Enumeration literals and #define naming convention	FR505, FR506, FR507, FR508, FR509, FR511, FR512, FR514
BSW00442	Debugging Support in Modules	FR625, FR626
BSW00443	Enabling / disabling defensive behavior of BSW	missing
BSW00444	Error reporting and logging for defensive behavior	missing
BSW00445	Protection against untimely call of BSW initialization	missing
BSW00446	Protection against untimely call of BSW de-initialization	missing
BSW00447	Standardizing Include file structure of BSW Modules Implementing Autosar Service	not applicable for Fr SWS
BSW00448	Module SWS shall not contain requirements from Other Modules	whole document
BSW00449	BSW Service APIs used by Autosar Application Software shall return a Std_ReturnType	not applicable for Fr SWS
BSW00450	Main Function Processing for Un-Initialized Modules	not applicable for Fr SWS
BSW005	No hard coded horizontal interfaces within MCAL	not applicable for Fr SWS
BSW006	Platform independency	not applicable for Fr SWS
BSW007	HIS MISRA C	FR073
BSW009	Module User Documentation	not applicable for Fr SWS
BSW010	Memory resource documentation	not applicable for Fr SWS
BSW101	Initialization interface	FR032
BSW158	Separation of configuration from implementation	FR115, FR116
BSW159	Tool-based configuration	Chapter 7.6
BSW160	Human-readable configuration data	FR072
BSW161	Microcontroller abstraction	not applicable for Fr SWS
BSW162	ECU layout abstraction	not applicable for Fr SWS
BSW164	Implementation of interrupt service routines	not applicable for Fr SWS
BSW167	Static configuration checking	Chapter 7.6
BSW168	Diagnostic Interface of SW components	not applicable for Fr SWS
BSW170	Data for reconfiguration of AUTOSAR SW-Components	not applicable for Fr SWS
BSW171	Configurability of optional functionality	Chapter 7.6
BSW172	Compatibility and documentation of scheduling strategy	not applicable for Fr SWS

Document: AUTOSAR requirements on FlexRay

Requirement		***Satisfied by***
BSW05000	Support of Synchronous SW Modules	not applicable for Fr SWS
BSW05001	Support of Asynchronous SW Modules	not applicable for Fr SWS
BSW05002	FlexRay Interface and FlexRay Driver as Only Necessarily Synchronous SW Modules	not applicable for Fr SWS
BSW05003	Support of Slot/Cycle Multiplexing	FR005, FR092, FR093, FR094
BSW05169	Avoid Timer Interrupts during Start-up	FR152

BSW05055	Avoid Timer Interrupts during Shutdown	FR106
BSW05064	Abstraction of FlexRay CC-specific Implementation	FR465, FR466
BSW05065	Number of FlexRay CCs per Driver	FR467
BSW05005	Support of Hardware FIFO Mechanism	FR593, FR594, FR595, FR596, FR597
BSW05006	Support of Message ID Filter Mechanism	FR593, FR006_Conf, FR011_Conf, FR012_Conf
BSW05007	Support of MTM Mode	FR083_Conf
BSW05024	Abstraction from CC Buffer Configuration	FR005, FR092, FR093, FR094, FR440, FR441, FR610
BSW05066	L-SDU-Based API	FR005, FR092, FR093, FR094, FR440, FR441, FR610
BSW05058	Configuration of FlexRay Driver at System Configuration Time	FR480
BSW05059	Transmit/Receive Buffer Configuration	FR148, FR524, FR539
BSW05116	Initialization of FlexRay CC	FR017
BSW05011	Initialize Low-Level Parameters	FR017
BSW05012	Initialize FlexRay CC Transmit/Receive Buffers	FR148
BSW05109	Start-up of a FlexRay CC	FR010
BSW05117	Sending of Wake-Up Pattern	FR009, FR091
BSW05120	Get FlexRay CC POC Status	FR012
BSW05121	Get FlexRay CC Sync State	FR021
BSW05106	Buffer Reconfiguration in Normal Active Mode	FR107
BSW05125	Interrupt Handling	FR034, FR036, FR035, FR108
BSW05114	Abortion of FlexRay CC Communication	FR011
BSW05115	Halt of FlexRay CC Communication	FR014, FR617
BSW05033	Tick Conversion	not applicable for Fr SWS
BSW05053	Cluster External Clock Synchronization	not applicable for Fr SWS
BSW05156	Controller External Clock Synchronization	FR041
BSW05044	Set Absolute Timer	FR033, FR095
BSW05046	Enable Absolute Alarms	FR034
BSW05047	Disable Absolute Alarms	FR035
BSW05048	Acknowledge Absolute Alarms	FR036
BSW05052	Get Cycle Length in Macroticks	not applicable for Fr SWS
BSW05019	Get FlexRay Global Time	FR042
BSW05072	FlexRay Time Services Access if CC is Out of Sync	FR044

7 Functional specification

7.1 General description

[FR465]「 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. 」(BSW05064)

[FR466]「 The Fr module maps abstract functional requests to sequences of CC specific hardware accesses. 」(BSW05064)

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.

[FR030]「 The Flexray Driver module shall perform Inter Module Checks to avoid integration of incompatible files. The imported included files shall be checked by preprocessing directives.
The following version numbers shall be verified:
- <MODULENAME>_AR_RELEASE_MAJOR_VERSION
- <MODULENAME>_AR_RELEASE_MINOR_VERSION
where <MODULENAME> is the module short name of the other (external) modules which provide header files included by the Flexray Driver module.
If the values are not identical to the expected values, an error shall be reported. 」(BSW004)

[FR029]「 In case development error detection is enabled for the Fr module, then the Fr module shall check the validity of API parameters and report detected errors to the DET. 」(BSW00350)

See chapter 8 for a detailed DET specification for each API function.

[FR073]「 The Fr module's implementation shall conform to the HIS subset of the MISRA C Standard (see document [14]). 」(BSW007)

[FR076]「 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. 」(BSW00347)

[FR097]「 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. 」(BSW00342)

[FR479]「 The rationale of FR097 is to allow object code module integration. 」()

[FR102]「 None of the Fr module's header files shall define global variables. 」(BSW00308)

[FR625]「 All data/variables that shall be debugged by AUTOSAR Debugging have to be defined as global variables. 」(BSW00442)

[FR626]「 The type definitions of debugging variables shall be accessable by Fr.h in a way that allows to calculate the size of the elements by C"-sizeof" and decode the structure elements. 」(BSW00442)

[FR106]「 The Fr module or the underlying hardware or both shall stop FlexRay timers in case of loss of synchronization. 」(BSW05055)

The implementation may assume that

- The Fr module's environment shall call all LPdu-based services (`Fr_TransmitTxLPdu()`, `Fr_ReceiveRxLPdu()`, `Fr_CheckLPduTxStatus()`, `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.

[FR103]「 If buffer reconfiguration is disabled (configuration parameter `FrBufferReconfig` is `false`), then buffers must not be configured by any Fr API function other than `Fr_ControllerInit()`. 」()

7.3 Indexing Scheme

Users of the Fr identify Fr resources by using an indexing scheme as depicted in Figure 3.

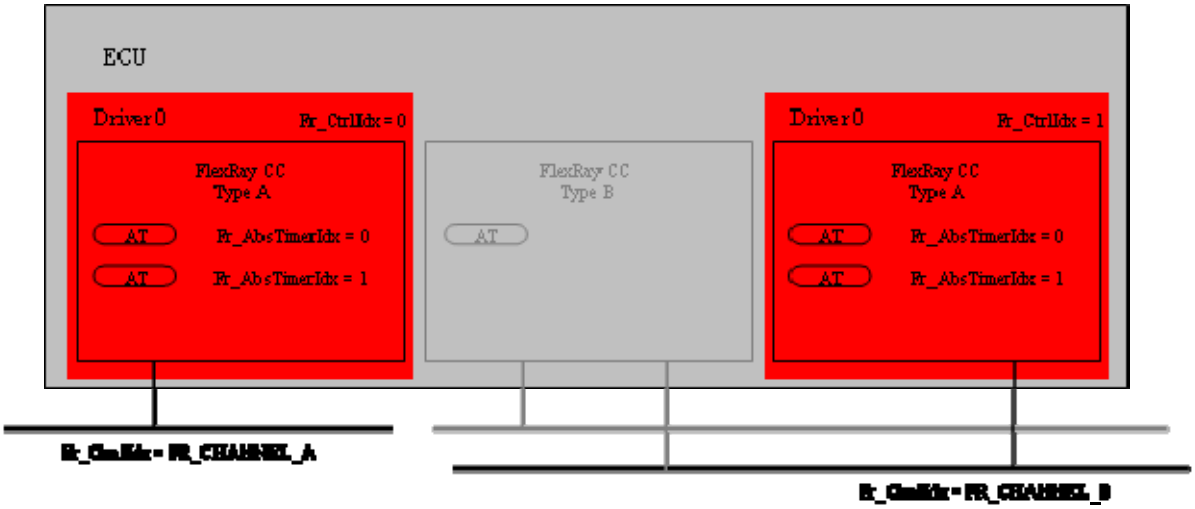

Figure 3 FlexRay Driver indexing scheme

The following Fr resources are available:

- **[FR075]** 「 CCs are identified via controller indices (`Fr_CtrlIdx`). 」(BSW00413)

  [FR467] 「 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. 」(BSW05065)

- **[FR344]** 「 For each FlexRay CC the connected channels are identified by channel indices (`Fr_ChnlIdx`). 」()

  [FR468] 「 A dedicated type that holds the enumerations `FR_CHANNEL_A`, `FR_CHANNEL_B` or `FR_CHANNEL_AB` represents the channel index. 」()

  [FR469] 「 Channel indices are only valid within a tuple <`Fr_CtrlIdx`, `Fr_ChnlIdx`>. 」()

- **[FR005]** 「 Each FlexRay frame processed by Fr API functions is identified by an LPdu index (`Fr_LPduIdx`). 」(BSW05003, BSW05024)

  [FR470] 「 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. 」()

  [FR471] 「 LPdu indices are only valid within a tuple <`Fr_CtrlIdx`, `Fr_LPduIdx`>. 」()

[FR472] ⌈ An `Fr_LPduIdx` uniquely identifies the following parameters of a FlexRay frame as a key: {Slot ID, Channel, cycle repetition, base cycle, transmit/receive}. ⌋ ()

- **[FR345]** ⌈ Each FlexRay CC contains absolute timers. Absolute FlexRay timers are identified via absolute timer indices (`Fr_AbsTimerIdx`). ⌋ ()

  [FR473] ⌈ 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. ⌋ ()

  [FR474] ⌈ Absolute timer indices are only valid within a tuple <`Fr_CtrlIdx`, `Fr_AbsTimerIdx`>. ⌋ ()

The FlexRay Driver numbering scheme (Figure 3) 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

[FR477] ⌈ 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 [12] for details). ⌋ ()

[FR478] ⌈ 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()` ⌋ ()

[FR438] ⌈ All API functions other than those listed above shall not change the POC state of the FlexRay CC.

Figure 4 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 4 – compare to [5]).

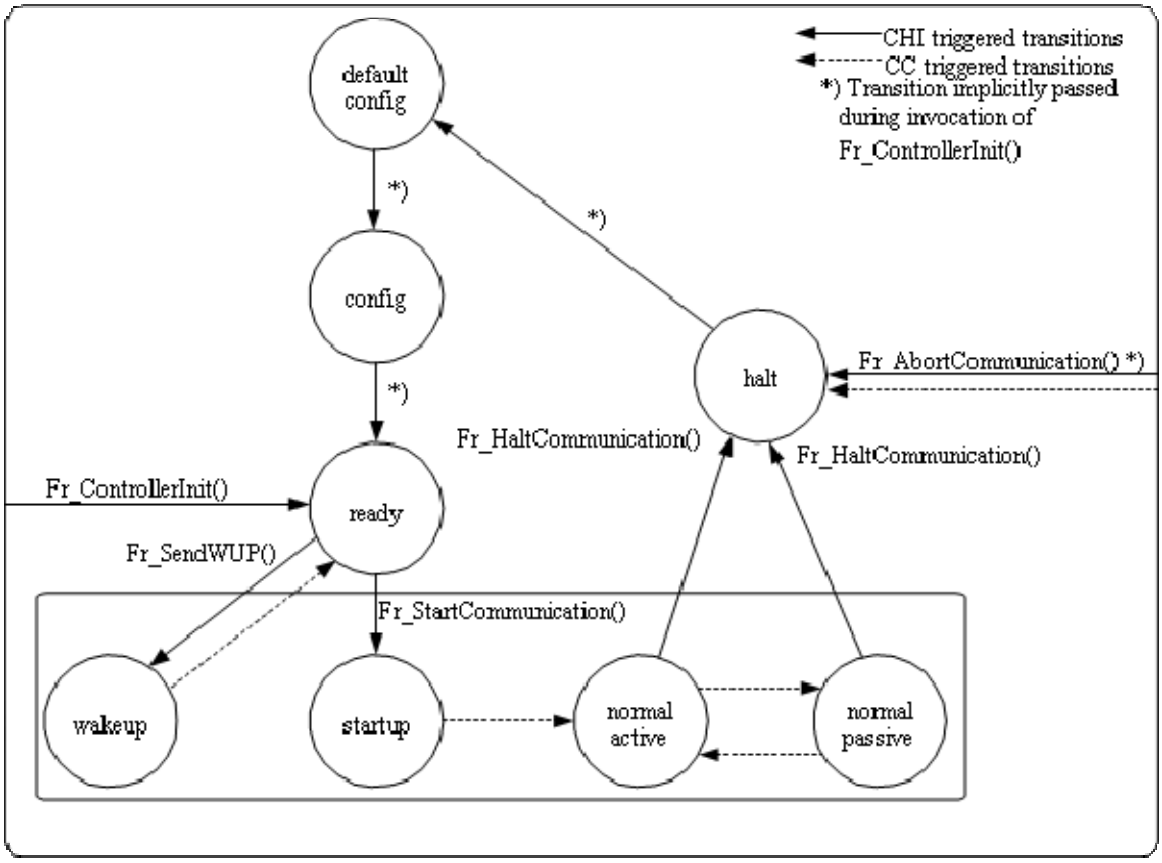

Figure 4 FlexRay Driver POC state machine control 」()

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.

[FR593] 「 The hardware receive-FIFO shall be used if the FIFO filter-criterias as configured can be applied to the hardware FIFO. 」(BSW05005, BSW05006)

[FR594] ⌈ All LPdus (as configured within FrIf) matching a receive-FIFO's filter-criteria shall be assigned to the respective receive-FIFO. ⌋ (BSW05005)

[FR595] ⌈ No specific buffers shall be assigned to LPdus that are assigned to a receive-FIFO. ⌋ (BSW05005)

[FR596] ⌈ 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. ⌋ (BSW05005)

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.

[FR597] ⌈ LPdus received via the FIFO shall be returned in the same order as they were received on the FlexRay network. ⌋ (BSW05005)

7.6 Configuration description

[FR080] ⌈ 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. ⌋ (BSW003, BSW00334, BSW00374, BSW00379)

[FR480] ⌈ 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. ⌋ (BSW05058)

[FR481] ⌈ 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. 」()

[FR482] 「 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 [7] for the FrIf configuration parameters). 」()

[FR483] 「 Based on those invocation times the Fr MCG might decide certain resource alignment optimizations for transmission and reception (share buffers among different LPdus). 」()

[FR003] 「 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. 」()

[FR624] 「 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. 」()

[FR484] 「 The Fr MCG shall have knowledge about the capabilities of the CC and the corresponding driver, therefore this tool is called driver dependent. 」()

[FR485] 「 If an Fr MCG is unable to map all required communication operations to the available resources, then it has to report that conflict[1]. 」()

[FR486] 「 The number of supported FlexRay CCs is defined at configuration time. 」()

[FR487] 「 The MCG shall ensure the consistency of the generated configuration. 」()

[FR027] 「 The Fr module shall support the pre-compile-time and post-build-time configuration classes. 」(BSW00338, BSW00345, BSW00404)

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

H[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 [12], but the device is is compliant to [13]).

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 [7]) shall be evaluated for Fr module configuration.

7.7 Error classification

[FR124] ⌈ Values for production code Event Ids are assigned externally by the configuration of the Dem. ⌋ ()

[FR125] ⌈ Development error values are of type `uint8`. ⌋ ()

The following errors and diagnostic events shall be detectable by the Fr module:

Description	*Relevance*	*Error / EventId name*	*Value*
FR025:parameter timer index exceeds number of available timers	Development	`FR_E_INV_TIMER_IDX`	0x01
FR488: invalid pointer in parameter list	Development	`FR_E_INV_POINTER`	0x02
FR489: parameter offset exceeds bounds	Development	`FR_E_INV_OFFSET`	0x03
FR490: invalid controller index	Development	`FR_E_INV_CTRL_IDX`	0x04
FR491: invalid channel index	Development	`FR_E_INV_CHNL_IDX`	0x05
FR492: parameter cycle exceeds 63	Development	`FR_E_INV_CYCLE`	0x06
FR494: Fr module was not initialized	Development	`FR_E_NOT_INITIALIZED`	0x08
FR495: Fr CC is not in the expected POC state.	Development	`FR_E_INV_POCSTATE`	0x09
FR496: Payload length parameter has an invalid value.	Development	`FR_E_INV_LENGTH`	0x0A
FR497: invalid LPdu index	Development	`FR_E_INV_LPDU_IDX`	0x0B
FR633: invalid FlexRay header CRC	Development	`FR_E_INV_HEADERCRC`	0x0C
invalid value passed as parameter Fr_ConfigParamIdx.	Development	`FR_E_INV_CONFIG_IDX`	0x0D
FR498: Access to FlexRay CC event id	Production	*`FrDemCtrlTestResultRef`*[2]	Assigned by DEM
FR600: SlotStatus of a particular FlexRay FrameTriggering	Production	*`FrIfDemFTSlotStatusRef`*[3]	Assigned by DEM

[FR078] ⌈ The DET error values and EventIds are named in capital letters according to the scheme FR_E_<NAME>, where NAME describes the error/EventId and may consist of several words separated by underscores. ⌋ (BSW00327)

[2] Since there are multiple instances for each DEM EventId, the value is configurable. The names listed here (*`FrDemCtrlTestResultRef, FrIfDemFTSlotStatusRef,`*) is the name of the configuration parameter.
[3] The configuration parameter *`FrIfDemFTSlotStatusRef`* is part of FrIf configuration.

7.8 Error detection

[FR026] ⌈ The detection of development errors is configurable at pre-compile-time. The switch `FrDevErrorDetect` (see chapter 10) shall activate (`true`) or deactivate (`false`) the detection of all development errors. ⌋(BSW00338, BSW00350)

[FR126] ⌈ The detection of production code errors cannot be switched off. ⌋()

7.9 Error notification

[FR127] ⌈ Detected development errors shall be reported to the `Det_ReportError` service of the Development Error Tracer (DET) if the pre-processor switch `FrDevErrorDetect` is set (see chapter 10) to `true`. ⌋()

[FR028] ⌈ The status of EventIds shall be reported to the Diagnostic Event Manager (see chapter 8.7). ⌋(BSW00339)

8 API specification

[FR098]「 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. 」(BSW00307)

8.1 Imported types

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

[FR099]「

Module	*Imported Type*
Dem	Dem_EventIdType
	Dem_EventStatusType
Std_Types	Std_ReturnType
	Std_VersionInfoType

」(BSW00348, BSW00353, BSW00361)

8.2 Macro definitions

[FR657]「 The macros listed in this chapter shall be defined in Fr_GeneralTypes.h.

」()

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 chapter 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

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	FrIfGNetworkManagementVectorLength
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
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

[FR110] ⌈ The content of *Fr_GeneralTypes.h* consists of types specified within [7], [8] and the following type specifications within this document except Fr_ConfigType. ⌋ (BSW00305)

[FR499] ⌈ The content of *Fr_GeneralTypes.h* shall be protected by a FR_GENERAL_TYPES define. ⌋ ()

[FR500] ⌈ 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. ⌋ ()

[FR077] ⌈ 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. ⌋ (BSW00305)

8.3.1 Fr_ConfigType

[FR501] ⌈ This type contains the definition of the implementation-specific post build configuration structure. ⌋ ()

[FR646] ⌈ Fr_ConfigType shall be provided by the headerfile Fr.h. ⌋ ()

[FR648] ⌈ Rules of FR076 shall be applied to Fr_ConfigType. ⌋ ()

8.3.2 Fr_POCStateType

[FR505] ⌈

Name:	`Fr_POCStateType`	
Type:	`Enumeration`	
Range:	`FR_POCSTATE_CONFIG (=0)`	Represents literal CONFIG of formal type definition T_POCState.
	`FR_POCSTATE_DEFAULT_CONFIG`	Represents literal DEFAULT_CONFIG of formal type definition T_POCState.
	`FR_POCSTATE_HALT`	Represents literal HALT of formal type definition T_POCState.

	FR_POCSTATE_NORMAL_ACTIVE	Represents literal NORMAL_ACTIVE of formal type definition T_POCState.
	FR_POCSTATE_NORMAL_PASSIVE	Represents literal NORMAL_PASSIVE of formal type definition T_POCState.
	FR_POCSTATE_READY	Represents literal READY of formal type definition T_POCState.
	FR_POCSTATE_STARTUP	Represents literal STARTUP of formal type definition T_POCState.
	FR_POCSTATE_WAKEUP	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].	

」(BSW00441)

8.3.3 Fr_SlotModeType

[FR506]「 [4]

Name:	Fr_SlotModeType	
Type:	Enumeration	
Range:	FR_SLOTMODE_KEYSLOT	Represents literal KEYSLOT of formal type definition T_SlotMode.
	FR_SLOTMODE_ALL_PENDING	Represents literal ALL_PENDING of formal type definition T_SlotMode.
	FR_SLOTMODE_ALL	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].	

」(BSW00441)

[FR599]「 There shall be a preprocessor macro with name FR_SLOTMODE_SINGLE that maps to value FR_SLOTMODE_KEYSLOT in file *Fr_GeneralTypes.h*. 」()

8.3.4 Fr_ErrorModeType

[FR507]「

Name:	Fr_ErrorModeType	
Type:	Enumeration	
Range:	FR_ERRORMODE_ACTIVE (=0)	Represents literal ACTIVE of formal type definition T_ErrorMode.
	FR_ERRORMODE_PASSIVE	Represents literal PASSIVE of formal type definition T_ErrorMode.
	FR_ERRORMODE_COMM_HALT	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].	

」(BSW00441)

[4] For FlexRay 2.1 Rev A compliant FlexRay controllers see literal SINGLESLOT instead of KEYSLOT in [13].

8.3.5 Fr_WakeupStatusType

[FR508]⌜

Name:	Fr_WakeupStatusType	
Type:	Enumeration	
Range:	FR_WAKEUP_UNDEFINED (=0)	Represents literal UNDEFINED of formal type definition T_WakeupStatus.
	FR_WAKEUP_RECEIVED_HEADER	Represents literal RECEIVED_HEADER of formal type definition T_WakeupStatus.
	FR_WAKEUP_RECEIVED_WUP	Represents literal RECEIVED_WUP of formal type definition T_WakeupStatus.
	FR_WAKEUP_COLLISION_HEADER	Represents literal COLLISION_HEADER of formal type definition T_WakeupStatus.
	FR_WAKEUP_COLLISION_WUP	Represents literal COLLISION_WUP of formal type definition T_WakeupStatus.
	FR_WAKEUP_COLLISION_UNKNOWN	Represents literal COLLISION_UNKNOWN of formal type definition T_WakeupStatus.
	FR_WAKEUP_TRANSMITTED	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].	

⌟(BSW00441)

8.3.6 Fr_StartupStateType

[FR509] ⌜

Name:	Fr_StartupStateType	
Type:	Enumeration	
Range:	FR_STARTUP_UNDEFINED (=0)	Represents literal UNDEFINED of formal type definition T_StartupState.
	FR_STARTUP_COLDSTART_LISTEN	Represents literal COLDSTART_LISTEN of formal type definition T_StartupState.
	FR_STARTUP_INTEGRATION_COLDSTART_CHECK	Represents literal INTEGRATION_COLDSTART_CHECK of formal type definition T_StartupState.
	FR_STARTUP_COLDSTART_JOIN	Represents literal COLDSTART_JOIN of formal type definition T_StartupState.
	FR_STARTUP_COLDSTART_COLLISION_RESOLUTION	Represents literal COLDSTART_COLLISION_RESOLUTION of formal type definition T_StartupState.
	FR_STARTUP_COLDSTART_CONSISTENCY_CHECK	Represents literal COLDSTART_CONSISTENCY_CHECK of formal type definition T_StartupState.
	FR_STARTUP_INTEGRATION_LISTEN	Represents literal INTEGRATION_LISTEN of formal type definition T_StartupState.
	FR_STARTUP_INITIALIZE_SCHEDULE	Represents literal INITIALIZE_SCHEDULE of formal type definition T_StartupState.
	FR_STARTUP_INTEGRATION_CONSISTENCY_CHECK	Represents literal INTEGRATION_CONSISTENCY_CHECK of formal type definition T_StartupState.
	FR_STARTUP_COLDSTART_GAP	Represents literal COLDSTART_GAP of formal type definition T_StartupState.

	FR_STARTUP_EXTERNAL_STARTUP	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].	

」(BSW00441)

Note: Fr_StartupStateType contains the superset of FlexRay 2.1 and FlexRay 3.0 specification. Thus state FR_STARTUP_EXTERNAL_STARTUP canot be reached on FlexRay 2.1 compliant FlexRay controllers.

8.3.7 Fr_POCStatusType

[FR510]「

Name:	Fr_POCStatusType		
Type:	Structure		
Element:	boolean	CHIHaltRequest	--
	boolean	ColdstartNoise	--
	Fr_ErrorModeType	ErrorMode	--
	boolean	Freeze	--
	Fr_SlotModeType	SlotMode	--
	Fr_StartupStateType	StartupState	--
	Fr_POCStateType	State	--
	Fr_WakeupStatusType	WakeupStatus	--
	boolean	CHIReadyRequest	--
Description:	This formal definition refers to the description of type T_POCStatus in chapter 2.2.1.3 POC status of [12].		

」()

8.3.8 Fr_TxLPduStatusType

[FR511]「

Name:	Fr_TxLPduStatusType	
Type:	Enumeration	
Range:	FR_TRANSMITTED (=0)	LPdu has been transmitted
	FR_NOT_TRANSMITTED	LPdu has not been transmitted
Description:	These values are used to determine whether a LPdu has been transmitted or not.	

」(BSW00441)

8.3.9 Fr_RxLPduStatusType

[FR512]「

Name:	Fr_RxLPduStatusType	
Type:	Enumeration	
Range:	FR_RECEIVED (=0)	LPdu has been received
	FR_NOT_RECEIVED	LPdu has not been received
	FR_RECEIVED_MORE_DATA_AVAILABLE	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.	

」(BSW00441)

8.3.10 Fr_ChannelType

[FR514] ⌈

Name:	`Fr_ChannelType`	
Type:	`Enumeration`	
Range:	`FR_CHANNEL_A (=0)`	Refers to channel A of a CC.
	`FR_CHANNEL_B`	Refers to channel B of a CC.
	`FR_CHANNEL_AB`	Refers to both channels (A and B) of a CC.
Description:	The values are used to reference channels on a CC.	

」(BSW00441)

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

[FR032] ⌈

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:	Initalizes the Fr.

」(BSW00358, BSW00404, BSW00405, BSW00414, BSW101)

CC precondition for the function Fr_Init: None.

[FR137]「 The function Fr_Init shall internally store the configuration address to enable subsequent API calls to access the configuration. 」(BSW00438)

[FR135]「 If development error detection for the Fr module is enabled, then the function Fr_Init shall check whether the parameter Fr_ConfigPtr is a NULL pointer (NULL_PTR). If Fr_ConfigPtr is a NULL pointer, then the function Fr_Init shall raise development error FR_E_INV_POINTER and return. 」()

8.4.2 Fr_ControllerInit

[FR017]「

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.	

」(BSW05116, BSW05011)

[FR148]「 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. 」(BSW05059, BSW05012)

CC post condition for the function Fr_ControllerInit: CC Fr_CtrIdx shall be left in POCState ‘POC:ready’.

[FR149] ⌈ The function Fr_ControllerInit shall ensure that no transmission requests are pending. ⌋ ()

[FR150] ⌈ The function Fr_ControllerInit shall ensure that no reception indications are pending. ⌋ ()

[FR151] ⌈ The function Fr_ControllerInit shall ensure that no interrupts are pending. ⌋ ()

[FR152] ⌈ The function Fr_ControllerInit shall ensure that all timers are disabled. ⌋ (BSW05169)

[FR153] ⌈ The function Fr_ControllerInit shall ensure that all interrupts are disabled. ⌋ ()

[FR515] ⌈ The function Fr_ControllerInit shall disable all LPdus which are dynamically reconfigurable (see Fr_ReconfigLPdu, Fr_DisableLPdu). ⌋ ()

[FR147] ⌈ 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_ReportErrorStatus (*FrDemCtrlTestResultRef*, DEM_EVENT_STATUS_FAILED) and returns E_NOT_OK. ⌋ ()

[FR647] ⌈ The CC test as described in FR147 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. ⌋ ()

[FR598] ⌈ If the function Fr_ControllerInit passes the CC test within a number of configurable (*FrCtrlTestCount*) times, then it calls Dem_ReportErrorStatus (*FrDemCtrlTestResultRef*, DEM_EVENT_STATUS_PASSED). ⌋ ()

[FR143] ⌈ 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_NOT_INITIALIZED and return E_NOT_OK. ⌋ ()

[FR144] ⌈ 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 and return E_NOT_OK. ⌋ ()

8.4.3 Fr_StartCommunication

[FR010]⌈

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.	

⌋ (BSW05109)

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

[FR177] ⌈ 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.

[FR176] ⌈ If the function Fr_StartCommunication is able to and detects a hardware error while performing the requested functionality, then it shall call Dem_ReportErrorStatus (*FrDemCtrlTestResultRef*, DEM_EVENT_STATUS_FAILED) and return E_NOT_OK. ⌋ ()

[FR173] ⌈ 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_NOT_INITIALIZED and return E_NOT_OK. ⌋ ()

[FR174] ⌈ 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 and return E_NOT_OK. ⌋ ()

[FR175] ⌈ If development error detection for the Fr module is enabled, then 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 development error FR_E_INV_POCSTATE and return E_NOT_OK. ⌋ ()

8.4.4 Fr_AllowColdstart

[FR114] ⌈

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'.	

⌋ ()

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.

[FR182] ⌈ 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. ⌋ ()

[FR181] ⌈ If the function Fr_AllowColdstart is able to and detects a hardware error while performing the requested functionality, then it shall call Dem_ReportErrorStatus

(*FrDemCtrlTestResultRef*, DEM_EVENT_STATUS_FAILED) and return E_NOT_OK. 」()

[FR178] 「 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_NOT_INITIALIZED and return E_NOT_OK. 」()

[FR179] 「 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 and return E_NOT_OK. 」()

[FR180] 「 If development error detection for the Fr module is enabled, then 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 development error FR_E_INV_POCSTATE and return E_NOT_OK. 」()

8.4.5 Fr_AllSlots

[FR516] 「

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'.	

」()

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

[FR518] 「 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.

[FR520] 「 If the function Fr_AllSlots is able to and detects a hardware error while performing the requested functionality, then it shall call Dem_ReportErrorStatus (*FrDemCtrlTestResultRef*, DEM_EVENT_STATUS_FAILED) and return E_NOT_OK. 」()

[FR521] 「 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_NOT_INITIALIZED and return E_NOT_OK. 」()

[FR522] 「 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 and return E_NOT_OK. 」()

[FR523] 「 If development error detection for the Fr module is enabled, then 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 development error FR_E_INV_POCSTATE and return E_NOT_OK. 」()

8.4.6 Fr_HaltCommunication

[FR014] [5]「

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.

[5] Invoke the command ‘HALT’ for FlexRay Controllers compliant to [13].

		E_NOT_OK: API call aborted due to errors.
Description:	Invokes the CC CHI command 'DEFERRED_HALT'.	

」(BSW00336, BSW05115)

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

[FR187] 「 The function Fr_HaltCommunication shall perform the following tasks on FlexRay CC Fr_CtrIdx:

1. Invoke the CC CHI command 'DEFERRED_HALT'[5].
2. Return E_OK. 」()

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.

[FR186] 「 If the function Fr_HaltCommunication is able to and detects a hardware error while performing the requested functionality, then it shall call Dem_ReportErrorStatus (*FrDemCtrlTestResultRef*, DEM_EVENT_STATUS_FAILED) and return E_NOT_OK. 」()

[FR183] 「 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_NOT_INITIALIZED and return E_NOT_OK. 」()

[FR184] 「 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 and return E_NOT_OK. 」()

[FR185] 「 If development error detection for the Fr module is enabled, then 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 development error FR_E_INV_POCSTATE and return E_NOT_OK. 」()

8.4.7 Fr_AbortCommunication

[FR011] 「

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'.	

」(BSW05114)

[FR191]「 The function Fr_AbortCommunication shall perform the following tasks on FlexRay CC Fr_CtrIdx:

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

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

[FR190]「 If the function Fr_AbortCommunication is able to and detects a hardware error while performing the requested functionality, then it shall call Dem_ReportErrorStatus (*FrDemCtrlTestResultRef*, DEM_EVENT_STATUS_FAILED) and return E_NOT_OK. 」()

[FR188]「 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_NOT_INITIALIZED and return E_NOT_OK. 」()

[FR189]「 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 and return E_NOT_OK. 」()

8.4.8 Fr_SendWUP

[FR009]「

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'.	

」(BSW05117)

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

[FR196]「 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.

[FR195]「 If the function Fr_SendWUP is able to and detects a hardware error while performing the requested functionality, then it shall call Dem_ReportErrorStatus (*FrDemCtrlTestResultRef*, DEM_EVENT_STATUS_FAILED) and return E_NOT_OK. 」()

[FR192]「 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_NOT_INITIALIZED and return E_NOT_OK. 」()

[FR193]「 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 and return E_NOT_OK. 」()

[FR194]「 If development error detection for the Fr module is enabled, then 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 development error FR_E_INV_POCSTATE and return E_NOT_OK. 」()

8.4.9 Fr_SetWakeupChannel

[FR091] ⌈

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.	

」(BSW05117)

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

[FR202] ⌈ The function Fr_SetWakeupChannel shall perform the following tasks on FlexRay CC Fr_CtrIdx:

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. 」()

[FR201] ⌈ If the function Fr_SetWakeupChannel is able to and detects a hardware error while performing the requested functionality, then it shall call Dem_ReportErrorStatus (*FrDemCtrlTestResultRef*, DEM_EVENT_STATUS_FAILED) and return E_NOT_OK. 」()

[FR197] ⌈ 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_NOT_INITIALIZED and return E_NOT_OK. 」()

[FR198] 「 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 and return E_NOT_OK. 」()

[FR199] 「 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 and return E_NOT_OK. 」()

[FR200] 「 If development error detection for the Fr module is enabled, then 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 development error FR_E_INV_POCSTATE and return E_NOT_OK. 」()

8.4.10 Fr_GetPOCStatus

[FR012] 「

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.	

」(BSW05120)

CC precondition for the function Fr_GetPOCStatus: None.

[FR217] 「 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. 」()

[FR216] 「 If the function Fr_GetPOCStatus is able to and detects a hardware error while performing the requested functionality, then it shall call Dem_ReportErrorStatus (*FrDemCtrlTestResultRef*, DEM_EVENT_STATUS_FAILED) and return E_NOT_OK. 」()

[FR213] 「 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_NOT_INITIALIZED and return E_NOT_OK. 」()

[FR214] 「 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 and return E_NOT_OK. 」()

[FR215] 「 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_INV_POINTER and return E_NOT_OK. 」()

8.4.11 Fr_TransmitTxLPdu

[FR092] 「

Service name:	Fr_TransmitTxLPdu	

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

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):	None	
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.	

」(BSW05003, BSW05024)

CC precondition for the function Fr_TransmitTxLPdu: None.

[FR224]「 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 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.
3. Copy `Fr_LsduLength` bytes from address Fr_LsduPtr into the FlexRay CC's transmission resource and then activate it for transmission.
4. Return E_OK. 」()

[FR440]「 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. 」(BSW05024)

[FR225]「 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). 」()

[FR223]「 If the function Fr_TransmitTxLPdu is able to and detects a hardware error while performing the requested functionality, then it shall call Dem_ReportErrorStatus (*FrDemCtrlTestResultRef*, DEM_EVENT_STATUS_FAILED) and return E_NOT_OK. 」()

[FR218]「 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_NOT_INITIALIZED and return E_NOT_OK. 」()

[FR219]「 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 and return E_NOT_OK. 」()

[FR220] 「 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 and return E_NOT_OK. 」()

[FR221] 「 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 and return E_NOT_OK. 」()

[FR222] 「 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_INV_POINTER and return E_NOT_OK. 」()

8.4.12 Fr_CancelTxLPdu

[FR610] 「

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).	

」(BSW05024)

CC precondition for the function Fr_CancelTxLPdu: None.

[FR611] ⌈ The function Fr_CancelTxLPdu 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 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. ⌋ ()

[FR612] ⌈ 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. ⌋ ()

[FR613] ⌈ If the function Fr_CancelTxLPdu is able to and detects a hardware error while performing the requested functionality, then it shall call Dem_ReportErrorStatus (*FrDemCtrlTestResultRef*, DEM_EVENT_STATUS_FAILED) and return E_NOT_OK. ⌋ ()

[FR614] ⌈ 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_NOT_INITIALIZED and return E_NOT_OK. ⌋ ()

[FR615] ⌈ 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 and return E_NOT_OK. ⌋ ()

[FR616] ⌈ 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 and return E_NOT_OK. ⌋ ()

8.4.13 Fr_ReceiveRxLPdu

[FR093] ⌈

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)	
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.
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.	

」(BSW05003, BSW05024)

CC precondition for the function Fr_ReceiveRxLPdu: None.

[FR233]「 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 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.
4. 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.
5. Return E_OK.」()

[FR441]「 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 FrLPduIdx 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. 」(BSW05024)

[FR234] 「 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). 」()

[FR604] 「 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. 」()

[FR603] 「 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. 」()

[FR236] 「 The function Fr_ReceiveRxLPdu shall ensure that FR_RECEIVED is returned only for non-Nullframes. 」()

[FR237] 「 The function Fr_ReceiveRxLPdu shall ensure that the function returns FR_RECEIVED only once per received frame. 」()

[FR645] 「 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`. 」()

[FR239] 「 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. 」()

[FR239] 「 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). 」()

[FR232]「 If the function Fr_ReceiveRxLPdu is able to and detects a hardware error while performing the requested functionality, then it shall call Dem_ReportErrorStatus (FrDemCtrlTestResultRef, DEM_EVENT_STATUS_FAILED) and return E_NOT_OK. 」()

[FR605] 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_ReportErrorStatus (*FrIfDemFTSlotStatusRef*, DEM_EVENT_STATUS_FAILED).

[FR627]「 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_ReportErrorStatus (*FrIfDemFTSlotStatusRef*, DEM_EVENT_STATUS_PASSED). 」()

[FR628]「 Dem_ReportErrorStatus() shall only be called if the optional configuration parameter *FrIfDemFTSlotStatusRef* exists. 」()

[FR226]「 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_NOT_INITIALIZED and return E_NOT_OK. 」()

FR227: 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 and return E_NOT_OK.

[FR228]「 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 and return E_NOT_OK. 」()

[FR229]「 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_INV_POINTER and return E_NOT_OK. 」()

[FR230] ⌈ 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_INV_POINTER and return E_NOT_OK. ⌋ ()

[FR231] ⌈ 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_INV_POINTER and return E_NOT_OK. ⌋ ()

8.4.14 Fr_CheckTxLPduStatus

[FR094] ⌈

Service name:	Fr_CheckTxLPduStatus	

Syntax:	`Std_ReturnType Fr_CheckTxLPduStatus(`
	`  uint8 Fr_CtrlIdx,`
	`  uint16 Fr_LPduIdx,`
	`  Fr_TxLPduStatusType* Fr_TxLPduStatusPtr`
	`)`	

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
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.	

⌋ (BSW05003, BSW05024)

CC precondition for the function Fr_CheckTxLPduStatus: None.

[FR244] ⌈ 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[6].
3. If a transmission request is pending, then store the status FR_NOT_TRANSMITTED to Fr_TxLPduStatusPtr.

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

4. If no transmission request is pending, then store the status FR_TRANSMITTED to Fr_TxLPduStatusPtr.
5. Return E_OK. 」()

[FR243] 「 If the function Fr_CheckTxLPduStatus is able to and detects a hardware error while performing the requested functionality, then it shall call Dem_ReportErrorStatus (FrDemCtrlTestResultRef, DEM_EVENT_STATUS_FAILED) and return E_NOT_OK. 」()

[FR606] 「 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_ReportErrorStatus (*FrIfDemFTSlotStatusRef*, DEM_EVENT_STATUS_FAILED). 」()

[FR629] 「 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_ReportErrorStatus (*FrIfDemFTSlotStatusRef*, DEM_EVENT_STATUS_PASSED). 」()

[FR630] 「 Dem_ReportErrorStatus() shall be called only if the optional configuration parameter *FrIfDemFTSlotStatusRef* exists. 」()

[FR240] 「 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_NOT_INITIALIZED and return E_NOT_OK. 」()

[FR241] 「 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 and return E_NOT_OK. 」()

[FR242] 「 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 and return E_NOT_OK. 」()

[FR343] 「 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_INV_POINTER and return E_NOT_OK. 」()

8.4.15 Fr_PrepareLPdu

[FR107] 「

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.	

」(BSW05106)

CC precondition for the function Fr_PrepareLPdu: None.

[FR249] 「 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[7].
3. Return E_OK. 」()

[FR250] 「 The function Fr_PrepareLPdu shall be pre compile time configurable On/Off by the configuration parameter: *FrBufferReconfig*. 」()

[FR248] 「 If the function Fr_PrepareLPdu is able to and detects a hardware error while performing the requested functionality, then it shall call Dem_ReportErrorStatus

[7] 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.

(*FrDemCtrlTestResultRef*, DEM_EVENT_STATUS_FAILED) and return E_NOT_OK. 」()

[FR245]「 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_NOT_INITIALIZED and return E_NOT_OK. 」()

[FR246]「 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 and return E_NOT_OK. 」()

[FR247]「 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 and return E_NOT_OK. 」()

8.4.16 Fr_ReconfigLPdu

[FR524]「

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.	

」(BSW05059)

CC precondition for the function Fr_ReconfigLPdu: None.

[FR525]「 The function Fr_ReconfigLPdu 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 as identified by Fr_LpduIdx.
2. 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.
3. Return E_OK. 」()

[FR526]「 Whether an Lpdu is dynamically reconfigurable is determined via the configuration parameter *FrIfReconfigurable* which is a property of the FrIfLPdu configuration parameter container. 」()

[FR527]「 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. 」()

[FR528]「 The function Fr_ReconfigLPdu shall be pre compile time configurable On/Off by the configuration parameter: *FrBufferReconfig*. 」()

[FR529]「 If the function Fr_ReconfigLPdu is able to and detects a hardware error while performing the requested functionality, then it shall call Dem_ReportErrorStatus (*FrDemCtrlTestResultRef*, DEM_EVENT_STATUS_FAILED) and return E_NOT_OK. 」()

[FR530]「 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_NOT_INITIALIZED and return E_NOT_OK. 」()

[FR531]「 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 and return E_NOT_OK. 」()

[FR532] ⌈ 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 and return E_NOT_OK. ⌋ ()

[FR533] ⌈ 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 and return E_NOT_OK. ⌋ ()

[FR534] ⌈ 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 and return E_NOT_OK. ⌋ ()

[FR535] ⌈ Valid values[8] for parameter Fr_CycleRepetition are 1, 2, 4, 5, 8, 10, 16, 20, 32, 40, 50 and 64. ⌋ ()

[FR536] ⌈ If development error detection for the Fr module is enabled, then the function Fr_ReconfigLPdu shall check the parameter Fr_BaseCycle for being valid. If Fr_BaseCycle is invalid, then the function Fr_ReconfigLPdu shall raise the development error FR_E_INV_CYCLE and return E_NOT_OK. ⌋ ()

[FR537] ⌈ Valid values for parameter Fr_BaseCycle are 0 to (Fr_CycleRepetition – 1). ⌋ ()

[FR538] ⌈ 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 and return E_NOT_OK. ⌋ ()

[FR634] ⌈ If development error detection for the Fr module is enabled, then the function Fr_ReconfigLPdu shall check the parameter Fr_HeaderCRC for being valid[9]. If Fr_HeaderCRC is invalid, then the function Fr_ReconfigLPdu shall raise the development error FR_E_INV_HEADERCRC and return E_NOT_OK. ⌋ ()

[8] For FlexRay Controllers compliant to [13] only cycle repetition values 1, 2, 4, 8, 16, 32 and 64 shall be supported.
[9] 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.

8.4.17 Fr_DisableLPdu

[FR539] ⌈

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	
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.	

⌋ (BSW05059)

CC precondition for the function Fr_DisableLPdu: None.

[FR540] ⌈ 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. ⌋ ()

[FR541] ⌈ Only Lpdus that can be dynamically reconfigured can be disabled by this service (see Fr_ReconfigLPdu). ⌋ ()

[FR542] ⌈ The function Fr_DisableLPdu shall be pre compile time configurable On/Off by the configuration parameter: *FrBufferReconfig*. ⌋ ()

[FR543] ⌈ If the function Fr_DisableLPdu is able to and detects a hardware error while performing the requested functionality, then it shall call Dem_ReportErrorStatus (*FrDemCtrlTestResultRef*, DEM_EVENT_STATUS_FAILED) and return E_NOT_OK. ⌋ ()

[FR544] ⌈ 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_NOT_INITIALIZED and return E_NOT_OK. 」()

[FR545] 「 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 and return E_NOT_OK. 」()

[FR546] 「 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 and return E_NOT_OK. 」()

8.4.18 Fr_GetGlobalTime

[FR042] 「

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.	

」(BSW05019)

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

[FR256] 「 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. 」()

[FR257] ⌈ 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). ⌋ ()

[FR044] ⌈ 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. ⌋ (BSW05072)

[FR255] ⌈ If the function Fr_GetGlobalTime is able to and detects a hardware error while performing the requested functionality, then it shall call Dem_ReportErrorStatus (*FrDemCtrlTestResultRef*, DEM_EVENT_STATUS_FAILED) and return E_NOT_OK. ⌋ ()

[FR251] ⌈ 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_NOT_INITIALIZED and return E_NOT_OK. ⌋ ()

[FR252] ⌈ 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 and return E_NOT_OK. ⌋ ()

[FR253] ⌈ 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_INV_POINTER and return E_NOT_OK. ⌋ ()

[FR254] ⌈ 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_INV_POINTER and return E_NOT_OK. ⌋ ()

8.4.19 Fr_GetNmVector

[FR113] ⌈

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.	

」()

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 [12]). At this point of time the service Fr_GetNmVector shall not be called, in order to ensure a consistent NM-Vector value.

[FR262] 「 The function Fr_GetNmVector shall perform the following tasks on FlexRay CC Fr_CtrIdx:

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. 」()

[FR263] 「 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. 」()

[FR264] 「 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). 」()

[FR261] 「 If the function Fr_GetNmVector is able to and detects a hardware error while performing the requested functionality, then it shall call Dem_ReportErrorStatus (*FrDemCtrlTestResultRef*, DEM_EVENT_STATUS_FAILED) and return E_NOT_OK. 」()

[FR258] 「 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_NOT_INITIALIZED and return E_NOT_OK. 」()

[FR259] 「 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 and return E_NOT_OK. 」()

[FR260] 「 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_INV_POINTER and return E_NOT_OK. 」()

8.4.20 Fr_GetNumOfStartupFrames

[FR547] 「 [10]

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_NumOfStartupFramesPtr	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.	

」()

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

[FR549] 「 The function Fr_GetNumOfStartupFrames shall perform the following tasks on FlexRay CC Fr_CtrlIdx:

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.

[10] FlexRay 2.1 Rev A compliant controllers do not support vStartupPairs. See FR550 for FlexRay 2.1 Rev A controllers implementation constraints.

2. Return E_OK. 」()

[FR550] 「 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. 」()

[FR551] 「 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. 」()

[FR552] 「 If the function Fr_GetNumOfStartupFrames is able to and detects a hardware error while performing the requested functionality, then it shall call Dem_ReportErrorStatus (*FrDemCtrlTestResultRef*, DEM_EVENT_STATUS_FAILED) and return E_NOT_OK. 」()

[FR553] 「 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_NOT_INITIALIZED and return E_NOT_OK. 」()

[FR554] 「 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 and return E_NOT_OK. 」()

[FR555] 「 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_INV_POINTER and return E_NOT_OK. 」()

8.4.21 Fr_GetChannelStatus

[FR556]「

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.**	

」()

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

[FR558] ⌈ 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)[11]:
   - 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. 」()

[FR559] ⌈ 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. 」()

[11] 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.

[FR560] ⌈ If the function Fr_GetChannelStatus is able to and detects a hardware error while performing the requested functionality, then it shall call Dem_ReportErrorStatus (*FrDemCtrlTestResultRef*, DEM_EVENT_STATUS_FAILED) and return E_NOT_OK. ⌋ ()

[FR561] ⌈ 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_NOT_INITIALIZED and return E_NOT_OK. ⌋ ()

[FR562] ⌈ 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 and return E_NOT_OK. ⌋ ()

[FR563] ⌈ 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_INV_POINTER and return E_NOT_OK. ⌋ ()

[FR607] ⌈ 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_INV_POINTER and return E_NOT_OK. ⌋ ()

8.4.22 Fr_GetClockCorrection

[FR564] ⌈ [12],[13]

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	None	

[12] vInterimRate Correction maps to vRateCorrection for FlexRay 2.1 compliant controllers, see [13]
[13] vInterimOffsetCorrection maps to vOffsetCorrection for FlexRay 2.1 compliant controllers, see [13]

(inout):		
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 vInterimOffsetCorrection of [12] for details.	

」()

[FR566] 「 The function Fr_GetClockCorrection shall perform the following tasks on FlexRay CC Fr_CtrIdx:

1. Read the rate correction value (vInterimRateCorrection[12]) and write it as signed integer to the output parameter Fr_RateCorrectionPtr. Read the offset correction value (vInterimOffsetCorrection[13]) and write it as signed integer to the output parameter Fr_OffsetCorrectionPtr
2. Return E_OK. 」()

[FR568] 「 If the function Fr_GetClockCorrection is able to and detects a hardware error while performing the requested functionality, then it shall call Dem_ReportErrorStatus (*FrDemCtrlTestResultRef*, DEM_EVENT_STATUS_FAILED) and return E_NOT_OK. 」()

[FR569] 「 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_NOT_INITIALIZED and return E_NOT_OK. 」()

[FR570] 「 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 and return E_NOT_OK. 」()

[FR571] 「 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_INV_POINTER and return E_NOT_OK. 」()

[FR572] 「 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_INV_POINTER and return E_NOT_OK. 」()

8.4.23 Fr_GetSyncFrameList

[FR573] 「

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_ChannelAEvenListPtr Fr_ChannelBEvenListPtr Fr_ChannelAOddListPtr Fr_ChannelBOddListPtr. 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.

」()

[FR575]「 The function Fr_ GetSyncFrameList shall perform the following tasks on FlexRay CC Fr_CtrIdx:

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.
   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.
   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.
   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.
2. Return E_OK. 」()

[FR576]「 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)). 」()

[FR577]「 Unused array elements shall be set to 0, indicating no valid sync frame. 」()

[FR578]「 A maximum number of 15 syncframes shall be supported. 」()

[FR580]「 If the function Fr_GetSyncFrameList is able to and detects a hardware error while performing the requested functionality, then it shall call Dem_ReportErrorStatus (*FrDemCtrlTestResultRef*, DEM_EVENT_STATUS_FAILED) and return E_NOT_OK. 」()

[FR581]「 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_NOT_INITIALIZED and return E_NOT_OK. 」()

[FR582]「 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 and return E_NOT_OK. 」()

[FR583] ⌈ 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_INV_POINTER and return E_NOT_OK. ⌋ ()

[FR584] ⌈ 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_INV_POINTER and return E_NOT_OK. ⌋ ()

[FR585] ⌈ 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_INV_POINTER and return E_NOT_OK. ⌋ ()

[FR586] ⌈ 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_INV_POINTER and return E_NOT_OK. ⌋ ()

8.4.24 Fr_GetWakeupRxStatus

[FR587]⌈

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.	

」()

[FR588]「 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. 」()

[FR589]「 If the function Fr_GetWakeupRxStatus is able to and detects a hardware error while performing the requested functionality, then it shall call Dem_ReportErrorStatus (*FrDemCtrlTestResultRef*, DEM_EVENT_STATUS_FAILED) and return E_NOT_OK. 」()

[FR590]「 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_NOT_INITIALIZED and return E_NOT_OK. 」()

[FR591]「 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 and return E_NOT_OK. 」()

[FR592]「 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_INV_POINTER and return E_NOT_OK. 」()

8.4.25 Fr_SetAbsoluteTimer

[FR033] ⌈

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.	

⌋ (BSW05044)

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).

[FR273] ⌈ 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. ⌋ ()

[FR272] ⌈ If the function Fr_SetAbsoluteTimer is able to and detects a hardware error while performing the requested functionality, then it shall call Dem_ReportErrorStatus (*FrDemCtrlTestResultRef*, DEM_EVENT_STATUS_FAILED) and return E_NOT_OK. ⌋ ()

[FR267] ⌈ 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_NOT_INITIALIZED and return E_NOT_OK. ⌋ ()

[FR268] ⌈ 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 and return E_NOT_OK. ⌋ ()

[FR269] ⌈ 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 and return E_NOT_OK. ⌋ ()

[FR270] ⌈ 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 and return E_NOT_OK. ⌋ ()

[FR271] ⌈ 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 and return E_NOT_OK. ⌋ ()

[FR436] ⌈ If development error detection for the Fr module is enabled, then 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 development error FR_E_INV_POCSTATE and return E_NOT_OK. ⌋ ()

8.4.26 Fr_CancelAbsoluteTimer

[FR095] ⌈

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.	

⌋ (BSW05044)

CC precondition for the function Fr_CancelAbsoluteTimer: None.

[FR287] ⌈ The function Fr_CancelAbsoluteTimer shall perform the following tasks:

1. Stop the absolute timer Fr_AbsTimerIdx.
2. Return E_OK. ⌋ ()

[FR286] ⌈ If the function Fr_CancelAbsoluteTimer is able to and detects a hardware error while performing the requested functionality, then it shall call Dem_ReportErrorStatus (*FrDemCtrlTestResultRef*, DEM_EVENT_STATUS_FAILED) and return E_NOT_OK. ⌋ ()

[FR283] ⌈ 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_NOT_INITIALIZED and return E_NOT_OK. ⌋ ()

[FR284] ⌈ 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 and return E_NOT_OK. ⌋ ()

[FR285] ⌈ 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 and return E_NOT_OK. ⌋ ()

8.4.27 Fr_EnableAbsoluteTimerIRQ

[FR034] ⌈

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.	

⌋ (BSW05125, BSW05046)

CC precondition for the function Fr_EnableAbsoluteTimerIRQ: None.

[FR298] ⌈ The function Fr_EnableAbsoluteTimerIRQ shall perform the following tasks:

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

[FR297] ⌈ If the function Fr_EnableAbsoluteTimerIRQ is able to and detects a hardware error while performing the requested functionality, then it shall call Dem_ReportErrorStatus (*FrDemCtrlTestResultRef*, DEM_EVENT_STATUS_FAILED) and return E_NOT_OK. ⌋ ()

[FR294] ⌈ 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_NOT_INITIALIZED and return E_NOT_OK. ⌋ ()

[FR295] ⌈ 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 and return E_NOT_OK. ⌋ ()

[FR296] ⌈ 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 and return E_NOT_OK. ⌋ ()

8.4.28 Fr_AckAbsoluteTimerIRQ

[FR036]⌈

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.	

」(BSW05125, BSW05048)

CC precondition for the function Fr_AckAbsoluteTimerIRQ: None.

[FR309] ⌈ The function Fr_AckAbsoluteTimerIRQ shall perform the following tasks:

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

[FR308] ⌈ If the function Fr_AckAbsoluteTimerIRQ is able to and detects a hardware error while performing the requested functionality, then it shall call Dem_ReportErrorStatus (*FrDemCtrlTestResultRef*, DEM_EVENT_STATUS_FAILED) and return E_NOT_OK. 」()

[FR305] ⌈ 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_NOT_INITIALIZED and return E_NOT_OK. 」()

[FR306] ⌈ 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 and return E_NOT_OK. 」()

[FR307] ⌈ 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 and return E_NOT_OK. 」()

8.4.29 Fr_DisableAbsoluteTimerIRQ

[FR035] ⌈

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.	

」(BSW05125, BSW05047)

CC precondition for the function Fr_DisableAbsoluteTimerIRQ: None.

[FR320]「 The function Fr_DisableAbsoluteTimerIRQ shall perform the following tasks:

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

[FR319]「 If the function Fr_DisableAbsoluteTimerIRQ is able to and detects a hardware error while performing the requested functionality, then it shall call Dem_ReportErrorStatus (*FrDemCtrlTestResultRef*, DEM_EVENT_STATUS_FAILED) and return E_NOT_OK.」()

[FR316]「 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_NOT_INITIALIZED and return E_NOT_OK.」()

[FR317]「 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 and return E_NOT_OK.」()

[FR318]「 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 and return E_NOT_OK.」()

8.4.30 Fr_GetAbsoluteTimerIRQStatus

[FR108]「

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.	

」(BSW05125)

CC precondition for the function Fr_GetAbsoluteTimerIRQStatus: None.

[FR332]「 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.」()

[FR331]「 If the function Fr_GetAbsoluteTimerIRQStatus is able to and detects a hardware error while performing the requested functionality, then it shall call Dem_ReportErrorStatus (*FrDemCtrlTestResultRef*, DEM_EVENT_STATUS_FAILED) and return E_NOT_OK.」()

[FR327]「 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_NOT_INITIALIZED and return E_NOT_OK.」()

[FR328]「 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 and return E_NOT_OK.」()

[FR329]「 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 and return E_NOT_OK.」()

[FR330]「 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_INV_POINTER and return E_NOT_OK. 」()

8.4.31 Fr_GetVersionInfo

[FR070] 「

Service name:	Fr_GetVersionInfo

Syntax:	`void Fr_GetVersionInfo(`
	`  Std_VersionInfoType* VersioninfoPtr`
	`)`

Service ID[hex]:	0x1b
Sync/Async:	Synchronous
Reentrancy:	Non 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.

」(BSW00407, BSW00411)

[FR340] 「 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_INV_POINTER and return. 」(BSW00411)

[FR341] 「 The function Fr_GetVersionInfo shall return the version information of this module. The version information includes:

- Module Id
- Vendor Id
- Vendor specific version numbers. 」(BSW00411)

[FR342] 「 The function Fr_GetVersionInfo shall be pre compile time configurable On/Off by the configuration parameter: *FrVersionInfoApi*. 」(BSW00411)

8.4.32 Fr_ReadCCConfig

[FR651:

]「

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.	

」()

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.

[FR652]「 If the function Fr_ReadCCConfig is able to and detects a hardware error while performing the requested functionality, then it shall call Dem_ReportErrorStatus (*FrDemCtrlTestResultRef*, DEM_EVENT_STATUS_FAILED) and return E_NOT_OK. 」()

[FR653]「 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_NOT_INITIALIZED and return E_NOT_OK. 」()

[FR654]「 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 and return E_NOT_OK. 」()

[FR655]「 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_ConfigParamIdxIdx is invalid[14], then the function Fr_ReadCCConfig shall raise the development error FR_E_INV_CONFIG_IDX and return E_NOT_OK. 」()

[14] Valid values are listed in chapter 8.2.1 Configuration parameter index macros and and in requirements FR662, FR663, FR664, FR665, FR666.

[FR656]「 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_INV_POINTER and return E_NOT_OK. 」()

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:

- **[FR658]**「 *integers* are mapped 1 to 1. 」()
- **[FR659]**「 *floats* (units of seconds) are converted to units of nanoseconds (with nanosecond granularity) and converted to uint32. 」()
- **[FR661]**「 *booleans* shall output 1 for true and 0 for false. 」()

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

[FR662HH]「 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
」()

[FR663HH]「 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
」()

[FR664HH]「 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
」()

[FR665HH]「 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

」()

[FR666HXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX XXXX]「 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 Call-back 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.

[FR390]「

API function	***Description***
Dem_ReportErrorStatus	Queues the reported events from the BSW modules (API is only used by BSW modules). The interface has an asynchronous behavior, because the processing of the event is done within the Dem main function.

」()

8.7.2 Optional Interfaces

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

[FR391] ⌈

API function	*Description*
Det_ReportError	Service to report development errors.
SchM_Enter_Fr	--
SchM_Exit_Fr	--

⌋ ()

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.

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

In addition to this section, it is highly recommended to read the documents:

- AUTOSAR Layered Software Architecture [2]
- AUTOSAR ECU Configuration Specification [4]
  This document describes the AUTOSAR configuration methodology and the AUTOSAR configuration metamodel in detail.

The following is only a short survey of the topic and it will not replace the ECU Configuration Specification document.

10.1.1 Configuration and configuration parameters

Configuration parameters define the variability of the generic part(s) of an implementation of a module. This means that only generic or configurable module implementation can be adapted to the environment (software/hardware) in use during system and/or ECU configuration.

The configuration of parameters can be achieved at different times during the software process: before compile time, before link time or after build time. In the following, the term “configuration class” (of a parameter) shall be used in order to refer to a specific configuration point in time.

10.1.2 Variants

Variants describe sets of configuration parameters. E.g., variant 1: only pre-compile time configuration parameters; variant 2: mix of pre-compile- and post build time-configuration parameters. In one variant, a parameter can only be of one configuration class.

10.1.3 Containers

[FR067] ⌈ Containers structure the set of configuration parameters. This means:

- *all* configuration parameters are kept in containers.
- (sub-) containers can reference (sub-) containers. It is possible to assign a multiplicity to these references. The multiplicity then defines the possible number of instances of the contained parameters. ⌋ (BSW00388)

10.1.4 Specification template for configuration parameters

The following tables consist of three sections:

- the general section
- the configuration parameter section
- the section of included/referenced containers

Pre-compile time - specifies whether the configuration parameter shall be of configuration class *Pre-compile time* or not

Label	***Description***
x	The configuration parameter shall be of configuration class *Pre-compile time*.
--	The configuration parameter shall never be of configuration class *Pre-compile time*.

Link time - specifies whether the configuration parameter shall be of configuration class *Link time* or not

Label	***Description***
x	The configuration parameter shall be of configuration class *Link time*.
--	The configuration parameter shall never be of configuration class *Link time*.

Post Build - specifies whether the configuration parameter shall be of configuration class *Post Build* or not

Label	***Description***
x	The configuration parameter shall be of configuration class *Post Build* and no specific implementation is required.
L	*Loadable* – the configuration parameter shall be of configuration class *Post Build* and only one configuration parameter set resides in the ECU.
M	*Multiple* – the configuration parameter shall be of configuration class *Post Build* and is selected out of a set of multiple parameters by passing a dedicated pointer to the init function of the module.
--	The configuration parameter shall never be of configuration class *Post Build*.

10.2 Containers and configuration parameters

The following chapters summarize all configuration parameters.

10.2.1 Variants

VARIANT-POST-BUILD: All configuration parameters in container 'FrGeneral' shall be configurable at pre-compile time. All other configuration parameters shall be configurable at post-build-time.
Use case: Object code delivery, selectable configuration

VARIANT-PRE-COMPILE: All configuration parameters shall be configurable at pre-compile time.
Use case: Execution time optimizations

10.2.2 Fr

Module Name	*Fr*
Module Description	Configuration of the Fr (FlexRay driver) module.

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
FrGeneral	1	General configuration (parameters) of the FlexRay Driver module.
FrMultipleConfiguration	1	Configuration of the individual controllers.

10.2.3 FrGeneral

SWS Item	**FR392_Conf :**
Container Name	FrGeneral
Description	General configuration (parameters) of the FlexRay Driver module.
Configuration Parameters	

SWS Item	**FR443_Conf :**		
Name	FrBufferReconfig		
Description	Enables or disables buffer reconfiguration at runtime.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**FR001_Conf :**		
Name	FrCtrlTestCount		
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		
ConfigurationClass	***Pre-compile time***	X	All Variants

	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**FR393_Conf :**		
Name	FrDevErrorDetect		
Description	Switches the Development Error Detection and Notification on or off. true: Development Error Detection and Notification enabled. false: Development Error Detection and Notification disabled.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**FR439_Conf :**		
Name	FrIndex		
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	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**FR394_Conf :**		
Name	FrNumCtrlSupported		
Description	Determines the maximum number of communication controllers that the driver supports.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 256		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**FR002_Conf :**		
Name	FrRxStringentCheck		
Description	If stringent check is enabled (true), received frames are only accepted if no slot status error occured.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**FR016_Conf :**

Name	FrRxStringentLengthCheck		
Description	If stringent check is enabled (true), received frames are only accepted the received payload length matches the configured payload length.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**FR396_Conf :**		
Name	FrVersionInfoApi		
Description	Enables/disables the existence of the Fr_GetVersionInfo API.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

No Included Containers

10.2.4 FrController

SWS Item	**FR083_Conf :**
Container Name	FrController
Description	Configuration of the individual controller.
Configuration Parameters	

SWS Item	**FR400_Conf :**		
Name	FrCtrlIdx		
Description	Determines index of CC within Fr.		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 255		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**FR402_Conf :**		
Name	FrPAllowHaltDueToClock		
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	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD

Scope / Dependency	scope: Module

SWS Item	**FR403_Conf :**		
Name	FrPAllowPassiveToActive		
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	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**FR404_Conf :**		
Name	FrPChannels		
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
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**FR405_Conf :**		
Name	FrPClusterDriftDamping		
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	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**FR406_Conf :**		
Name	FrPDecodingCorrection		
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 FlexRay Protocol 2.1 Rev. A compliance. Upper limit 136 for FlexRay Protocol 3.0 compliance.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	12 .. 143		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE

	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**FR407_Conf :**		
Name	FrPDelayCompensationA		
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	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**FR408_Conf :**		
Name	FrPDelayCompensationB		
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	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**FR448_Conf :**		
Name	FrPExternalSync		
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	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**FR447_Conf :**		
Name	FrPFallBackInternal		
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	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**FR411_Conf :**		
Name	FrPKeySlotId		
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.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 1023		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**FR425_Conf :**		
Name	FrPKeySlotOnlyEnabled		
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 pSingleSlotEnabled.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**FR412_Conf :**		
Name	FrPKeySlotUsedForStartup		
Description	Flag indicating whether the key slot is used to transmit a startup frame. If FrPKeySlotUsedForStartup is set to true then FrPKeySlotUsedForSync must also be set to true. If FrPTwoKeySlotMode is set to true then both FrPKeySlotUsedForSync and FrPKeySlotUsedForStartup must also be set to true.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**FR413_Conf :**		
Name	FrPKeySlotUsedForSync		
Description	Flag indicating whether the key slot is used to transmit a sync frame. If FrPKeySlotUsedForStartup is set to true then FrPKeySlotUsedForSync must also be set to true. If FrPTwoKeySlotMode is set to true then both		

	FrPKeySlotUsedForSync and FrPKeySlotUsedForStartup must also be set to true.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**FR414_Conf :**		
Name	FrPLatestTx		
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	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**FR415_Conf :**		
Name	FrPMacroInitialOffsetA		
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	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**FR416_Conf :**		
Name	FrPMacroInitialOffsetB		
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	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**FR417_Conf :**		
Name	FrPMicroInitialOffsetA		
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	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**FR418_Conf :**		
Name	FrPMicroInitialOffsetB		
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	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**FR419_Conf :**		
Name	FrPMicroPerCycle		
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	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**FR444_Conf :**		
Name	FrPNmVectorEarlyUpdate		
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	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**FR421_Conf :**		
Name	FrPOffsetCorrectionOut		
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	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**FR450_Conf :**		
Name	FrPOffsetCorrectionStart		
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 FlexRay 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	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**FR422_Conf :**		
Name	FrPPayloadLengthDynMax		
Description	Maximum payload length for dynamic frames [16 bit words].		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 127		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**FR423_Conf :**		
Name	FrPRateCorrectionOut		
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	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE

	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**FR424_Conf :**		
Name	FrPSamplesPerMicrotick		
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	
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**FR445_Conf :**		
Name	FrPSecondKeySlotId		
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	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**FR446_Conf :**		
Name	FrPTwoKeySlotMode		
Description	Flag indicating whether node operates as a coldstart node in a TT-E or TT-L cluster. If pTwoKeySlotMode is set to true then both pKeySlotUsedForSync and pKeySlotUsedForStartup 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	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**FR426_Conf :**	
Name	FrPWakeupChannel	
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	--	
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**FR427_Conf :**		
Name	FrPWakeupPattern		
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	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**FR428_Conf :**		
Name	FrPdAcceptedStartupRange		
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	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**FR429_Conf :**		
Name	FrPdListenTimeout		
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	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**FR431_Conf :**
Name	FrPdMicrotick
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	
	T25NS	25 ns	
	T50NS	50 ns	
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

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_ReportErrorStatus API in case the corresponding error occurs. The EventId is taken from the referenced DemEventParameter's DemEventId value. The standardized errors are provided in the 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.

10.2.5 FrAbsoluteTimer

SWS Item	**FR432_Conf :**
Container Name	FrAbsoluteTimer
Description	Specifies the absolute timer configuration parameters of the Fr.
Configuration Parameters	

SWS Item	**FR433_Conf :**		
Name	FrAbsTimerIdx		
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	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

No Included Containers

10.2.6 FrControllerDemEventParameterRefs

SWS Item	**FR452_Conf :**
Container Name	FrControllerDemEventParameterRefs
Description	Container for the references to DemEventParameter elements which shall be invoked using the API Dem_ReportErrorStatus API in case the corresponding error occurs. The EventId is taken from the referenced

	DemEventParameter's DemEventId value. The standardized errors are provided in the container and can be extended by vendor specific error references.
Configuration Parameters	

SWS Item	**FR005_Conf :**		
Name	FrDemCtrlTestResultRef		
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	Reference to [DemEventParameter]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

No Included Containers

10.2.7 FrFifo

SWS Item	**FR009_Conf :**
Container Name	FrFifo
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	**FR006_Conf :**		
Name	FrAdmitWithoutMessageId		
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	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**FR007_Conf :**		
Name	FrBaseCycle		
Description	FIFO cycle counter acceptance criteria.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 63		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**FR449_Conf :**
Name	FrChannels
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	
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**FR008_Conf :**		
Name	FrCycleRepetition		
Description	FIFO cylce 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	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**FR010_Conf :**		
Name	FrFifoDepth		
Description	Fifo Depth.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 2048		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**FR011_Conf :**		
Name	FrMsgIdMask		
Description	FIFO message identifier acceptance criteria (Mask filter).		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**FR012_Conf :**		
Name	FrMsgIdMatch		
Description	FIFO message identifier acceptance criteria (Match filter).		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	

	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

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

10.2.8 FrRange

SWS Item	**FR013_Conf :**		
Container Name	FrRange		
Description	FIFO Frame Id range acceptance criteria.		
Configuration Parameters			

SWS Item	**FR014_Conf :**		
Name	FrRangeMax		
Description	Last FrameId of this range that will be accepted by the FIFO.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 2047		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**FR015_Conf :**		
Name	FrRangeMin		
Description	First FrameId of this range that will be accepted by the FIFO.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 2047		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

No Included Containers

10.2.9 FrMultipleConfiguration

SWS Item	**FR397_Conf :**
Container Name	FrMultipleConfiguration [Multi Config Container]
Description	Configuration of the individual controllers.
Configuration Parameters	

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

10.3 Published Information

[FR667]「 The standardized common published parameters as required by BSW00402 in the General Requirements on Basic Software Modules [3] shall be published within the header file of this module and need to be provided in the BSW Module Description. The according module abbreviation can be found in the List of Basic Software Modules [1]. 」()

Additional module-specific published parameters are listed below if applicable.

[FR650]「 :
The version numbers of AUTOSAR Basic Software Modules shall be enumerated according to the following rules:
- Increasing a more significant digit of a version number resets all less significant digits
- The PATCH_VERSION is incremented if the module is still upwards and downwards compatible (e.g. bug fixed)
- The MINOR_VERSION is incremented if the module is still downwards compatible (e.g. new functionality added)
- The MAJOR_VERSION is incremented if the module is not compatible any more (e.g. existing API changed) 」(BSW00318)

11 Release Change History

Changes from Release 4.0 Rev. 1 to Release 4.0 Rev. 3

11.1 Added SWS Items

SWS Item	*Rationale*
FR651 – FR659 FR661 – FR666	Added API function Fr_ReadCCConfig

11.2 Changed SWS Items

SWS Item	*Rationale*
FR030	Change of requirement BSW004
FR176, FR181, FR520, FR186, FR190, FR195, FR201, FR216, FR223, FR613, FR232, FR243, FR248, FR529, FR543, FR255, FR261, FR552, FR560, FR568, FR580, FR589, FR272, FR286, FR297, FR308, FR319, FR331	Modification of hardware access check.
FR030	Fixed wording in intra-module pre-compile-time version check.
FR624	Adapted reconfigurable LPdu config description

11.3 Deleted SWS Items

SWS Item	*Rationale*
FR031	Change of requirement BSW004

12 Not applicable requirements

[FR999] ⌈ These requirements are not applicable to this specification. ⌋ (BSW00306, BSW00312, BSW00314, BSW00325, BSW00327, BSW00328, BSW00329, BSW00330, BSW00331, BSW00333, BSW00335, BSW00341, BSW00343, BSW00344, BSW00359, BSW00360, BSW00370, BSW00371, BSW00373, BSW00375, BSW00376, BSW00377, BSW00386, BSW00387, BSW00410, BSW00415, BSW00416, BSW00417, BSW00422, BSW00423, BSW00424, BSW00425, BSW00426, BSW00427, BSW00428, BSW00429, BSW00432, BSW00433, BSW00437, BSW00439, BSW00440, BSW00447, BSW00449, BSW00450, BSW005, BSW006, BSW009, BSW010, BSW161, BSW162, BSW164, BSW168, BSW170, BSW172, BSW05000, BSW05001, BSW05002, BSW05033, BSW05053, BSW05052)