MICROSAR FR

Technical Reference

Base Content

Version 1.04.00

Authors	Roland Hocke, Matthias Müller
Status	Released

1 Document Information

1.1 History

Author	Date	Version	Remarks
Matthias Müller	2012-11-15	1.0	Initial Version of Autosar 4
Matthias Müller	2013-07-18	1.0.1	ESCAN00069139: Added new restriction to section 3.14 FIFO reception
Matthias Müller	2013-08-01	1.1	ESCAN00067407 Remove obsolete MTS APIs
Roland Hocke	2013-10-24	1.2	Added Limitations and the description of StringentLength- and StringentCheck
Matthias Müller	2014-11-05	1.3	Added feature MICROSAR Identity Manager using Post-Build Selectable
Matthias Müller	2017-07-05	1.4	Adapted Features and added description of ApplFr_ISR_Timer0_1 and ApplFr_ISR_CycleStart_1.

Table 1-1 History of the document

1.2 Reference Documents

No.	Title	Version
[1]	AUTOSAR_SWS_FlexRayDriver.pdf	4.3.0
[2]	AUTOSAR_SWS_FlexRayDriver.pdf	2.3.0
[3]	AUTOSAR_SRS_FlexRay.pdf	3.1.0
[4]	AUTOSAR_SRS_FlexRay.pdf	4.3.0
[5]	AUTOSAR_SWS_DET.pdf	2.2.0
[6]	AUTOSAR_SWS_DEM.pdf	2.2.1
[7]	AUTOSAR_BasicSoftwareModules.pdf	1.2.0
[8]	TechnicalReference_ASR_Fr_<CCName>_<platform>.pdf	1.10 or later
[9]	FlexRay Communications System Protocol Specification	2.1A
[10]	TechnicalReference_ASR_FrIf.pdf	3.0.7 or later
[11]	TechnicalReference_Asr_EcuM.pdf	2.1.0 or later

[12]	TechnicalReference_Asr_FrTp.pdf	1.14.0 or later
[13]	TechnicalReference_Asr_AMDRunTimeMeasurement.pdf	V1.0 or later
[14]	AN-ISC-8-1118 MICROSAR BSW Compatibility Check	1.0
[15]	AUTOSAR_InterruptHandling_Explanation.pdf	1.0.0 or later
[16]	http://www.autosar.org/bugzilla/	n/a

Table 1-2 Reference documents

1.3 Scope of the Document

This technical reference describes the general use of the FlexRay driver basis software. All aspects which are Communication controller specific are described in a separate document [8], which is also part of the delivery.

!

Please note
We have configured the programs in accordance with your specifications in the questionnaire. Whereas the programs do support other configurations than the one specified in your questionnaire, Vector´s release of the programs delivered to your company is expressly restricted to the configuration you have specified in the questionnaire.

Contents

Illustrations

Tables

2 Introduction

This document describes the functionality, API and configuration of the AUTOSAR BSW module FR as specified in [8].

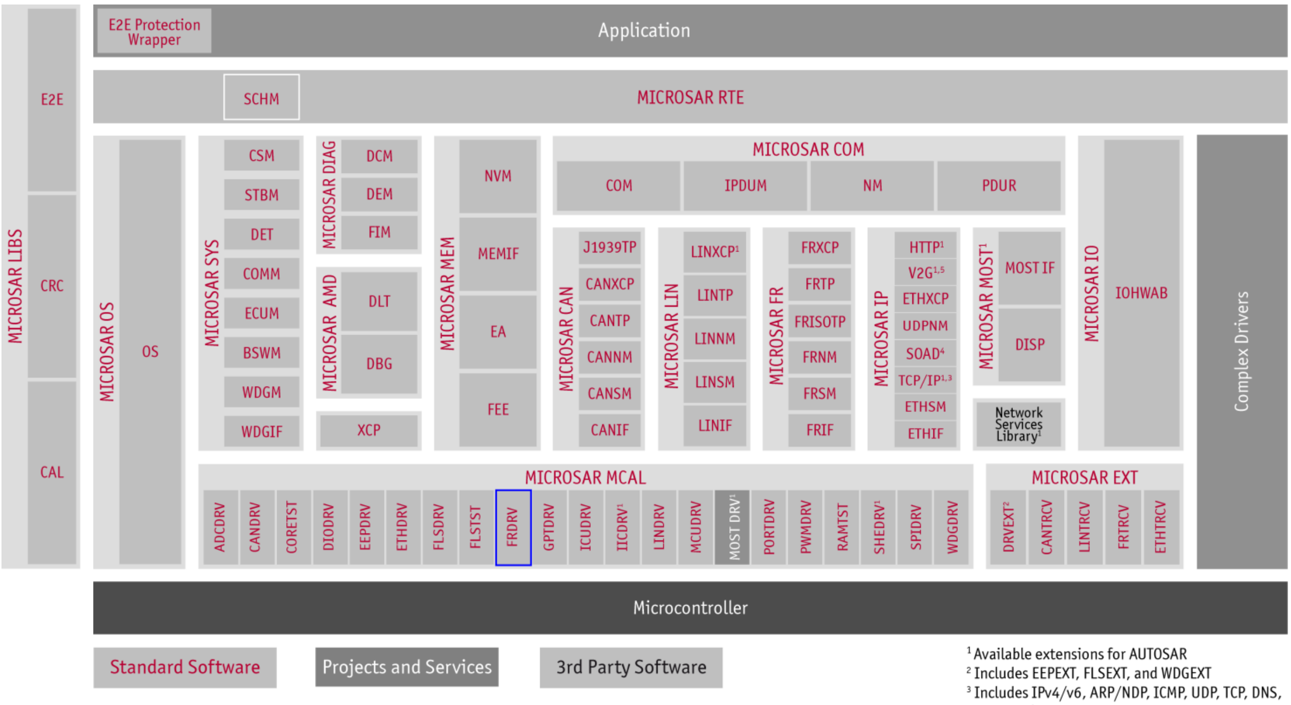
Supported AUTOSAR Release*:	4	
Supported Configuration Variants:	pre-compile, post-build	
Vendor ID:	FR_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	FR_MODULE_ID	81 decimal (according to ref. [7])

* For the precise AUTOSAR Release 4.x please see the release specific documentation.

Caution
Deviations can be described in [8].

2.1 Architecture Overview

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



[1] Available extensions for AUTOSAR

[2] Includes EEPEXT, FLSEXT, and WDGEXT

[3] Includes IPv4/v6, ARP/NDP, ICMP, UDP, TCP, DNS, DHCP and TLS

[4] Includes Diagnostics over IP (DoIP)

[5] Includes SCC according to ISO 15118 and DIN 70121

Figure 2-1 AUTOSAR architecture

The next figure shows the interfaces to adjacent modules of the Fr. These interfaces are described in chapter “API Description”.

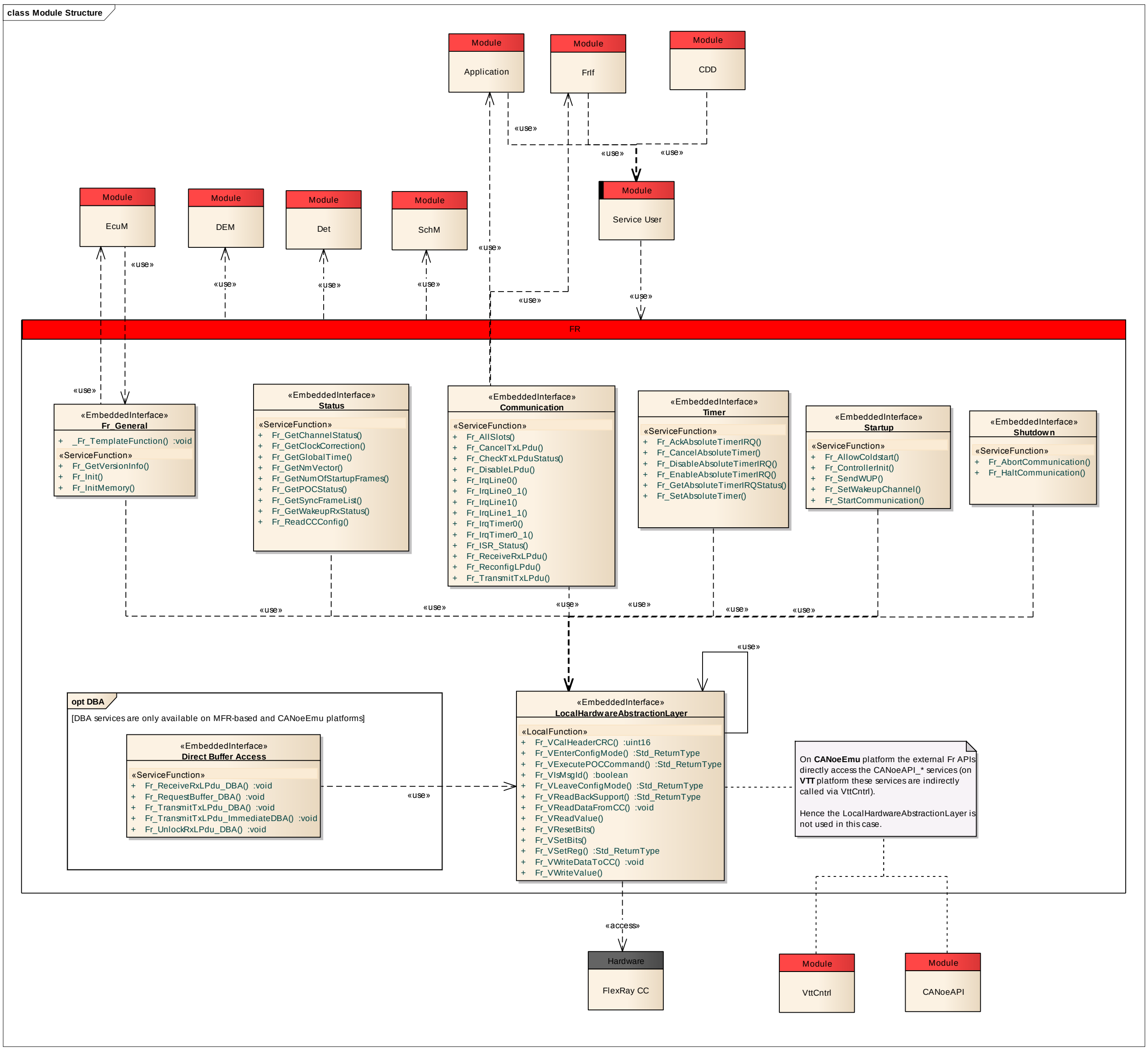


Figure 2-2 Interfaces to adjacent modules of the Fr

3 Functional Description

3.1 Features

he features listed in the following tables cover the complete functionality specified for the FR.

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

- Table 3-1 Supported AUTOSAR standard conform features
- Table 3-2 Not supported AUTOSAR standard conform features

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

- Table 3-3 Features provided beyond the AUTOSAR standard

The following features specified in [1] are supported:

Supported AUTOSAR Standard Conform Features
Initialization: • FlexRay Communication Controller initialization • FlexRay Communication handling with startup, halt or abort communication
LPdu transmission: • Transmission • Dynamic payload handling • Transparent hardware buffer reconfiguration •
LPdu reception: • Reception • FIFO support
Status information: • Synchronization information • FlexRay time • Sync frame list • NM vector • Version info
Hardware Abstraction: • Handle two FlexRay CCs (E-Ray) • Disable LPdu • Reconfigure LPdu
Configuration variants: • Precompile • Linktime • Post-build Selectable

Supported AUTOSAR Standard Conform Features
• Post-build Loadable
Error detection and handling: • Dev error reporting • FlexRay controller hardware error with DEM reporting

Table 3-1 Supported AUTOSAR standard conform features

3.1.1 Deviations

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

Category	Description	Version
Functional	Implementation Requirements: Module vendor identification.	3.0.0
Functional	Extended Production Errors: The FlexRay protocol communication error which indicates errors on the network for a particular LPdu is not supported. (FrIfDemFTSlotStatusRef)	3.2.0
Functional	Indexing Scheme: Handling of two FlexRay CCs is only supported by E-Ray based platforms. Mfr based platforms can still only handle one FlexRay CC.	1.0.0

Table 3-2 Not supported AUTOSAR standard conform features

3.1.2 Additions/Extensions

The following features are provided beyond the AUTOSAR standard:

Features provided beyond the AUTOSAR standard
BSW debug parameters
Message ID filtering
Direct buffer access
AMD runtime measurement

Table 3-3 Features provided beyond the AUTOSAR standard

3.2 Initialization

Before the Fr can be used it has to be initialized by Fr_InitMemory() which initialize local variables and Fr_Init(&Fr_Config) which performs the basic initialization but does not enable the FlexRay communication.

All other API calls that are required for proper FlexRay communication are done by FrIf layer.

3.3 Configuration Variants

Fr supports the precompile time configuration variant, linktime configuration variant and postbuild configuration variant. Precompile is faster due to a direct data access. At postbuild variant the configuration data section can be (over-)flashed without compiling the application. Respectively the configuration data is then only accessible by pointer.

3.4 States

The FlexRay driver life cycle comprises three states: configuration, communication and end. Figure 3-1 shows the entire life cycle except interrupt and timer handling. Interrupts are handled during communication. The following interrupts are concerned:

- Absolute Timer Interrupt (depending on configuration)
- Cycle Start Interrupt (depending on configuration)
- Receive Interrupt (depending on configuration)
- Transmit Interrupt (depending on configuration)

3.5 Dual bus network usage

Fr supports two physical layer channels identified as channel A and channel B. When using the dual bus redundancy mode remember that the chronological first valid frame that reaches the CC on channel A or B is copied into the message buffer.

Caution
Please keep in mind that a Null Frame is also a valid frame regarding to [9].

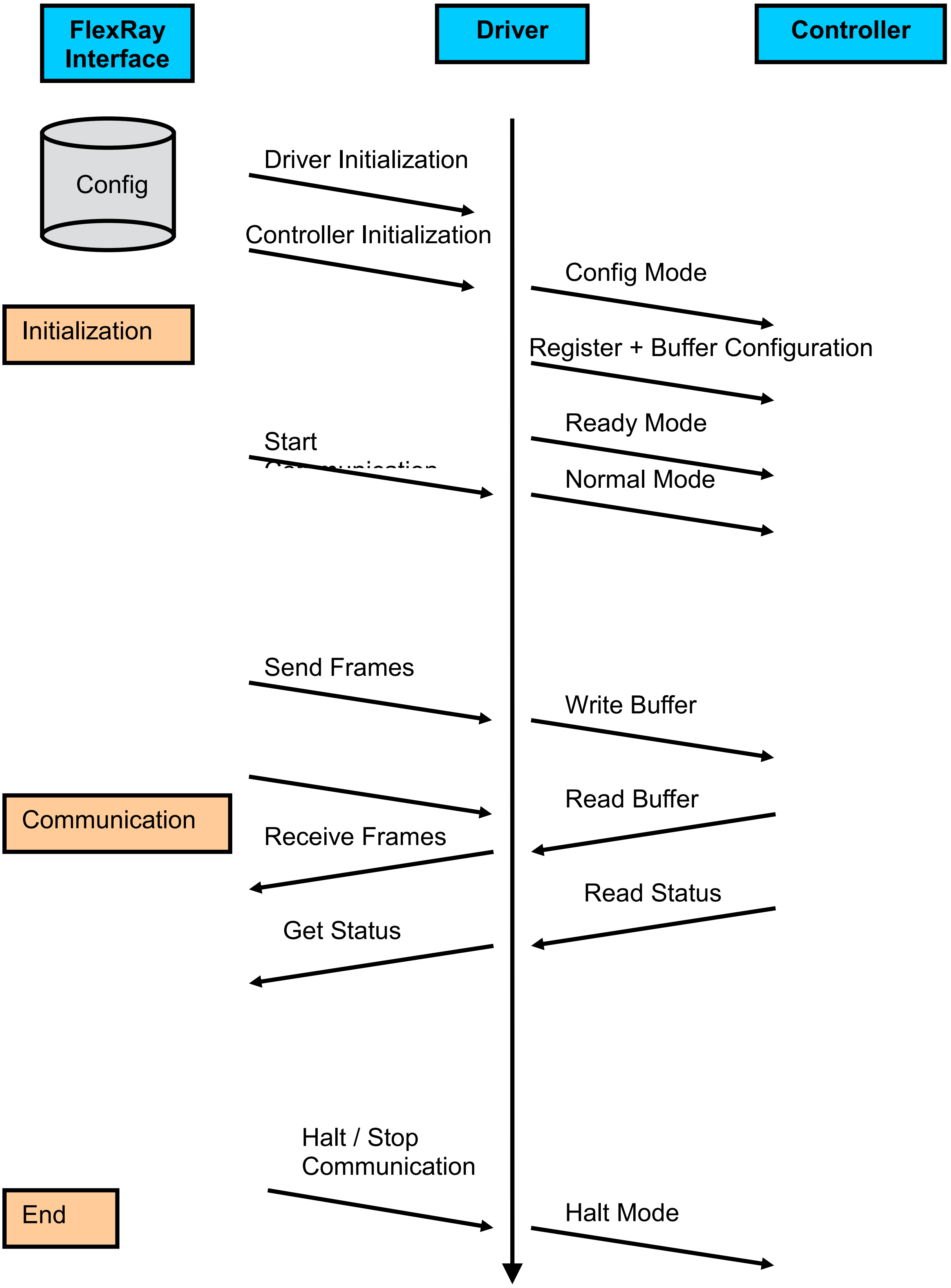


Figure 3-1 FlexRay Driver Sequence Diagram

3.6 Main Functions

After configuration the communication can be started and in case of success FlexRay frames can be sent and received using driver interfaces. The driver itself has no main function that needs to be called cyclically.

During bus operation the driver handles cycle start, timer and optional send / receives interrupts by clearing the interrupt flags and calling the interrupt specific handler routines.

In addition the driver offers interfaces to obtain the synchronization status of the FlexRay bus and the POC status.

The driver offers interfaces to handle the Absolute Timer.

3.7 Error Handling

3.7.1 Development Error Reporting

By default, development errors are reported to the DET using the service `Det_ReportError()as` specified in [3], if development error reporting is enabled (i.e. GenTool switch 'Dev Error Detect' is set).

If another module is used for development error reporting, the function prototype for reporting the error can be configured by the integrator, but must have the same signature as the service `Det_ReportError()`.

The reported FR ID is 81u.

The reported service IDs identify the services which are described in 5.3. The following table presents the service IDs and the related services:

Service ID	Service
0x00	FR_API_ID_CONTROLLER_INIT
0x03	FR_API_ID_START_COMMUNICATION
0x04	FR_API_ID_HALT_COMMUNICATION
0x05	FR_API_ID_ABORT_COMMUNICATION
0x06	FR_API_ID_SEND_WUP
0x07	FR_API_ID_SET_WAKEUP_CHANNEL
0x0A	FR_API_ID_GET_POC_STATUS
0x0B	FR_API_ID_TRANSMIT_TX_LPDU
0x0C	FR_API_ID_RECEIVE_RX_LPDU
0x0D	FR_API_ID_CHECK_TX_LPDU_STATUS
0x10	FR_API_ID_GET_GLOBAL_TIME
0x11	FR_API_ID_SET_ABSOLUTE_TIMER
0x13	FR_API_ID_CANCEL_ABSOLUTE_TIMER
0x15	FR_API_ID_ENABLE_ABSOLUTE_TIMER_IRQ
0x17	FR_API_ID_ACK_ABSOLUTE_TIMER_IRQ
0x19	FR_API_ID_DISABLE_ABSOLUTE_TIMER_IRQ

Service ID	Service
0x1B	FR_API_ID_GET_VERSION_INFO
0x1C	FR_API_ID_INIT
0x1F	FR_API_ID_PREPARE_LPDU
0x20	FR_API_ID_GET_ABSOLUTE_TIMER_IRQ_STATUS
0x22	FR_API_ID_GET_NM_VECTOR
0x23	FR_API_ID_ALLOW_COLDSTART
0x24	FR_API_ID_ALLSLOTS (optional)
0x25	FR_API_ID_RECONFIG_LPDU
0x26	FR_API_ID_DISABLE_LPDU
0x27	FR_API_ID_GETNUMOFSTARTUPFRAMES (optional)
0x28	FR_API_ID_GET_CHANNEL_STATUS
0x29	FR_API_ID_GET_CLOCK_CORRECTION
0x2A	FR_API_ID_GET_SYNC_FRAME_LIST(optional)
0x2B	FR_API_ID_GETWAKEUPRXSTATUS (optional)
0x2D	FR_API_ID_CANCELTXLPDU (optional)
0x2E	FR_API_ID_READCCCONFIG
0x30	FR_API_ID_REQUEST_BUFFER_DBA (optional)
0x31	FR_API_ID_TRANSMIT_TX_LPDU_DBA (optional)
0x32	FR_API_ID_RECEIVE_RX_LPDU_DBA (optional)
0x33	FR_API_ID_UNLOCK_RX_LPDU_DBA (optional)
0x34	FR_API_ID_LOCK_FTU (optional)
0x35	FR_API_ID_UNLOCK_FTU (optional)
0x36	FR_API_ID_TRANSMIT_TX_LPDU_IMMEDIATE_DBA (optional)

Table 3-4 Mapping of service IDs to services

The errors reported to DET are described in the following table:

Error Code		Description
0x01	FR_E_INV_TIMER_IDX	parameter timer index exceeds number of available timers
0x02	FR_E_INV_POINTER	invalid pointer in parameter list
0x03	FR_E_INV_OFFSET	parameter offset exceeds bounds
0x04	FR_E_INV_CTRL_IDX	invalid controller index
0x05	FR_E_INV_CHNL_IDX	invalid channel index
0x06	FR_E_INV_CYCLE	parameter cycle exceeds 63
0x08	FR_E_NOT_INITIALIZED	Fr module was not initialized
0x09	FR_E_INV_POCSTATE	Fr CC is not in the expected POC state.
0x0A	FR_E_INV_LENGTH	Payload length parameter has an invalid value.
0x0B	FR_E_INV_LPDU_IDX	invalid LPdu index

Error Code		Description
0x0C	FR_E_INV_HEADERCRC	Invalid header CRC
0x0D	FR_E_INV_CONFIG_IDX	Invalid value passed as parameter Fr_ConfigParamIdx.
0x40	FR_E_INV_LISTSIZE	Invalid Listsize in function Fr_GetSyncFrameList

Table 3-5 Errors reported to DET

3.7.1.1 Parameter Checking

The following table shows which parameter checks are performed on which services:

Info

Note that the FR_E_INV_CTRL_IDX error is only reported to DET by the FlexRay driver if the "Single Channel API" feature is disabled. Refer to section 3.11 how to disable the "Single Channel API" feature.

Check Service	FR_E_INV_TIMER_IDX	FR_E_INV_POINTER	FR_E_INV_OFFSET	FR_E_INV_LISTSIZE	FR_E_INV_CTRL_IDX	FR_E_INV_CHNL_IDX	FR_E_INV_CYCLE	FR_E_INV_HEADERCRC	FR_E_INV_CONFIG	FR_E_NOT_INITIALIZED	FR_E_INV_POCSTATE	FR_E_INV_LENGTH	FR_E_INV_LPDU_IDX
Fr_InitMemory													
Fr_Init		■											
Fr_ControllerInit					■				■	■			
Fr_AllSlots					■					■	■		
Fr_StartCommunication					■					■	■		
Fr_AllowColdstart					■					■			
Fr_HaltCommunication					■					■	■		
Fr_AbortCommunication					■					■			
Fr_SendWUP					■					■	■		
Fr_SetWakeupChannel					■	■				■	■		
Fr_GetPOCStatus		■			■					■			
Fr_GetNumOfStartupFrames		■			■					■			
Fr_GetWakeupRxStatus		■			■					■			
Fr_RequestBuffer_DBA		■			■					■		■	■

Check Service	FR_E_INV_TIMER_IDX	FR_E_INV_POINTER	FR_E_INV_OFFSET	FR_E_INV_LISTSIZE	FR_E_INV_CTRL_IDX	FR_E_INV_CHNL_IDX	FR_E_INV_CYCLE	FR_E_INV_HEADERCRC	FR_E_INV_CONFIG	FR_E_NOT_INITIALIZED	FR_E_INV_POCSTATE	FR_E_INV_LENGTH	FR_E_INV_LPDU_IDX
Fr_TransmitTxLPdu_DBA					■					■		■	■
Fr_ReceiveRxLPdu		■			■					■			■
Fr_ReceiveRxLPdu_DBA		■			■					■			■
Fr_TransmitTxLPdu_ImmediateDBA		■			■					■		■	■
Fr_TransmitTxLPdu		■			■					■		■	■
Fr_UnlockRxLPdu_DBA					■					■			■
Fr_CheckTxLPduStatus		■			■					■			■
Fr_GetSyncFrameList		■		■	■					■			
Fr_DisableLPdu					■					■			■
Fr_ReconfigLPdu					■	■	■	■		■		■	■
Fr_PrepareLPdu					■					■			
Fr_GetGlobalTime		■			■					■			
Fr_NmVectorPtr		■			■					■			
Fr_GetVersionInfo		■											
Fr_GetClockCorrection		■			■					■			
Fr_GetChannelStatus		■			■					■			
Fr_ReadCCConfig		■			■				■	■			
Fr_SetAbsoluteTimer	■		■		■		■			■	■		
Fr_CancelAbsoluteTimer	■				■					■			
Fr_EnableAbsoluteTimerIRQ	■				■					■			
Fr_AckAbsoluteTimerIRQ	■				■					■			
Fr_DisableAbsoluteTimerIRQ	■				■					■			
Fr_GetAbsoluteTimerIRQStatus	■	■			■					■			

3.7.2 Production Code Error Reporting

By default, production code related errors are reported to the DEM using the service `Dem_ReportErrorStatus()` as specified in [6], if production error reporting is enabled (i.e. GenTool switch ‘Prod Error Detect’ is set).

If another module is used for production code error reporting, the function prototype for reporting the error can be configured by the integrator, but must have the same signature as the service `Dem_ReportErrorStatus()`.

The errors reported to DEM are described in the following table:

Error Code	Description
FrDemCtrlTestResultRef	Access to FlexRay CC event ID

Table 3-6 Errors reported to DEM

3.8 Buffer Reconfiguration

The FlexRay Communication Controllers (CC) that are currently available, only offer a limited amount of message buffers. A FlexRay Schedule with a large number of frames might exceed the number of available message buffers.

The buffer reconfiguration feature reduces the amount of hardware message buffers that are used by the FR. This is done by changing the message buffer configuration during runtime.

The buffer reconfiguration can be enabled with the GenTool option “Enable Buffer Reconfiguration” (MICROSAR PARAMETER DEFINITION: 'FrEnableBufferReconfig').

Caution
When a frame f1 cannot be transmitted because it has been preempted by other frames (i.e. latestTx has been exceeded) and another frame f2 shall be transmitted in the next cycle using the same message buffer, the message buffer will be reconfigured. Hence f1 will not be transmitted.

Info
The reconfiguration of the message buffer is done at the runtime. Note that this feature uses more processor resources.

3.9 Reconfig LPdu Support

The Autosar 4.0 feature “Reconfig LPdu Support” (MICROSAR PARAMETER DEFINITION: 'FrReconfigLPduSupport') allows to reconfigure LPDUs at runtime. However, there can be some restrictions based on hardware properties.

1) It is not allowed to reconfigure a buffer with larger payload as it was configured at startup of ECU.

2) It is not allowed to reconfigure FIFO buffers or to configure a buffer into a FIFO range.
3) Depending on hardware and cluster It is not allowed to configure the sync frame

The reconfigurable LPDUs are disabled by default in Fr_ControllerInit. Disable means that LPDUs were not transmitted nor received. This is done to avoid collisions as more than one ECU can use the same LPDU for transmission. To avoid reception of LPDUs without explicit knowledge of the transmitter, receive LPDu are also disabled at initialization of CC. The LPDUs have to be manually enabled by API “Fr_ReconfigLPdu” (see 5.3.21)

3.10 Dynamic Payload

The payload of frames that was selected as “dynamic payload” within FrIf module is change by the FR at the runtime (refer to [10] for how to set “dynamic payload”). The FR updates the buffer configuration with the new payload and new calculated header CRC at runtime.

Caution
The values of the new payload shall be smaller or equal as the payload value that was define within FIBEX or EcuC.

Info
The calculation of the header CRC is done at the runtime. Note that this feature uses more processor resources

3.11 Single Channel API

According to the FlexRay controller driver software specification every FlexRay controller driver service contains a controller or timer handle.

If only one FlexRay controller driver instance is used this handle is not necessary, except for development error detection. Therefore, if only one FlexRay controller driver instance is used, the handle can be removed by a macro. These macros still expect channel handles, which remain unused.

The described prototypes in chapter 5 are written similar to the functions in the corresponding C file. In the C-File the macro `FR_VCTRL_SYSTEMTYPE_ONLY` is used for the single channel API feature. In the H- file the same function has the prototype with the macro `FR_VCTRL_SYSTEMTYPE_FIRST`.

If the feature ‘Single Channel API’ is not used, the macros are replaced with the Autosar prototype ‘`uint8 Fr_CtrlIdx’`.

If the feature ‘Single Channel API’ is used, the two macros are left empty and the code is smaller and faster in this module and also in the modules uses the Fr.

3.12 Direct Buffer Access

Without direct buffer access, received and transmitted FlexRay data are copied from Fr driver into RAM where CC reads data from or writes data to.

Some CC provides the possibility that FlexRay data is located in host RAM and can be accessed directly from application. To save execution time, the goal of direct buffer access is to save the copy of Fr-data from RAM to RAM in Fr driver and provide the possibility, that upper layers copy their data directly to RAM where CC reads/writes from.

Depending on CC, access to the RAM and additional necessary functionality (like locking of RAM) are described in [8].

3.13 Hardware Loop with Cancellation

The FlexRay driver sends commands which sometimes require confirmation to go on. It will enter a waiting loop (i.e. hardware loop). To avoid an infinite waiting time (e.g. caused by hardware defects), only a limited amount of loops will be performed. That means that after expiration of a configurable time the loop will be breaked. The actual executed function then exits with a DEM notification.

For the determination of an adequate timeout value refer to [8].

3.14 FIFO reception

FIFO is used to define Slot ID ranges for receive frames. All Slot ID within the Slot ID range are received from driver. That can be used to received frames with the same use-case e.g. network management frames. The driver supports more than one FIFO configuration set.

Please ensure that your configuration meets the following requirements:

- the FIFO needs at least one Rx frame triggering within the range
- one FIFO range supports either channel A or channel B. It is not possible to configure a FIFO range for both channels
- All frames within a FIFO range must have the same frame type (i.e. only NM frames or only TP frames)
- the FIFO ranges that are defined for one channel must not overlap
- payload length of the frames that are received by FIFO must be identical
- FIFO depth must not be greater than 255, i.e. RangeMax - RangeMin must not be greater than 254

3.15 Reading out the Flexray parameters (Read CC Parameters)

With activation of this feature the FlexRay low level parameters (see [2] for coverage) can be read and verified at runtime. The FlexRay driver offers an API `Fr_ReadCCConfig()` to read the FlexRay low level parameters. Refer to [2] for detail description.

The following parameters are intended to be used in FlexRay Protocol 3.0 and will return 0 for FlexRay Protocol 2.1 Rev A compliance:

- PSecondKeySlotId
- GCycleCountMax
- GdSymbolWindowActionPointOffset
- GdIgnoreAfterTx
- PExternalSync
- PFallBackInternal
- PKeySlotOnlyEnabled
- PNmVectorEarlyUpdate
- PTwoKeySlotMode

!

Caution
The value of parameter GdBit is returned as an enumeration value instead numerical value as defined in [2]. The reason for this behaviour is the still open Issue 49741 at please refer to [16]. The enumeration values correlation: 0→T100NS; 1→T200NS; 2→T400NS.

4 Integration

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

4.1 Scope of Delivery

The delivery of the FR contains the files which are described in the chapters 4.1.1 and 4.1.2:

4.1.1 Static Files

File Name	Description
Fr.c	Main Module contains the main driver functionality
Fr.h	AUTOSAR driver API
Fr_Timer.c	Timer handling contains timer related functionality
Fr_Irq.c	References of the Interrupt service routínes
Fr_Ext.h	Driver callback functions, additional function driver API
Fr_Priv.h	Macros
Fr_<CC>.h	Macros and Defines for CC
Fr_GeneralTypes.h	Datatype definitions according to AUTOSAR

Table 4-1 Static files

4.1.2 Dynamic Files

The dynamic files are generated by the GenTool Cfg5

File Name	Description
Fr_Cfg.h	General configuration data
Fr_Lcfg.c	Linker configuration data
Fr_PBcfg.c	Postbuild configuration data

Table 4-2 Generated files

4.2 Include Structure

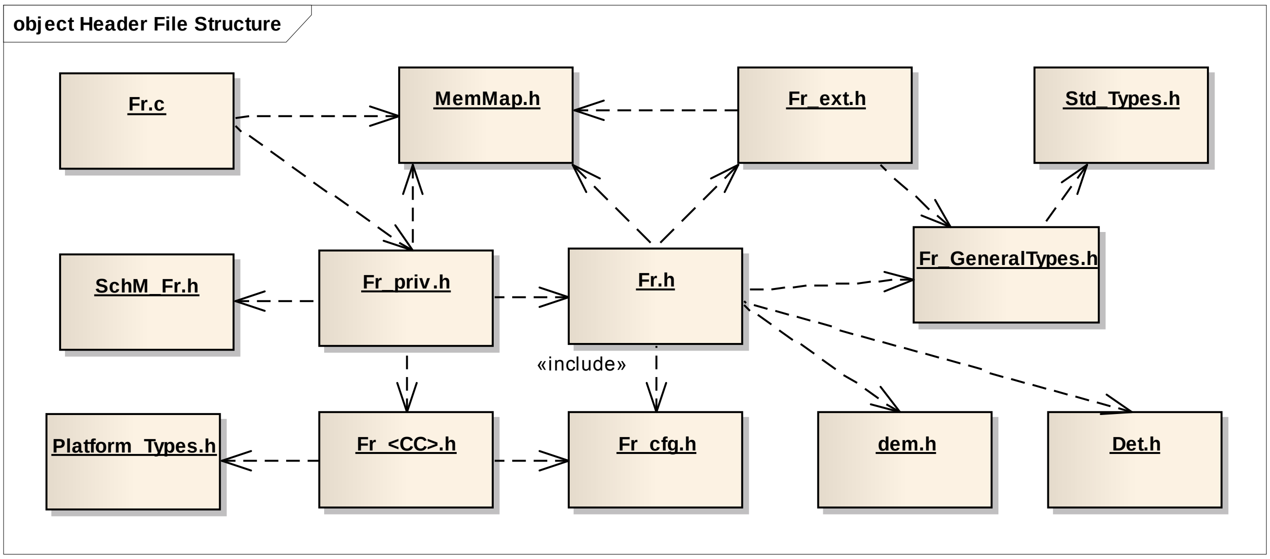


Figure 4-1 Include structure

4.3 Compiler Abstraction and Memory Mapping

The objects (e.g. variables, functions, constants) are declared by compiler independent definitions – the compiler abstraction definitions. Each compiler abstraction definition is assigned to a memory section.

For a detailed table please refer to [8].

4.4 Interrupt Handling

The interrupts are to be enabled by OS or the application. The following interrupts are supported by Fr:

- Timer0 (Absolute-Timer) Interrupt (must be enabled if FrIf is used),
- Cyclestart Interrupt (depending on configuration).

Caution
The Timer0 (Absolute-Timer) Interrupt is mandatory for the FlexRay functionality if FrIf is used.

Depending on the used CC module the interrupt service routines have specific names. Their implementation is provided by Fr_Irq.c.

The Fr provides optional callout functions triggered by Timer1- resp. Cyclestart Interrupt. At usage they may be implemented by the application. Prototypes of these functions are defined in Fr_Ext.h. Please refer to chapter “API Description” for more details.

4.5 Critical Sections

Special attention is given to critical code sections, that must not be interrupted. For details see [8].

5 API Description

5.1 Type Definitions

The software module FlexRay Driver uses the standard AUTOSAR data types that are defined within Std_Types.h and the platform specific data types that are defined within Platform_Types.h.

If the FlexRay Driver Interface is used the Communication Stack Types defined within ComStack_Types.h are used too.

For specific type definitions please refer to [8].

5.2 Interrupt Service Routines provided by FR

The GenTool option “Type of interrupt function” has to be adapted accordingly to the usage of the FlexRay driver interrupt services. If OS is used, the interrupts are of type “Category 2”. CC specific interrupt routine names have to be entered in OS configurator.

Otherwise on usage of “void func(void)” the addresses of the interrupt routines have to be entered in interrupt table or interrupt dispatcher. For details please refer to [8].

Category 1 interrupts are not allowed.

For a detailed description of Interrupt types please refer to see [15].

Caution
If “void func(void)” is used, secure that the interrupt function is not executed within a category 1 interrupt. Further it is possible to call a “void func(void)” within a user task (not recommended) or an interrupt of type category 2.

Caution
The implementation is hardware specific. For details please refer to [8].

5.3 Services provided by FR

The FR API consists of services, which are realized by function calls.

5.3.1 Fr_InitMemory

Prototype	
FUNC(void, FR_CODE) **Fr_InitMemory** (void);	
Parameter	
-	-
Return code	
-	-
Functional Description	
This method initializes the global variables of the module. It shall be called first of all services.	
Particularities and Limitations	
▶ Particularities, limitations, post-conditions, pre-conditions	
Expected Caller Context	
▶ Asynchronous	

Table 5-1 Fr_InitMemory

5.3.2 Fr_Init

Prototype	
FUNC(void, FR_CODE) **Fr_Init**(P2CONST(Fr_ConfigType, AUTOMATIC, FR_PBCFG) Fr_ConfigPtr);	
Parameter	
Fr_ConfigPtr	Pointer to post-build configuration. If Variant 1 (Pre-compile Configuration) is used, the parameter is ignored and a Null pointer can be passed.
Return code	
-	
Functional Description	
This method initializes the access to the Post Build Configuration. A pointer to the configuration gets passed in and is stored. This API must be called when the FlexRay Stack gets started or if a new configuration gets flashed	
Particularities and Limitations	
▶ Particularities, limitations, post-conditions, pre-conditions Fr_InitMemory is already executed.	
Expected Caller Context	
▶ Asynchronous	

Table 5-2 Fr_Init

5.3.3 Fr_ControllerInit

<table>
<tr><th colspan="2">Prototype</th></tr>
<tr><td colspan="2">FUNC(Std_ReturnType, FR_CODE) <b>Fr_ControllerInit</b> (
FR_VCTRL_SYSTEMTYPE_ONLY);</td></tr>
<tr><th colspan="2">Parameter</th></tr>
<tr><td>FR_VCTRL_SYSTEMTYPE_ONLY</td><td>Depending on preconfiguration parameter ‘DrvFrBaseApiOptimization’ this macro is either empty or contains the value for uint8 Fr_CtrlIdx. Fr_CtrlIdx is a zero based index identifies the CC controller for which the API call has to be applied.</td></tr>
<tr><th colspan="2">Return code</th></tr>
<tr><td>Std_ReturnType</td><td>E_OK: API call finished successfully.
E_NOT_OK: API call aborted due to errors.</td></tr>
<tr><th colspan="2">Functional Description</th></tr>
<tr><td colspan="2">This method configures the Communication Controller (CC). Initially, the CC will be set into Config State. The internal RAM gets cleared and configured with the content of the Post Build Configuration into the CC register and the frame buffers. Finally the CC is put into Cold Start State preparing the start of FlexRay communication.</td></tr>
<tr><th colspan="2">Particularities and Limitations</th></tr>
<tr><td colspan="2">▶ Particularities, limitations, post-conditions, pre-conditions
It is required that a valid configuration is set using function ‘Fr_Init’.</td></tr>
<tr><td colspan="2">Expected Caller Context</td></tr>
<tr><td colspan="2">▶ Asynchronous</td></tr>
</table>

Table 5-3 Fr_ControllerInit

5.3.4 Fr_AllSlots

<table>
<tr><th colspan="2">Prototype</th></tr>
<tr><td colspan="2">FUNC(Std_ReturnType, FR_CODE) <b>Fr_AllSlots</b> (
FR_VCTRL_SYSTEMTYPE_ONLY)</td></tr>
<tr><th colspan="2">Parameter</th></tr>
<tr><td>FR_VCTRL_SYSTEMTYPE_ONLY</td><td>Depending on preconfiguration parameter ‘DrvFrBaseApiOptimization’ this macro is either empty or contains the value for uint8 Fr_CtrlIdx. Fr_CtrlIdx is a zero based index identifies the CC controller for which the API call has to be applied.</td></tr>
<tr><th colspan="2">Return code</th></tr>
<tr><td>Std_ReturnType</td><td>E_OK: API call finished successfully.
E_NOT_OK: API call aborted due to errors.</td></tr>
<tr><th colspan="2">Functional Description</th></tr>
<tr><td colspan="2">Invokes the CC CHI command 'ALL_SLOTS'.</td></tr>
<tr><th colspan="2">Particularities and Limitations</th></tr>
<tr><td colspan="2">▶ Particularities, limitations, post-conditions, pre-conditions
The method is only available with enabled GenTool FrIf option “All Slots Support”. It is required that Fr_Init() is called before.</td></tr>
</table>

Expected Caller Context
▶ Asynchronous

Table 5-4 Fr_AllSlots

5.3.5 Fr_StartCommunication

Prototype	

`FUNC(Std_ReturnType, FR_CODE) `**`Fr_StartCommunication`**` (`
`FR_VCTRL_SYSTEMTYPE_ONLY)`	

Parameter	
FR_VCTRL_SYSTEMTYPE_ONLY	Depending on preconfiguration parameter 'DrvFrBaseApiOptimization' this macro is either empty or contains the value for uint8 Fr_CtrlIdx. Fr_CtrlIdx is a zero based index identifies the CC controller for which the API call has to be applied.
Return code	
Std_ReturnType	E_OK: API call finished successfully. E_NOT_OK: API call aborted due to errors.
Functional Description	
This method starts the FlexRay communication. The POC state changes from FR_POCSTATE_READY state into FR_POCSTATE_STARTUP state	
Particularities and Limitations	
▶ Particularities, limitations, post-conditions, pre-conditions It is required that the Controller is in Ready state by previous call of Fr_ControllerInit.	
Expected Caller Context	
▶ Asynchronous	

Table 5-5 Fr_StartCommunication

5.3.6 Fr_HaltCommunication

Prototype	

`FUNC(Std_ReturnType, FR_CODE) `**`Fr_HaltCommunication`**` (`
`FR_VCTRL_SYSTEMTYPE_ONLY)`	

Parameter	
FR_VCTRL_SYSTEMTYPE_ONLY	Depending on preconfiguration parameter 'DrvFrBaseApiOptimization' this macro is either empty or contains the value for uint8 Fr_CtrlIdx. Fr_CtrlIdx is a zero based index identifies the CC controller for which the API call has to be applied.
Return code	
Std_ReturnType	E_OK: API call finished successfully. E_NOT_OK: API call aborted due to errors.
Functional Description	
This method halts the specified Communication Controller at the end of the current communication cycle.	

Particularities and Limitations	
▶ Particularities, limitations, post-conditions, pre-conditions The physical layer is in sync with the network.	
Expected Caller Context	
▶ Asynchronous	

Table 5-6 Fr_HaltCommunication

5.3.7 Fr_AbortCommunication

Prototype	
FUNC(Std_ReturnType, FR_CODE) **Fr_AbortCommunication** (FR_VCTRL_SYSTEMTYPE_ONLY)	
Parameter	
FR_VCTRL_SYSTEMTYPE_ONLY	Depending on preconfiguration parameter 'DrvFrBaseApiOptimization' this macro is either empty or contains the value for uint8 Fr_CtrlIdx. Fr_CtrlIdx is a zero based index identifies the CC controller for which the API call has to be applied.
Return code	
Std_ReturnType	E_OK: API call finished successfully. E_NOT_OK: API call aborted due to errors.
Functional Description	
This method halts the specified Communication Controller independent of its current state	
Particularities and Limitations	
▶ Particularities, limitations, post-conditions, pre-conditions This API can be removed from code depending on the GenTool configuration.	
Expected Caller Context	
▶ Asynchronous	

Table 5-7 Fr_AbortCommunication

5.3.8 Fr_AllowColdstart

Prototype	
FUNC(Std_ReturnType, FR_CODE) **Fr_AllowColdstart** (FR_VCTRL_SYSTEMTYPE_ONLY)	
Parameter	
FR_VCTRL_SYSTEMTYPE_ONLY	Depending on preconfiguration parameter 'DrvFrBaseApiOptimization' this macro is either empty or contains the value for uint8 Fr_CtrlIdx. Fr_CtrlIdx is a zero based index identifies the CC controller for which the API call has to be applied.
Return code	
Std_ReturnType	E_OK: API call finished successfully. E_NOT_OK: API call aborted due to errors.

Functional Description
This method invokes the CC CHI command 'ALLOW_COLDSTART'
Particularities and Limitations
▶ Particularities, limitations, post-conditions, pre-conditions The CC is in any POCState except POC:default config, POC:config or POC:halt.
Expected Caller Context
▶ Asynchronous

Table 5-8 Fr_AllowColdstart

5.3.9 Fr_SendWUP

Prototype	
FUNC(Std_ReturnType, FR_CODE) **Fr_SendWUP** (FR_VCTRL_SYSTEMTYPE_ONLY)	
Parameter	
FR_VCTRL_SYSTEMTYPE_ONLY	Depending on preconfiguration parameter 'DrvFrBaseApiOptimization' this macro is either empty or contains the value for uint8 Fr_CtrlIdx. Fr_CtrlIdx is a zero based index identifies the CC controller for which the API call has to be applied.
Return code	
Std_ReturnType	E_OK: API call finished successfully. E_NOT_OK: API call aborted due to errors.
Functional Description	
This method sends the Wakeup pattern.	
Particularities and Limitations	
▶ Particularities, limitations, post-conditions, pre-conditions It is required that the Controller is in Ready state by previous call of Fr_ControllerInit.	
Expected Caller Context	
▶ Asynchronous	

Table 5-9 Fr_SendWUP

5.3.10 Fr_SetWakeupChannel

Prototype	

FUNC(Std_ReturnType, FR_CODE) **Fr_SetWakeupChannel** (
  FR_VCTRL_SYSTEMTYPE_ONLY,
  Fr_ChannelType Fr_ChnlIdx)	

Parameter	
FR_VCTRL_SYSTEMTYPE_ONLY	Depending on preconfiguration parameter 'DrvFrBaseApiOptimization' this macro is either empty or contains the value for uint8 Fr_CtrlIdx. Fr_CtrlIdx is a zero based index identifies the CC controller for which the API call has to be applied.
Fr_ChnlIdx	Index of the FlexRay channel

Return code	
Std_ReturnType	E_OK: API call finished successfully. E_NOT_OK: API call aborted due to errors.
Functional Description	
This method activates the given channel for sending the Wakeup pattern. The Wakeup pattern can not be sent on both Channels simultaneous. Trying issues a DET error.	
Particularities and Limitations	
▶ Particularities, limitations, post-conditions, pre-conditions It is required that the Controller is in Ready state. This API can be removed from code depending on the GenTool configuration.	
Expected Caller Context	
▶ Asynchronous	

Table 5-10 Fr_SetWakeupChannel

5.3.11 Fr_GetPOCStatus

Prototype	

`FUNC(Std_ReturnType, FR_CODE)` **`Fr_GetPOCStatus`**`(`
`  FR_VCTRL_SYSTEMTYPE_ONLY,`
`  P2VAR(Fr_POCStatusType, AUTOMATIC, FR_APPL_DATA) Fr_POCStatusPtr)`	

Parameter	
FR_VCTRL_SYSTEMTYPE_ONLY	Depending on preconfiguration parameter ‘DrvFrBaseApiOptimization’ this macro is either empty or contains the value for uint8 Fr_CtrlIdx. Fr_CtrlIdx is a zero based index identifies the CC controller for which the API call has to be applied.
Fr_POCStatusPtr	Current POC status information
Return code	
Std_ReturnType	E_OK: API call finished successfully. E_NOT_OK: API call aborted due to errors.
Functional Description	
This method gives back the POC status	
Particularities and Limitations	
▶ Particularities, limitations, post-conditions, pre-conditions	
Expected Caller Context	
▶ Synchronous	

Table 5-11 Fr_GetPOCStatus

5.3.12 Fr_TransmitTxLPdu

Prototype	

`FUNC(Std_ReturnType, FR_CODE) Fr_TransmitTxLPdu(`
`  FR_VCTRL_SYSTEMTYPE_ONLY,`
`  uint16 Fr_LPduIdx,`
`  P2CONST(uint8, AUTOMATIC, FR_APPL_DATA) Fr_LPduPtr,`
`  uint8 Fr_LPduLength)`	

Parameter	
FR_VCTRL_SYSTEMTYPE_ONLY	Depending on preconfiguration parameter 'DrvFrBaseApiOptimization' this macro is either empty or contains the value for uint8 Fr_CtrlIdx. Fr_CtrlIdx is a zero based index identifies the CC controller for which the API call has to be applied.
Fr_LPduIdx	This index is used to uniquely identify a FlexRay frame
Fr_LPduPtr	This reference points to a buffer where the assembled LSdu to be transmitted within this LPdu is stored at.
Fr_LPduLength	Determines the length of the data (in Bytes) to be transmitted. The value is not modified by this method
Return code	
Std_ReturnType	E_OK: API call finished successfully. E_NOT_OK: API call aborted due to errors.
Functional Description	
This method sends a frame on the bus thus the bus must be initialized. The method figures out the physical buffer where the LPdu should be copied to. The method copies the data to that physical buffer and activates it for transmission. This method reconfigures the message buffer in case dynamic payload or buffer reconfigure is enabled.	
Particularities and Limitations	
▶ Particularities, limitations, post-conditions, pre-conditions It is required that the physical layer is in sync with the network.	
Expected Caller Context	
▶ Synchronous	

Table 5-12 Fr_TransmitTxLPdu

Caution

The method does not check whether the "old" data is successfully send in the last configured frame. The message buffer is always overwrite with the new requested data rather frame in case enabled buffer reconfigure.

5.3.13 Fr_ReceiveRxLPdu

Prototype

```
FUNC(Std_ReturnType, FR_CODE) Fr_ReceiveRxLPdu(
  FR_VCTRL_SYSTEMTYPE_ONLY,
  uint16 Fr_LPduIdx,
  P2VAR(uint8, AUTOMATIC, FR_APPL_DATA) Fr_LPduPtr,
  P2VAR(Fr_RxLPduStatusType, AUTOMATIC, FR_APPL_DATA) Fr_LPduStatusPtr,
  P2VAR(uint8, AUTOMATIC, FR_APPL_DATA) Fr_LPduLengthPtr)
```

Parameter	
FR_VCTRL_SYSTEMTYPE_ONLY	Depending on preconfiguration parameter ‘DrvFrBaseApiOptimization’ this macro is either empty or contains the value for uint8 Fr_CtrlIdx. Fr_CtrlIdx is a zero based index identifies the CC controller for which the API call has to be applied.
Fr_LPduIdx	This index is used to uniquely identify a FlexRay frame.
Fr_LPduPtr	This reference points to the buffer where the LPdu 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_LPduLengthPtr	This reference points to the memory location where the length of the LPdu (in bytes) shall be stored.

Return code	
Std_ReturnType	E_OK: API call finished successfully. E_NOT_OK: API call aborted due to errors.

Functional Description

This method figures out if a new FlexRay frame identified by Fr_LPduIdx was received. It copies the payload to data of Fr_LPduPtr. The length parameter specifies the length in bytes whereas FlexRay is using 2-byte lengths.

The method returns “`FR_RECEIVED_MORE_DATA_AVAILABLE`” at the “`Fr_LPduStatusPtr`” in case the requested frame is placed within a FIFO range and the FIFO is not empty.

Particularities and Limitations

- Particularities, limitations, post-conditions, pre-conditions

Expected Caller Context

- Synchronous

Table 5-13 Fr_ReceiveRxLPdu

5.3.14 Fr_CancelTxLPdu

Prototype

```
FUNC(Std_ReturnType, FR_CODE) Fr_CancelTxLPdu(
  FR_VCTRL_SYSTEMTYPE_ONLY,
  uint16 Fr_LPduIdx)
```

Parameter	
FR_VCTRL_SYSTEMTYPE_ONLY	Depending on preconfiguration parameter 'Single Channel API' this macro is either empty or contains the value for uint8 Fr_CtrlIdx. Fr_CtrlIdx is a zero based index identifies the CC controller for which the API call has to be applied.
Fr_LPduIdx	This index is used to uniquely identify a FlexRay frame
Return code	
Std_ReturnType	E_OK: API call finished successfully. E_NOT_OK: API call aborted due to errors.
Functional Description	
Cancels the already pending transmission of a LPdu contained in a controllers physical transmit resource (e.g. message buffer).	
Particularities and Limitations	
> Particularities, limitations, post-conditions, pre-conditions The method is only available with enabled GenTool FrIf option "Cancel Transmit Support". It is required that Fr_Init() is called before.	
Expected Caller Context	
▶ Synchronous	

Table 5-14 Fr_CancelTxLPdu

5.3.15 Fr_CheckTxLPduStatus

Prototype	

`FUNC(Std_ReturnType, FR_CODE)` **`Fr_CheckTxLPduStatus`**`(`
`FR_VCTRL_SYSTEMTYPE_ONLY,`
`uint16 Fr_LPduIdx,`
`P2VAR(Fr_TxLPduStatusType, AUTOMATIC, FR_APPL_DATA) Fr_LPduStatusPtr)`	

Parameter	
FR_VCTRL_SYSTEMTYPE_ONLY	Depending on preconfiguration parameter 'Single Channel API' this macro is either empty or contains the value for uint8 Fr_CtrlIdx. Fr_CtrlIdx is a zero based index identifies the CC controller for which the API call has to be applied.
Fr_LPduIdx	This index is used to uniquely identify a FlexRay frame
Fr_LPduStatusPtr	This reference is used to store the transmit status of the LSdu
Return code	
Std_ReturnType	E_OK: API call finished successfully. E_NOT_OK: API call aborted due to errors.
Functional Description	
This method checks if previous send request succeeded. In addition this method checks the configuration of the message buffer in case buffer reconfiguration is enabled. The method returns E_NOT_OK and FR_NOT_TRANSMITTED in case the message buffer is already used for other frame.	
Particularities and Limitations	
▶ -	

Expected Caller Context
▶ Synchronous

Table 5-15 Fr_CheckTxLPduStatus

5.3.16 Fr_GetGlobalTime

Prototype	

`FUNC(Std_ReturnType, FR_CODE) **Fr_GetGlobalTime**(`
`FR_VCTRL_SYSTEMTYPE_ONLY,`
`P2VAR(uint8,  AUTOMATIC, FR_APPL_DATA) Fr_CyclePtr,`
`P2VAR(uint16, AUTOMATIC, FR_APPL_DATA) Fr_MacroTickPtr)`	

Parameter	
FR_VCTRL_SYSTEMTYPE_ONLY	Depending on preconfiguration parameter 'DrvFrBaseApiOptimization' this macro is either empty or contains the value for uint8 Fr_CtrlIdx. Fr_CtrlIdx is a zero based index identifies the CC controller for which the API call has to be applied.
Fr_CyclePtr	Buffer for current Cycle Counter
Fr_MacroTickPtr	Buffer for current Macrotick Counter
Return code	
Std_ReturnType	E_OK: API call finished successfully. E_NOT_OK: API call aborted due to errors.
Functional Description	
This method returns the current time specified in Cycle Counts and Macroticks if the bus is running (SYNC) i.e. Normal Active.	
Particularities and Limitations	
▶ Particularities, limitations, post-conditions, pre-conditions It is required that the physical layer is in sync with the network.	
Expected Caller Context	
▶ Synchronous	

Table 5-16 Fr_GetGlobalTime

5.3.17 Fr_GetNmVector

Prototype	

`FUNC(Std_ReturnType, FR_CODE) **Fr_GetNmVector**(`
`FR_VCTRL_SYSTEMTYPE_ONLY,`
`P2VAR(uint8, AUTOMATIC, FR_APPL_DATA) Fr_NmVectorPtr)`	

Parameter	
FR_VCTRL_SYSTEMTYPE_ONLY	Depending on preconfiguration parameter 'DrvFrBaseApiOptimization' this macro is either empty or contains the value for uint8 Fr_CtrlIdx. Fr_CtrlIdx is a zero based index identifies the CC controller for which the API call has to be applied.
Fr_NmVectorPtr	Address where the NmVector of the last communication cycle shall be stored.

Return code	
Std_ReturnType	E_OK: API call finished successfully. E_NOT_OK: API call aborted due to errors.
Functional Description	
Gets the network management vector of the last communication cycle	
Particularities and Limitations	
▶ Particularities, limitations, post-conditions, pre-conditions GenTool option 'NM Vector Support' is enabled. It is required that the physical layer is in sync with the network.	
Expected Caller Context	
▶ Synchronous	

Table 5-17 Fr_GetNmVector

5.3.18 Fr_GetVersionInfo

Prototype	

`FUNC(void, FR_CODE)` **`Fr_GetVersionInfo`** `(`
`  P2VAR(Std_VersionInfoType, AUTOMATIC, FR_APPL_DATA) VersionInfo )`	

Parameter	
VersionInfo	Pointer to where to store the version information of this module.
Return code	
-	
Functional Description	
This method returns the version info.	
Particularities and Limitations	
▶ Particularities, limitations, post-conditions, pre-conditions GenTool option 'Version Info Api' is switched on and 'Version Info Api As Macro' is switched off	
Expected Caller Context	
▶ Synchronous	

Table 5-18 Fr_GetVersionInfo

5.3.19 Fr_GetSyncFrameList

Prototype

`FUNC(Std_ReturnType, FR_CODE)` **`Fr_GetSyncFrameList`**`(`
`  FR_VCTRL_SYSTEMTYPE_ONLY,`
`  uint8 Fr_ListSize,`
`  (uint16, AUTOMATIC, FR_APPL_DATA) Fr_ChannelAEvenListPtr,`
`  (uint16, AUTOMATIC, FR_APPL_DATA) Fr_ChannelBEvenListPtr,`
`  (uint16, AUTOMATIC, FR_APPL_DATA) Fr_ChannelAOddListPtr,`
`  (uint16, AUTOMATIC, FR_APPL_DATA) Fr_ChannelBOddListPtr )`

Parameter	
FR_VCTRL_SYSTEMTYPE_ONLY	Depending on preconfiguration parameter 'DrvFrBaseApiOptimization' this macro is either empty or contains the value for uint8 Fr_CtrlIdx. Fr_CtrlIdx is a zero based index identifies the CC controller for which the API call has to be applied.
Fr_ListSize	Size of the parameter arrays
Fr_ChannelAEvenListPtr	the list of sync frames on channel A within the even communication cycle
Fr_ChannelBEvenListPtr	the list of sync frames on channel B within the even communication cycle
Fr_ChannelAOddListPtr	the list of sync frames on channel A within the odd communication cycle
Fr_ChannelBOddListPtr	the list of sync frames on channel B within the odd communication cycle
Return code	
Std_ReturnType	E_OK: API call finished successfully. E_NOT_OK: API call aborted due to errors.
Functional Description	
This method reads the list of sync frames received in the last communication cycle and write it as array to the appropriate array. A maximum of 15 for parameter Fr_ListSize is used if Fr_ListSize is greater than 15. In this case additionally a DET error (FR_E_INV_LISTSIZE) is thrown if DET is enabled.	
Particularities and Limitations	
► Particularities, limitations, post-conditions, pre-conditions It is required that the physical layer is in sync with the network.	
Expected Caller Context	
► Synchronous	

Table 5-19 Fr_GetSyncFrameList

5.3.20 Fr_DisableLPdu (optional)

Prototype	
FUNC(Std_ReturnType, FR_CODE) **Fr_DisableLPdu** (FR_VCTRL_SYSTEMTYPE_ONLY, uint16 Fr_LPduIdx)	
Parameter	
FR_VCTRL_SYSTEMTYPE_ONLY	Depending on preconfiguration parameter 'DrvFrBaseApiOptimization' this macro is either empty or contains the value for uint8 Fr_CtrlIdx. Fr_CtrlIdx is a zero based index identifies the CC controller for which the API call has to be applied.
Fr_LPduIdx	This index is used to uniquely identify a FlexRay frame.
Return code	
Std_ReturnType	E_OK: API call finished successfully. E_NOT_OK: API call aborted due to errors.
Functional Description	
This method configures the corresponding HW-buffer in a way that it does not take part in the FlexRay communication.	

Particularities and Limitations
▶ Particularities, limitations, post-conditions, pre-conditions It is required that the physical layer is in sync with the network.
Expected Caller Context
▶ Synchronous

Table 5-20 Fr_DisableLPdu

5.3.21 Fr_ReconfigLPdu (optional)

Prototype

```
FUNC(Std_ReturnType, FR_CODE) Fr_ReconfigLPdu (
  FR_VCTRL_SYSTEMTYPE_ONLY,
  uint16 Fr_LPduIdx,
  uint16 Fr_FrameId,
  Fr_ChannelType Fr_ChnlIdx,
  uint8 Fr_CycleRepetition,
  uint8 Fr_CycleOffset,
  uint8 Fr_PayloadLength,
  uint16 Fr_HeaderCRC )
```

Parameter	
FR_VCTRL_SYSTEMTYPE_ONLY	Depending on preconfiguration parameter ‘DrvFrBaseApiOptimization’ this macro is either empty or contains the value for uint8 Fr_CtrlIdx. Fr_CtrlIdx is a zero based index identifies the CC controller for which the API call has to be applied.
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.

Return code	
Std_ReturnType	E_OK: API call finished successfully. E_NOT_OK: API call aborted due to errors.

Functional Description
This method reconfigures during runtime the corresponding HW buffer according the given parameters. If LPDU reconfiguration is enabled for a specific LPDU, this LPDU is not sent after initialization of Fr but has to be enabled with this method.
Particularities and Limitations
▶ Particularities, limitations, post-conditions, pre-conditions It is required that the physical layer is in sync with the network.
Expected Caller Context

▶ Synchronous

Table 5-21 Fr_ReconfigLPdu

5.3.22 Fr_SetAbsoluteTimer

Prototype	

`FUNC(Std_ReturnType, FR_CODE)` **`Fr_SetAbsoluteTimer`** `(`
`FR_VCTRL_SYSTEMTYPE_ONLY,`
`uint8 Fr_AbsTimerIdx,`
`uint8 Cycle, uint16 MacrotickOffset)`	

Parameter	
FR_VCTRL_SYSTEMTYPE_ONLY	Depending on preconfiguration parameter 'DrvFrBaseApiOptimization' this macro is either empty or contains the value for uint8 Fr_CtrlIdx. Fr_CtrlIdx is a zero based index identifies the CC controller for which the API call has to be applied.
Fr_AbsTimerIdx	Index of the Timer
Cycle	Cycle in which the timer will fire
MacrotickOffset	MacrotickOffset
Return code	
Std_ReturnType	E_OK: API call finished successfully. E_NOT_OK: API call aborted due to errors.
Functional Description	
This method sets the absolute timer. The time is specified in FlexRay terms i.e. Cycle Counts and Macroticks. The timer fires an interrupt after n Macroticks in given Cycle. Hence the maximum duration is limited to one complete communication cycle (i.e. 64 Cycles).	
Particularities and Limitations	
▶ Particularities, limitations, post-conditions, pre-conditions It is required that the physical layer is in sync with the network.	
Expected Caller Context	
▶ Synchronous	

Table 5-22 Fr_SetAbsoluteTimer

5.3.23 Fr_CancelAbsoluteTimer

Prototype	

`FUNC(Std_ReturnType, FR_CODE)` **`Fr_CancelAbsoluteTimer`**`(`
`FR_VCTRL_SYSTEMTYPE_ONLY,`
`uint8 Fr_AbsTimerIdx)`	

Parameter	
FR_VCTRL_SYSTEMTYPE_ONLY	Depending on preconfiguration parameter 'DrvFrBaseApiOptimization' this macro is either empty or contains the value for uint8 Fr_CtrlIdx. Fr_CtrlIdx is a zero based index identifies the CC controller for which the API call has to be applied.

Fr_AbsTimerIdx	Index of the Timer
Return code	
Std_ReturnType	E_OK: API call finished successfully. E_NOT_OK: API call aborted due to errors.
Functional Description	
This method cancels the absolute timer	
Particularities and Limitations	
▶ Particularities, limitations, post-conditions, pre-conditions	
Expected Caller Context	
▶ Synchronous	

Table 5-23 Fr_CancelAbsoluteTimer

5.3.24 Fr_EnableAbsoluteTimerIRQ

Prototype	

`FUNC(Std_ReturnType, FR_CODE)` **`Fr_EnableAbsoluteTimerIRQ`**`(`
`  FR_VCTRL_SYSTEMTYPE_ONLY,`
`  uint8 Fr_AbsTimerIdx)`	

Parameter	
FR_VCTRL_SYSTEMTYPE_ONLY	Depending on preconfiguration parameter 'DrvFrBaseApiOptimization' this macro is either empty or contains the value for uint8 Fr_CtrlIdx. Fr_CtrlIdx is a zero based index identifies the CC controller for which the API call has to be applied.
Fr_AbsTimerIdx	Index of the Timer
Return code	
Std_ReturnType	E_OK: API call finished successfully. E_NOT_OK: API call aborted due to errors.
Functional Description	
This method enables the absolute timer.	
Particularities and Limitations	
▶ Particularities, limitations, post-conditions, pre-conditions	
Expected Caller Context	
▶ Synchronous	

Table 5-24 Fr_EnableAbsoluteTimerIRQ

5.3.25 Fr_AckAbsoluteTimerIRQ

Prototype

`FUNC(Std_ReturnType, FR_CODE)` **`Fr_AckAbsoluteTimerIRQ`**`(`
`  FR_VCTRL_SYSTEMTYPE_ONLY,`
`  uint8 Fr_AbsTimerIdx)`

Parameter	
FR_VCTRL_SYSTEMTYPE_ONLY	Depending on preconfiguration parameter 'DrvFrBaseApiOptimization' this macro is either empty or contains the value for uint8 Fr_CtrlIdx. Fr_CtrlIdx is a zero based index identifies the CC controller for which the API call has to be applied.
Fr_AbsTimerIdx	Index of the Timer
Return code	
Std_ReturnType	E_OK: API call finished successfully. E_NOT_OK: API call aborted due to errors.
Functional Description	
This method acknowledges the absolute timer.	
Particularities and Limitations	
▶ Particularities, limitations, post-conditions, pre-conditions This API can be removed from code depending on the GenTool configuration.	
Expected Caller Context	
▶ Synchronous	

Table 5-25 Fr_AckAbsoluteTimerIRQ

5.3.26 Fr_DisableAbsoluteTimerIRQ

Prototype	

`FUNC(Std_ReturnType, FR_CODE) Fr_DisableAbsoluteTimerIRQ(`
`  FR_VCTRL_SYSTEMTYPE_ONLY,`
`  uint8 Fr_AbsTimerIdx)`	

Parameter	
FR_VCTRL_SYSTEMTYPE_ONLY	Depending on preconfiguration parameter 'DrvFrBaseApiOptimization' this macro is either empty or contains the value for uint8 Fr_CtrlIdx. Fr_CtrlIdx is a zero based index identifies the CC controller for which the API call has to be applied.
Fr_AbsTimerIdx	Index of the Timer
Return code	
Std_ReturnType	E_OK: API call finished successfully. E_NOT_OK: API call aborted due to errors.
Functional Description	
This method disables the absolute timer.	
Particularities and Limitations	
▶ Particularities, limitations, post-conditions, pre-conditions This API can be removed from code depending on the GenTool configuration.	
Expected Caller Context	
▶ Synchronous	

Table 5-26 Fr_DisableAbsoluteTimerIRQ

5.3.27 Fr_GetAbsoluteTimerIRQStatus

Prototype	

`FUNC(Std_ReturnType, FR_CODE) **Fr_GetAbsoluteTimerIRQStatus**(`
`FR_VCTRL_SYSTEMTYPE_ONLY,`
`uint8 Fr_AbsTimerIdx,`
`P2VAR(boolean, AUTOMATIC, FR_APPL_DATA) Fr_IRQStatusPtr)`	

Parameter	
FR_VCTRL_SYSTEMTYPE_ONLY	Depending on preconfiguration parameter 'DrvFrBaseApiOptimization' this macro is either empty or contains the value for uint8 Fr_CtrlIdx. Fr_CtrlIdx is a zero based index identifies the CC controller for which the API call has to be applied.
Fr_AbsTimerIdx	Index of the Timer
Fr_IRQStatusPtr	Result value of the Timer IRQ Status
Return code	
Std_ReturnType	E_OK: API call finished successfully. E_NOT_OK: API call aborted due to errors.
Functional Description	
Gets IRQ status of the absolute timer	
Particularities and Limitations	
▶ Particularities, limitations, post-conditions, pre-conditions This API can be removed from code depending on the GenTool configuration.	
Expected Caller Context	
▶ Synchronous	

Table 5-27 Fr_GetAbsoluteTimerIRQStatus

5.3.28 Fr_GetChannelStatus

Prototype	

`FUNC(Std_ReturnType, FR_CODE) **Fr_GetChannelStatus** (`
`FR_VCTRL_SYSTEMTYPE_ONLY,`
`P2VAR(uint16, AUTOMATIC, FR_APPL_DATA) Fr_ChannelAStatusPtr,`
`P2VAR(uint16, AUTOMATIC, FR_APPL_DATA) Fr_ChannelBStatusPtr)`	

Parameter	
FR_VCTRL_SYSTEMTYPE_ONLY	Depending on preconfiguration parameter 'DrvFrBaseApiOptimization' this macro is either empty or contains the value for uint8 Fr_CtrlIdx. Fr_CtrlIdx is a zero based index identifies the CC controller for which the API call has to be applied.
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 code	
Std_ReturnType	E_OK: API call finished successfully. E_NOT_OK: API call aborted due to errors.

Functional Description
Returns the current aggregated channel status information. The information is bitcoded as follow:
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 (This bit is not supportet at FlexRay 2.1 hardware) 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) The aggregated channel status information is cleared after the read operation.
Particularities and Limitations
▶ Particularities, limitations, post-conditions, pre-conditions The method is only available with enabled GenTool option “Get Channel Status Support”. It is required that the physical layer is in sync with the network.
Expected Caller Context
▶ Asynchronous

Table 5-28 Fr_GetChannelStatus

5.3.29 Fr_GetClockCorrection

Prototype

```
FUNC(Std_ReturnType, FR_CODE) Fr_GetClockCorrection(
  FR_VCTRL_SYSTEMTYPE_ONLY,
  P2VAR(sint16, AUTOMATIC, FR_APPL_DATA) Fr_RateCorrectionPtr,
  P2VAR(sint32, AUTOMATIC, FR_APPL_DATA) Fr_OffsetCorrectionPtr )
```

Parameter	
FR_VCTRL_SYSTEMTYPE_ONLY	Depending on preconfiguration parameter ‘DrvFrBaseApiOptimization’ this macro is either empty or contains the value for uint8 Fr_CtrlIdx. Fr_CtrlIdx is a zero based index identifies the CC controller for which the API call has to be applied.
Fr_RateCorrectionPtr	Address where the rate correction value shall be stored.
Fr_OffsetCorrectionPtr	Address where the offset correction value shall be stored.
Return code	
Std_ReturnType	E_OK: API call finished successfully. E_NOT_OK: API call aborted due to errors.
Functional Description	
Returns the values of the rate and offset correction.	

Particularities and Limitations
▶ Particularities, limitations, post-conditions, pre-conditions The method is only available in a delivery with extended channel status and enabled GenTool option "Get Clock Correction Support". It is required that the physical layer is in sync with the network.
Expected Caller Context
▶ Asynchronous

Table 5-29 Fr_GetClockCorrection

5.3.30 Fr_GetWakeupRxStatus

Prototype	
`Std_ReturnType Fr_GetWakeupRxStatus(FR_VCTRL_SYSTEMTYPE_ONLY, uint8* Fr_WakeupRxStatusPtr)`	
Parameter	
FR_VCTRL_SYSTEMTYPE_ONLY	Depending on preconfiguration parameter 'DrvFrBaseApiOptimization' this macro is either empty or contains the value for uint8 Fr_CtrlIdx. Fr_CtrlIdx is a zero based index identifies the CC controller for which the API call has to be applied.
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 code	
E_OK	API call finished successfully.
E_NOT_OK	API call aborted due to errors.
Functional Description	
Gets the wakeup received information from the FlexRay controller.	
Particularities and Limitations	
▶ Particularities, limitations, post-conditions, pre-conditions The method is only available with enabled GenTool FrIf option "Get Wakeup Rx Status Support". It is required that Fr_Init() is called before.	
Expected Caller Context	
▶ This function can be called in any context.	

Table 5-30 Fr_GetWakeupRxStatus

5.3.31 Fr_GetNumOfStartupFrames

Prototype
`Std_ReturnType Fr_GetNumOfStartupFrames(FR_VCTRL_SYSTEMTYPE_ONLY, uint8* Fr_NumOfStartupFramesPtr)`

Parameter	
FR_VCTRL_SYSTEMTYPE_ONLY	Depending on preconfiguration parameter 'DrvFrBaseApiOptimization' this macro is either empty or contains the value for uint8 Fr_CtrlIdx. Fr_CtrlIdx is a zero based index identifies the CC controller for which the API call has to be applied.
Fr_NumOfStartupFramesPtr	Address where the number of startup frames seen within the last even/odd cycle pair shall be stored.
Return code	
E_OK	API call finished successfully.
E_NOT_OK	API call aborted due to errors.
Functional Description	
Gets the current number of startup frames seen on the cluster.	
Particularities and Limitations	
▶ Particularities, limitations, post-conditions, pre-conditions The method is only available with enabled GenTool FrIf option "Get Num Of Startup Frames Support". It is required that Fr_Init() is called before.	
Expected Caller Context	
▶ This function can be called in any context.	

Table 5-31 Fr_GetNumOfStartupFrames

5.3.32 Fr_TransmitTxLPdu_DBA (optional)

Prototype	

`FUNC(Std_ReturnType, FR_CODE) Fr_TransmitTxLPdu_DBA(`
`  FR_VCTRL_SYSTEMTYPE_ONLY,`
`  uint16 Fr_LPduIdx,`
`  uint8 Fr_LPduLength)`	

Parameter	
FR_VCTRL_SYSTEMTYPE_ONLY	Depending on preconfiguration parameter 'DrvFrBaseApiOptimization' this macro is either empty or contains the value for uint8 Fr_CtrlIdx. Fr_CtrlIdx is a zero based index identifies the CC controller for which the API call has to be applied.
Fr_LPduIdx	This index is used to uniquely identify a FlexRay frame
Fr_LPduLength	Determines the length of the data (in Bytes) to be transmitted. The value is not modified by this method
Return code	
Std_ReturnType	E_OK: API call finished successfully. E_NOT_OK: API call aborted due to errors.
Functional Description	
This method sends a frame on the bus thus the bus must be initialized. The method figures out the physical buffer where the LPdu should be copied to. The method does not copy the data to that physical buffer. This must be done by other modules. The method activates the prepared buffer for transmission. This method reconfigures the message buffer in case dynamic payload or buffer reconfigure is enabled.	

Prototype
Particularities and Limitations
▶ It is required that the physical layer is in sync with the network. The location of the buffer can be get from method “Fr_RequestBuffer_DBA” ▶ The API availability depends on the platform please refer to [8].
Expected Caller Context
▶ Synchronous

Table 5-32 Fr_TransmitTxLPdu_DBA

5.3.33 Fr_TransmitTxLPdu_ImmediateDBA (optional)

Prototype	
`FUNC(Std_ReturnType, FR_CODE) Fr_TransmitTxLPdu_ImmediateDBA ( FR_VCTRL_SYSTEMTYPE_ONLY, uint16 Fr_LPduIdx, P2CONST(uint8, AUTOMATIC, FR_APPL_DATA) Fr_LPduPtr, uint8 Fr_LPduLength)`	
Parameter	
`Fr_CtrlIdx`	This zero based index identifies the driver within the context of the Fr to which the API call has to be applied.
Fr_LPduIdx	This index is used to uniquely identify a FlexRay frame.
*Fr_LPduPtr	This reference points to the buffer where the LPdu to be received shall be stored.
Fr_LPduLengthPtr	This reference points to the memory location where the length of the LPdu (in bytes) shall be stored.
Return code	
Std_ReturnType	E_OK: API call finished successfully. E_NOT_OK: API call aborted due to errors.
Functional Description	
This method requests a buffer, copies the payload and prepares and configures a buffer to be sent on FlexRay bus. The function brings FTU in halt mode and disables the interrupts. Afterwards it copies the payload, transmits the buffer and set the FTU in normal mode again and enables the interrupts.	
Particularities and Limitations	
▶ It is required that the physical layer is in sync with the network. ▶ The API availability depends on the platform please refer to [8].	
Expected Caller Context	
▶ Synchronous	

Table 5-33 Fr_TransmitTxLPdu_ImmediateDBA

5.3.34 Fr_RequestBuffer_DBA (optional)

Prototype	

FUNC(Std_ReturnType, FR_CODE) Fr_RequestBuffer_DBA (
  FR_VCTRL_SYSTEMTYPE_ONLY,
  uint16 Fr_LPduIdx,
  P2VAR(Fr_LPduPtrType, AUTOMATIC, FR_APPL_DATA) Fr_LPduPtr,
  uint8 Fr_LPduLength)	

Parameter	
FR_VCTRL_SYSTEMTYPE_ONLY	Depending on preconfiguration parameter 'DrvFrBaseApiOptimization' this macro is either empty or contains the value for uint8 Fr_CtrlIdx. Fr_CtrlIdx is a zero based index identifies the CC controller for which the API call has to be applied.
Fr_LPduIdx	This index is used to uniquely identify a FlexRay frame
Fr_LPduPtr	This pointer to a reference gives back the pointer of the buffer where the assembled LSdu to be transmitted within this LPdu is stored at
Fr_LPduLength	Determines the length of the data (in Bytes) to be transmitted. The value is not modified by this method
Return code	
Std_ReturnType	E_OK: API call finished successfully. E_NOT_OK: API call aborted due to errors.
Functional Description	
This method gives back the pointer to the buffer of the corresponding LPDU The method figures out the physical buffer where the LPdu should be copied to. This method should be called before Fr_TransmitTxLPdu_DBA to get the message buffer where the data should be written to.	
Particularities and Limitations	
▶ It is required that the physical layer is in sync with the network. ▶ The API availability depends on the platform please refer to [8].	
Expected Caller Context	
▶ Synchronous	

Table 5-34 Fr_RequestBuffer_DBA

5.3.35 Fr_ReceiveRxLPdu_DBA (optional)

Prototype	

FUNC(Std_ReturnType, FR_CODE) Fr_ReceiveRxLPdu_DBA(
  FR_VCTRL_SYSTEMTYPE_ONLY,
  uint16 Fr_LPduIdx, P2VAR(Fr_LPduPtrType, AUTOMATIC, FR_APPL_DATA)
  Fr_LPduPtr,
  P2VAR(Fr_RxLPduStatusType, AUTOMATIC, FR_APPL_DATA) Fr_LPduStatusPtr,
  P2VAR(uint8, AUTOMATIC, FR_APPL_DATA) Fr_LPduLengthPtr)	

Parameter	
FR_VCTRL_SYSTEMTYPE_ONLY	Depending on preconfiguration parameter 'DrvFrBaseApiOptimization' this macro is either empty or contains the value for uint8 Fr_CtrlIdx. Fr_CtrlIdx is a zero based index identifies the CC controller for which the API call has to be applied.

Prototype	
Fr_LPduIdx	This index is used to uniquely identify a FlexRay frame.
Fr_LPduPtr	This pointer to a reference gives back the pointer to the buffer where the LPdu to be received is stored.
Fr_LPduStatusPtr	This reference points to the memory location where the status of the LPdu shall be stored.
Fr_LPduLengthPtr	This reference points to the memory location where the length of the LPdu (in bytes) shall be stored.
Return code	
Std_ReturnType	E_OK: API call finished successfully. E_NOT_OK: API call aborted due to errors.
Functional Description	
This method figures out if a new FlexRay frame identified by Fr_LPduIdx was received. It gives back the pointer to the data of Fr_LPduPtr. The length parameter specifies the length in bytes whereas FlexRay is using 2-byte lengths. The method locks the buffer from unsynchronized access of the CC. The method returns “`FR_RECEIVED_MORE_DATA_AVAILABLE`” at the “`Fr_LPduStatusPtr`” in case the requested frame is placed within a FIFO range and the FIFO is not empty.	
Particularities and Limitations	
▶ It is required that the physical layer is in sync with the network. ▶ The method Fr_UnlockRxLPdu_DBA should be called after the data of Fr_LPduIdx was processed. ▶ The API availability depends on the platform please refer to [8].	
Expected Caller Context	
▶ Synchronous	

Table 5-35 Fr_ReceiveRxLPdu_DBA

5.3.36 Fr_UnlockRxLPdu_DBA (optional)

Prototype	

`FUNC(Std_ReturnType, FR_CODE) Fr_UnlockRxLPdu_DBA(`
`  FR_VCTRL_SYSTEMTYPE_ONLY,  uint16 Fr_LPduIdx)`	

Parameter	
FR_VCTRL_SYSTEMTYPE_ONLY	Depending on preconfiguration parameter ‘DrvFrBaseApiOptimization’ this macro is either empty or contains the value for uint8 Fr_CtrlIdx. Fr_CtrlIdx is a zero based index identifies the CC controller for which the API call has to be applied.
Fr_LPduIdx	This index is used to uniquely identify a FlexRay frame.
Return code	
Std_ReturnType	E_OK: API call finished successfully. E_NOT_OK: API call aborted due to errors.
Functional Description	
This method unlocks the message buffer identified by Fr_LPduIdx. The method should be called after the data, received by method Fr_ReceiveRxLPdu_DBA, was processed.	

Prototype
Particularities and Limitations
▶ It is required that the physical layer is in sync with the network. ▶ The method Fr_UnlockRxLPdu_DBA should be called after the data of Fr_LPduIdx was processed. ▶ The API availability depends on the platform please refer to [8].
Expected Caller Context
▶ Synchronous

Table 5-36 Fr_UnlockRxLPdu_DBA

5.3.37 Fr_ReadCCConfig (optional)

Prototype	

`FUNC(Std_ReturnType, FR_CODE) Fr_ReadCCConfig(`
`  FR_VCTRL_SYSTEMTYPE_ONLY,`
`  uint8 Fr_CCLLParamIndex,`
`  P2VAR(uint32, AUTOMATIC, FR_APPL_DATA) Fr_CCLLParamValue)`	

Parameter	
FR_VCTRL_SYSTEMTYPE_ONLY	Depending on preconfiguration parameter 'DrvFrBaseApiOptimization' this macro is either empty or contains the value for uint8 Fr_CtrlIdx. Fr_CtrlIdx is a zero based index identifies the CC controller for which the API call has to be applied.
Fr_CCLLParamIndex	Parameter index.
Fr_CCLLParamValue	Value of index.
Return code	
Std_ReturnType	E_OK: API call finished successfully. E_NOT_OK: API call aborted due to errors.
Functional Description	
This method returns the requested ECUC parameter. Please refer to [2] to find out the index of corresponding low level parameter.	
Particularities and Limitations	
▶ Particularities, limitations, post-conditions, pre-conditions The API is only available if GenTool option “Read CC Parameters” is set.	
Expected Caller Context	
▶ Synchronous	

Table 5-37 Fr_ReadCCConfig

5.4 Services used by FR

In Table 5-38 services provided by other components, which are used by the FR are listed. For details about prototype and functionality refer to the documentation of the providing component.

Component	API
DET	Det_ReportError
DEM	Dem_ReportErrorStatus
Vstdlib.c	Usage of this API is CC specfic please refer to [8]
SchM	`SchM_Enter_Fr` `SchM_Exit_Fr`
EcuM	`EcuM_GeneratorCompatibilityError`

Table 5-38 Services used by the FR

5.5 Callback Functions

The MICROSAR Fr does not use any callback functions.

5.6 Configurable Interfaces

5.6.1 Notifications

The MICROSAR Fr does not use any notifications.

5.6.2 Callout Functions

At its configurable interfaces the Fr defines callout functions. The declarations of the callout functions are provided by the BSW module, i.e. the Fr. It is the integrator's task to provide the corresponding function definitions. The definitions of the callouts can be adjusted to the system's needs. The callout functions are optional and can be invoked by other modules. They are described in the following tables:

5.6.2.1 ApplFr_ISR_Timer0

Prototype	
`FUNC(void, FR_APPL_CODE)` **`ApplFr_ISR_Timer0`** `(void);`	
Parameter	
-	
Return code	
-	
Functional Description	
The method is called when the absolute timer fires. The Interrupt is cleared within the handler. The method is only used with enabled compiler switch ‘FR_CFG_APPL_CALLBACK_TIMER0’	
Particularities and Limitations	
▶ Particularities, limitations, post-conditions, pre-conditions Called within the interrupt handler. It is required that the physical layer is in sync with the network. Depending on	

Expected Caller Context
▶ Interrupt context

Table 5-39 ApplFr_ISR_Timer0

5.6.2.2 ApplFr_ISR_Timer0_1

Prototype	
`FUNC(void, FR_APPL_CODE)` **`ApplFr_ISR_Timer0_1`** `(void);`	
Parameter	
-	
Return code	
-	
Functional Description	
The method is called when the absolute timer fires. The Interrupt is cleared within the handler. The method is only used with enabled compiler switch ‘FR_CFG_APPL_CALLBACK_TIMER0’ and ‘FR_NUM_CTRL_USED > 1’	
Particularities and Limitations	
▶ Particularities, limitations, post-conditions, pre-conditions Called within the interrupt handler. It is required that the physical layer is in sync with the network. Depending on	
Expected Caller Context	
▶ Interrupt context	

Table 5-40 ApplFr_ISR_Timer0

5.6.2.3 ApplFr_ISR_CycleStart

Prototype	
`FUNC(void, FR_APPL_CODE)` **`ApplFr_ISR_CycleStart`** `(void)`	
Parameter	
-	None
Return code	
-	None
Functional Description	
The method is called when a new communication cycle starts. The Interrupt is cleared within the handler. The method is only used with enabled compiler switch ‘FR_CFG_APPL_CALLBACK_CYCLE_START’	

Particularities and Limitations
▶ Particularities, limitations, post-conditions, pre-conditions Called within the interrupt handler. It is required that the physical layer is in sync with the network.
Expected Caller Context
▶ Interrupt context

Table 5-41 ApplFr_ISR_CycleStart

5.6.2.4 ApplFr_ISR_CycleStart_1

Prototype	
`FUNC(void, FR_APPL_CODE)` **`ApplFr_ISR_CycleStart_1`** `(void)`	
Parameter	
-	None
Return code	
-	None
Functional Description	
The method is called when a new communication cycle starts. The Interrupt is cleared within the handler. The method is only used with enabled compiler switch 'FR_CFG_APPL_CALLBACK_CYCLE_START' and 'FR_NUM_CTRL_USED > 1'	
Particularities and Limitations	
▶ Particularities, limitations, post-conditions, pre-conditions Called within the interrupt handler. It is required that the physical layer is in sync with the network.	
Expected Caller Context	
▶ Interrupt context	

Table 5-42 ApplFr_ISR_CycleStart

6 Glossary and Abbreviations

6.1 Glossary

Term	Description
EAD	Embedded Architecture Designer; generation tool for MICROSAR components
Cfg5	Generation tool for AUTOSAR4 MICROSAR components

Table 6-1 Glossary

6.2 Abbreviations

Abbreviation	Description
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSW	Basis Software
CC	FlexRay Communication Controller
DEM	Diagnostic Event Manager
DET	Development Error Tracer
EAD	Embedded Architecture Designer
ECU	Electronic Control Unit
HIS	Hersteller Initiative Software
Host	Specific microcontroller where Fr CC is implemented and on which the Fr software is executed.
ISR	Interrupt Service Routine
JLE	Job List Execution (see [10])
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
MTS	Media Test Symbol
PPort	Provide Port
RPort	Require Port
RTE	Runtime Environment
SRS	Software Requirement Specification
SWC	Software Component
SWS	Software Specification
EcuC	ECU Configuration file

Table 6-2 Abbreviations

7 Contact

Visit our website for more information on

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

www.vector-informatik.com