VECTOR

MICROSAR FlexRay Interface

Technical Reference

Version 4.03.00

Authors	visore, visath
Status	Released

Document Information

History

Author	Date	Version	Remarks
visath	2016-04-24	4.00.00	First SafeBSW release.
visath	2016-07-13	4.00.01	Added deviation related to the job list resynchronization.
visath	2017-03-24	4.01.00	Support 2 FlexRay Clusters
visath	2019-04-12	4.01.01	Removed outdated single controller limitations.
visore	2020-06-23	4.02.00	Added FrIf_GetConflictStatus Removed relative timer API descriptions.
visore	2021-03-22	4.03.00	Updated Dynamic Payload section.

Reference Documents

No.	Title	Version
[1]	AUTOSAR_FlexRayInterface.pdf	V3.2.0
[2]	AUTOSAR_SWS_DET.pdf	V2.2.0
[3]	AUTOSAR_SWS_DEM.pdf	V2.2.1
[4]	AUTOSAR_BasicSoftwareModules.pdf	V1.0.0
[5]	TechnicalReference_Asr_Fr.pdf	V1.14 or later
[6]	TechnicalReference_Asr_FrTrcv_Tja1080.pdf	V.1.08 or later
[7]	TechnicalReference_Asr_FrNm.pdf	V1.3.0 or later
[8]	TechnicalReference_Asr_EcuM.pdf	V2.1.0 or later
[9]	TechnicalReference_Asr_Com.pdf	V2.6.0 or later
[10]	TechnicalReference_Asr_PduR.pdf	V3.4.0 or later
[11]	TechnicalReference_Asr_SchM.pdf	V2.3.0 or later
[12]	AN-ISC-8-1118_MICROSAR_BSW_Compatibility_Check.pdf	V1.0
[13]	TechnicalReference_IdentityManager.pdf	V1.1.7 or later

Scope of the Document

This technical reference describes the general use of the FlexRay Interface basic software.

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

1 Introduction

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

Supported AUTOSAR Release*:	4	
Supported Configuration Variants:	pre-compile, link-time, post-build	
Vendor ID:	FrIf_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	FrIf_MODULE_ID	61 decimal (according to ref. [4])

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

The main purpose of the FlexRay Interface is to provide a PDU based API that can be used asynchronously to the FlexRay global time by upper software layer modules. The FlexRay Interface assembles PDUs of the upper software layers into frames and vice versa. E.g. multiple bus-independent PDUs with a length of 8 bytes can be assembled into a frame with a length of up to 254 bytes. Thus bus-independent modules can take advantage of the large payload that is possible with FlexRay.

Another task of the FlexRay Interface is to abstract the usage of multiple FlexRay Communication Controllers and multiple Transceiver Drivers within an ECU.

The behaviour of the FlexRay Interface can be summarized as follows:

- The FlexRay Interface does not have any RAM buffers to store the content of PDUs, e.g. for periodic transmission.
- The FlexRay Interface cannot be polled for received data. When a frame is received, the reception of the PDUs that are contained in the frame is actively indicated by the FlexRay Interface.
- The FlexRay Interface provides the possibility to actively indicate the transmission of PDUs when the frame that contains the PDUs is actually transmitted.

1.1 Architecture Overview

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

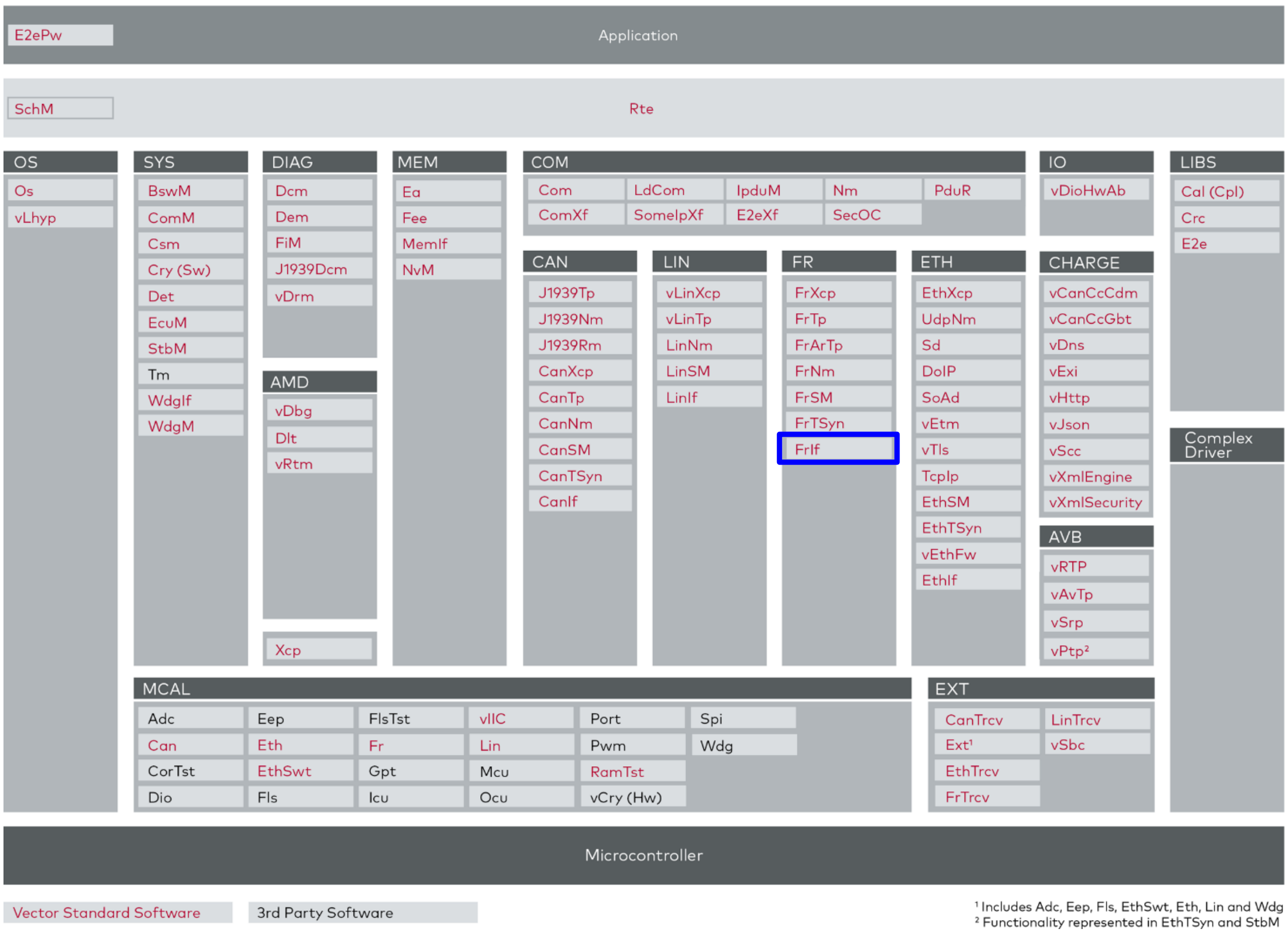

Figure 1-1 AUTOSAR 4.2 Architecture Overview

The next figure shows the interfaces to adjacent modules of the FrIf. This interfaces are explained in section 4.2.

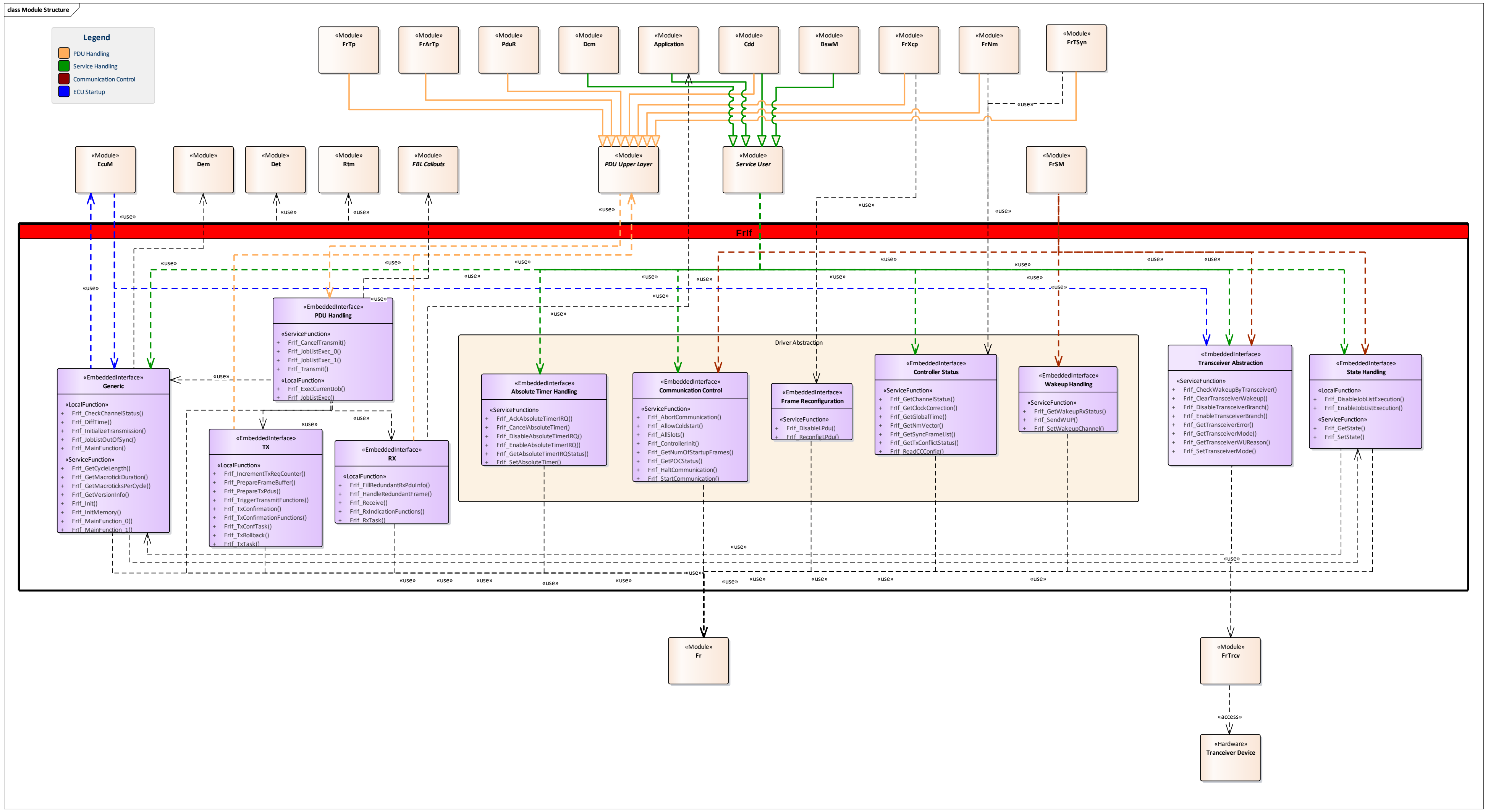
class Module Structure
Legend
🟧 PDU Handling
🟩 Service Handling
🟥 Communication Control
🟦 ECU Startup
«Module» FrTp
«Module» FrArTp
«Module» PduR
«Module» Dcm
«Module» Application
«Module» Cdd
«Module» BswM
«Module» FrXcp
«Module» FrNm
«Module» FrTSyn
«Module» EcuM
«Module» Dem
«Module» Det
«Module» Rtm
«Module» FBL Callouts
«Module» PDU Upper Layer
«Module» Service User
«Module» FrSM
FrIf
«use»
«EmbeddedInterface» PDU Handling
«ServiceFunction»
+ FrIf_CancelTransmit()
+ FrIf_JobListExec_0()
+ FrIf_JobListExec_1()
+ FrIf_Transmit()
«LocalFunction»
+ FrIf_ExecCurrentJob()
+ FrIf_JobListExec()
«EmbeddedInterface» Generic
«LocalFunction»
+ FrIf_CheckChannelStatus()
+ FrIf_DiffTime()
+ FrIf_InitializeTransmission()
+ FrIf_JobListOutOfSync()
+ FrIf_MainFunction()
«ServiceFunction»
+ FrIf_GetCycleLength()
+ FrIf_GetMacrotickDuration()
+ FrIf_GetMacroticksPerCycle()
+ FrIf_GetVersionInfo()
+ FrIf_Init()
+ FrIf_InitMemory()
+ FrIf_MainFunction_0()
+ FrIf_MainFunction_1()
«EmbeddedInterface» TX
«LocalFunction»
+ FrIf_IncrementTxReqCounter()
+ FrIf_PrepareFrameBuffer()
+ FrIf_PrepareTxPdus()
+ FrIf_TriggerTransmitFunctions()
+ FrIf_TxConfirmation()
+ FrIf_TxConfirmationFunctions()
+ FrIf_TxConfTask()
+ FrIf_TxRollback()
+ FrIf_TxTask()
«EmbeddedInterface» RX
«LocalFunction»
+ FrIf_FillRedundantRxPduInfo()
+ FrIf_HandleRedundantFrame()
+ FrIf_Receive()
+ FrIf_RxIndicationFunctions()
+ FrIf_RxTask()
Driver Abstraction
«EmbeddedInterface» Absolute Timer Handling
«ServiceFunction»
+ FrIf_AckAbsoluteTimerIRQ()
+ FrIf_CancelAbsoluteTimer()
+ FrIf_DisableAbsoluteTimerIRQ()
+ FrIf_EnableAbsoluteTimerIRQ()
+ FrIf_GetAbsoluteTimerIRQStatus()
+ FrIf_SetAbsoluteTimer()
«EmbeddedInterface» Communication Control
«ServiceFunction»
+ FrIf_AbortCommunication()
+ FrIf_AllowColdstart()
+ FrIf_AllSlots()
+ FrIf_ControllerInit()
+ FrIf_GetNumOfStartupFrames()
+ FrIf_GetPOCStatus()
+ FrIf_HaltCommunication()
+ FrIf_StartCommunication()
«EmbeddedInterface» Frame Reconfiguration
«ServiceFunction»
+ FrIf_DisableLPdu()
+ FrIf_ReconfigLPdu()
«EmbeddedInterface» Controller Status
«ServiceFunction»
+ FrIf_GetChannelStatus()
+ FrIf_GetClockCorrection()
+ FrIf_GetGlobalTime()
+ FrIf_GetNmVector()
+ FrIf_GetSyncFrameList()
+ FrIf_GetTxConflictStatus()
+ FrIf_ReadCCConfig()
«EmbeddedInterface» Wakeup Handling
«ServiceFunction»
+ FrIf_GetWakeupRxStatus()
+ FrIf_SendWUP()
+ FrIf_SetWakeupChannel()
«EmbeddedInterface» Transceiver Abstraction
«ServiceFunction»
+ FrIf_CheckWakeupByTransceiver()
+ FrIf_ClearTransceiverWakeup()
+ FrIf_DisableTransceiverBranch()
+ FrIf_EnableTransceiverBranch()
+ FrIf_GetTransceiverError()
+ FrIf_GetTransceiverMode()
+ FrIf_GetTransceiverWUReason()
+ FrIf_SetTransceiverMode()
«EmbeddedInterface» State Handling
«LocalFunction»
+ FrIf_DisableJobListExecution()
+ FrIf_EnableJobListExecution()
«ServiceFunction»
+ FrIf_GetState()
+ FrIf_SetState()
«Module» Fr
«Module» FrTrcv
«access»
«Hardware» Tranceiver Device

Figure 1-2 Interfaces to adjacent modules of the **FrIf**

2 Functional Description

2.1 Features

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

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

> Table 2-1 Supported AUTOSAR standard conform features

> Table 2-2 Not supported AUTSAR standard conform features

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

> Table 2-3 Features provided beyond the AUTOSAR standard

<u>The following features specified in [1] are supported:</u>

Supported AUTOSAR Standard Conform Features
PDU Reception: • Reception Indication • FIFO Support
PDU Transmission: • Transmission Request • Transmission Confirmation • Transmit Request Queuing • PDU-based Continuous Transmission • Frame-based Continuous Transmission • Transmit Cancellation
Frame Assembly and Disassembly: • Update-bit Handling • Frame Layout and PDU owner handling (for FrTp, FrArTp, FrTSyn, FrNm, FrXcp, CDD, PduR) • Cycle Multiplexing
Hardware Abstraction: • FR Driver API Abstraction • FRTRCV Driver API Abstraction
Configuration Variants: • Precompile • Linktime • Post-build Selectable • Post-build Loadable
Error Detection and Handling: • Dev Error Reporting • Channel Status Checks with DEM reporting

Supported AUTOSAR Standard Conform Features
• Job list monitoring

Table 2-1 Supported AUTOSAR standard conform features

2.1.1 Deviations

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

Category	Description	ASR Version
Functional	7.2 Indexing Scheme: The following features are partly supported: • Multiple FlexRay Drivers: a single FlexRay Driver is currently supported. • Multiple FlexRay Clusters: a maximum of two FlexRay Clusters is currently supported. • Multiple FlexRay Communication Controllers: a single FlexRay Communication Controller per cluster is currently supported.	4.0.3
Functional	7.6.3 Communication Operations: The following Communication Operations are not supported: • RECEIVE_AND_STORE • RX_INDICATION • PREPARE_LPDU • FREE_OP_A • FREE_OP_B	4.0.3
Functional	7.6.1 PDU Packing, PDU update bits, and Frame Construction Plans: Received frames that are shorter than the configured length are not padded, even if the FrIfUnusedBitValue exists.	4.0.3
Functional	7.3.1 FlexRay Interface Main Function and 7.6.2.2 FlexRay Job List Execution Function: Pending transmissions and transmission confirmations are cleared and forgotten during the resynchronization of the job list.	4.0.3
API	FrIf_CancelTransmit: The request counters of the PDUs sharing a frame with a cancelled PDU are not increased after cancelling the transmission of the frame.	4.0.3
Config	FrIfByteOrder: Only little endian is currently supported.	4.0.3
Config	FrIfCounterLimit: Can only be configured to 1 or 255.	4.0.3
Config	FrIfDisableTransceiverBranchSupport and FrIfEnableTransceiverBranchSupport: The functions FrIf_DisableTransceiverBranch and FrIf_EnableTransceiverBranch will be available as long as a FrIf/FrIfConfig/FrIfCluster/FrIfController/FrIfTransceiver container exists.	4.0.3
Config	FrIfRxComOpMaxLoop: A limit when emptying a FIFO can’t be configured.	4.0.3

Table 2-2 Not supported AUTSAR standard conform features

2.1.2 Additions/ Extensions

The following features are provided beyond the AUTOSAR standard:

Features Provided Beyond The AUTOSAR Standard
Dual channel redundancy support
AMD runtime measurement
BSW debug parameters
TX conflict status support

Table 2-3 Features provided beyond the AUTOSAR standard

2.2 Initialization

The FlexRay Interface gets initialized by call of **`FrIf_InitMemory`** and then **`FrIf_Init`** out of EcuM. If EcuM is not used the application has to call **`FrIf_InitMemory`** and then **`FrIf_Init`**.

Example

```
FrIf_Init(&FrIf_Config);
```

Figure 2-4 Initialization example for the FlexRay Interface

Caution

Starting with AUTOSAR Release 3 the FlexRay Interface is no longer responsible for initializing the FlexRay Driver and the FlexRay Transceiver Driver.

Depending on the configuration variant the `FrIf_Config` parameter of **`FrIf_Init`** has different meaning:

2.2.1 Configuration Variants 1 and 2 (Pre-Compile and Link-Time Configuration)

At Variant 1 (Pre-compile Configuration) and Variant 2 (Link-Time Configuration) the pointer given to **`FrIf_Init`** is ignored. At these configuration variants the FlexRay interface is configured at compile or link time and has direct access to all configuration data which is stored in the files FrIf_Cfg.h and FrIf_LCfg.c.

2.2.2 Configuration Variant 3 (Post-build Configuration)

In this configuration variant, the FlexRay interface has to be initialized using the **`FrIf_Init`** function with the address of the post-build configuration data passed as parameter. The declaration of the post-build configuration data is contained in the files FrIf_PBcfg.h and FrIf_PBcfg.c.

2.3 States

The FlexRay Interface is shown in Figure 2-1 and comprises the following states:

State Name	Description
`FRIF_STATE_OFFLINE`	The job list of the FlexRay Interface is not executed.
`FRIF_STATE_ONLINE`	The job list of the FlexRay Interface is executed.

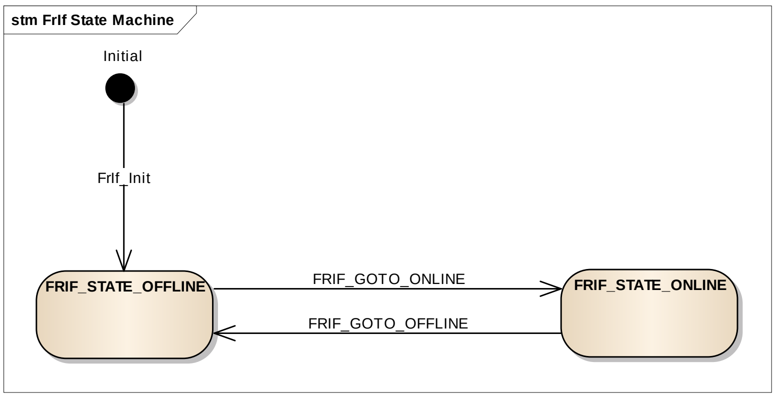

Figure 2-1 FrIf's state machine.

The state of the state machine can be changed using the **`FrIf_SetState`** function, see section 4.2.41.

2.4 Main Functions

2.4.1 Cyclic function

The **`FrIf_MainFunction_<ClstIdx>`** monitors the job execution. This function should be called cyclically with a cycle time that is equal to or smaller than the FlexRay cycle length.

There is no need to synchronize this function to the FlexRay global time. This function can be called even when the FlexRay communication has not been started.

2.4.2 Job List Execution

The communication via FlexRay is executed by a FlexRay communication controller that provides a number of transmit- and receive-buffers. These buffers are accessed by both the FlexRay-Stack and the communication controller. As some controllers prohibit concurrent access, the FlexRay-Stack may only access the buffers when they are not busy. This is ensured by using a time-triggered access that is defined by means of a job list. Each entry of the job list defines at which point in time (in terms of cycle and macro tick offset) which receive buffer has to be read or which transmit buffer has to be filled with data or checked for transmission success.

The MICROSAR FlexRay Interface has a number of so called Rx- and Tx- jobs that execute the job list. An Rx job executes the job list entries for reception; a Tx-job executes those for transmission. The numbers of Rx- and Tx-job are independent, but usually at least 2 Rx- and 2 Tx-Job must be defined. The Rx- and Tx-jobs are activated by using one absolute timer of the FlexRay communication controller (see section 495H2.8). As described in the following two chapters the job list can be activated in the interrupt context of the FlexRay timer interrupt or in the context of an OS task. As the **<UL>_TriggerTransmit** and **<UL>_RxIndication** call-back functions are called during the

FrIf_JobListExec_<ClstIdx> routine, these functions can also be executed in interrupt context.

Info

You can use the "Job Concatenation Enable" feature, to reduce the interrupt load of the FlexRay Interface to a minimum of two timer interrupts per cycle, even if 2 Rx and 2 Tx jobs are defined. If this feature is enabled, two FrIf jobs with the same configured "Macrotick" parameter will be executed by one call of the **`FrIf_JobListExec_<ClstIdx>`** function.

2.4.2.1 Job List Execution in Interrupt Context

The job list execution is activated by a FlexRay timer interrupt. The **`FrIf_JobListExec_<ClstIdx>`** function can be directly called in the call-back function of the FlexRay timer interrupt that is provided by the FlexRay Driver. An example for the Vector FlexRay Driver is depicted in the following figure.

Example

```
void ApplFr_ISR_Timer0(void)
{
  FrIf_JobListExec_0();
}
```

Figure 2-5 The FrIf_JobListExec_0() function is directly called in the call-back function for FlexRay timer 0.

2.4.2.2 Job List Execution in Task Context

Depending on the platform being used, the execution of the job list can take up to several hundred microseconds. Thus it is not always advisable to execute the job list in the context of the FlexRay timer interrupt.

An example for the job list execution in the context of an OS task is depicted in the following figure.

Example

```
void ApplFr_ISR_Timer0(void)
{
   ActivateTask(FrIfJobExecTask);
}

TASK(FrIfJobExecTask)
{
  FrIf_JobListExec_0();
  (void)TerminateTask();
}
```

Figure 2-6 The FrIf_JobListExec_0() function is called inside a task that is activated in the call-back function for FlexRay timer 0.

2.5 Error Handling

2.5.1 Development Error Reporting

Development errors are reported to DET using the service Det_ReportError(), (specified in [2]), if the pre-compile option "Dev Error Detect" is set.

The reported **FrIf** ID is 61.

The service IDs from the following table identify the services which are described in section 4.2.

Service ID	Service
`FrIf_ReconfigLPdu`	0x00
`FrIf_GetVersionInfo`	0x01
`FrIf_Init`	0x02
`FrIf_ControllerInit`	0x03
`FrIf_StartCommunication`	0x04
`FrIf_HaltCommunication`	0x05
`FrIf_AbortCommunication`	0x06
`FrIf_GetState`	0x07
`FrIf_SetState`	0x08
`FrIf_SetWakeupChannel`	0x09
`FrIf_SendWUP`	0x0A
`FrIf_GetPOCStatus`	0x0D
`FrIf_GetGlobalTime`	0x0E
`FrIf_GetNmVector`	0x0F
`FrIf_AllowColdstart`	0x10
`FrIf_AllSlots`	0x33
`FrIf_GetNumOfStartupFrames`	0x34
`FrIf_GetMacroticksPerCycle`	0x11
`FrIf_GetWakeupRxStatus`	0x2B
`FrIf_Transmit`	0x30
`FrIf_SetTransceiverMode`	0x13
`FrIf_GetTransceiverMode`	0x14
`FrIf_GetTransceiverWUReason`	0x15
`FrIf_EnableTransceiverWakeup`	0x36
`FrIf_DisableTransceiverWakeup`	0x37

Service ID	Service
`FrIf_ClearTransceiverWakeup`	0x18
`FrIf_EnableTransceiverBranch`	0x36
`FrIf_DisableTransceiverBranch`	0x37
`FrIf_GetTransceiverError`	0x35
`FrIf_CheckWakeupByTransceiver`	0x39
`FrIf_SetAbsoluteTimer`	0x19
`FrIf_CancelAbsoluteTimer`	0x1B
`FrIf_EnableAbsoluteTimerIRQ`	0x1D
`FrIf_GetAbsoluteTimerIRQStatus`	0x1F
`FrIf_AckAbsoluteTimerIRQ`	0x21
`FrIf_DisableAbsoluteTimerIRQ`	0x23
`FrIf_Cbk_WakeupByTransceiver`	0x39
`FrIf_GetChannelStatus (optional)`	0x26
`FrIf_MainFunction_<ClstIdx>`	0x27
`FrIf_DisableLPdu (optional)`	0x28
`FrIf_GetClockCorrection (optional)`	0x29
`FrIf_GetCycleLength`	0x3A
`FrIf_GetSyncFrameList (optional)`	0x2A
`FrIf_GetMacrotickDuration`	0x31
`FrIf_JobListExec_<ClstIdx>`	0x32
`FrIf_CancelTransmit (optional)`	0x30
`FrIf_ReadCCConfig (optional)`	0x3B
`FrIf_ExecCurrentJob`	0x40
`FrIf_TriggerTransmitFunctions`	0x41
`FrIf_GetTxConflictStatus`	0x44

Table 2-7 Mapping of service IDs to services

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

Error Code		Description
0x01	`FRIF_E_INV_POINTER`	Invalid pointer
0x02	`FRIF_E_INV_CTRL_IDX`	Invalid Controller index
0x03	`FRIF_E_INV_CLST_IDX`	Invalid Cluster index
0x04	`FRIF_E_INV_CHNL_IDX`	Invalid Channel index
0x05	`FRIF_E_INV_TIMER_IDX`	Invalid timer index
0x06	`FRIF_E_INV_TXPDUID`	Invalid FrIf_TxPdu Index
0x08	`FRIF_E_NOT_INITIALIZED`	FrIf not initialized
0x0A	`FRIF_E_INV_LPDU_IDX`	Invalid L-PDU Index
0x0B	`FRIF_E_INV_FRAME_ID`	Invalid Frame Index
0x26	`FRIF_E_TXTASK_RET_E_NOT_OK`	FrIf_TxTask_0 returned E_NOT_OK
0x27	`FRIF_E_INVALID_PDU_OWNER`	Invalid Pdu Owner
0x09	`FRIF_E_JLE_SYNC`	Job List Execution lost synchronization to the FlexRay Global Time

Table 2-8 Errors reported to DET

2.5.1.1 Parameter Checking

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

Info
Note that the FRIF_E_INV_CTRL_IDX error is only reported to DET by the FlexRay Interface if the "Single Channel API" feature is disabled.

Check / Service	FRIF_E_INV_POINTER	FRIF_E_INV_CTRL_IDX	FRIF_E_INV_CLST_IDX	FRIF_E_INV_CHNL_IDX	FRIF_E_INV_TIMER_IDX	FRIF_E_INV_TXPDUID	FRIF_E_NOT_INITIALIZED	FRIF_E_TXTASK_RET_E_NOT_OK	FRIF_E_INVALID_PDU_OWNER	FRIF_E_INV_LPDU_IDX	FRIF_E_INV_FRAME_ID
FrIf_ReconfigLPdu		■		■			■			■	■
FrIf_GetVersionInfo	■										
FrIf_Init	■										
FrIf_ControllerInit		■					■				
FrIf_StartCommunication		■					■				
FrIf_HaltCommunication		■					■				
FrIf_AbortCommunication		■					■				
FrIf_GetState	■		■				■				
FrIf_SetState			■				■				
FrIf_SetWakeupChannel		■					■				
FrIf_SendWUP		■					■				
FrIf_GetPOCStatus	■	■					■				
FrIf_GetGlobalTime	■	■					■				
FrIf_GetNmVector	■	■					■				
FrIf_GetTxConflictStatus	■	■					■			■	
FrIf_AllowColdstart		■					■				
FrIf_GetMacrotickDuration							■				
FrIf_Transmit	■					■	■				
FrIf_SetTransceiverMode		■		■			■				
FrIf_GetTransceiverMode	■	■		■			■				
FrIf_GetTransceiverWUReason	■	■		■			■				
FrIf_ClearTransceiverWakeup		■		■			■				
FrIf_SetAbsoluteTimer		■			■		■				
FrIf_CancelAbsoluteTimer		■			■		■				
FrIf_EnableAbsoluteTimerIRQ		■			■		■				
FrIf_GetAbsoluteTimerIRQStatus	■	■			■		■				
FrIf_AckAbsoluteTimerIRQ		■			■		■				
FrIf_DisableAbsoluteTimerIRQ		■			■		■				
FrIf_GetChannelStatus	■	■					■				

Check / Service	FRIF_E_INV_POINTER	FRIF_E_INV_CTRL_IDX	FRIF_E_INV_CLST_IDX	FRIF_E_INV_CHNL_IDX	FRIF_E_INV_TIMER_IDX	FRIF_E_INV_TXPDUID	FRIF_E_NOT_INITIALIZED	FRIF_E_TXTASK_RET_E_NOT_OK	FRIF_E_INVALID_PDU_OWNER	FRIF_E_INV_LPDU_IDX	FRIF_E_INV_FRAME_ID
FrIf_MainFunction_<ClstIdx>							■[1]				
FrIf_DisableLPdu		■					■			■	
FrIf_GetClockCorrection	■	■					■				
FrIf_GetSyncFrameList	■	■					■				
FrIf_GetMacrotickDuration	■	■									
FrIf_JobListExec_<ClstIdx>							■				
FrIf_CancelTransmit						■	■				
FrIf_ReadCCConfig	■	■					■				
FrIf_ExecCurrentJob_0								■			
FrIf_GetCycleLength	■	■									
FrIf_TriggerTransmitFunctions									■		
FrIf_TxConfirmationFunctions									■		
FrIf_RxIndicationFunctions									■		
FrIf_AllSlots		■					■				
FrIf_GetMacroticksPerCycle		■					■				
FrIf_GetNumOfStartupFrames	■	■					■				
FrIf_GetWakeupRxStatus	■	■					■				
FrIf_EnableTransceiverBranch		■		■			■				
FrIf_DisableTransceiverBranch		■		■			■				
FrIf_GetTransceiverError	■	■		■			■				
FrIf_CheckWakeupByTransceiver		■		■			■				

Table 2-9 Development Error Reporting: Assignment of checks to services

2.5.2 Production Code Error Reporting

Production code related errors are reported to DEM using the service Dem_ReportErrorStatus() (specified in [3]), if the pre-compile parameter "Prod Error Detect" option is enabled.

[1] Note: this DET error is not reported when using a multiple configuration ECUC.

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

Error Code	Description
`FRIF_E_NIT_CH_A`	DEM_EVENT_STATUS_FAILED: is reported when error bits for the NIT on channel A are set in the return value of Fr_GetChannelStatus. DEM_EVENT_STATUS_PASSED: is reported when no error bits are set in the return value of Fr_GetChannelStatus.
`FRIF_E_NIT_CH_B`	DEM_EVENT_STATUS_FAILED: is reported when error bits for the NIT on channel B are set in the return value of Fr_GetChannelStatus. DEM_EVENT_STATUS_PASSED: is reported when no error bits are set in the return value of Fr_GetChannelStatus.
`FRIF_E_SW_CH_A`	DEM_EVENT_STATUS_FAILED: is reported when error bits for the SW on channel A are set in the return value of Fr_GetChannelStatus. DEM_EVENT_STATUS_PASSED: is reported when no error bits are set in the return value of Fr_GetChannelStatus.
`FRIF_E_SW_CH_B`	DEM_EVENT_STATUS_FAILED: is reported when error bits for the SW on channel B are set in the return value of Fr_GetChannelStatus. DEM_EVENT_STATUS_PASSED: is reported when no error bits are set in the return value of Fr_GetChannelStatus.
`FRIF_E_ACS_CH_A`	DEM_EVENT_STATUS_FAILED: is reported when error bits for the ACS on channel A are set in the return value of Fr_GetChannelStatus. DEM_EVENT_STATUS_PASSED: is reported when no error bits are set in the return value of Fr_GetChannelStatus.

`FRIF_E_ACS_CH_B`	DEM_EVENT_STATUS_FAILED: is reported when error bits for the ACS on channel B are set in the return value of Fr_GetChannelStatus. DEM_EVENT_STATUS_PASSED: is reported when no error bits are set in the return value of Fr_GetChannelStatus.

Table 2-10 Errors reported to DEM

2.6 Transmission

2.6.1 Decoupled Transmission

Usually the Decoupled Transmission is used for the transmission of PDUs. Decoupled Transmission is mandatory in case a frame contains multiple PDUs.

When an upper layer software module calls the **`FrIf_Transmit`** method of the FlexRay Interface, the parameter that holds the pointer to the PDU content is not evaluated. It is just stored that the given PDU shall be transmitted. When the frame that contains the PDU is assembled, the FlexRay Interface calls the call-back function **`<UL>_TriggerTransmit`** where **`<UL>`** is the name of the upper layer module. The upper layer module then copies the PDU content to the memory location that is given as a parameter of the call-back function.

A frame that contains multiple PDUs will be transmitted, if the **`FrIf_Transmit`** function was called for at least one of its PDUs. The FlexRay Interface does not buffer any PDU content. The **`<UL>_TriggerTransmit`** function is only called for those PDUs for which **`FrIf_Transmit`** was called.

When a frame is transmitted and **`FrIf_Transmit`** has not been called for each PDU contained in the frame, the FlexRay Interface cannot transmit the previous value of those PDUs that have not been updated. Instead, the FlexRay Interface uses update-bits to indicate which PDUs of a frame are valid.

2.6.2 Immediate Transmission

If a PDU is configured for immediate transmission, the **`FrIf_Transmit`** will immediately copy the PDU content to the FlexRay Driver, i.e. the **`<UL>_TriggerTransmit`** call-back will not be used.

However, the following constraints have to hold true:

- Immediate Transmission cannot be used, if there are multiple PDUs in a frame. I.e. a PDU that is configured for immediate transmission must be the only PDU of the frame.
- The upper layer software module must not call the **`FrIf_Transmit`** function when the transmit buffer of the FlexRay communication controller is transmitting data. I.e. the upper layer software module must be synchronized to the FlexRay global time.

2.7 Reception

The FlexRay Interface cannot be polled for received data. When a frame is received, the reception of the PDUs that are contained in the frame is actively indicated by the FlexRay Interface using the <UL>_RxIndication call-back function.

2.8 Timer Handling

The FlexRay Interface uses one absolute timer of the FlexRay communication controller for the execution of the job list. Therefore the application can only use one FlexRay timer if the communication controller provides at least two timers.

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

Info
The FlexRay protocol standard only requires one absolute timer. Many FlexRay communication controllers provide two timers, but there are controllers that only have one timer.

The FlexRay Interface uses timer 0 of the FlexRay communication controller. Thus the application can only use timer 1, provided that it is available.

2.9 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 message buffers that are used by the FlexRay Driver. The MICROSAR FlexRay Driver triggers the buffer reconfiguration automatically (the FrIf does not need to call the `Fr_PrepareLPdu` function). For a detailed description of the buffer reconfiguration feature of the MICROSAR FlexRay Driver refer to [5].

2.10 L-PDU Reconfiguration

The MICROSAR FlexRay Interface supports the AUTOSAR 4.0 L-PDU reconfiguration feature. This feature allows the FlexRay Interface to reconfigure the frame ID, channel, cycle repetition, cycle offset, payload length and the header CRC at runtime for a given L-PDU.

The L-PDU reconfiguration can be enabled with the attribute “Reconfig LPdu Support” within the **FrIf** module

Only FrIf PDUs with the “Reconfigurable” option enabled can be reconfigured/disabled by using the API services `FrIf_ReconfigLPdu` and `FrIf_DisableLPdu`.

Info
Note that if L-PDU reconfiguration is enabled for a specific L-PDU, this L-PDU is not sent after `FrIf_ControllerInit` but has to be enabled with `FrIf_ReconfigLPdu`. Therewith it is not allowed to set the "Reconfigurable" flag for a sync frame.

2.11 Dual Channel Redundancy Support

The MICROSAR FlexRay Interface supports optional feature for the replicated reception and transmission on a FlexRay dual channel (or single channel) cluster according to Bugzillla 42025. This feature allows the FlexRay Interface to transmit a L-PDU in different slots either on one channel or on both channels with the same payload.

If two Tx frames (i.e. frame1 and frame2 of Figure 2-2) are configured to be redundant to each other the FlexRay Interface assembles the frame only once for both frames in order to ensure that the identical frame content is transmitted.

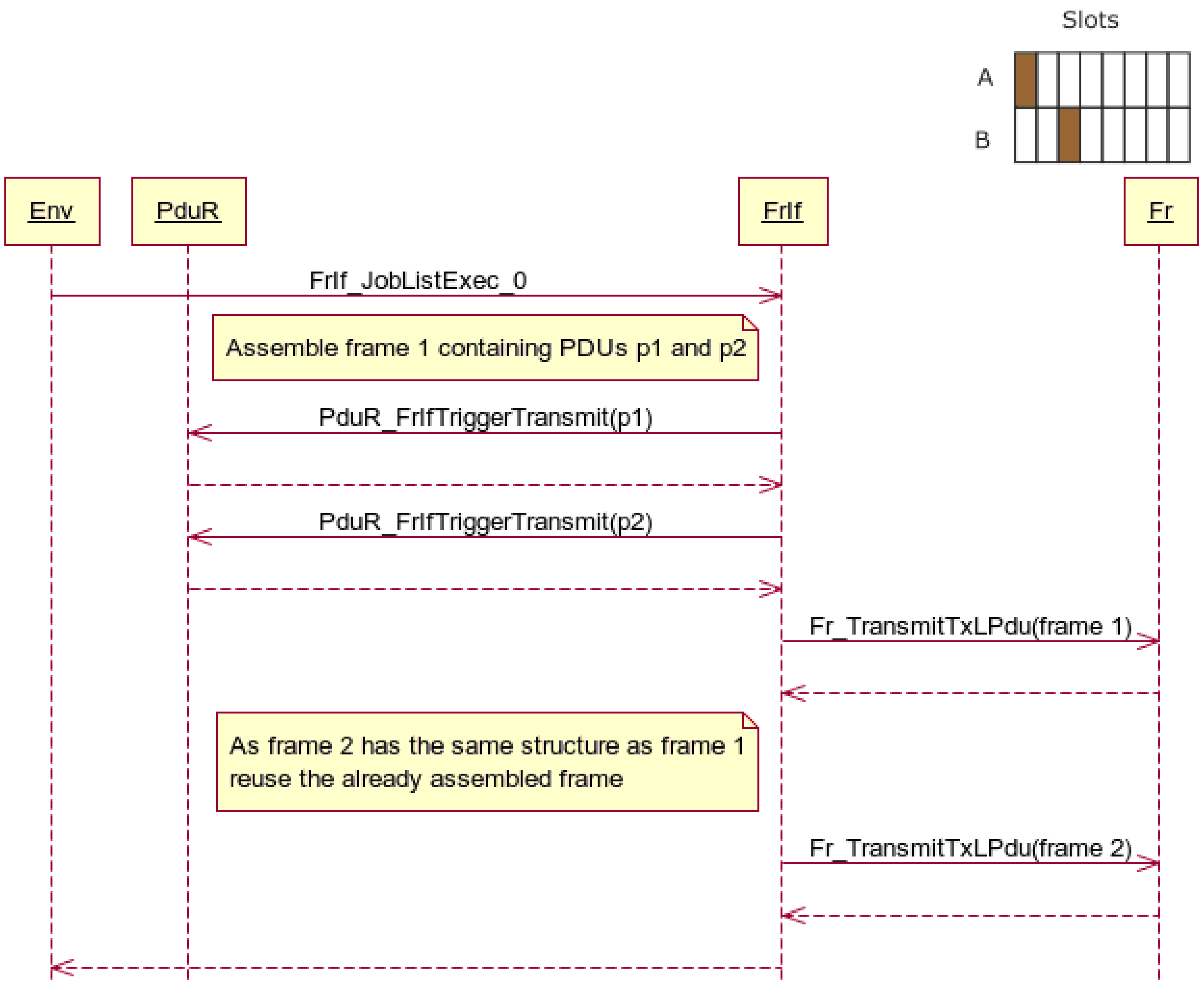
Slots
A
B
Env
PduR
FrIf
Fr
FrIf_JobListExec_0
Assemble frame 1 containing PDUs p1 and p2
PduR_FrIfTriggerTransmit(p1)
PduR_FrIfTriggerTransmit(p2)
Fr_TransmitTxLPdu(frame 1)
As frame 2 has the same structure as frame 1 reuse the already assembled frame
Fr_TransmitTxLPdu(frame 2)

Figure 2-2 Behaviour of FrIf for replicated transmission

Additionally this feature allows the application above the FlexRay Interface to use a payload comparison between the received PDUs of two redundant Rx frames as depicted in Figure 2-3. This payload comparison can be used to check whether a I-PDU was received on both channels or not.

For a detailed API description of the voting function refer to section 4.5.2.4.

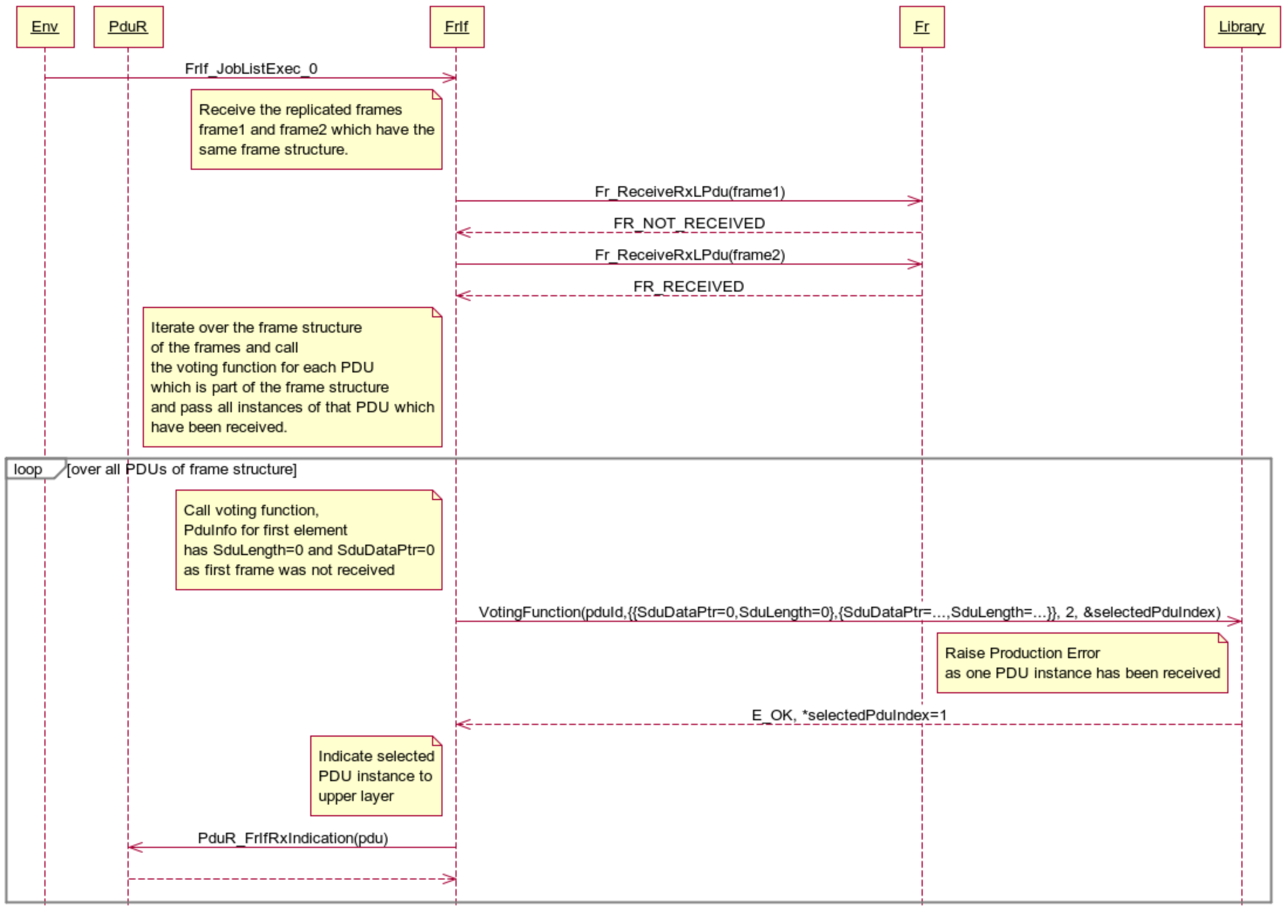
Env
PduR
FrIf
Fr
Library
FrIf_JobListExec_0
Receive the replicated frames frame1 and frame2 which have the same frame structure.
Fr_ReceiveRxLPdu(frame1)
FR_NOT_RECEIVED
Fr_ReceiveRxLPdu(frame2)
FR_RECEIVED
Iterate over the frame structure of the frames and call the voting function for each PDU which is part of the frame structure and pass all instances of that PDU which have been received.
loop
[over all PDUs of frame structure]
Call voting function, PduInfo for first element has SduLength=0 and SduDataPtr=0 as first frame was not received
VotingFunction(pduId,{{SduDataPtr=0,SduLength=0},{SduDataPtr=...,SduLength=...}}, 2, &selectedPduIndex)
Raise Production Error as one PDU instance has been received
E_OK, *selectedPduIndex=1
Indicate selected PDU instance to upper layer
PduR_FrIfRxIndication(pdu)

Figure 2-3 Behaviour of FrIf for replicated reception

Info

Only frames that have frame triggerings with the same base cycle, cycle repetition, LSdu length and the same frame layout can be configured to be redundant to each other.

The frame layout of two frames is identical if both frames use the same number of PDUs and the redundant PDUs within the frames have the same:

- PDU length, PDU offset and PDU owner
- PDU update bit configuration

The Tx PDUs for replicated transmission may not have the "Immediate" flag enabled.

Redundant reception and FIFOs are mutually exclusive, so no redundant frames are allowed within the slot range of FIFOs.

2.12 Dynamic Payload

The dynamic payload feature allows the **FrIf** to pass only the actual used L-PDU length via `FrIf_Transmit`() to the driver (Fr_TransmitTxLPdu()). If the feature is enabled, the FR module 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 defined within FIBEX or EcuC.

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

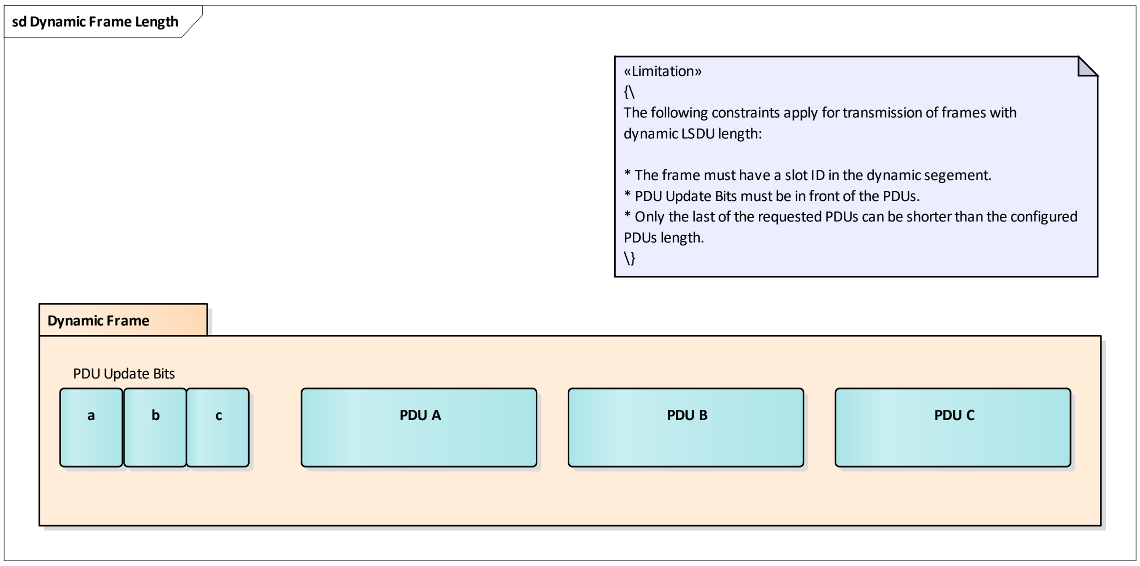
sd Dynamic Frame Length
«Limitation»
{\
The following constraints apply for transmission of frames with dynamic LSDU length:
* The frame must have a slot ID in the dynamic segement.
* PDU Update Bits must be in front of the PDUs.
* Only the last of the requested PDUs can be shorter than the configured PDUs length.
\}
Dynamic Frame
PDU Update Bits
a
b
c
PDU A
PDU B
PDU C

Figure 2-4 Usage scenarios of Dynamic Payload

As mentioned in Figure 2-4 there are some limitations for the usage of dynamic L-SDU length:

- Frames in the static segment cannot be shortened at runtime because the FlexRay protocol enforces that all frames in the static segement are transmitted with the same payload length (due to FrIfGPayloadLengthStatic).
- Also if the PDUs Update Bits are configured to be at the end of the frame, the frame cannot be shortened as expected because otherwise the PDU Update Bit information would be cut off and the receiver cannot determine the PDUs reception properly.

- If there are multiple PDUs within a frame it is not possible to request the transmission of a shortened PDU which has a lower FrIfPduOffset than another PDU which is also requested for transmission. If for example both PDU A and PDU C of Figure 2-4 are transmitted with a lower PDU length than configured the FrIf will use padding to define the frame content between the end of PDU A and the beginning of PDU C. In this case the payload of PDU A will partly be overwritten with the FrIfUnusedBitValue and the receiver of the PDU cannot distinquish between the actual payload and the padded content of the PDU. If only PDU A or PDU C is transmitted this issue does not occur.

3 Integration

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

3.1 Scope of Delivery

The delivery of the **FrIf** contains the files which are described in the chapters 3.1.1 and 3.1.2:

3.1.1 Static Files

File Name	Description
`FrIf.c`	Static source for FlexRay Interface core.
`FrIf.h`	Static header file for FlexRay Interface API.
`FrIf_Cbk.h`	Static header file for callbacks.
`FrIf_LCfg.h`	Static header file for link time configuration data.
`FrIf_AbsTimer.c`	Static source for absolute time handling.
`FrIf_Priv.h`	Static header file for private prototypes and macros.
`FrIf_Ext.c`	Static source for external synchronization API.
`FrIf_Rx.c`	Static source for reception handling.
`FrIf_Time.c`	Static source for time services.
`FrIf_Trcv.c`	Static source for transceiver handling.
`FrIf_Tx.c`	Static source for transmission handling.

Table 3-1 Static files

3.1.2 Dynamic Files

The dynamic files are generated by the configuration tool

.

File Name	Description
`FrIf_Cfg.h`	Generated header file for pre-compile time configuration data.
`FrIf_LCfg.c`	Generated source for link time configuration data.
`FrIf_PBcfg.c`	Generated source for post-build time configuration data.
`FrIf_PBcfg.h`	Generated header file for post-build time configuration data.
`FrIf_Types.h`	Generated header file for FlexRay Interface type definitions.

Table 3-2 Generated files

3.2 Include Structure

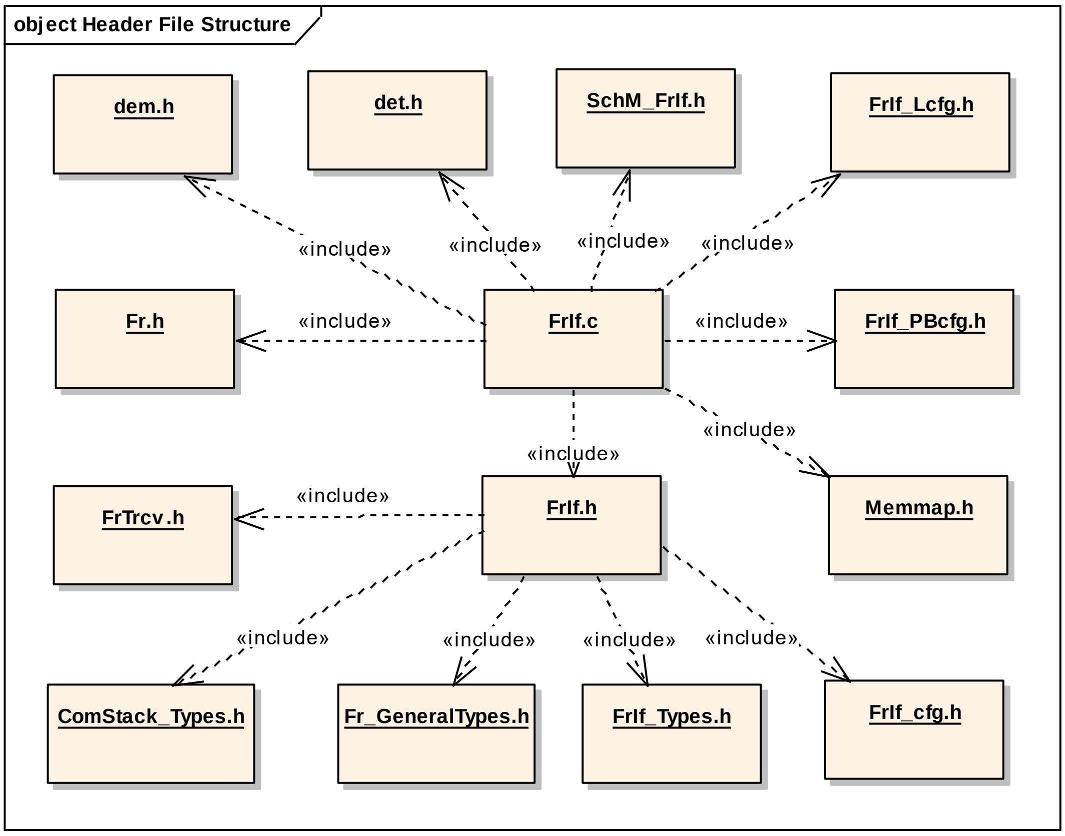

Figure 3-1 Include structure

3.3 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.

The following table contains the memory section names and the compiler abstraction definitions defined for the **FrIf** and illustrates their assignment among each other.

Memory Mapping Sections \ Compiler Abstraction Definitions	FRIF_VAR_NOINIT	FRIF_CONST	FRIF_PBCFG	FRIF_CODE	FRIF_APPL_DATA
`FRIF_START_SEC_CODE` `FRIF_STOP_SEC_CODE`				■	
`FRIF_START_SEC_PBCFG` `FRIF_STOP_SEC_PBCFG`			■		
`FRIF_START_SEC_CONST_32BIT` `FRIF_STOP_SEC_CONST_32BIT`		■			
`FRIF_START_SEC_CONST_UNSPECIFIED` `FRIF_STOP_SEC_CONST_UNSPECIFIED`		■			
`FRIF_START_SEC_VAR_NOINIT_UNSPECIFIED` `FRIF_STOP_SEC_VAR_NOINIT_UNSPECIFIED`	■				

Table 3-3 Compiler abstraction and memory mapping

The Compiler Abstraction Definition `FRIF_APPL_DATA` is used to address code, variables and constants which are declared by other modules and used by the FlexRay Interface.

3.4 Critical Sections and Exclusive Areas

3.4.1 FRIF_EXCLUSIVE_AREA_0

This exclusive area is used to avoid modifications to the FrIf_TxPduDirtyBits and/or FrIf_TxPduTxRequestCounters arrays from different contexts, by preventing that the functions **`FrIf_Transmit`**, **`FrIf_CancelTransmit`** and **`FrIf_JobListExec_<ClstIdx>`** interrupt themselves. It can be ommited if the 2 following conditions are fulfilled:

> The parameter FrIfPduDirtyByteUsage is set to true.

> The parameter FrIfCounterLimitDisable is set to true.

A global interrupt lock is recommended, since the 3 functions could be called from different interrupt contexts if they have different priorities (e.g., in a routing scenario **`FrIf_Transmit`** could be called from an CAN RxIndication interrupt while the **`FrIf_JobListExec_<ClstIdx>`** is being executed or vice versa).

This exclusive area has a medium duration if the flags and counters of all the PDUs in a frame must be rolled back. However, under normal conditions the whole processing shouldn't take long.

3.4.2 FRIF_EXCLUSIVE_AREA_1

This exclusive area is used to protect the time-critical estimation and setting of the timer for the next job list execution that takes place within the **`FrIf_JobListExec_<ClstIdx>`**, **`FrIf_MainFunction_<ClstIdx>`** and **`FrIf_SetState`** functions.

Any tasks or interrupts of higher priority must be blocked, since the estimation and setting of the next timer interrupt shall not be interrupted.

It's not possible to tell the exact duration of this exclusive area, since the functions Fr_GetGlobalTime and Fr_SetAbsoluteTimer from the Fr are called. However, if no errors take place within these functions the whole processing shouldn't take long.

3.4.3 FRIF_EXCLUSIVE_AREA_2

This exclusive area is used to ensure the required atomicity during initialization, by preventing that the function **`FrIf_SetState`** gets interrupted by the **`FrIf_JobListExec_<ClstIdx>`**.

At least the FlexRay timer interrupt must be blocked within this exclusive area, since the **`FrIf_JobListExec_<ClstIdx>`** is usually executed within that interrupt context.

It's not possible to tell the exact duration of this exclusive area, since the functions Fr_GetGlobalTime and Fr_SetAbsoluteTimer from the Fr are called. However, if no errors take place within these functions the whole processing shouldn't take long.

4 API Description

For an interfaces overview please see Figure 1-2.

4.1 Type Definitions

Type Name	C-Type	Description	Value Range
`FrIf_StateType`	Enum	Representation of the FrIf job list execution status	`FRIF_STATE_OFFLINE` Job list is not executed
			`FRIF_STATE_ONLINE` Job list is executed
`FrIf_StateTransitionType`	Enum	Start or stop the the FrIf job list execution	`FRIF_GOTO_OFFLINE` Stop the job list execution.
			`FRIF_GOTO_ONLINE` Start the job list is execution.

Table 4-1 Type definitions

4.2 Services provided by FrIf

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

4.2.1 FrIf_AbortCommunication

Prototype	
`Std_ReturnType` **`FrIf_AbortCommunication`**`(uint8 FrIf_CtrlIdx)`	
Parameter	
FrIf_CtrlIdx	Index of the FlexRay communication controller for which Fr_AbortCommunication shall be called.
Return code	
E_OK	The call of the FlexRay Driver API service Fr_AbortCommunication() has returned E_OK.
E_NOT_OK	The call of the FlexRay Driver API service Fr_AbortCommunication() has returned E_NOT_OK, or an error has been detected if development error detection is enabled.
Functional Description	
Call the function Fr_AbortCommunication of the FlexRay Driver.	
Particularities and Limitations	
> Precondition: The FlexRay Interface has to be initialized with a call of **`FrIf_Init`**. > This API function is only available if "AbortCommunication Disable" has not been set for FlexRay Interface. > This function is implemented as macro to reduce code size if the "Wrapper APIs As Macro" option is enabled for the FlexRay Interface.	
Expected Caller Context	

> This function can be called in any context.

Table 4-2 FrIf_AbortCommunication

4.2.2 FrIf_AckAbsoluteTimerIRQ

Prototype	
`Std_ReturnType` **`FrIf_AckAbsoluteTimerIRQ`**`(uint8 FrIf_CtrlIdx, uint8 FrIf_AbsTimerIdx)`	
Parameter	
FrIf_CtrlIdx	The index of the FlexRay Communication Controller.
FrIf_AbsTimerIdx	Index of the absolute timer.
Return code	
E_OK	The call of the FlexRay Driver API service Fr_AckAbsoluteTimerIRQ() has returned E_OK.
E_NOT_OK	The call of the FlexRay Driver API service Fr_AckAbsoluteTimerIRQ() has returned E_NOT_OK, or an error has been detected if development error detection is enabled.
Functional Description	
This function wraps the Fr_AckAbsoluteTimerIRQ function of the FlexRay Driver.	
Particularities and Limitations	
> Precondition: The FlexRay Interface has to be initialized with a call of **`FrIf_Init`**. > This function is not available if the parameter FrIf/FrIfGeneral/FrIfAbsTimerIdx is set to 0. > This function is implemented as macro to reduce code size if the “Wrapper APIs As Macro” option is enabled for the FlexRay Interface.	
Expected Caller Context	
> This function must be called by the application or integration code in the context of the timer interrupt service routine.	

Table 4-3 FrIf_AckAbsoluteTimerIRQ

4.2.3 FrIf_AllowColdstart

Prototype	
`Std_ReturnType` **`FrIf_AllowColdstart`**`(uint8 FrIf_CtrlIdx)`	
Parameter	
FrIf_CtrlIdx	Index of the FlexRay communication controller for which Fr_AllowColdstart shall be called.
Return code	
E_OK	The call of the FlexRay Driver API service Fr_AllowColdstart() has returned E_OK.
E_NOT_OK	The call of the FlexRay Driver API service Fr_AllowColdstart() has returned E_NOT_OK, or an error has been detected if development error detection is enabled.

Functional Description
Call the function Fr_AllowColdstart of the FlexRay Driver.
Particularities and Limitations
> Precondition: The FlexRay Interface has to be initialized with a call of **`FrIf_Init`**. > This function is implemented as macro to reduce code size if the "Wrapper APIs As Macro" option is enabled for the FlexRay Interface.
Expected Caller Context
> This function can be called in any context.

Table 4-4 FrIf_AllowColdstart

4.2.4 FrIf_AllSlots

Prototype	
`Std_ReturnType` **`FrIf_AllSlots`**`(uint8 FrIf_CtrlIdx)`	
Parameter	
FrIf_CtrlIdx	Index of the FlexRay communication controller for which Fr_AllSlots shall be called.
Return code	
E_OK	The call of the FlexRay Driver API service Fr_AllSlots() has returned E_OK.
E_NOT_OK	The call of the FlexRay Driver API service Fr_AllSlots() has returned E_NOT_OK, or an error has been detected if development error detection is enabled.
Functional Description	
Call the function Fr_AllSlots of the FlexRay Driver.	
Particularities and Limitations	
> Precondition: The FlexRay Interface has to be initialized with a call of **`FrIf_Init`**. > This API function is only available if the "All Slots Support" has been enabled for FlexRay Interface. > This function is implemented as macro to reduce code size if the "Wrapper APIs As Macro" option is enabled for the FlexRay Interface.	
Expected Caller Context	
> This function can be called in any context.	

Table 4-5 FrIf_AllSlots

4.2.5 FrIf_CancelAbsoluteTimer

Prototype
`Std_ReturnType` **`FrIf_CancelAbsoluteTimer`**`(uint8 FrIf_CtrlIdx, uint8 FrIf_AbsTimerIdx)`

Parameter	
FrIf_CtrlIdx	The index of the FlexRay Communication Controller.
FrIf_AbsTimerIdx	Index of the absolute timer.
Return code	
E_OK	The call of the FlexRay Driver API service Fr_CancelAbsoluteTimer() has returned E_OK.
E_NOT_OK	The call of the FlexRay Driver API service Fr_CancelAbsoluteTimer() has returned E_NOT_OK, or an error has been detected if development error detection is enabled.
Functional Description	
This function wraps the Fr_CancelAbsoluteTimer function of the FlexRay Driver.	
Particularities and Limitations	
> Precondition: The FlexRay Interface has to be initialized with a call of **`FrIf_Init`**. > This function is not available if the parameter FrIf/FrIfGeneral/FrIfAbsTimerIdx is set to 0. > As one FlexRay timer is used by the FlexRay Interface this function must not be used for timer 0. > This function is implemented as macro to reduce code size if the "Wrapper APIs As Macro" option is enabled for the FlexRay Interface.	
Expected Caller Context	
> This function can be called in any context.	

Table 4-6 FrIf_CancelAbsoluteTimer

4.2.6 FrIf_CancelTransmit (optional)

Prototype	
`Std_ReturnType` **`FrIf_CancelTransmit`**`(PduIdType FrIf_TxPduId)`	
Parameter	
FrIf_TxPduId	FrIf-ID of the PDU to be cancelled.
Return code	
E_OK	The cancelation request has been accepted.
E_NOT_OK	The cancelation request has been rejected, or an error has been detected if development error detection is enabled.
Functional Description	
This function cancels the transmission of a PDU by clearing the number of indications of ready PDU data.	
Particularities and Limitations	
> Precondition: The FlexRay Interface has to be initialized with a call of **`FrIf_Init`**. > This API function is only available if the "Cancel Transmit Support" has been enabled for FlexRay Interface. > During its runtime this function temporarily disables all interrupts.	

Expected Caller Context
> This function can be called in any context.

Table 4-7 FrIf_CancelTransmit (optional)

4.2.7 FrIf_CheckWakeupByTransceiver

Prototype	
`void` **`FrIf_CheckWakeupByTransceiver`**`(uint8 FrIf_CtrlIdx, Fr_ChannelType FrIf_ChnlIdx)`	
Parameter	
FrIf_CtrlIdx	The index of the FlexRay communication controller to which the transceiver is connected.
FrIf_ChnlIdx	The index of the FlexRay channel to which the transceiver is connected.
Return code	
E_OK	The call of the FlexRay Transceiver Driver API service FrTrcv_CheckWakeupByTransceiver() has returned E_OK.
E_NOT_OK	The call of the FlexRay Transceiver Driver API service FrTrcv_CheckWakeupByTransceiver() has returned E_NOT_OK, or an error has been detected if development error detection is enabled.
Functional Description	
This function wraps the FrTrcv_CheckWakeupByTransceiver function of the FlexRay Transceiver Driver.	
Particularities and Limitations	
> Precondition: The FlexRay Interface has to be initialized with a call of **`FrIf_Init`**. > This API function is only available if the MICROSAR FlexRay Transceiver component is enabled in configuration tool. > This function is implemented as macro to reduce code size if the "Wrapper APIs As Macro" option is enabled for the FlexRay Interface.	
Expected Caller Context	
> This function can be called in any context.	

Table 4-8 FrIf_CheckWakupByTransceiver

4.2.8 FrIf_ClearTransceiverWakeup

Prototype	
`Std_ReturnType` **`FrIf_ClearTransceiverWakeup`**`(uint8 FrIf_CtrlIdx, Fr_ChannelType FrIf_ChnlIdx)`	
Parameter	
FrIf_CtrlIdx	The index of the FlexRay communication controller to which the transceiver is connected.
FrIf_ChnlIdx	The index of the FlexRay channel to which the transceiver is connected.

Return code	
E_OK	The call of the FlexRay Transceiver Driver API service FrTrcv_ClearTransceiverWakeup() has returned BUSTRCV_E_OK.
E_NOT_OK	The call of the FlexRay Transceiver Driver API service FrTrcv_ClearTransceiverWakeup() has returned BUSTRCV_E_ERROR, or an error has been detected if development error detection is enabled.
Functional Description	
This function wraps the FrTrcv_ClearTransceiverWakeup function of the FlexRay Transceiver Driver.	
Particularities and Limitations	
> Precondition: The FlexRay Interface has to be initialized with a call of **`FrIf_Init`**. > This API function is only available if the MICROSAR FlexRay Transceiver component is enabled in the configuration tool. > This function is implemented as macro to reduce code size if the "Wrapper APIs As Macro" option is enabled for the FlexRay Interface.	
Expected Caller Context	
> This function can be called in any context.	

Table 4-9 FrIf_ClearTransceiverWakeup

4.2.9 FrIf_ControllerInit

Prototype	
`void` **`FrIf_ControllerInit`**`(uint8 FrIf_CtrlIdx)`	
Parameter	
FrIf_CtrlIdx	Index of the FlexRay communication controller to be initialized.
Return code	
E_OK	The call of the FlexRay Driver API service Fr_ControllerInit() has returned E_OK.
E_NOT_OK	The call of the FlexRay Driver API service Fr_ControllerInit() has returned E_NOT_OK, or an error has been detected if development error detection is enabled.
Functional Description	
This function initializes one FlexRay communication controller by calling Fr_ControllerInit of the FlexRay Driver.	
Particularities and Limitations	
> Precondition: The FlexRay Interface has to be initialized with a call of **`FrIf_Init`**. > If this function is called during active communication, the communication of all FlexRay communication controllers will be terminated. > This function is implemented as macro to reduce code size if the "Wrapper APIs As Macro" option is enabled for the FlexRay Interface.	

Expected Caller Context
> This function can be called in any context.

Table 4-10 FrIf_ControllerInit

4.2.10 FrIf_DisableLPdu (optional)

Prototype	
`Std_ReturnType` **`FrIf_DisableLPdu`**`(uint8 FrIf_CtrlIdx, int16 FrIf_LPduIdx)`	
Parameter	
FrIf_CtrlIdx	Index of the FlexRay communication controller for which Fr_DisableLPdu shall be called.
FrIf_LPduIdx	This index is used to uniquely identify a FlexRay frame.
Return code	
E_OK	The call of the FlexRay Driver API service Fr_DisableLPdu() has returned E_OK.
E_NOT_OK	The call of the FlexRay Driver API service Fr_DisableLPdu() has returned E_NOT_OK, or an error has been detected if development error detection is enabled.
Functional Description	
According to AUTOSAR 4.0 this function wraps the Fr_DisableLPdu of the FlexRay Driver to disable the message buffer of the given L-PDU.	
Particularities and Limitations	
> Precondition: The FlexRay Interface has to be initialized with a call of **`FrIf_Init`**. > This function is not available if the parameter FrIf/FrIfGeneral/FrIfDisableLPduSupport is set to true. > Only the L-PDUs with the “Reconfigurable” flag enabled can be disabled by this API. > This function is implemented as macro to reduce code size if the “Wrapper APIs As Macro” option is enabled for the FlexRay Interface.	
Expected Caller Context	
> This function can be called in any context.	

Table 4-11 FrIf_DisableLPdu

4.2.11 FrIf_DisableAbsoluteTimerIRQ

Prototype	
`Std_ReturnType` **`FrIf_DisableAbsoluteTimerIRQ`**`(uint8 FrIf_CtrlIdx, uint8 FrIf_AbsTimerIdx)`	
Parameter	
FrIf_CtrlIdx	The index of the FlexRay Communication Controller.
FrIf_AbsTimerIdx	Index of the absolute timer.

Return code	
E_OK	The call of the FlexRay Driver API service Fr_DisableAbsoluteTimerIRQ() has returned E_OK.
E_NOT_OK	The call of the FlexRay Driver API service Fr_DisableAbsoluteTimerIRQ() has returned E_NOT_OK, or an error has been detected if development error detection is enabled.
Functional Description	
This function wraps the Fr_DisableAbsoluteTimerIRQ function of the FlexRay Driver.	
Particularities and Limitations	
> Precondition: The FlexRay Interface has to be initialized with a call of **`FrIf_Init`**. > This function is not available if the parameter FrIf/FrIfGeneral/FrIfAbsTimerIdx is set to 0. > This function is implemented as macro to reduce code size if the "Wrapper APIs As Macro" option is enabled for the FlexRay Interface.	
Expected Caller Context	
> This function can be called in any context.	

Table 4-12 FrIf_DisableAbsoluteTimerIRQ

4.2.12 FrIf_DisableTransceiverBranch

Prototype	
`Std_ReturnType` **`FrIf_DisableTransceiverWakeup`**`(uint8 FrIf_CtrlIdx, Fr_ChannelType FrIf_ChnlIdx, uint8 FrIf_BranchIdx)`	
Parameter	
FrIf_CtrlIdx	The index of the FlexRay communication controller to which the transceiver is connected.
FrIf_ChnlIdx	The index of the FlexRay channel to which the transceiver is connected.
FrIf_BranchIdx	This zero based index identifies the branch of the (active star) transceiver to which the API call has to be applied.
Return code	
E_OK	The call of the FlexRay Transceiver Driver API service FrTrcv_DisableTransceiverBranch() has returned E_OK.
E_NOT_OK	The call of the FlexRay Transceiver Driver API service FrTrcv_DisableTransceiverBranch() has returned E_NOT_OK, or an error has been detected if development error detection is enabled.
Functional Description	
This function wraps the FrTrcv_DisableTransceiverBranch function of the FlexRay Transceiver Driver.	

Particularities and Limitations
> Precondition: The FlexRay Interface has to be initialized with a call of **FrIf_Init**. > This API function is only available if the MICROSAR FlexRay Transceiver component is enabled in the configuration tool. > This function is implemented as macro to reduce code size if the "Wrapper APIs As Macro" option is enabled for the FlexRay Interface.
Expected Caller Context
> This function can be called in any context.

Table 4-13 FrIf_DisableTransceiverBranch

4.2.13 FrIf_EnableAbsoluteTimerIRQ

Prototype	
`Std_ReturnType **FrIf_EnableAbsoluteTimerIRQ**(uint8 FrIf_CtrlIdx, uint8 FrIf_AbsTimerIdx)`	
Parameter	
FrIf_CtrlIdx	The index of the FlexRay Communication Controller.
FrIf_AbsTimerIdx	Index of the absolute timer.
Return code	
E_OK	The call of the FlexRay Driver API service Fr_EnableAbsoluteTimerIRQ() has returned E_OK.
E_NOT_OK	The call of the FlexRay Driver API service Fr_EnableAbsoluteTimerIRQ() has returned E_NOT_OK, or an error has been detected if development error detection is enabled.
Functional Description	
This function wraps the Fr_EnableAbsoluteTimerIRQ function of the FlexRay Driver.	
Particularities and Limitations	
> Precondition: The FlexRay Interface has to be initialized with a call of **FrIf_Init**. > This function is not available if the parameter FrIf/FrIfGeneral/FrIfAbsTimerIdx is set to 0. > This function is implemented as macro to reduce code size if the "Wrapper APIs As Macro" option is enabled for the FlexRay Interface.	
Expected Caller Context	
> This function can be called in any context.	

Table 4-14 FrIf_EnableAbsoluteTimerIRQ

4.2.14 FrIf_EnableTransceiverBranch

Prototype
`Std_ReturnType **FrIf_EnableTransceiverBranch**(uint8 FrIf_CtrlIdx, Fr_ChannelType FrIf_ChnlIdx, uint8 FrIf_BranchIdx)`

Parameter	
FrIf_CtrlIdx	The index of the FlexRay communication controller to which the transceiver is connected.
FrIf_ChnlIdx	The index of the FlexRay channel to which the transceiver is connected.
FrIf_BranchIdx	This zero based index identifies the branch of the (active star) transceiver to which the API call has to be applied.
Return code	
E_OK	The call of the FlexRay Transceiver Driver API service FrTrcv_EnableTransceiverBranch() has returned E_OK.
E_NOT_OK	The call of the FlexRay Transceiver Driver API service FrTrcv_EnableTransceiverBranch() has returned E_NOT_OK, or an error has been detected if development error detection is enabled.
Functional Description	
This function wraps the FrTrcv_EnableTransceiverBranch function of the FlexRay Transceiver Driver.	
Particularities and Limitations	
> Precondition: The FlexRay Interface has to be initialized with a call of **`FrIf_Init`**. > This API function is only available if the MICROSAR FlexRay Transceiver component is enabled in the configuration tool. > This function is implemented as macro to reduce code size if the “Wrapper APIs As Macro” option is enabled for the FlexRay Interface.	
Expected Caller Context	
> This function can be called in any context.	

Table 4-15 FrIf_EnableTransceiverBranch

4.2.15 FrIf_GetAbsoluteTimerIRQStatus

Prototype	
`Std_ReturnType` **`FrIf_GetAbsoluteTimerIRQStatus`**`(uint8 FrIf_CtrlIdx, uint8 FrIf_AbsTimerIdx)`	
Parameter	
FrIf_CtrlIdx	The index of the FlexRay Communication Controller.
FrIf_AbsTimerIdx	Index of the absolute timer.
Return code	
E_OK	The call of the FlexRay Driver API service Fr_GetAbsoluteTimerIRQStatus() has returned E_OK.
E_NOT_OK	The call of the FlexRay Driver API service Fr_GetAbsoluteTimerIRQStatus() has returned E_NOT_OK, or an error has been detected if development error detection is enabled.
Functional Description	
This function wraps the Fr_GetAbsoluteTimerIRQStatus function of the FlexRay Driver.	

Particularities and Limitations
> Precondition: The FlexRay Interface has to be initialized with a call of **`FrIf_Init`**. > This function is not available if the parameter FrIf/FrIfGeneral/FrIfAbsTimerIdx is set to 0. > This function is implemented as macro to reduce code size if the "Wrapper APIs As Macro" option is enabled for the FlexRay Interface.
Expected Caller Context
> This function can be called in any context.

Table 4-16 FrIf_GetAbsoluteTimerIRQStatus

4.2.16 FrIf_GetChannelStatus (optional)

Prototype	
`Std_ReturnType` **`FrIf_GetChannelStatus`**`(uint8 FrIf_CtrlIdx, uint16* FrIf_ChannelAStatusPtr, uint16* FrIf_ChannelBStatusPtr)`	
Parameter	
FrIf_CtrlIdx	Index of the FlexRay communication controller for which Fr_GetGlobalTime shall be called.
FrIf_ChannelAStatusPtr	Address where the bitcoded channel A status information shall be stored
FrIf_ChannelBStatusPtr	Address where the bitcoded channel B status information shall be stored
Return code	
E_OK	The call of the FlexRay Driver API service Fr_GetChannelStatus() has returned E_OK.
E_NOT_OK	The call of the FlexRay Driver API service Fr_GetChannelStatus() has returned E_NOT_OK, or an error has been detected if development error detection is enabled.
Functional Description	
This function wraps the Fr_GetChannelStatus of the FlexRay Driver according to AUTOSAR 4.0.	
Particularities and Limitations	
> Precondition: The FlexRay Interface has to be initialized with a call of **`FrIf_Init`**. > This API function is only available if the "Get Channel Status Support" has been enabled for FlexRay Interface and FlexRay Driver. > This function is implemented as macro to reduce code size if the "Wrapper APIs As Macro" option is enabled for the FlexRay Interface.	
Expected Caller Context	
> This function can be called in any context.	

Table 4-17 FrIf_GetChannelStatus (optional)

4.2.17 FrIf_GetClockCorrection (optional)

Prototype	
`Std_ReturnType FrIf_GetClockCorrection(uint8 FrIf_CtrlIdx, sint16* FrIf_RateCorrectionPtr, sint32* FrIf_OffsetCorrectionPtr)`	
Parameter	
FrIf_CtrlIdx	Index of the FlexRay communication controller for which Fr_GetGlobalTime shall be called.
FrIf_RateCorrectionPtr	Address where the current rate correction value shall be stored.
FrIf_OffsetCorrectionPtr	Address where the current offset correction value shall be stored.
Return code	
E_OK	The call of the FlexRay Driver API service Fr_GetClockCorrection() has returned E_OK.
E_NOT_OK	The call of the FlexRay Driver API service Fr_GetClockCorrection() has returned E_NOT_OK, or an error has been detected if development error detection is enabled.
Functional Description	
This function wraps the Fr_GetClockCorrection of the FlexRay Driver according to AUTOSAR 4.0..	
Particularities and Limitations	
> Precondition: The FlexRay Interface has to be initialized with a call of `FrIf_Init`. > This API function is only available if the "Get Clock Correction Support" has been enabled for FlexRay Interface and FlexRay Driver. > This function is implemented as macro to reduce code size if the "Wrapper APIs As Macro" option is enabled for the FlexRay Interface.	
Expected Caller Context	
> This function can be called in any context.	

Table 4-18 FrIf_GetClockCorrection (optional)

4.2.18 FrIf_GetCycleLength

Prototype	
`uint32 FrIf_GetCycleLength(uint8 FrIf_CtrlIdx)`	
Parameter	
FrIf_CtrlIdx	Index of the FlexRay CC to address.
Return code	
uint32	Time in unit of nanoseconds.
Functional Description	
This API returns the configured time of the configuration parameter "GdCycle" in nanoseconds for the FlexRay controller with index FrIf_CtrlIdx.	

Particularities and Limitations
> Precondition: The FlexRay Interface has to be initialized with a call of **`FrIf_Init`**.
Expected Caller Context
> This function can be called in any context.

Table 4-19 FrIf_GetCycleLength

4.2.19 FrIf_GetGlobalTime

Prototype	
`Std_ReturnType` **`FrIf_GetGlobalTime`**`(uint8 FrIf_CtrlIdx, uint8* FrIf_CyclePtr, uint16* FrIf_MacroTickPtr)`	
Parameter	
FrIf_CtrlIdx	Index of the FlexRay communication controller for which Fr_GetGlobalTime shall be called.
FrIf_CyclePtr	Reference to the memory location the current FlexRay communication cycle will be stored at.
FrIf_MacroTickPtr	Reference to the memory location the current macrotick value will be stored at.
Return code	
E_OK	The call of the FlexRay Driver API service Fr_GetGlobalTime() has returned E_OK.
E_NOT_OK	The call of the FlexRay Driver API service Fr_GetGlobalTime() has returned E_NOT_OK, or an error has been detected if development error detection is enabled.
Functional Description	
Call the function Fr_GetGlobalTime of the FlexRay Driver.	
Particularities and Limitations	
> Precondition: The FlexRay Interface has to be initialized with a call of **`FrIf_Init`**. > The FlexRay global time is only defined if the FlexRay bus is synchronized. If this function is called while the communication controller is not in POC state normal active, it will return E_NOT_OK. > This function is implemented as macro to reduce code size if the "Wrapper APIs As Macro" option is enabled for the FlexRay Interface.	
Expected Caller Context	
> This function can be called in any context.	

Table 4-20 FrIf_GetGlobalTime

4.2.20 FrIf_GetMacrotickDuration

Prototype
`uint16` **`FrIf_GetMacrotickDuration`**`(uint8 FrIf_CtrlIdx)`

Parameter	
FrIf_CtrlIdx	Index of the FlexRay controller for which the macro tick duration shall be determined.
Return code	
uint16	The duration of a macro tick in nanoseconds.
Functional Description	
This function returns the duration of a macro tick.	
Particularities and Limitations	
> Precondition: The FlexRay Interface has to be initialized with a call of `FrIf_Init`.	
Expected Caller Context	
> This function can be called in any context.	

Table 4-21 FrIf_GetMacrotickDuration

4.2.21 FrIf_GetMacroticksPerCycle

Prototype	
`uint16 FrIf_GetMacroticksPerCycle (uint8 FrIf_CtrlIdx)`	
Parameter	
FrIf_CtrlIdx	Index of the FlexRay controller for which the number of macro ticks per cycle shall be determined.
Return code	
uint16	The number of macro ticks per cycle.
Functional Description	
This function returns the number of macro ticks per cycle.	
Particularities and Limitations	
> Precondition: The FlexRay Interface has to be initialized with a call of `FrIf_Init`.	
Expected Caller Context	
> This function can be called in any context.	

Table 4-22 FrIf_GetMacrotickDuration

4.2.22 FrIf_GetNmVector

Prototype	
`Std_ReturnType FrIf_GetNmVector(uint8 FrIf_CtrlIdx, uint8* FrIf_NmVectorPtr)`	
Parameter	
FrIf_CtrlIdx	Index of the FlexRay communication controller for which Fr_GetNmVector shall be called.
FrIf_NmVectorPtr	Pointer to a memory location where the NM vector will be stored.

Return code	
E_OK	The call of the FlexRay Driver API service Fr_GetNmVector() has returned E_OK.
E_NOT_OK	The call of the FlexRay Driver API service Fr_GetNmVector() has returned E_NOT_OK, or an error has been detected if development error detection is enabled.
Functional Description	
Call the function Fr_GetNmVector of the FlexRay Driver.	
Particularities and Limitations	
> Precondition: The FlexRay Interface has to be initialized with a call of **`FrIf_Init`**. > This API function is only available if the "Get NM Vector Support" has been enabled for FlexRay Interface and FlexRay Driver. > This function is implemented as macro to reduce code size if the "Wrapper APIs As Macro" option is enabled for the FlexRay Interface.	
Expected Caller Context	
> This function can be called in any context.	

Table 4-23 FrIf_GetNmVector

4.2.23 FrIf_GetNumOfStartupFrames

Prototype	
`Std_ReturnType` **`FrIf_GetNumOfStartupFrames`**`(uint8 FrIf_CtrlIdx, uint8* FrIf_NumOfStartupFramesPtr)`	
Parameter	
FrIf_CtrlIdx	Index of the FlexRay communication controller for which Fr_GetNumOfStartupFrames() shall be called.
FrIf_NumOfStartupFramesPtr	Address where the number of startup frames seen within the last even/odd cycle pair shall be stored.
Return code	
E_OK	The call of the FlexRay Driver API service Fr_GetNumOfStartupFrames() has returned E_OK.
E_NOT_OK	The call of the FlexRay Driver API service Fr_GetNumOfStartupFrames() has returned E_NOT_OK, or an error has been detected if development error detection is enabled.
Functional Description	
Call the function Fr_GetNumOfStartupFrames of the FlexRay Driver.	
Particularities and Limitations	
> Precondition: The FlexRay Interface has to be initialized with a call of **`FrIf_Init`**. > This API function is only available if the "Get Num Of Startup Frames Support" has been enabled for FlexRay Interface. > This function is implemented as macro to reduce code size if the "Wrapper APIs As Macro" option is enabled for the FlexRay Interface.	

Expected Caller Context
> This function can be called in any context.

Table 4-24 FrIf_GetNumOfStartupFrames

4.2.24 FrIf_GetPOCStatus

Prototype	
`Std_ReturnType` **`FrIf_GetPOCStatus`**`(uint8 FrIf_CtrlIdx, Fr_POCStatusType* FrIf_POCStatusPtr)`	
Parameter	
FrIf_CtrlIdx	Index of the FlexRay communication controller for which Fr_GetPOCStatus shall be called.
FrIf_POCStatusPtr	Pointer to a memory location where the output value will be stored.
Return code	
E_OK	The call of the FlexRay Driver API service Fr_GetPOCStatus() has returned E_OK.
E_NOT_OK	The call of the FlexRay Driver API service Fr_GetPOCStatus() has returned E_NOT_OK, or an error has been detected if development error detection is enabled.
Functional Description	
Call the function Fr_GetPOCStatus of the FlexRay Driver.	
Particularities and Limitations	
> Precondition: The FlexRay Interface has to be initialized with a call of **`FrIf_Init`**. > This function is implemented as macro to reduce code size if the “Wrapper APIs As Macro” option is enabled for the FlexRay Interface.	
Expected Caller Context	
> This function can be called in any context.	

Table 4-25 FrIf_GetPOCStatus

4.2.25 FrIf_GetState

Prototype	
`Std_ReturnType` **`FrIf_GetState`**`(uint8 FrIf_ClstIdx, FrIf_StateType *FrIf_StatePtr)`	
Parameter	
FrIf_ClstIdx	Index of the FlexRay cluster for which the state of the FlexRay Interface shall be determined.
FrIf_StatePtr	Pointer to a memory location where the retrieved FrIf_State will be stored.
Return code	
E_OK	Function was successfully executed.

E_NOT_OK	Function execution failed due to detected errors.
Functional Description	
Determine the state of the FlexRay Interface for the given cluster which can be either `FRIF_STATE_OFFLINE` or `FRIF_STATE_ONLINE`.	
Particularities and Limitations	
> Precondition: The FlexRay Interface has to be initialized with a call of **`FrIf_Init`**.	
Expected Caller Context	
> This function can be called in any context.	

Table 4-26 FrIf_GetState

4.2.26 FrIf_GetSyncFrameList (optional)

Prototype	
`Std_ReturnType` **`FrIf_GetSyncFrameList`**`(uint8 FrIf_CtrlIdx, uint8 FrIf_ListSize, uint16* FrIf_ChannelAEvenListPtr, int16* FrIf_ChannelBEvenListPtr, int16* FrIf_ChannelAOddListPtr, uint16* FrIf_ChannelBOddListPtr)`	
Parameter	
FrIf_CtrlIdx	Index of the FlexRay communication controller for which Fr_GetSyncFrameList shall be called.
FrIf_ListSize	Size of the arrays passed via parameters: - FrIf_ChannelAEvenListPtr FrIf_ChannelBEvenListPtr - FrIf_ChannelAOddListPtr FrIf_ChannelBOddListPtr. The service must ensure to not write more entries into those arrays than granted by this parameter.
FrIf_ChannelAEvenListPtr	Address the list of sync frames on channel A within the even communication cycle is written to. The exact number of elements written to the list is limited by parameter FrIf_ListSize. Unused list elements are filled with the value '0' to indicate that no more sync frame has been seen.
FrIf_ChannelBEvenListPtr	Address the list of sync frames on channel B within the even communication cycle is written to. The exact number of elements written to the list is limited by parameter FrIf_ListSize. Unused list elements are filled with the value '0' to indicate that no more sync frame has been seen.
FrIf_ChannelAOddListPtr	Address the list of sync frames on channel A within the odd communication cycle is written to. The exact number of elements written to the list is limited by parameter FrIf_ListSize. Unused list elements are filled with the value '0' to indicate that no more sync frame has been seen.

FrIf_ChannelBOddListPtr	Address the list of sync frames on channel B within the odd communication cycle is written to. The exact number of elements written to the list is limited by parameter FrIf_ListSize. Unused list elements are filled with the value '0' to indicate that no more sync frame has been seen.
Return code	
E_OK	The call of the FlexRay Driver API service Fr_GetSyncFrameList() has returned E_OK.
E_NOT_OK	The call of the FlexRay Driver API service Fr_GetSyncFrameList() has returned E_NOT_OK, or an error has been detected if development error detection is enabled.
Functional Description	
According to AUTOSAR 4.0 this function wraps the Fr_GetSyncFrameList() of the FlexRay Driver to read the list of sync frames received in the last two communication cycles and write it to the given arrays.	
Particularities and Limitations	
> Precondition: The FlexRay Interface has to be initialized with a call of `FrIf_Init`. > This API function is only available if the "Get Sync Frame List Support" has been enabled for FlexRay Interface and FlexRay Driver. > The FrIf_ListSize parameter is limited to a maximum of 15 by the FlexRay Driver. > This function is implemented as macro to reduce code size if the "Wrapper APIs As Macro" option is enabled for the FlexRay Interface.	
Expected Caller Context	
> This function can be called in any context.	

Table 4-27 FrIf_GetSyncFrameList (optional)

4.2.27 FrIf_GetTransceiverError

Prototype	
`Std_ReturnType` **`FrIf_GetTransceiverMode`**`(uint8 FrIf_CtrlIdx, Fr_ChannelType FrIf_ChnlIdx, uint8 FrIf_BranchIdx, uint32* FrIf_BusErrorState)`	
Parameter	
FrIf_CtrlIdx	The index of the FlexRay communication controller to which the transceiver is connected.
FrIf_ChnlIdx	The index of the FlexRay channel to which the transceiver is connected.
FrIf_BranchIdx	This zero based index identifies the branch of the (active star) transceiver to which the API call has to be applied.
FrIf_BusErrorState	Address where the transceiver error state is stored.
Return code	
E_OK	The call of the FlexRay Transceiver Driver API service FrTrcv_GetTransceiverError() has returned E_OK.

E_NOT_OK	The call of the FlexRay Transceiver Driver API service FrTrcv_GetTransceiverError() has returned E_NOT_OK, or an error has been detected if development error detection is enabled.
Functional Description	
This function wraps the FrTrcv_GetTransceiverError() function of the FlexRay Transceiver Driver. The enum value "FR_CHANNEL_AB" shall not be used.	
Particularities and Limitations	
> Precondition: The FlexRay Interface has to be initialized with a call of **`FrIf_Init`**. > This API function is only available if the MICROSAR FlexRay Transceiver component is enabled in the configuration tool. > This function is implemented as macro to reduce code size if the "Wrapper APIs As Macro" option is enabled for the FlexRay Interface.	
Expected Caller Context	
> This function can be called in any context.	

Table 4-28 FrIf_GetTransceiverError

4.2.28 FrIf_GetTransceiverMode

Prototype	
`Std_ReturnType` **`FrIf_GetTransceiverMode`**`(uint8 FrIf_CtrlIdx, Fr_ChannelType FrIf_ChnlIdx, FrTrcv_TrcvModeType* FrIf_TrcvModePtr)`	
Parameter	
FrIf_CtrlIdx	The index of the FlexRay communication controller to which the transceiver is connected.
FrIf_ChnlIdx	The index of the FlexRay channel to which the transceiver is connected.
FrIf_TrcvModePtr	The memory location to which the mode shall be written.
Return code	
E_OK	The call of the FlexRay Transceiver Driver API service FrTrcv_GetTransceiverMode() has returned BUSTRCV_E_OK.
E_NOT_OK	The call of the FlexRay Transceiver Driver API service FrTrcv_GetTransceiverMode() has returned BUSTRCV_E_ERROR, or an error has been detected if development error detection is enabled.
Functional Description	
This function wraps the FrTrcv_GetTransceiverMode function of the FlexRay Transceiver Driver.	
Particularities and Limitations	
> Precondition: The FlexRay Interface has to be initialized with a call of **`FrIf_Init`**. > This API function is only available if the MICROSAR FlexRay Transceiver component is enabled in the configuration tool. > This function is implemented as macro to reduce code size if the "Wrapper APIs As Macro" option is enabled for the FlexRay Interface.	
Expected Caller Context	

> This function can be called in any context.

Table 4-29 FrIf_GetTransceiverMode

4.2.29 FrIf_GetTransceiverWUReason

Prototype	
`Std_ReturnType` **`FrIf_GetTransceiverWUReason`**`(uint8 FrIf_CtrlIdx, Fr_ChannelType FrIf_ChnlIdx, FrTrcv_TrcvWUReasonType* FrIf_TrcvWUReasonPtr)`	
Parameter	
FrIf_CtrlIdx	The index of the FlexRay communication controller to which the transceiver is connected.
FrIf_ChnlIdx	The index of the FlexRay channel to which the transceiver is connected.
FrIf_TrcvWUReasonPtr	The memory location to which the wake-up reason shall be written.
Return code	
E_OK	The call of the FlexRay Transceiver Driver API service FrTrcv_GetTransceiverWUReason() has returned BUSTRCV_E_OK.
E_NOT_OK	The call of the FlexRay Transceiver Driver API service FrTrcv_GetTransceiverWUReason() has returned BUSTRCV_E_ERROR, or an error has been detected if development error detection is enabled.
Functional Description	
This function wraps the FrTrcv_GetTransceiverWUReason function of the FlexRay Transceiver Driver.	
Particularities and Limitations	
> Precondition: The FlexRay Interface has to be initialized with a call of **`FrIf_Init`**. > This API function is only available if the MICROSAR FlexRay Transceiver component is enabled in the configuration tool. > This function is implemented as macro to reduce code size if the "Wrapper APIs As Macro" option is enabled for the FlexRay Interface.	
Expected Caller Context	
> This function can be called in any context.	

Table 4-30 FrIf_GetTransceiverWUReason

4.2.30 FrIf_GetWakeupRxStatus

Prototype	
`Std_ReturnType` **`FrIf_GetWakeupRxStatus`**`(uint8 FrIf_CtrlIdx, uint8* FrIf_WakeupRxStatusPtr)`	
Parameter	
FrIf_CtrlIdx	Index of the FlexRay communication controller for which Fr_GetWakeupRxStatus shall be called.

FrIf_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	The call of the FlexRay Driver API service Fr_GetWakeupRxStatus () has returned E_OK.
E_NOT_OK	The call of the FlexRay Driver API service Fr_GetWakeupRxStatus () has returned E_NOT_OK, or an error has been detected if development error detection is enabled.
Functional Description	
Call the function Fr_GetWakeupRxStatus of the FlexRay Driver.	
Particularities and Limitations	
> Precondition: The FlexRay Interface has to be initialized with a call of `FrIf_Init`. > This API function is only available if the "Get Wakeup Rx Status Support" has been enabled for FlexRay Interface. > This function is implemented as macro to reduce code size if the "Wrapper APIs As Macro" option is enabled for the FlexRay Interface.	
Expected Caller Context	
> This function can be called in any context.	

Table 4-31 FrIf_GetWakeupRxStatus

4.2.31 FrIf_GetTxConflictStatus (optional)

Prototype	
`Std_ReturnType **FrIf_GetTxConflictStatus**(uint8 FrIf_CtrlIdx, uint16 FrIf_LPduIdx, Fr_TxConflictStatusType* FrIf_TxConflictStatusPtr)`	
Parameter	
FrIf_CtrlIdx	Index of the FlexRay communication controller for which Fr_GetTxConflictStatus shall be called.
FrIf_LPduIdx	This index is used to uniquely identify a FlexRay frame.
FrIf_TxConflictStatusPtr	Address where the TxConflict status information shall be stored.
Return code	
E_OK	The call of the FlexRay Driver API service Fr_GetTxConflictStatus() has returned E_OK.
E_NOT_OK	The call of the FlexRay Driver API service Fr_GetTxConflictStatus() has returned E_NOT_OK, or an error has been detected if development error detection is enabled.
Functional Description	
This function wraps the Fr_GetTxConflictStatus function of the FlexRay Driver and gets the TxConflict status information for the given LPdu.	

Particularities and Limitations
> Precondition: The FlexRay Interface must be initialized with a call of **`FrIf_Init`**. > This API function is only available if the "Get Tx Conflict Status" has been enabled for FlexRay Interface and FlexRay Driver. > This function is implemented as macro to reduce code size if the "Wrapper APIs As Macro" option is enabled for the FlexRay Interface.
Expected Caller Context
> This function can be called in any context.

Table 4-32 FrIf_GetTxConflictStatus

4.2.32 FrIf_HaltCommunication

Prototype	
`Std_ReturnType` **`FrIf_HaltCommunication`**`(uint8 FrIf_CtrlIdx)`	
Parameter	
FrIf_CtrlIdx	Index of the FlexRay communication controller for which Fr_HaltCommunication shall be called.
Return code	
E_OK	The call of the FlexRay Driver API service Fr_HaltCommunication() has returned E_OK.
E_NOT_OK	The call of the FlexRay Driver API service Fr_HaltCommunication() has returned E_NOT_OK, or an error has been detected if development error detection is enabled.
Functional Description	
Call the function Fr_HaltCommunication of the FlexRay Driver.	
Particularities and Limitations	
> Precondition: The FlexRay Interface has to be initialized with a call of **`FrIf_Init`**. > This function is implemented as macro to reduce code size if the "Wrapper APIs As Macro" option is enabled for the FlexRay Interface.	
Expected Caller Context	
> This function can be called in any context.	

Table 4-33 FrIf_HaltCommunication

4.2.33 FrIf_Init

Prototype	
`void` **`FrIf_Init`**`(const FrIf_ConfigType *FrIf_ConfigPtr)`	
Parameter	
FrIf_ConfigPtr	Pointer to the post-build configuration data structure of the FlexRay Interface. If the configuration variant pre-compile is used, the pointer given to **`FrIf_Init`** is ignored.

Return code	
Void	-
Functional Description	
This function is used to initialize the FlexRay Interface. The configuration data that shall be used by the FlexRay Interface is passed as parameter.	
Particularities and Limitations	
> If this function is called during active communication, the communication of all FlexRay communication controllers will be terminated.	
Expected Caller Context	
> This function can be called in any context.	

Table 4-34 FrIf_Init

4.2.34 FrIf_InitMemory

Prototype	
`void FrIf_InitMemory(void)`	
Parameter	
Void	-
Return code	
Void	-
Functional Description	
This function is used to initialize the global variables of the FlexRay Interface at startup.	
Particularities and Limitations	
> This function shall be called at startup before `FrIf_Init`.	
Expected Caller Context	
> This function can be called in any context.	

Table 4-35 FrIf_InitMemory

4.2.35 FrIf_JobListExec_<ClstIdx>

Prototype	
`void FrIf_JobListExec_<ClstIdx> (void)`	
Parameter	
Void	-
Return code	
Void	-

Functional Description
This is the common call-back function for the activation of the Rx- or Tx-tasks (job execution) of the FlexRay Interface. Depending on the current FlexRay global time and the configured start times of the Rx- and Tx-tasks, the FlexRay Interface will either execute the Rx- or Tx-task of the corresponding FlexRay Cluster.
Particularities and Limitations
> Precondition: The FlexRay Interface has to be initialized with a call of **`FrIf_Init`**.   > During its runtime this function temporarily disables all interrupts.
Expected Caller Context
> This function must be called by the application or integration code either directly in the context of the timer interrupt or in the context of an OS task that is activated in the context of the timer interrupt.

Table 4-36 FrIf_JobListExec_<ClstIdx>

4.2.36 FrIf_MainFunction_<ClstIdx>

Prototype	
`void` **`FrIf_MainFunction_<ClstIdx>`**`(void)`	
Parameter	
Void	-
Return code	
Void	-
Functional Description	
This function checks whether the job execution of its related FlexRay Cluster is synchronized.	
Particularities and Limitations	
> Precondition: The FlexRay Interface has to be initialized with a call of **`FrIf_Init`**.	
Expected Caller Context	
> This function is called cyclically by the BSW Scheduler.	

Table 4-37 FrIf_MainFunction_<ClstIdx>

4.2.37 FrIf_ReadCCConfig (optional)

Prototype	
`Std_ReturnType` **`FrIf_ReadCCConfig`**`(uint8 FrIf_CtrlIdx, uint8 FrIf_CCLLParamIndex, uint32* FrIf_CCLLParamValue)`	
Parameter	
FrIf_CtrlIdx	Index of the FlexRay communication controller for which Fr_SendWUP shall be called.
FrIf_CCLLParamIndex	This index selects the low level parameter value that shall be copied. Refer to Fr_GeneralTypes.h for the list of supported low level parameters.

FrIf_CCLLParamValue	Value of the read parameter.
Return code	
E_OK	The call of the FlexRay Driver API service Fr_ReadCCConfig() has returned E_OK.
E_NOT_OK	The call of the FlexRay Driver API service Fr_ReadCCConfig() has returned E_NOT_OK, or an error has been detected if development error detection is enabled.
Functional Description	
This function wraps the Fr_ReadCCConfig of the FlexRay Driver to read the CC configuration registers.	
Particularities and Limitations	
> Precondition: The FlexRay Interface has to be initialized with a call of **`FrIf_Init`**. > This API function is only available if the "Read CC Config Support" has been enabled for FlexRay Interface and FlexRay Driver. > This function is implemented as macro to reduce code size if the "Wrapper APIs As Macro" option is enabled for the FlexRay Interface.	
Expected Caller Context	
> This function can be called in any context.	

Table 4-38 FrIf_ReadCCConfig (optional)

4.2.38 FrIf_ReconfigLPdu (optional)

Prototype	
`Std_ReturnType` **`FrIf_ReconfigLPdu`**`(uint8 FrIf_CtrlIdx, int16 FrIf_LPduIdx, int16 FrIf_FrameId, Fr_ChannelType FrIf_ChnlIdx, int8 FrIf_CycleRepetition, int8 FrIf_CycleOffset, uint8 FrIf_PayloadLength, int16 FrIf_HeaderCRC )`	
Parameter	
FrIf_CtrlIdx	Index of the FlexRay communication controller for which Fr_ReconfigLPdu shall be called.
FrIf_LPduIdx	This index is used to uniquely identify a FlexRay frame.
FrIf_FrameId	FlexRay Frame ID the FrIf_LPdu shall be configured to.
FrIf_ChnlIdx	FlexRay Channel the FrIf_LPdu shall be configured to.
FrIf_CycleRepetition	Cycle Repetition part of the cycle filter mechanism FrIf_LPdu shall be configured to.
FrIf_CycleOffset	Cycle Offset part of the cycle filter mechanism FrIf_LPdu shall be configured to.
FrIf_PayloadLength	Payloadlength in units of bytes the FrIf_LPduIdx shall be configured to.
FrIf_HeaderCRC	Header CRC the FrIf_LPdu shall be configured to.
Return code	
E_OK	The call of the FlexRay Driver API service Fr_ReconfigLPdu() has returned E_OK.

E_NOT_OK	The call of the FlexRay Driver API service Fr_ReconfigLPdu() has returned E_NOT_OK, or an error has been detected if development error detection is enabled.
Functional Description	
According to AUTOSAR 4.0 this function wraps the Fr_ReconfigLPdu of the FlexRay Driver to reconfigure the corresponding HW buffer according the given parameters during runtime.	
Particularities and Limitations	
> Precondition: The FlexRay Interface has to be initialized with a call of `FrIf_Init`. > This API function is only available if the "Reconfig LPdu Support" has been enabled for FlexRay Interface and FlexRay Driver. > Only the L-PDUs with the "Reconfigurable" flag enabled can be reconfigured by this API. > If L-PDU reconfiguration is enabled for a specific L-PDU, this L-PDU is not sent after FrIf_ControllerInit but has to be enabled with this function. Therewith it is not allowed to set the "Reconfigurable" flag for a sync frame. > This function is implemented as macro to reduce code size if the "Wrapper APIs As Macro" option is enabled for the FlexRay Interface.	
Expected Caller Context	
> This function can be called in any context.	

Table 4-39 FrIf_ReconfigLPdu (optional)

4.2.39 FrIf_SendWUP

Prototype	
`Std_ReturnType FrIf_SendWUP(uint8 FrIf_CtrlIdx)`	
Parameter	
FrIf_CtrlIdx	Index of the FlexRay communication controller for which Fr_SendWUP shall be called.
Return code	
E_OK	The call of the FlexRay Driver API service Fr_SendWUP() has returned E_OK.
E_NOT_OK	The call of the FlexRay Driver API service Fr_SendWUP() has returned E_NOT_OK, or an error has been detected if development error detection is enabled.
Functional Description	
Call the function Fr_SendWUP of the FlexRay Driver.	
Particularities and Limitations	
> Precondition: The FlexRay Interface has to be initialized with a call of `FrIf_Init`. > This function is implemented as macro to reduce code size if the "Wrapper APIs As Macro" option is enabled for the FlexRay Interface.	
Expected Caller Context	
> This function can be called in any context.	

Table 4-40 FrIf_SendWUP

4.2.40 FrIf_SetAbsoluteTimer

Prototype	
`Std_ReturnType **FrIf_SetAbsoluteTimer**(uint8 FrIf_CtrlIdx, uint8 FrIf_AbsTimerIdx, uint8 FrIf_Cycle, uint16 FrIf_Offset)`	
Parameter	
FrIf_CtrlIdx	The index of the FlexRay Communication Controller.
FrIf_AbsTimerIdx	Index of the absolute timer to be used for setting the alarm time.
FrIf_Cycle	Cycle in which the timer shall expire.
FrIf_Offset	Offset to the cycle start in macro ticks.
Return code	
E_OK	The call of the FlexRay Driver API service Fr_SetAbsoluteTimer() has returned E_OK.
E_NOT_OK	The call of the FlexRay Driver API service Fr_SetAbsoluteTimer() has returned E_NOT_OK, or an error has been detected if development error detection is enabled.
Functional Description	
This function wraps the Fr_SetAbsoluteTimer function of the FlexRay Driver.	
Particularities and Limitations	
> Precondition: The FlexRay Interface has to be initialized with a call of **`FrIf_Init`**. > As one FlexRay timer is used by the FlexRay Interface this function must not be used for timer 0. > The FlexRay global time is only defined if the FlexRay bus is synchronized. > This function is implemented as macro to reduce code size if the "Wrapper APIs As Macro" option is enabled for the FlexRay Interface.	
Expected Caller Context	
> This function can be called in any context.	

Table 4-41 FrIf_SetAbsoluteTimer

4.2.41 FrIf_SetState

Prototype	
`Std_ReturnType **FrIf_SetState**(uint8 FrIf_ClstIdx, FrIf_StateTransitionType FrIf_StateTransition )`	
Parameter	
FrIf_ClstIdx	Index of the FlexRay cluster for which the state of the FlexRay Interface shall be determined.
FrIf_StateTransition	Requested state transition, i.e. either `FRIF_GOTO_OFFLINE` or `FRIF_GOTO_ONLINE`

Return code	
E_OK	Function was successfully executed.
E_NOT_OK	Function execution failed due to detected errors.
Functional Description	
Change the state of the FlexRay Interface for the given cluster. > `FRIF_GOTO_ONLINE` will start the FrIf job list execution. ■ `FRIF_GOTO_OFFLINE` will stop the FrIf job list execution. Note that the state of the FlexRay communication controller is not influenced by this function. I.e. the function **`FrIf_HaltCommunication`** has to be used to actually stop the FlexRay communication.	
Particularities and Limitations	
> Precondition: The FlexRay Interface has to be initialized with a call of **`FrIf_Init`**. > The FlexRay Interface can only be set to `FRIF_STATE_ONLINE` after synchronization of the FlexRay bus has been achieved.	
Expected Caller Context	
> This function can be called in any context.	

Table 4-42 FrIf_SetState

4.2.42 FrIf_SetTransceiverMode

Prototype	
`Std_ReturnType` **`FrIf_SetTransceiverMode`**`(uint8 FrIf_CtrlIdx, Fr_ChannelType FrIf_ChnlIdx, FrTrcv_TrcvModeType FrIf_TrcvMode)`	
Parameter	
FrIf_CtrlIdx	The index of the FlexRay communication controller to which the transceiver is connected.
FrIf_ChnlIdx	The index of the FlexRay channel to which the transceiver is connected.
FrIf_TrcvMode	The mode that shall be set.
Return code	
E_OK	The call of the FlexRay Transceiver Driver API service FrTrcv_SetTransceiverMode() has returned BUSTRCV_E_OK.
E_NOT_OK	The call of the FlexRay Transceiver Driver API service FrTrcv_SetTransceiverMode() has returned BUSTRCV_E_ERROR, or an error has been detected if development error detection is enabled.
Functional Description	
This function wraps the FrTrcv_SetTransceiverMode function of the FlexRay Transceiver Driver.	

Particularities and Limitations
> Precondition: The FlexRay Interface has to be initialized with a call of **`FrIf_Init`**. > This API function is only available if the MICROSAR FlexRay Transceiver component is enabled in the configuration tool. > This function is implemented as macro to reduce code size if the "Wrapper APIs As Macro" option is enabled for the FlexRay Interface.
Expected Caller Context
> This function can be called in any context.

Table 4-43 FrIf_SetTransceiverMode

4.2.43 FrIf_SetWakeupChannel

Prototype	
`Std_ReturnType` **`FrIf_SetWakeupChannel`**`(uint8 FrIf_CtrlIdx, Fr_ChannelType FrIf_ChnlIdx)`	
Parameter	
FrIf_CtrlIdx	Index of the FlexRay communication controller for which Fr_SetWakeupChannel shall be called.
FrIf_ChnlIdx	Index of the FlexRay channel.
Return code	
E_OK	The call of the FlexRay Driver API service Fr_SetWakeupChannel() has returned E_OK.
E_NOT_OK	The call of the FlexRay Driver API service Fr_SetWakeupChannel() has returned E_NOT_OK, or an error has been detected if development error detection is enabled.
Functional Description	
Call the function Fr_SetWakeupChannel of the FlexRay Driver.	
Particularities and Limitations	
> Precondition: The FlexRay Interface has to be initialized with a call of **`FrIf_Init`**. > This API function is only available if "SetWakeupChannel Disable" has not been disabled for FlexRay Interface. > This function is implemented as macro to reduce code size if the "Wrapper APIs As Macro" option is enabled for the FlexRay Interface.	
Expected Caller Context	
> This function can be called in any context.	

Table 4-44 FrIf_SetWakeupChannel

4.2.44 FrIf_StartCommunication

Prototype	
`Std_ReturnType` **`FrIf_StartCommunication`**`(uint8 FrIf_CtrlIdx)`	
Parameter	
FrIf_CtrlIdx	Index of the FlexRay communication controller for which Fr_StartCommunication shall be called.
Return code	
E_OK	The call of the FlexRay Driver API service Fr_StartCommunication() has returned E_OK.
E_NOT_OK	The call of the FlexRay Driver API service Fr_StartCommunication() has returned E_NOT_OK, or an error has been detected if development error detection is enabled.
Functional Description	
Call the function Fr_StartCommunication of the FlexRay Driver.	
Particularities and Limitations	
> Precondition: The FlexRay Interface has to be initialized with a call of **`FrIf_Init`**. > This function is implemented as macro to reduce code size if the "Wrapper APIs As Macro" option is enabled for the FlexRay Interface.	
Expected Caller Context	
> This function can be called in any context.	

Table 4-45 FrIf_StartCommunication

4.2.45 FrIf_Transmit

Prototype	
`Std_ReturnType` **`FrIf_Transmit`**`(PduIdType FrIf_TxPduId, const PduInfoType * FrIf_PduInfoPtr)`	
Parameter	
FrIf_TxPduId	FrIf-ID of the PDU to be transmitted.
PduInfoPtr	Pointer to a structure with FlexRay PDU related data.
Return code	
E_OK	The immediate transmission request has been accepted.
E_NOT_OK	The transmission request has been rejected, or an error has been detected if development error detection is enabled.
Functional Description	
This function initiates the transmission of a PDU. If the PDU is configured for Decoupled Transmission, the PDU is marked as dirty. When a frame that contains the PDU shall be transmitted, the <UL>_TriggerTransmit call-back function of the upper-layer component will be called in order to get the PDU content. If the PDU is configured for Immediate Transmission the PDU content is directly given to the FlexRay Driver.	

Particularities and Limitations
> Precondition: The FlexRay Interface has to be initialized with a call of `FrIf_Init`. > During its runtime this function temporarily disables all interrupts.
Expected Caller Context
> This function can be called in any context.

Table 4-46 FrIf_Transmit

4.3 Services used by FrIf

In the following table services provided by other components, which are used by the **FrIf** are listed. For details about prototype and functionality refer to the documentation of the providing component.

Component	API
Fr	`Fr_AbortCommunication`
Fr	`Fr_AckAbsoluteTimerIRQ`
Fr	`Fr_AllowColdstart`
Fr	`Fr_CancelAbsoluteTimer`
Fr	`Fr_CheckTxLPduStatus`
Fr	`Fr_ControllerInit`
Fr	`Fr_DisableLPdu`
Fr	`Fr_DisableAbsoluteTimerIRQ`
Fr	`Fr_EnableAbsoluteTimerIRQ`
Fr	`Fr_GetAbsoluteTimerIRQStatus`
Fr	`Fr_GetChannelStatus`
Fr	`Fr_GetClockCorrection`
Fr	`Fr_GetGlobalTime`
Fr	`Fr_GetNmVector`
Fr	`Fr_GetPOCStatus`
Fr	`Fr_GetSyncFrameList`
Fr	`Fr_HaltCommunication`
Fr	`Fr_PrepareLPdu`
Fr	`Fr_ReadCCConfig`
Fr	`Fr_ReceiveRxLPdu`
Fr	`Fr_ReconfigLPdu`
Fr	`Fr_SendWUP`
Fr	`Fr_AllSlots`
Fr	`Fr_GetNumOfStartupFrames`
Fr	`Fr_GetWakeupRxStatus`
Fr	`Fr_SetAbsoluteTimer`
Fr	`Fr_SetWakeupChannel`
Fr	`Fr_StartCommunication`
Fr	`Fr_TransmitTxLPdu`

Component	API
Fr	`Fr_GetTxConflictStatus`
FrTrcv	`FrTrcv_ClearTransceiverWakeup`
FrTrcv	`FrTrcv_GetTransceiverMode`
FrTrcv	`FrTrcv_DisableTransceiverBranch`
FrTrcv	`FrTrcv_EnableTransceiverBranch`
FrTrcv	`FrTrcv_CheckWakeupByTransceiver`
FrTrcv	`FrTrcv_GetTransceiverError`
FrTrcv	`FrTrcv_GetTransceiverWUReason`
FrTrcv	`FrTrcv_SetTransceiverMode`
Det	`Det_ReportError`
Dem	`Dem_ReportErrorStatus`

Table 4-47 Services used by the **FrIf**

4.4 Callback Functions

The FlexRay Interface does not provide any callback function.

4.5 Configurable Interfaces

At its configurable interfaces the **FrIf** defines notifications that can be mapped to callback functions provided by other modules. The mapping is not statically defined by the BSW module but can be performed at configuration time. The function prototypes that can be used for the configuration have to match the appropriate function prototype signatures, which are described in the following tables.

4.5.1 Notifications

The MICROSAR **FrIf** does not use any notifications

4.5.2 Callout Functions

4.5.2.1 <UL>_TriggerTransmit

Prototype	
`Std_ReturnType` **`<UL>_TriggerTransmit`**`(PduIdType TxPduId, PduInfoType* PduInfoPtr)`	
Parameter	
TxPduId (in)	PDU-ID of FlexRay PDU that shall be copied to the FrIf
PduInfoPtr (inout)	Contains a pointer to a buffer (SduDataPtr) to where the SDU shall be copied to. On return, the service will indicate the length of the copied SDU data in SduLength.
Return code	
E_OK	SDU has been copied and SduLength indicates the number of copied bytes.
E_NOT_OK	No SDU has been copied. PduInfoPtr must not be used since it may contain a NULL pointer or point to invalid data.
Functional Description	
The FlexRay Interface calls this function to request the PDU content from the upper layer software module. For the different upper layer software module, the following function names are used: ▪ `FrNm_TriggerTransmit` ▪ `FrTp_TriggerTransmit` ▪ `PduR_FrIfTriggerTransmit` ▪ `Xcp_FrIfTriggerTransmit` ▪ `FrTSyn_TriggerTransmit` ▪ `FrArTp_TriggerTransmit`	
Particularities and Limitations	
Call Context	
▪ Depending on how the FrIf job list function is activated (see section 2.4.2) this function can be called in interrupt context.	

Table 4-48 <UL>_TriggerTransmit ("Use Pdu Info Type" enabled)

4.5.2.2 <UL>_TxConfirmation

Prototype	
`void <UL>_TxConfirmation( PduIdType TxPduId)`	
Parameter	
TxPduId	The ID of the PDU in the upper layer software module.
Return code	
Void	-
Functional Description	
The FlexRay Interface calls this function to confirm the transmission of a PDU. For the different upper layer software module, the following function names are used: ▪ `FrNm_TxConfirmation` ▪ `FrTp_TxConfirmation` ▪ `PduR_FrIfTxConfirmation` ▪ `Xcp_FrIfTxConfirmation` ▪ `FrArTp_TxConfirmation`	
Particularities and Limitations	
-	
Call Context	
▪ Depending on how the FrIf job list function is activated (see section 2.4.2) this function can be called in interrupt context.	

Table 4-49 <UL>_TxConfirmation

4.5.2.3 <UL>_RxIndication

Prototype	
`void <UL>_RxIndication(PduIdType RxPduId, const PduInfoType* PduInfoPtr)`	
Parameter	
RxPduId (in)	PDU-ID of FlexRay PDU that has been received
PduInfoPtr (in)	Contains the length (SduLength) of the received I-PDU and a pointer to a buffer (SduDataPtr) containing the I-PDU.
Return code	
void	-
Functional Description	
The FlexRay Interface calls this function to indicate the reception of a PDU. For the different upper layer software module, the following function names are used: ■ `FrNm_RxIndication` ■ `FrTp_RxIndication` ■ `PduR_FrIfRxIndication` ■ `Xcp_FrIfRxIndication` ■ `FrTSyn_RxIndication` ■ `FrArTp_RxIndication`	
Particularities and Limitations	
Call Context	
■ Depending on how the FrIf job list function is activated (see section 2.4.2) this function can be called in interrupt context.	

Table 4-50 <UL>_RxIndication ("Use Pdu Info Type" enabled)

4.5.2.4 Rx Voting Function (optional for Dual Channel Redundancy Support)

Prototype	
`Std_ReturnType <FrIfRxVotingFunction>(P2CONST(P2VAR(PduInfoType, AUTOMATIC, FRIF_VAR_NOINIT), AUTOMATIC, FRIF_DATA) PduInfo, CONST(uint8, FRIF_CONST) NumberOfPdus, P2VAR(uint8, AUTOMATIC, FRIF_DATA) SelectedPduIndex)`	
Parameter	
`PduInfo`	PduInfo array holds a list of received PDU instances for one PDU of a redundant frame
NumberOfPdus	NumberOfPdus indicates the size of the PduInfo array
SelectedPduIndex	SelectedPduIndex is set by application in order to select the PDU instance that shall be indicated to the upper layer.

Return code	
E_OK	The call of the service <FrIfRxVotingFunction> returns E_OK if the application was able to select a valid PDU that shall be indicated.
E_NOT_OK	The call of the service <FrIfRxVotingFunction> returns E_NOT_OK if the application was not able to select a valid PDU that shall be indicated (e.g. NumberOfPdus is zero)
Functional Description	
Voting function for dual channel redundancy. The FlexRay Interface calls this function to indicate the reception of redundant PDUs.	
Particularities and Limitations	
> This API function is only used by the FlexRay Interface if the "Dual Channel Redundancy Support" is enabled and at least one pair of redundant reception frames is configured (refer to "Redundant Frame Triggering Ref" attribute). > If the "Dual Channel Redundancy Support" is enabled the name of this API service can be configured by setting the "Rx Voting Function" attribute. For the declaration of this voting function the FlexRay interface includes the configured "Rx Voting Function Header File".	
Call Context	
■ Depending on how the FrIf job list function is activated (see section 2.4.2) this function can be called in interrupt context.	

Table 4-51 <FrIfRxVotingFunction>

4.5.3 Complex Device Driver Callout Functions

Instead of calling the <UL>_TriggerTransmit, <UL>_TxConfirmation and <UL>_RxIndication functions of the upperlayer modules FrNm, FrTp, FrXcp or PduR, the MICROSAR FlexRay Interface can call any user defined application callout function for all PDUs which "Pdu Owner" is CDD.

5 Glossary and Abbreviations

5.1 Glossary

Term	Description
GENy	Generation tool for CANbedded and MICROSAR components
CFG5	Generation tool for MICROSAR4 components
If_AsrIfFr	Vector Informatik component name of the MICROSAR FlexRay Interface module

Table 5-1 Glossary

5.2 Abbreviations

Abbreviation	Description
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSW	Basis Software
CDD	Complex Device Driver
CRC	Cyclic Redundancy Check
DEM	Diagnostic Event Manager
DET	Development Error Tracer
EAD	Embedded Architecture Designer
ECU	Electronic Control Unit
ECUC	ECU Configuration
ECUM	ECU Manager
ESCANXXXXXXXX	Vector PES Clearquest Database ID. Replace XXXXXXXX by the numeric identifier.
FIBEX	Field Bus Exchange
FR	FlexRay Driver
FRIF	FlexRay Interface
FRNM	FlexRay Network Management
FRSM	FlexRay State Manager
FRTRCV	FlexRay Transceiver Driver
FRTP	FlexRay Transport Protocol
HIS	Hersteller Initiative Software
ISR	Interrupt Service Routine
FlexRay L-PDU	Synonym - „FlexRay Frame“: A structure used by the communication system to exchange information within the system. A FlexRay Frame consists of a header segment, a payload segment and a trailer segment. The payload segment is used to convey application data.
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)

OEM	Original Equipment Manufacturer
OS	Operating System
PDU	Protocol Data Unit
PDUR	PDU Router
SCHM	Basic Software Scheduler
SRS	Software Requirement Specification
SWS	Software Specification
TP	Transport Protocol
UL	Upper layer component of the FlexRay Interface (FrNm, FrTp, FrXcp, PduR or CDD)

Table 5-2 Abbreviations

6 Contact

Visit our website for more information on

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

www.vector-informatik.com